

Goal Modeling

KAOS

Goal-Directed Requirements Acquisition (KAOS)

- ❖ *(Organizational) goals lead to requirements.*
- ❖ *Goals justify and explain requirements which are not necessarily comprehensible by stakeholders.*
- ❖ *Goals can be used to assign responsibilities to agents so that prescribed constraints can be met.*
- ❖ *Goals provide basic information for detecting and resolving conflicts that arise from multiple viewpoints*

The KAOS Modeling Philosophy

- *Modeling a social setting involves a variety of concepts, including **Goals, Agents, Concerned Objects, Actions, Constraints and Responsibilities.***
- *Goals lead to assignments of responsibilities and designs of actions and artifacts*
- *Use a metamodel to support reuse of generic domain modeling patterns [Dardenne93]*
- *For example, the library domain is an instance of the “resource allocation” meta-domain, which also covers car/room/dwelling rental and is similar to airline/hotel reservation, class registration etc.*

The KAOS Acquisition Process

- ***Identify Goals, and their Concerned Objects.***
- ***Identify potential Agents and their Capabilities.***
- ***Operationalize Goals into Constraints.***
- ***Refine Objects and Actions.***
- ***Derive strengthened Objects and Actions to Ensure***
- ***Constraints.***
- ***Identify alternative Responsibilities.***
- ***Assign Actions to responsible Agents.***

All this could be just as useful to someone doing organizational design, rather than software development!

Four KAOS models :

- Goal model
- Responsibility model
- Operation model
- Object model

Goals are **desired system properties** that have been expressed by **some stakeholders**.

Stakeholder (wikipedia): third party who temporarily holds money or property; more recent meaning (widely used in management): person or organization that has a legitimate interest in a project or entity; the new use of the term arose together with and due to the spread of corporate social responsibility ideas...

- ✓ *interviews of domain experts and current & future users*
- ✓ *analysis of existing systems*
- ✓ *reading of available technical documents*

- *Each goal is either a root or is justified by at least another goal that explain **WHY** it is needed.*
- *Each goal except a leave is refined as a collection of subgoals describing **HOW** it can be reached.*
- *Identifying goals proceeds from either **top-down**, **bottom-up** and **non-directional** approaches: from intermediate goals are identified higher-level (business or strategical) goals and lower-level (system requirements) ones.*

Clear distinction between two low-level types of goals:

- *Requirements*, to be achieved by a software agent,
- *Expectations*, to be achieved by an agent which is part of the system environment.

Conflicts (and more generally obstacles that prevent goals to be achieved) must be identified as soon as possible.

Domain properties:



Infrastructure available

Properties relevant to the application domain, used in refinements to prove that a refinement is complete. Ex: in order to satisfy a service request, an infrastructure to perform the service must be available.

There are two types of domain properties :

- **domain hypotheses**: domain object properties expected to hold ; ex: the elevator system has at least one cage to carry passengers,
- **domain invariants**: properties known to hold in every state of some domain object, e.g. a physical law, regulation or a constraint enforced by some environmental agent ; eg : for light buttons, we could state that « the light is either on or off ».

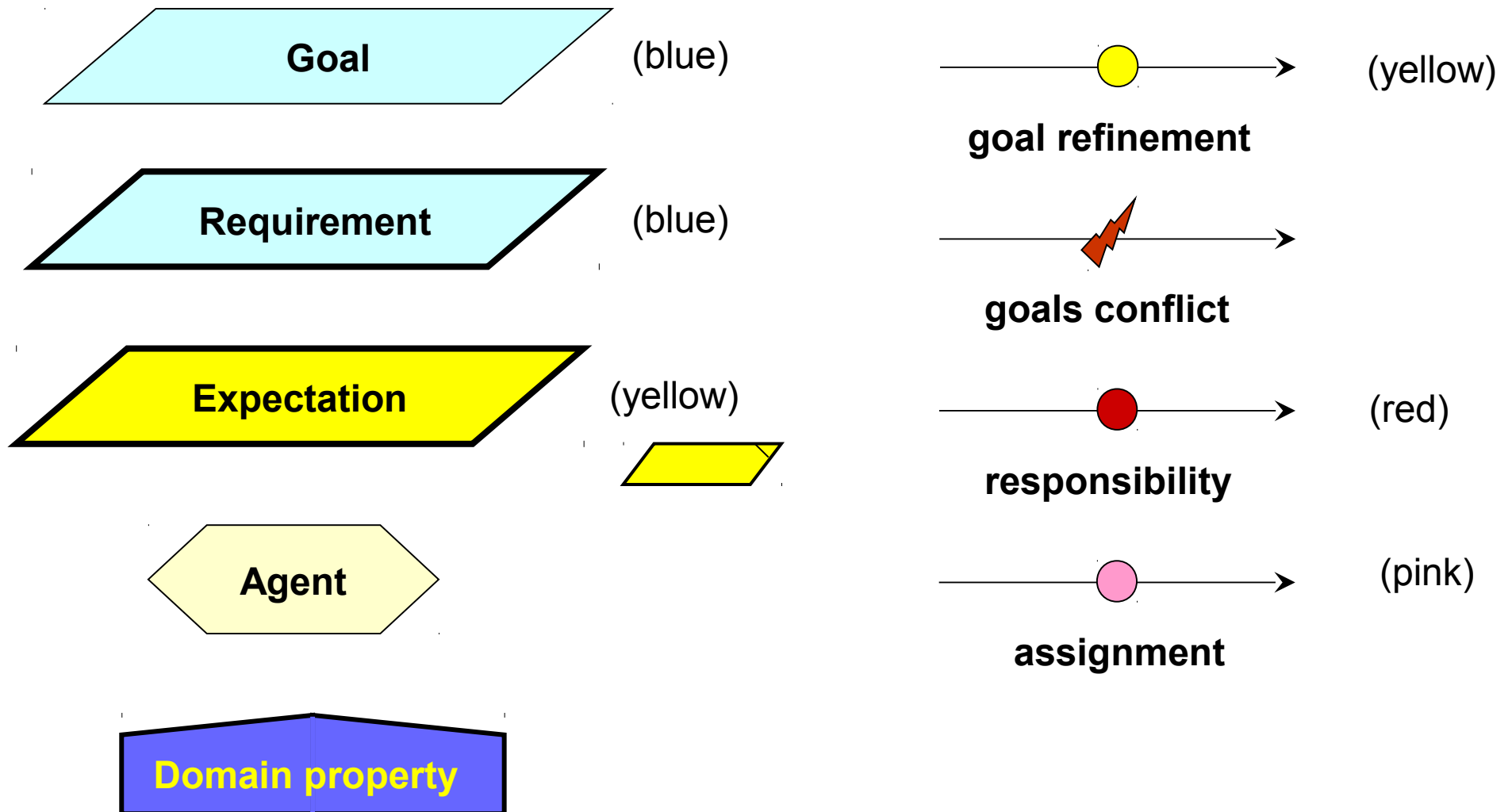
Requirements can be gathered by means of open interviews. A more efficient way is to conduct less open interviews by reusing requirements patterns.

