Rationale

Allen Dutoit
dutoit@in.tum.de
Technische Universität München
Institut für Informatik
Lehrstuhl für Angewandte Softwaretechnik (Prof. Bruegge)

An aircraft example

A320

- First fly-by-wire passenger aircraft
- 150 seats, short to medium haul

A319 & A321

- Derivatives of A320
- Same handling as A320

Design rationale

- Reduce pilot training & maintenance costs
- Increase flexibility for airline

An aircraft example (2)

A330 & A340

- Long haul and ultra long haul
- 2x seats, 3x range
- Similar handling than A320 family

Design rationale

- With minimum cross training, A320 pilots can be certified to fly A330 and A340 airplanes

Consequence

- Any change in these five airplanes must maintain this similarity

Another example

When buying a coffee in the Cafeteria, four types of currency can be used:

- Cash
- Tokens
- “Essenmarke”
- Card

Why? What is the reasoning that lead to such a system?

Overview: rationale

- What is rationale?
- Why should you care?
- Centralized traffic control
- Representing rationale
- Capturing rationale
- Maintaining rationale
- Open issues

- Questions?

What is rationale?

Rationale is the reasoning that lead to the system.

Rationale includes:

- the ***issues*** that were addressed,
- the ***alternatives*** that were considered,
- the ***decisions*** that were made to resolve the issues,
- the ***criteria*** that were used to guide decisions, and
- the ***debate*** developers went through to reach a decision.

Why is rationale important in software engineering?

Many software systems are like the currency system of the Mensa:

They result from a large number of decisions taken over an extended period of time.

- Evolving assumptions
- Legacy decisions
- Conflicting criteria

-> high maintenance cost

-> loss & rediscovery of information

Uses of rationale in software engineering

- Improve design support
  - Avoid duplicate evaluation of poor alternatives
  - Make consistent and explicit trade-offs
- Improve documentation support
  - Makes it easier for non developers (e.g., managers, lawyers, technical writers) to review the design
- Improve maintenance support
  - Provide maintainers with design context
- Improve learning
  - New staff can learn the design by replaying the decisions that produced it

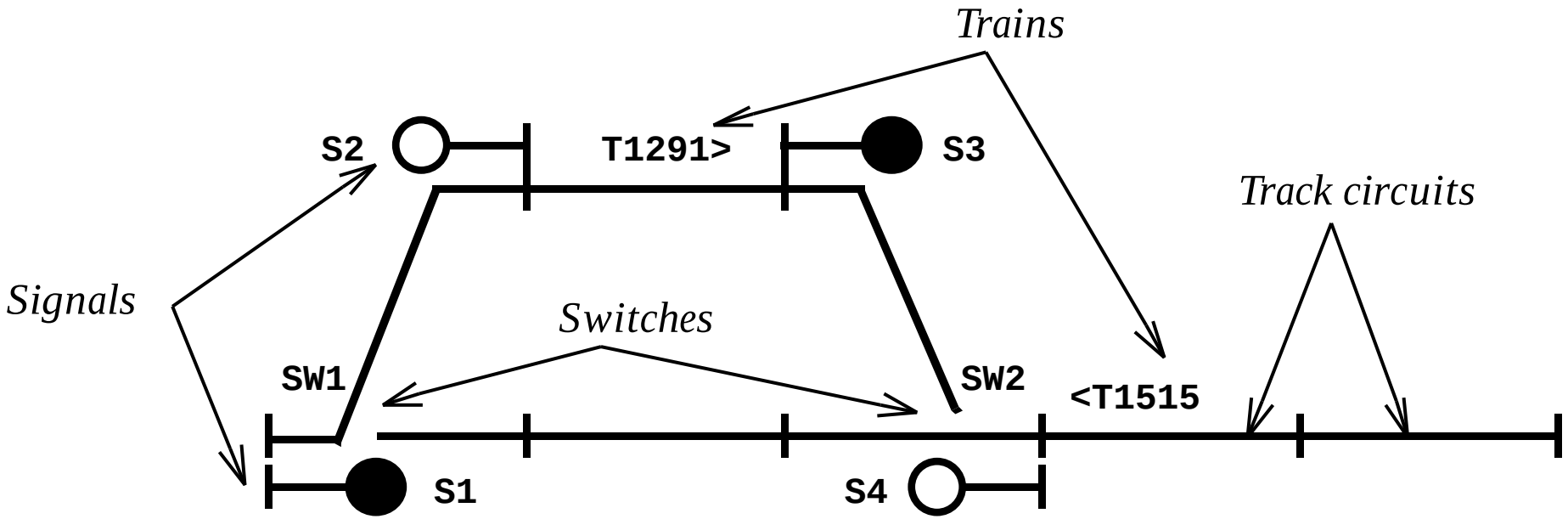
Example: sort algorithm

Requirements:	what should the system do?
Design:	how should it do it?
Rationale:	why does it dot it the way it does?

Rationale includes:

Decisions	`Let's use insert_sort`
Justifications	`The data are quasi sorted`
Alternatives	`quick_sort` `bubble_sort`
Tradeoffs	`worst vs. common case` `speed vs. space`
Argumentation	`Quick_sort performs badly` `on quasi sorted data`

Centralized traffic control



- CTC systems enable dispatchers to monitor and control trains remotely
- CTC allows the planning of routes and replanning in case of problems

Centralized traffic control (2)

CTC systems are ideal examples of rationale capture:

- Long lived systems (some systems include relays installed last century)
  - Extended maintenance life cycle
- Although not life critical, downtime is expensive
  - Low tolerance for bugs
  - Transition to mature technology

Representing rationale

Many media and forms are available for representing rationale information:

- Video & audio
- Transcripts
- Online communication traffic
- Paper

- Communication records
- Design documentation
- Argumentation

Representing rationale: issue models

Argumentation is the most promising approach so far:

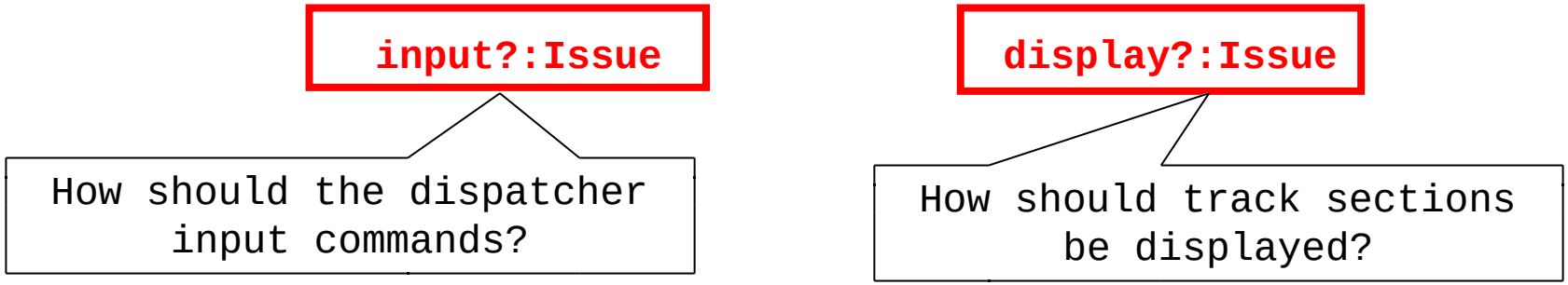
- More information than document: captures trade-offs and discarded alternatives that design documents do not.
- Less messy than communication records: communication records contain everything.

Issue models represent arguments in a semi structure form:

- Nodes represent argument steps
- Links represent their relationships

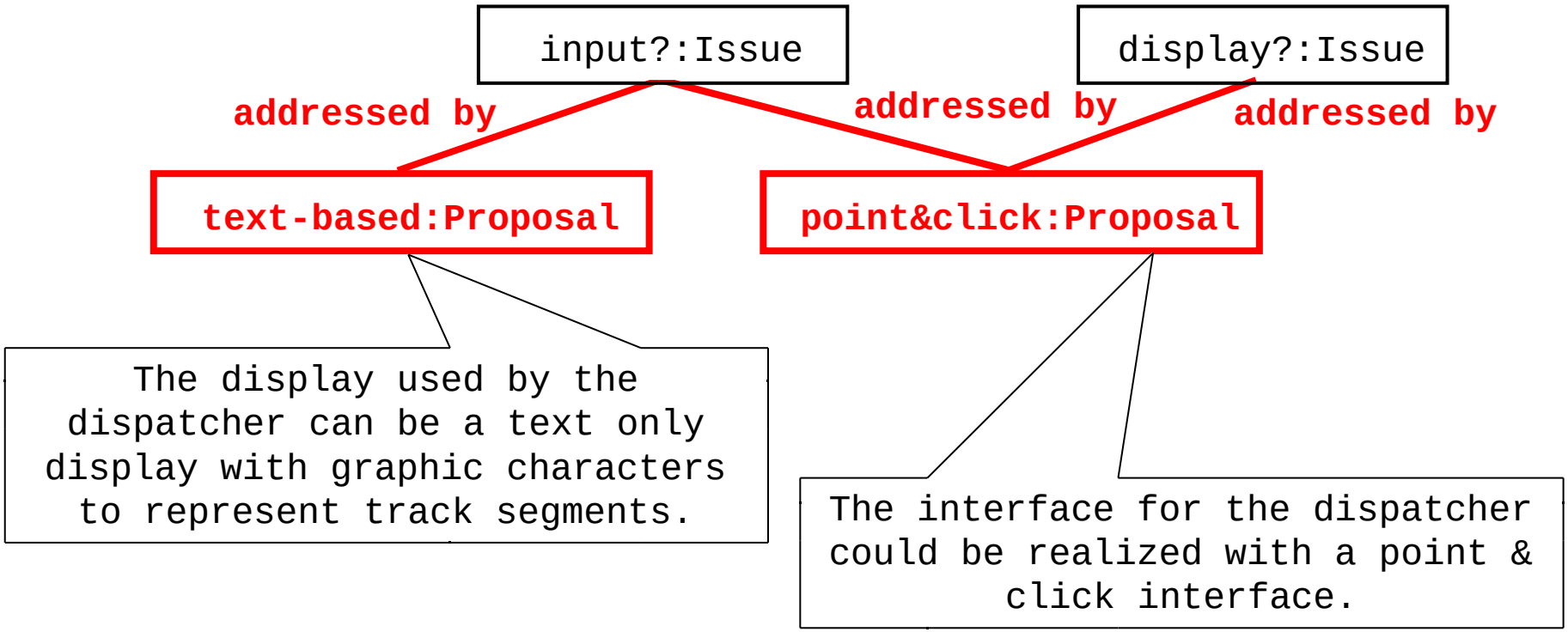
Issues

- Issues are concrete problem which usually do not have a unique, correct solution.
- Issues are phrased as questions.