One of the long-term benefits of investing in KAOS technology consists in progressively **modelling generic patterns of requirements**.

Completeness of the goal model is ensured by reviewing the different diagrams during validation sessions with stakeholders.

- **Information system**: software system to be and the part of the environment with which it interacts.
- **Agent**: human being or automated component that are responsible for achieving requirements and expectations.
 - We may stop refining goals into subgoals when they are placed (as requirements) under the **responsibility** of a single agent.
- **Assignment** is used to link expectations to related agents (several agents may be assigned to an expectation).

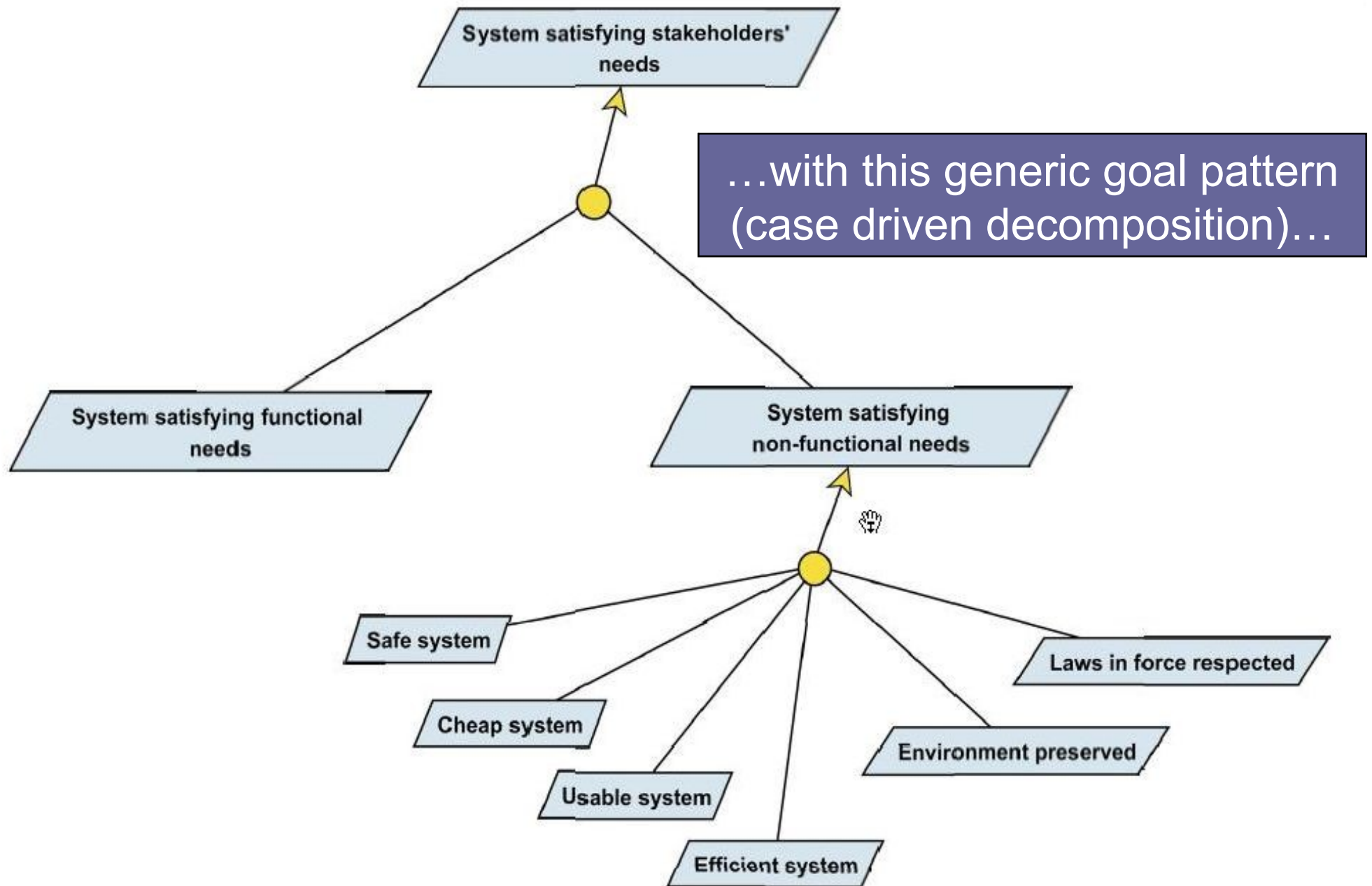
Graphical Conventions



Completeness criterion 1: a goal model is said to be complete with respect to the refinement relationship 'if and only if' every leaf goal is either an expectation, a domain property or a requirement.

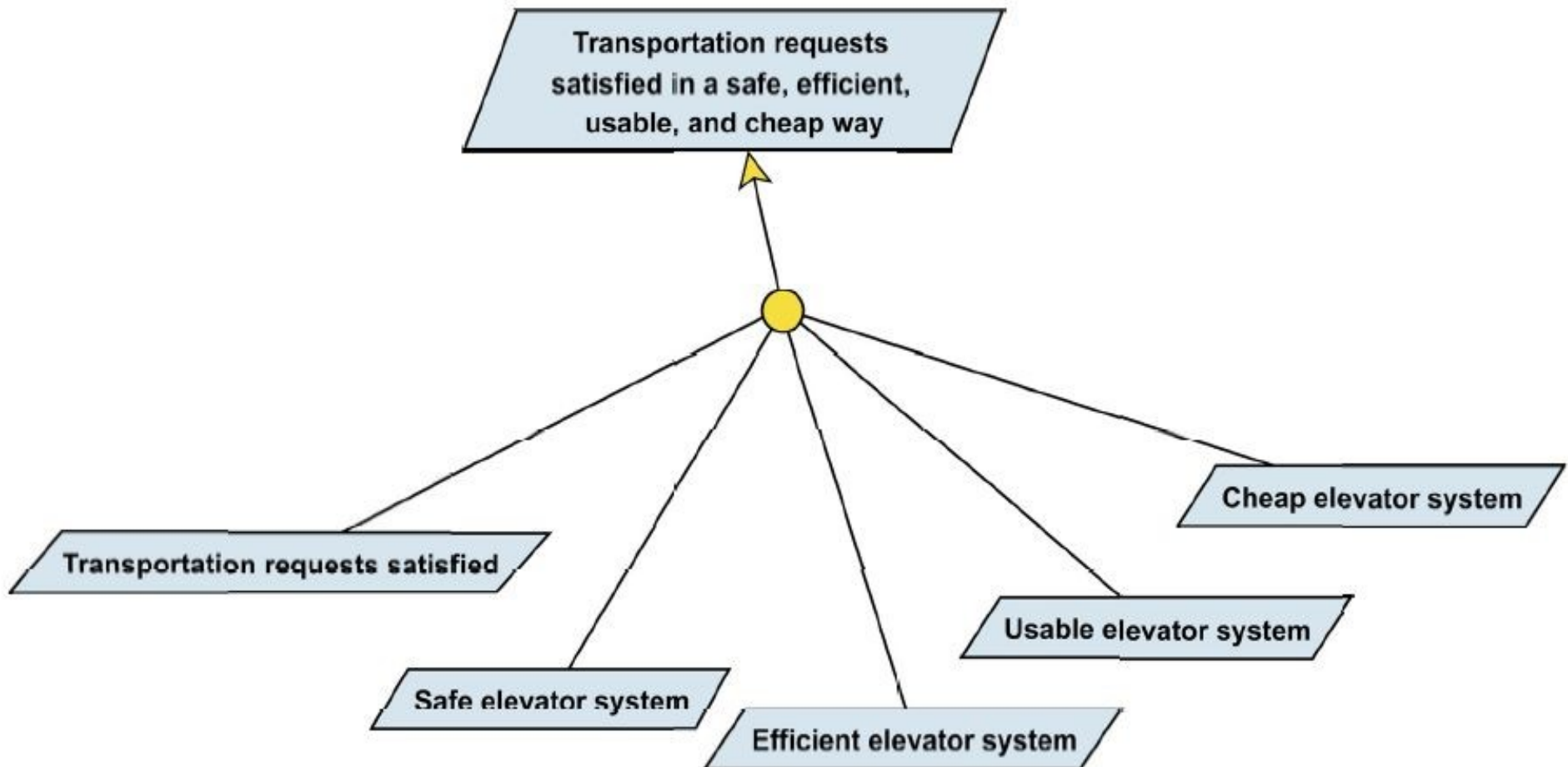
Completeness criterion 2: a goal model is complete with respect to the responsibility relationship 'if and only if' every requirement is placed under the responsibility of one and only one agent (either explicitly or implicitly).

Beginning the Elevator case study... *Goal model*



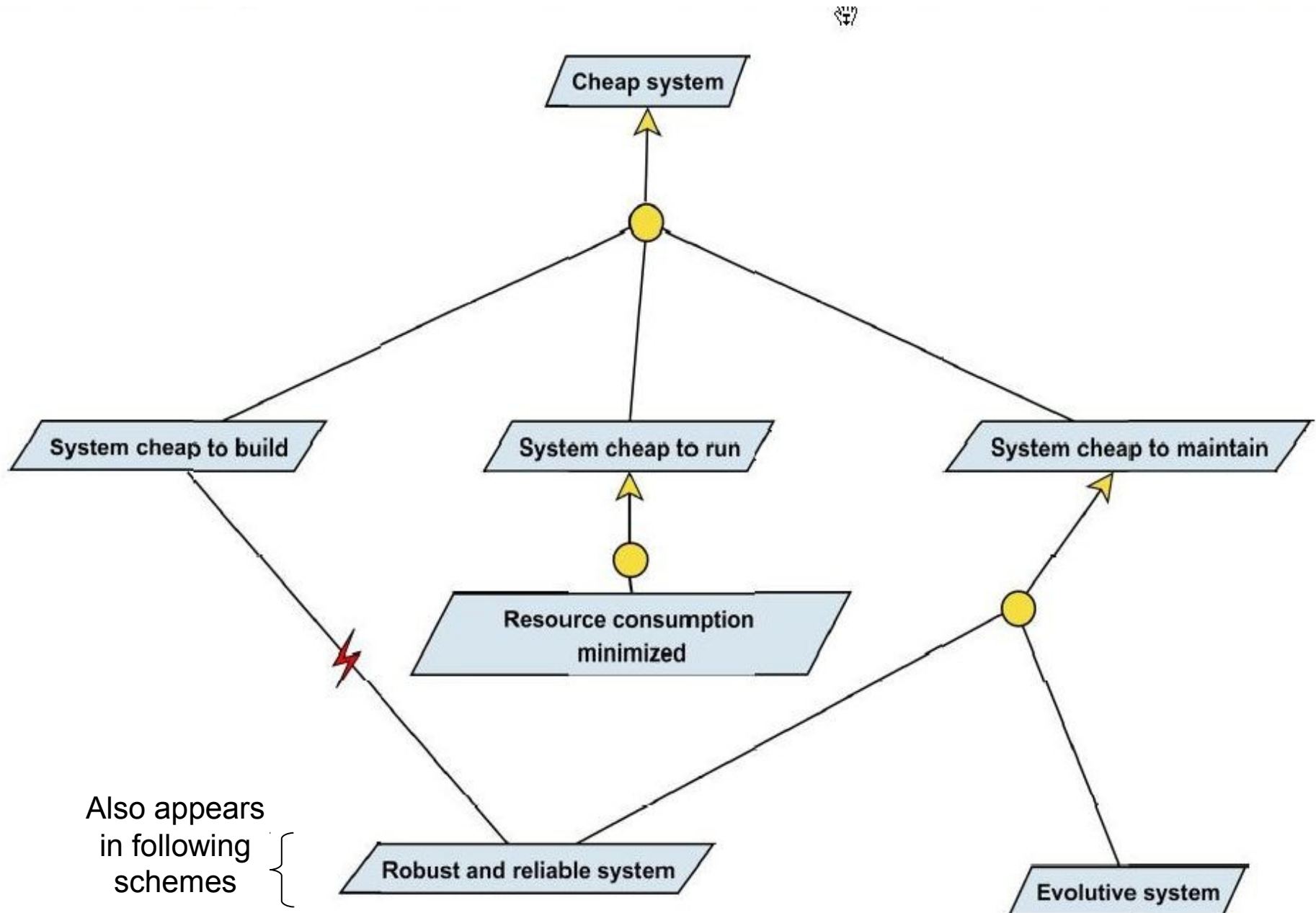
...applied to the elevator problem:

Goal model



Another generic goal pattern (milestone driven decomposition)

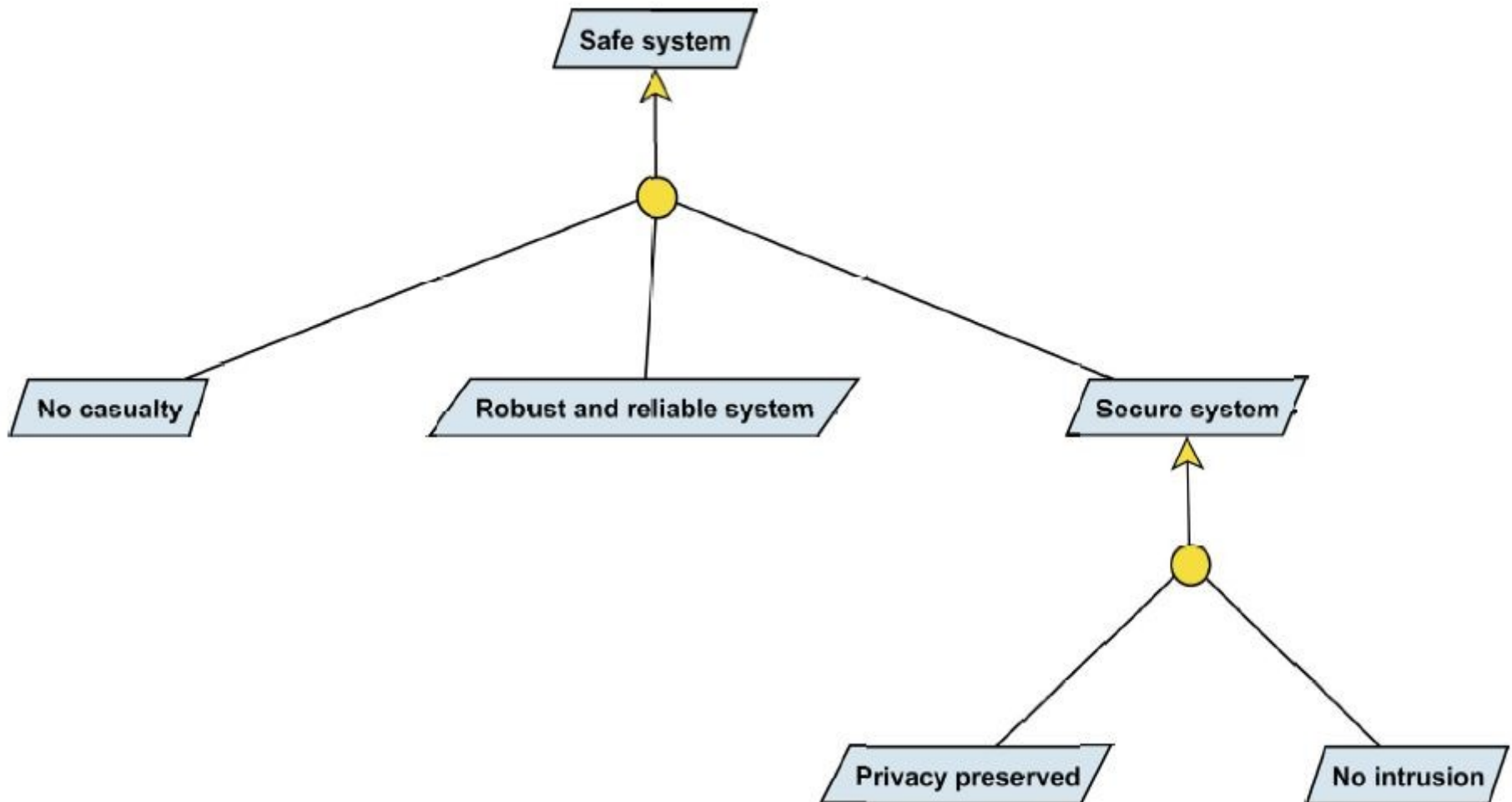
Goal model



Also appears
in following