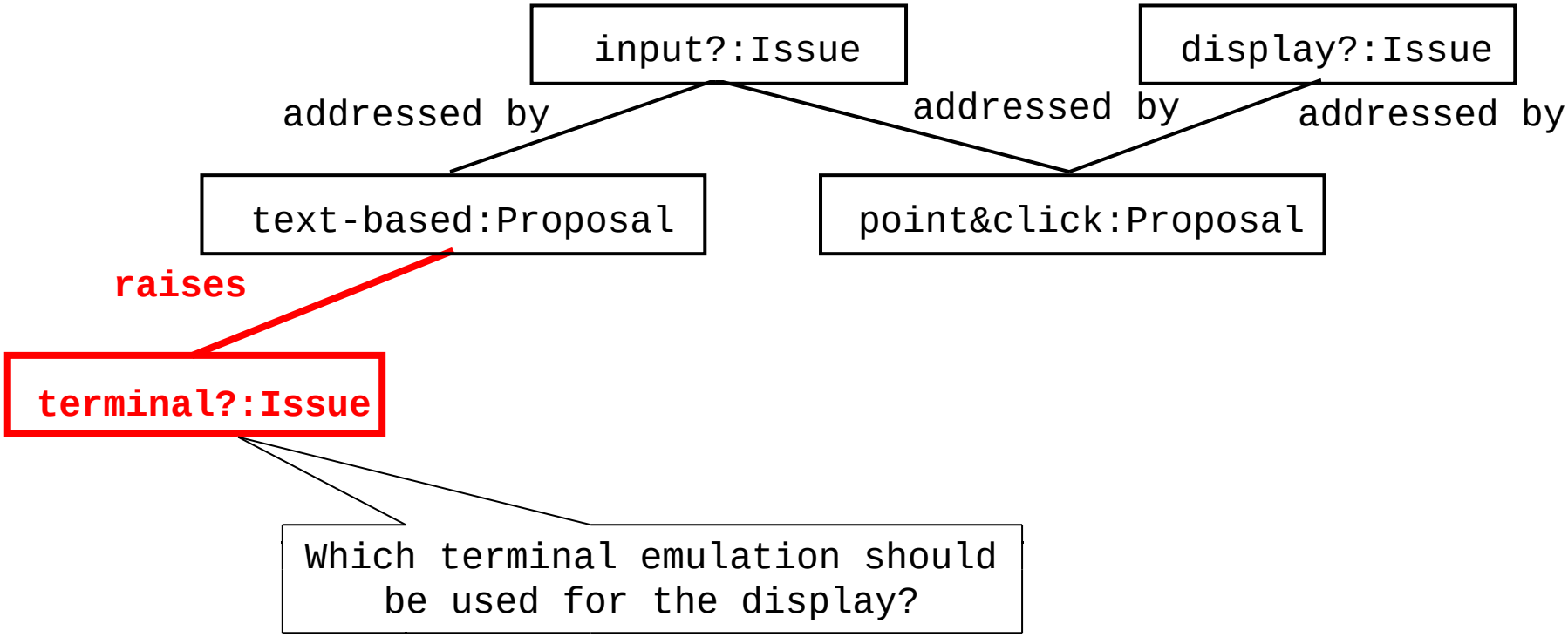
Proposals

- Proposals are possible alternatives to issues.
- One proposal can be shared across multiple issues.



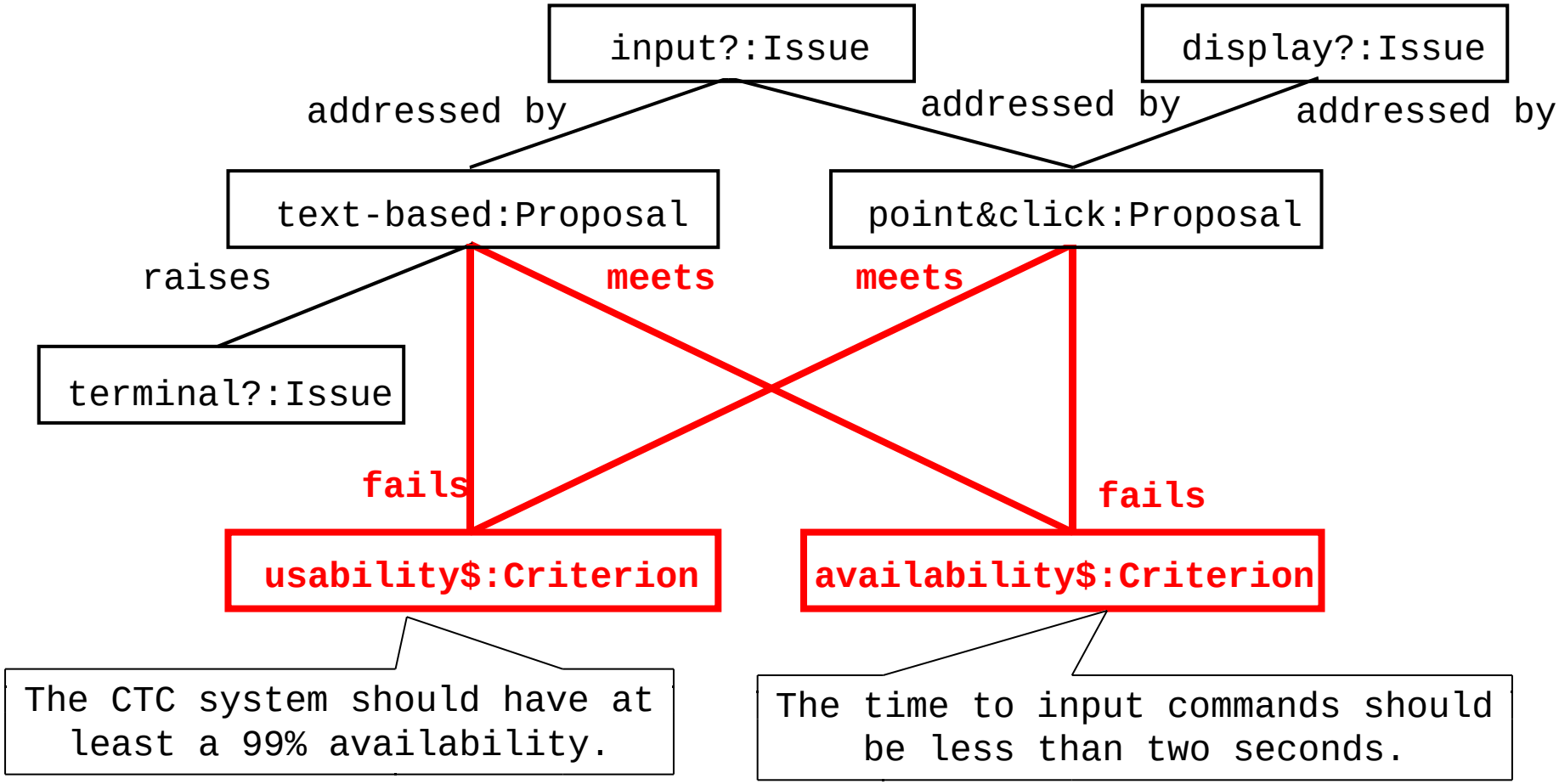
Consequent issue

- Consequent issues are issues raised by the introduction of a proposal.



Criteria

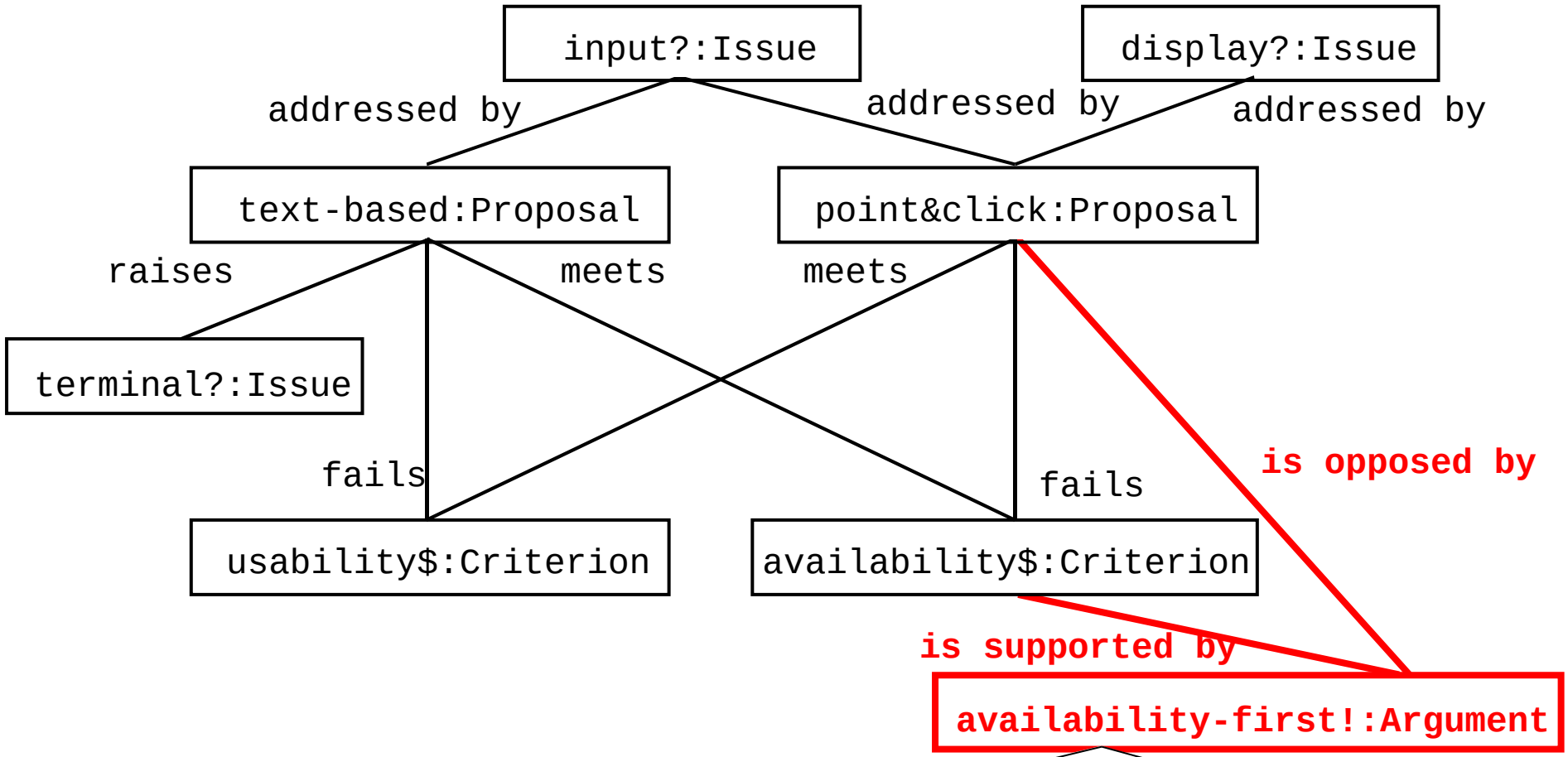
- A criteria represent a goodness measure.
- Criteria are often design goals or nonfunctional requirements.



Arguments

- Arguments represent the debate developers went through to arrive to resolve the issue.
- Arguments can support or oppose any other part of the rationale.
- Arguments constitute the most part of rationale.

Arguments (2)