schemes {

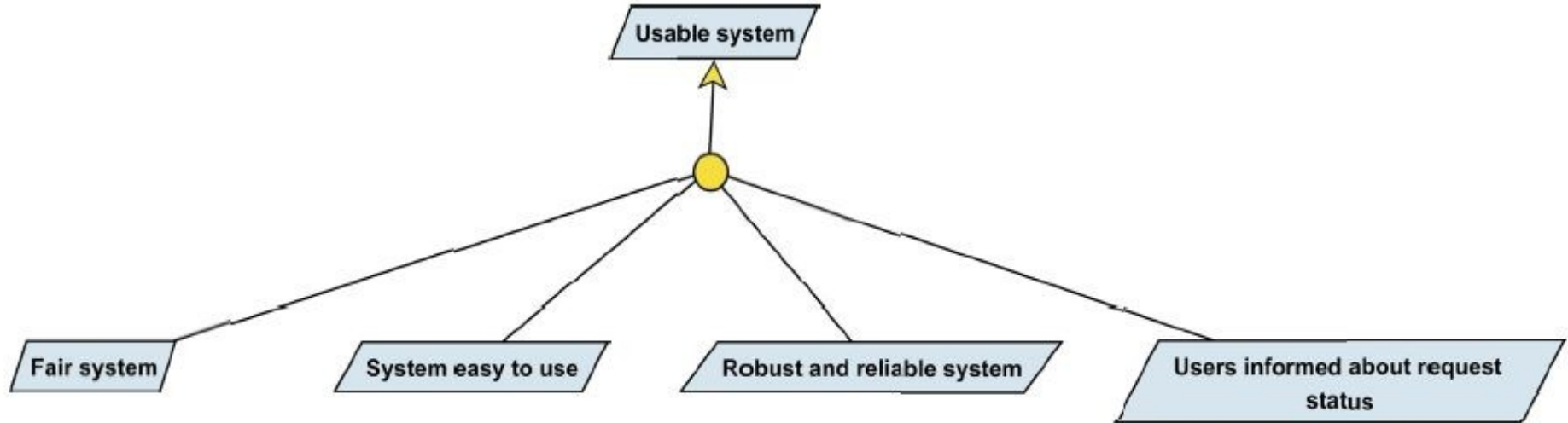
Generic goal « Safe system »

Goal model

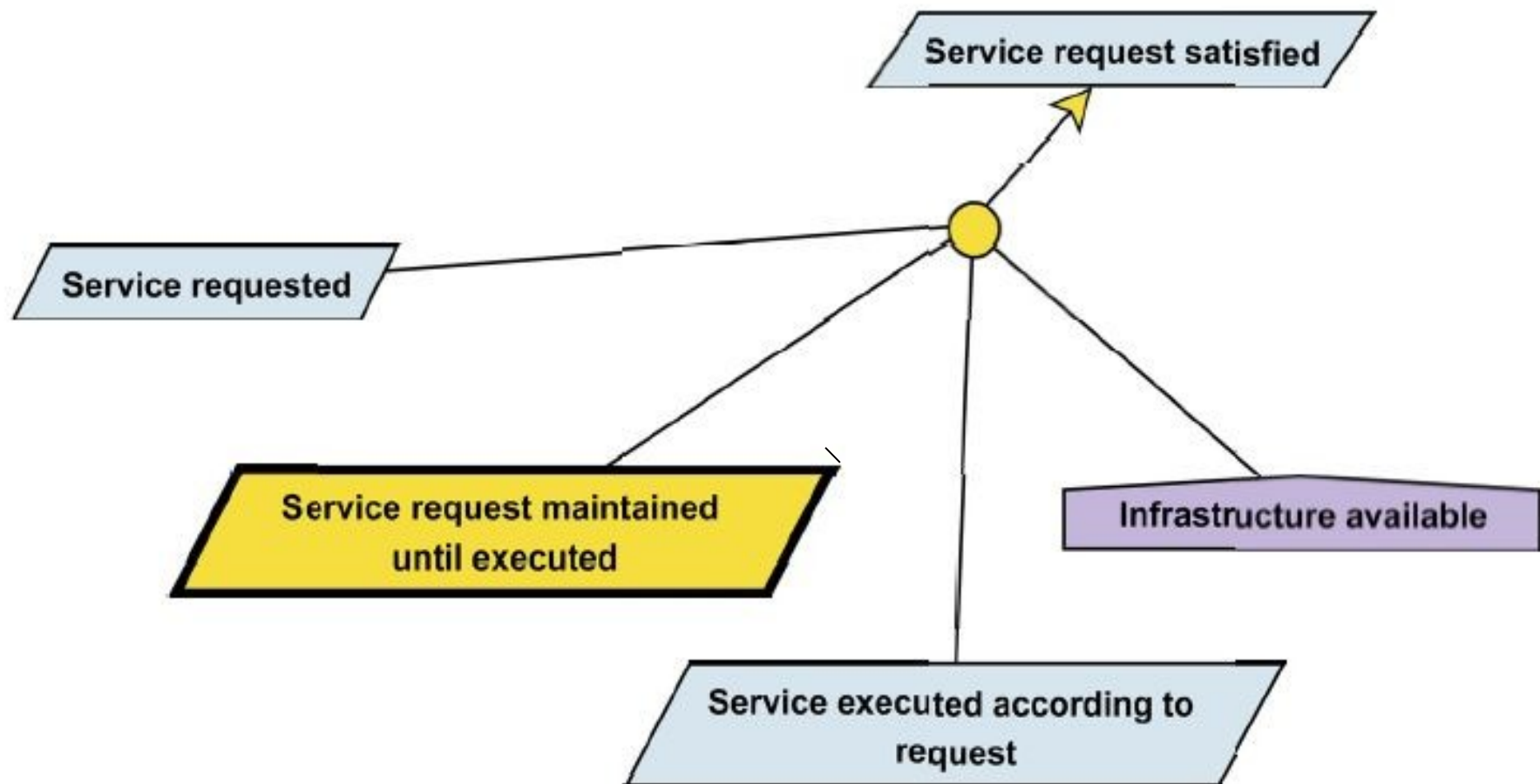


Generic goal « Usable system »

Goal model



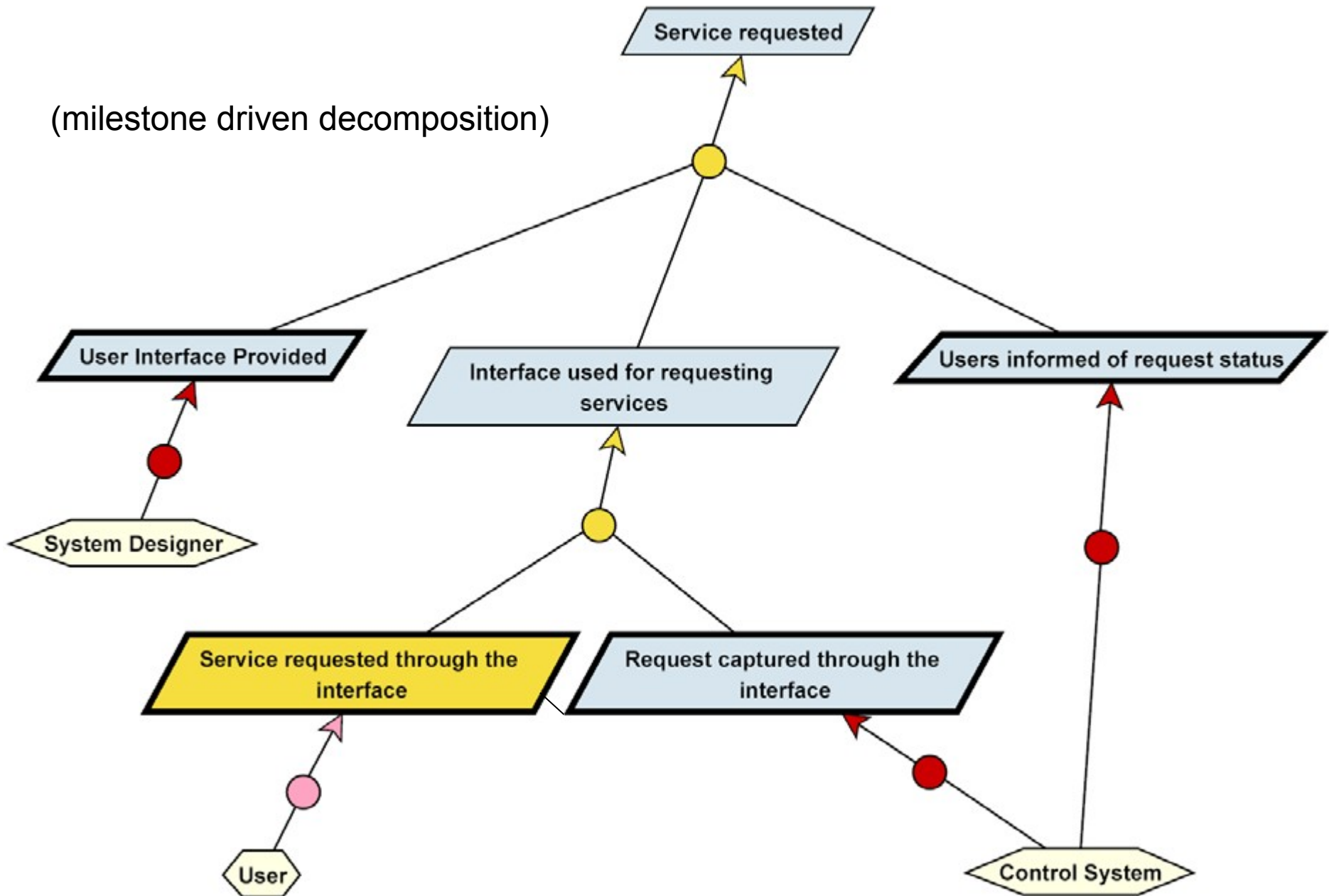
Generic goal « Service request satisfied »...