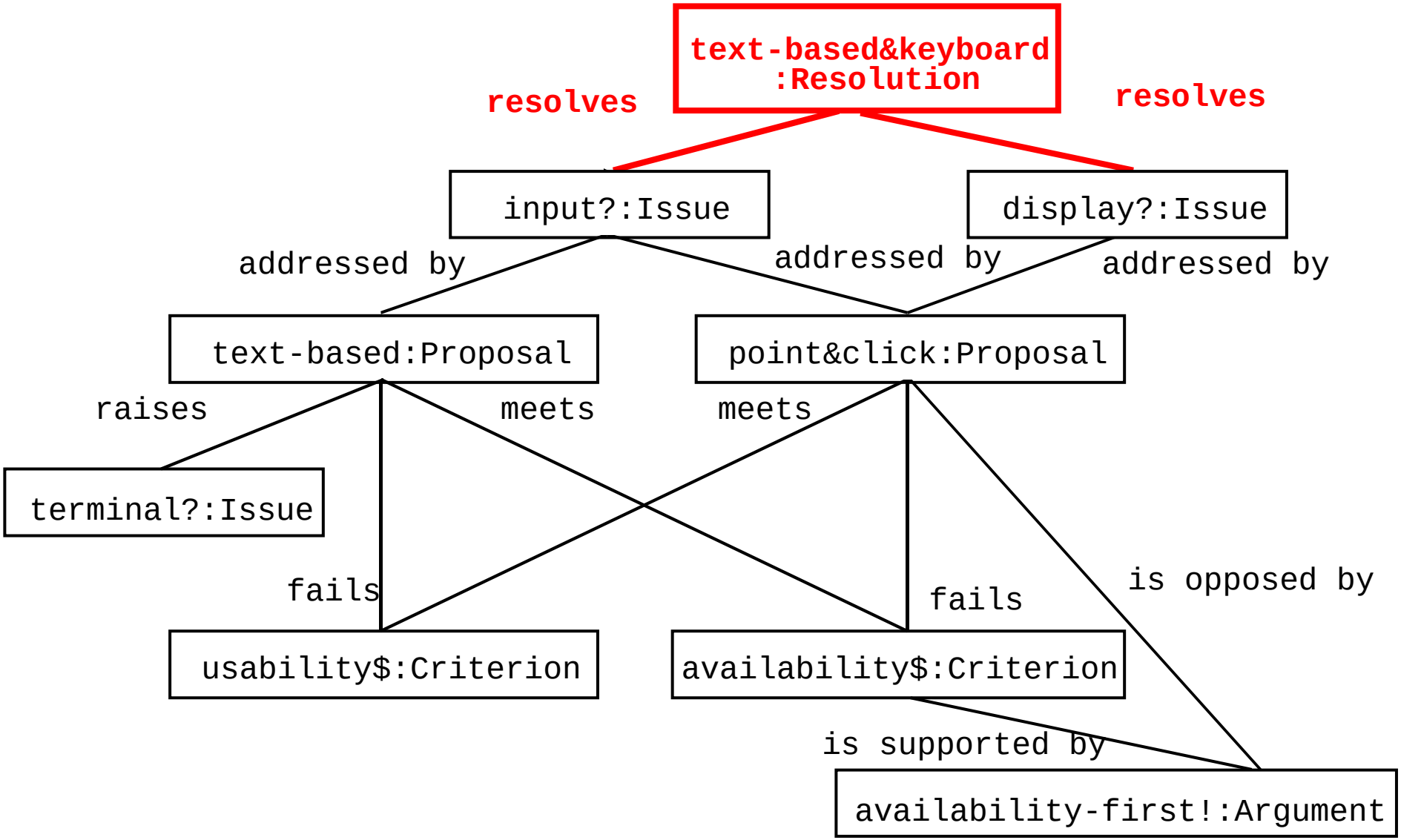


Point&click interfaces are more complex to implement than text-based interfaces. Hence, they are also more difficult to test. The point&click interface risks introducing fatal errors in the system that would offset any usability benefit the interface would provide.

Resolutions

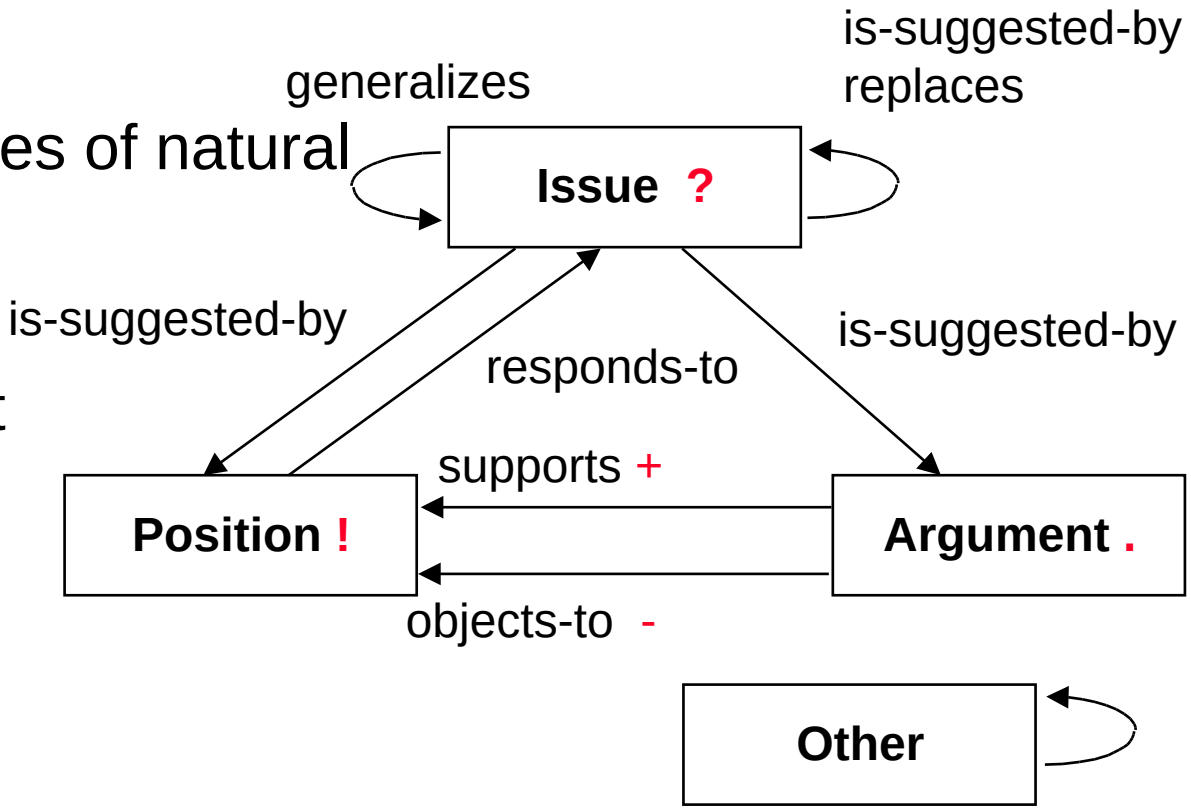
- Resolutions represent decisions.
- A resolution summarizes the chosen alternative and the argument supporting it.
- A resolved issue is said to be closed.
- A resolved issue can be re-opened if necessary, in which case the resolution is demoted.

Resolutions (2)



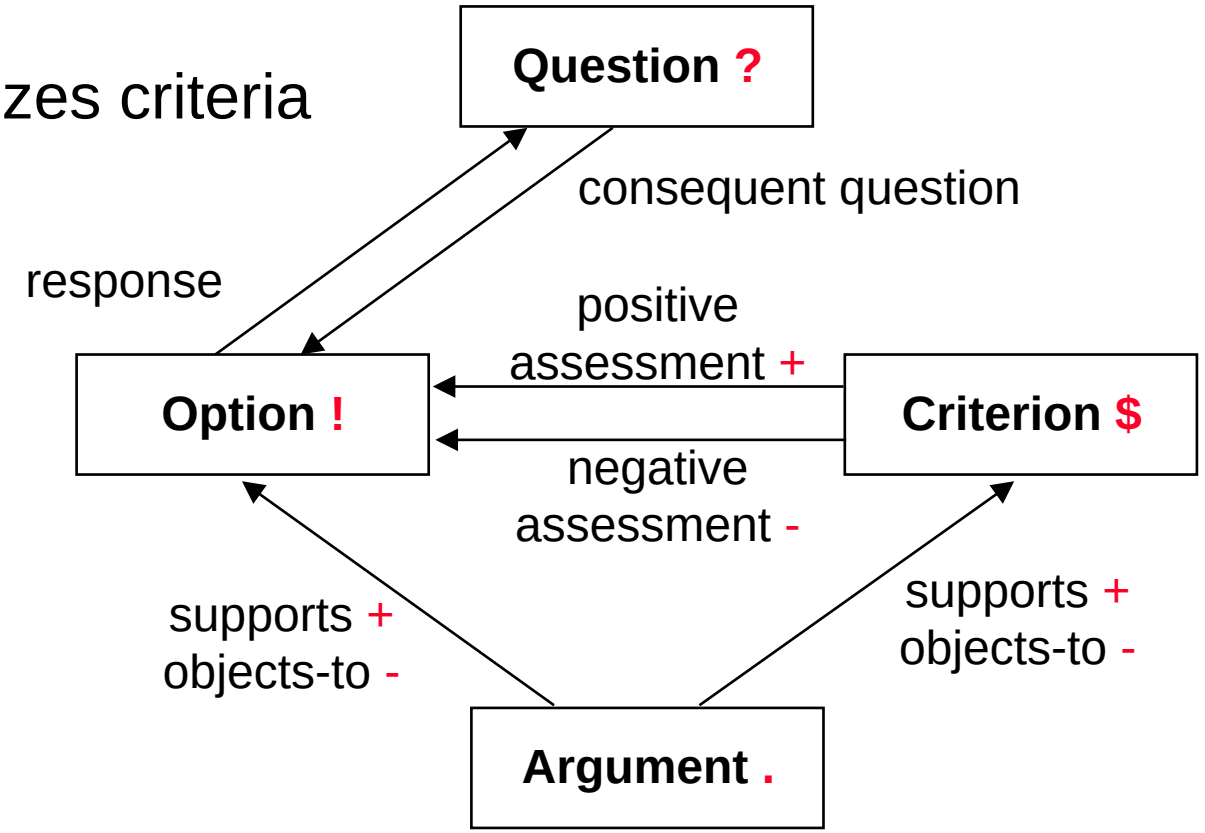
Other issue models: Issue-Based Information System

- Semi structured notation for capturing rationale as decisions are made.
- Nodes are pieces of natural language text
- Links represent relationships between nodes