Generic pattern for the « Elevator called » goal

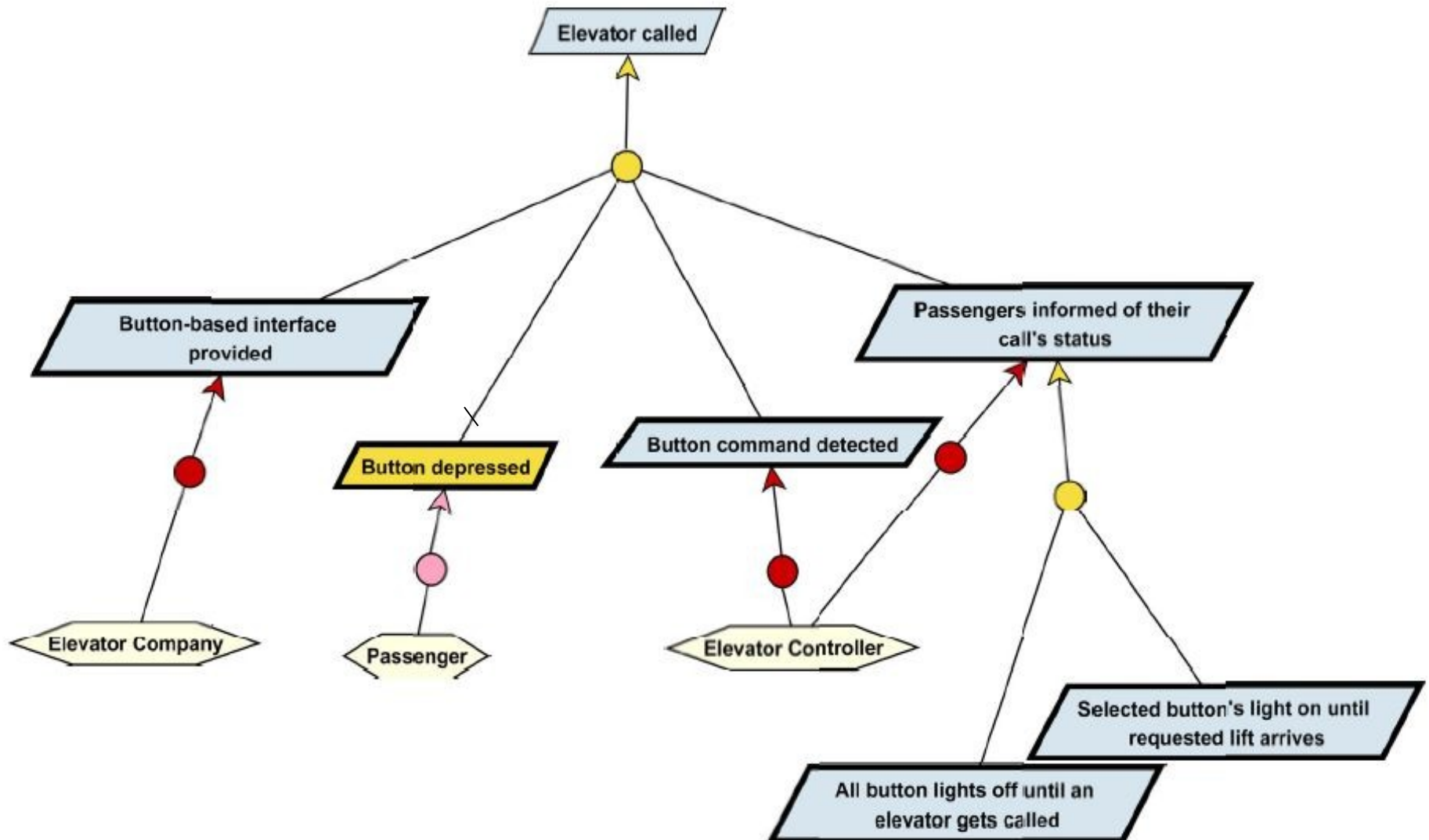
Goal model

(milestone driven decomposition)