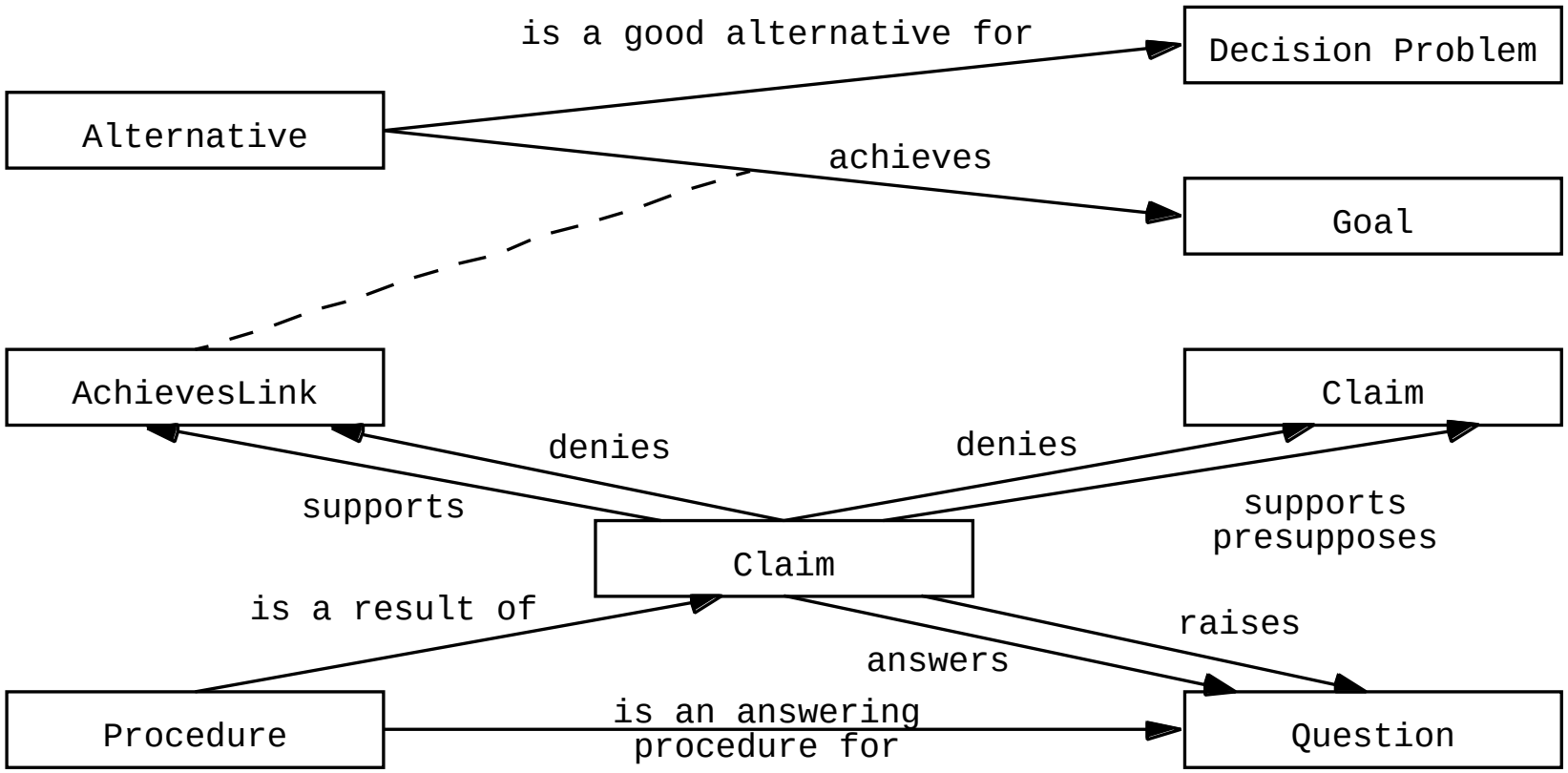


Other issue models:
Questions, Options, Criteria

- Designed for capturing rationale after the fact (e.g., quality assessment).
- Similar to IBIS
- QOC emphasizes criteria



Other issue models:
Decision Representation Language



Capturing rationale

Possible approaches to capturing rationale

- Reconstruction
- Record-and-replay
- Byproduct of development method

Requirements

- Non disruptive: it should not interfere with design
- Integrated with development

Capturing rationale: reconstruction

- A librarian is assigned to role to reconstruct the system's rationale
- Developers are interviewed and surveyed
- Reconstructing rationale is similar to technical documentation

- Advantages
  - Captures justifications of selected alternatives
  - Relatively accurate when done shortly after development
- Disadvantages
  - Misses discarded alternatives
  - Difficult to maintain history of changes

Rationale reconstruction: example

- JAMES'97
  - Smartcard architecture for automotive industry
  - Demonstration prototype
  - ~40 participants
- System Design Document (SDD)
  - Documents system level design decision and their rationale
  - Asked each participants to reconstruct the rationale for one issue
- Result
  - 17 fully documented design issues
  - ~20 pages, rationale is the largest section in the SDD
  - Positive feedback from the client

JAMES SDD excerpt

Proposal 1: Register only subsystems	**For:** ● Quick implementation time. ● Satisfies name server requirements. Provides means for communication between subsystem. ● Robust. Services always available. Subsystems designed with services built in. ● Extensible. Can be extended to allow service registry. **Against:** ● Inflexible. Must match subsystem to particular hardware ● Inefficient. Does not allow for sharing of services.
Proposal 2: Register services and subsystems	**For:** ● Efficient. Eases sharing of commonly used services by multiple subsystems. ● Easy subsystem development. Allows development of libraries of services, and development of front end interfaces for multiple services that provide same functionality using different hardware. ● Flexible. Developer always has option of hard coding little used services instead of using a name service. **Against:** ● Longer implementation time. Must work out issues of security and priority when subsystems are competing for resources. ● Not robust. Subsystem must fail if it cannot find required service.
Proposal 3: Treat services as subsystems - Peer to Peer	**For:** ● Peer to peer relationship between subsystems/services more flexible. All the same advantages as proposal II.

Capturing rationale: record and replay

- Participants use a semi-structured notation to record meetings and online discussions
- Can use a issue-base or text-based conventions

Advantages

- Captures arguments
- Occurs closely with the design

Disadvantages

- Requires post processing
- Can disrupt the design process

Example: capturing rationale in meetings

- Facilitator posts an agenda
- Participants respond to the agenda
- Facilitator updates the agenda and facilitates the meeting
- Minute taker records the meeting

Example: capturing rationale in meetings (2)

- Facilitator posts an agenda
  - Discussion items are ***issues***
- Participants respond to the agenda
  - Proposed amendments are ***proposals*** or additional ***issues***
- Facilitator updates the agenda and facilitates the meeting
  - The scope of each discussion is a single ***issue*** tree
- Minute taker records the meeting
  - The minute taker records discussions in terms of ***issues***, ***proposals***, ***arguments***, and ***criteria***.
  - The minute taker records decisions as ***resolutions*** and ***action items***.

Example: database discussion agenda

3. Discussion

I[1] Which policy for retrieving tracks from the database?

I[2] Which encoding for representing tracks in transactions?

I[3] Which query language for specifying tracks in the database request?

Example: database discussion

I[1] Which policy for retrieving tracks from the database?

Jim: How about we just retrieve the track specified by the query? It is straightforward to implement and we can always revisit it if it is too slow.

Ann: Prefetching neighboring tracks would not be much difficult and way faster.

Sam: During route planning, we usually need the neighbor tracks anyway. Queries for route planning are the most common queries.

Jim: Ok, let's go for the prefetch solution. We can revert to the simpler solution if it gets too complicated.

Example: database discussion minutes

3. Discussion

I[1] Which policy for retrieving tracks from the database?

P[1.1] Single tracks!

A- Lower throughput.

A+ Simpler.

P[1.2] Tracks + neighbors!

A+ Overall better performance: during route planning, we need the neighbors anyway.

{ref: 1/31 routing meeting}

R[1] Implement P[1.2]. However, the prefetch should be implemented in the database layer, allowing use to encapsulate this decision. If all else fails, we will fall back on P[1.1].

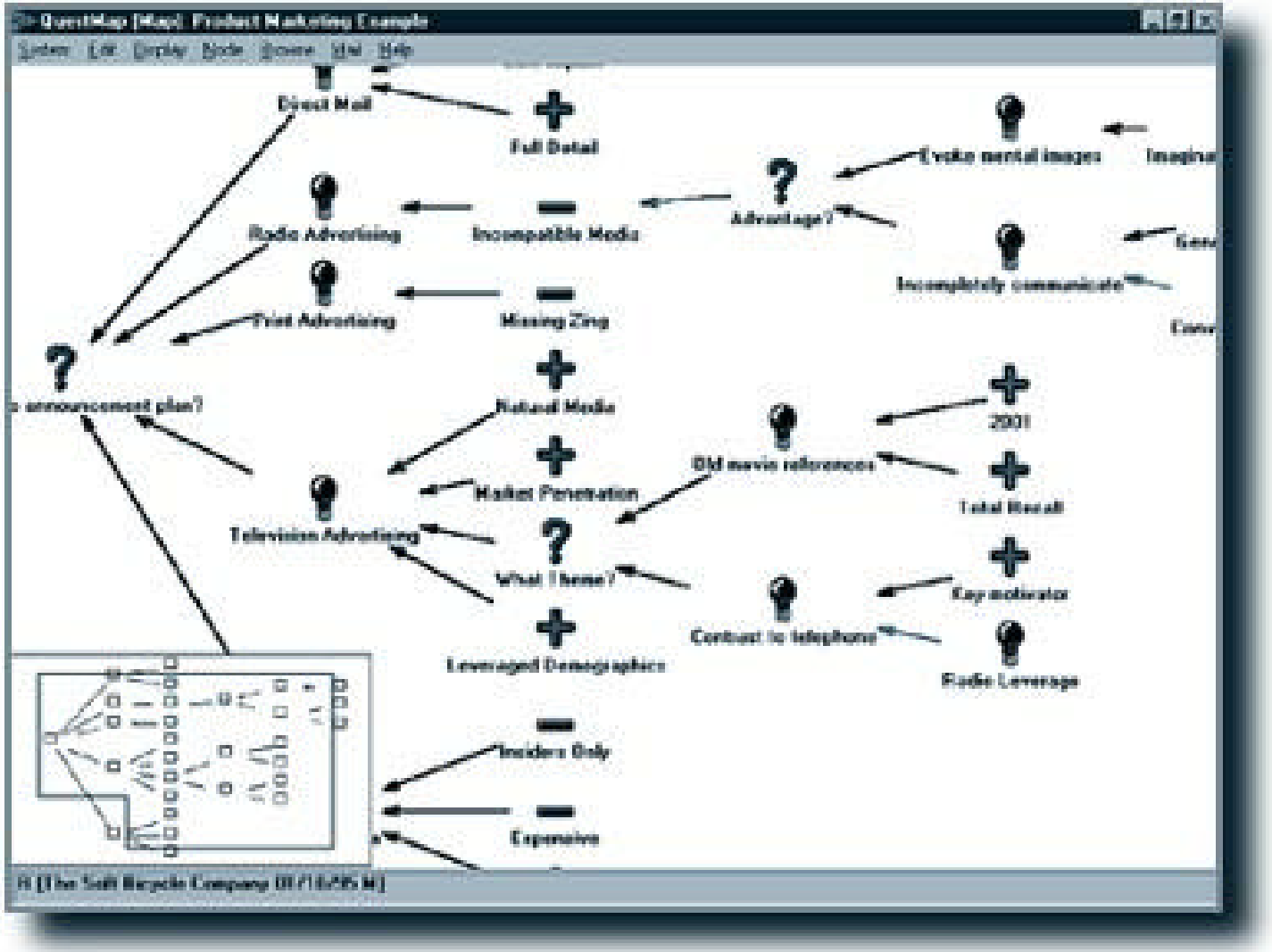
Maintaining rationale

- Rationale information grows as the system evolves.
- Rationale information needs to be updated to be useful.
- An issue base can be used to maintain the issue trees.
- The meeting agendas and minutes should be integrated with the issue base.

Example: Lotus Notes IBIS Discuss

Rationale in practice

- QuestMap, by the Soft Bicycle Company



Open issues

- Formalizing knowledge is costly.
  - Maintaining a consistent design model is expensive.
  - Capturing and maintaining rationale is worse.

- The benefits of rationale are perceived to be long term.
  - If the person who does the work is not the one who benefits from it, the work will have lower priority.
  - 90% of off-the-shelf software projects are terminated before the product ships.

- Capturing rationale can be disruptive.
  - Developers are reluctant to stop design to explain what they just did.

Rationale in the future

- As with many new methods and technologies, will appear as features of existing tools, rather than self contained tools.

- Examples:
  - Discussion support in **RequisitePro**, tool for requirements analysis of Rational
  - Complex schema for modeling change requests in **ClearQuest** and **ClearCase**, a configuration tool by Rational

- In the longer term, issue models or discussion models of multiple tools would be integrated into one issue-base.

Rationale summary

- Capturing rationale is critical:
  - argumentation of alternatives,
  - explicit design criteria,
  - consensus building, and
  - information relevant for future modifications.
- Issue models
  - offer a structured solution to capture rationale
  - make it easier to find rationale information
- Open issues
  - Integration of rationale with current development tools (e.g., communication, IDEs, CASE)
  - Cost-effectiveness
  - Developer incentives