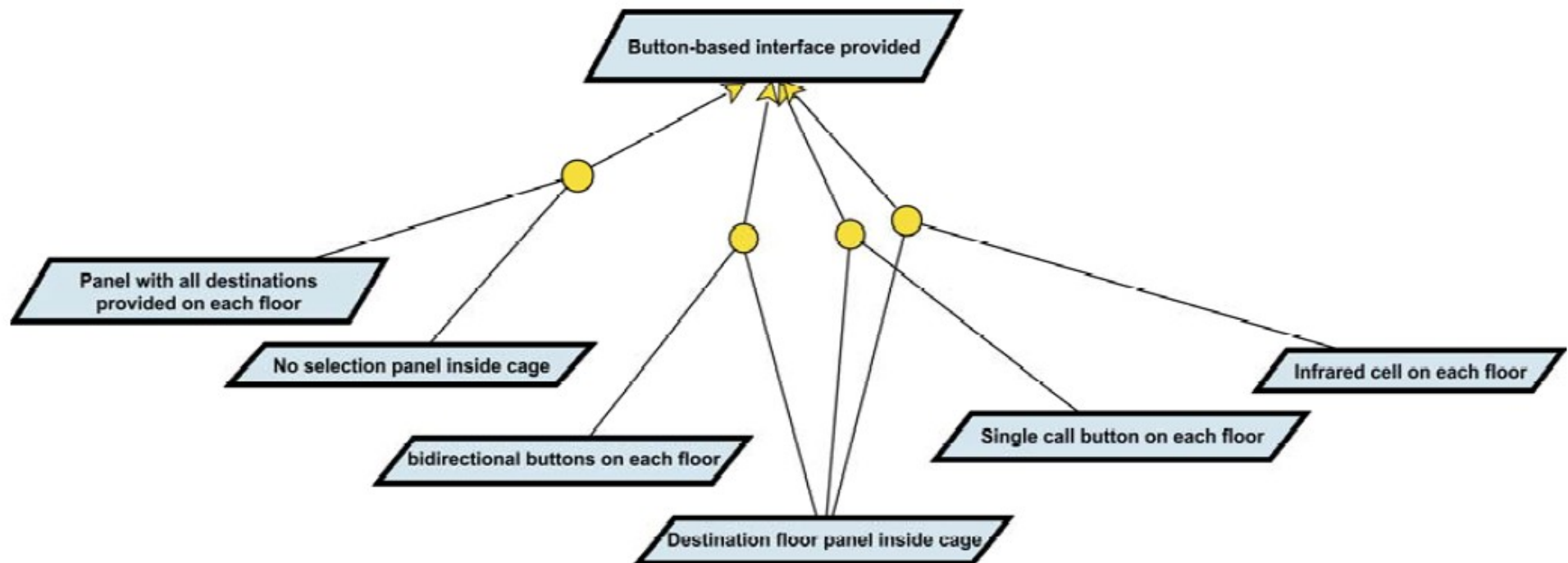


« Elevator called » goal

Goal model

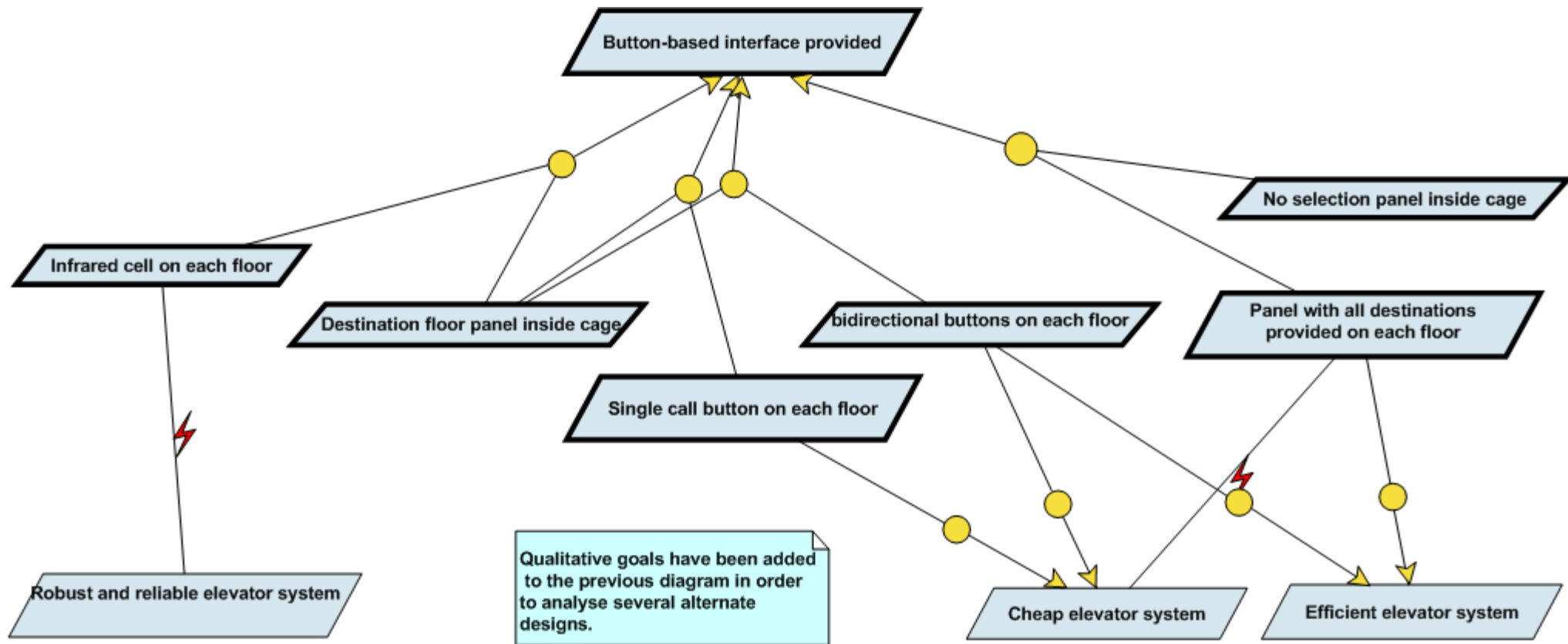


How to deal with Alternatives:



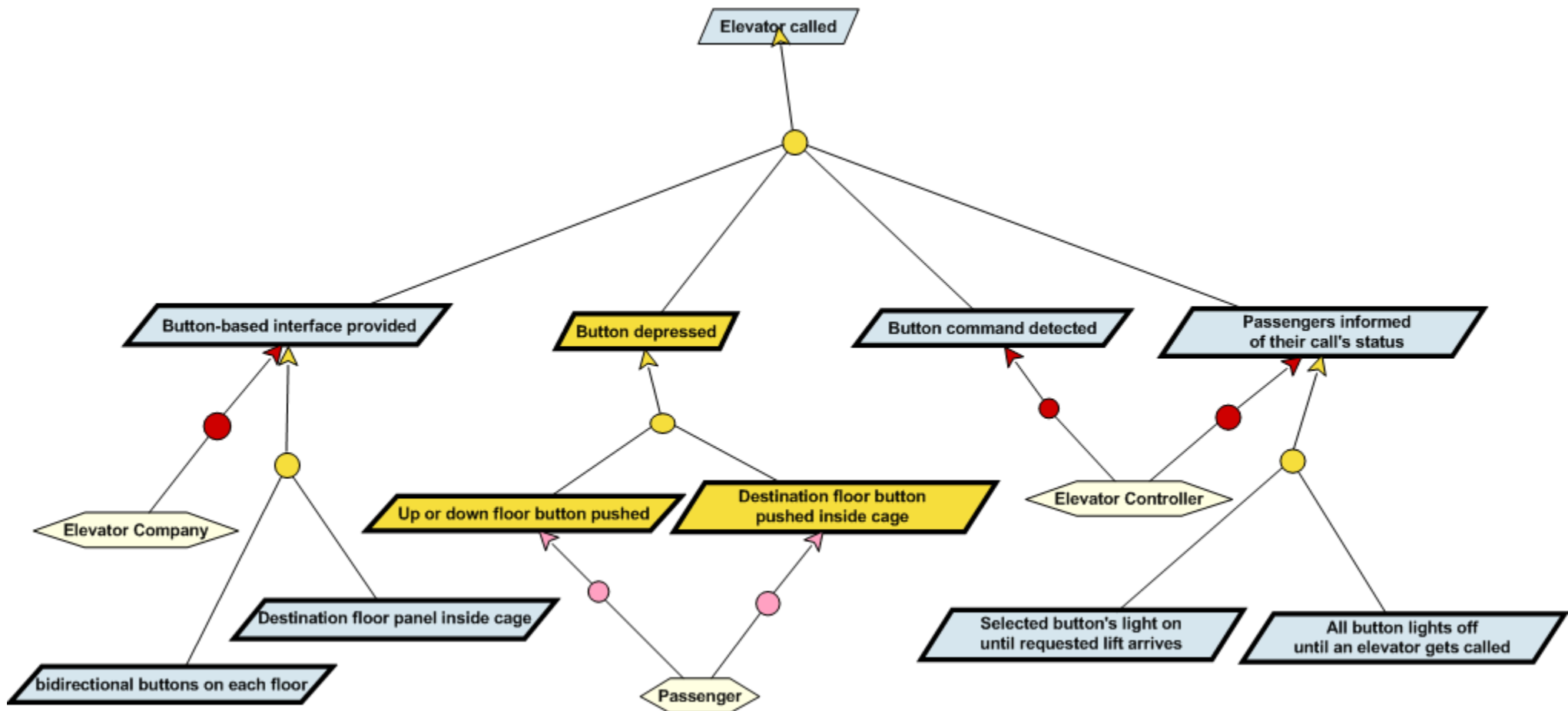
Qualitative comparison:

- links to parent goals to which each requirement contributes
- creating a conflict if it contributes negatively

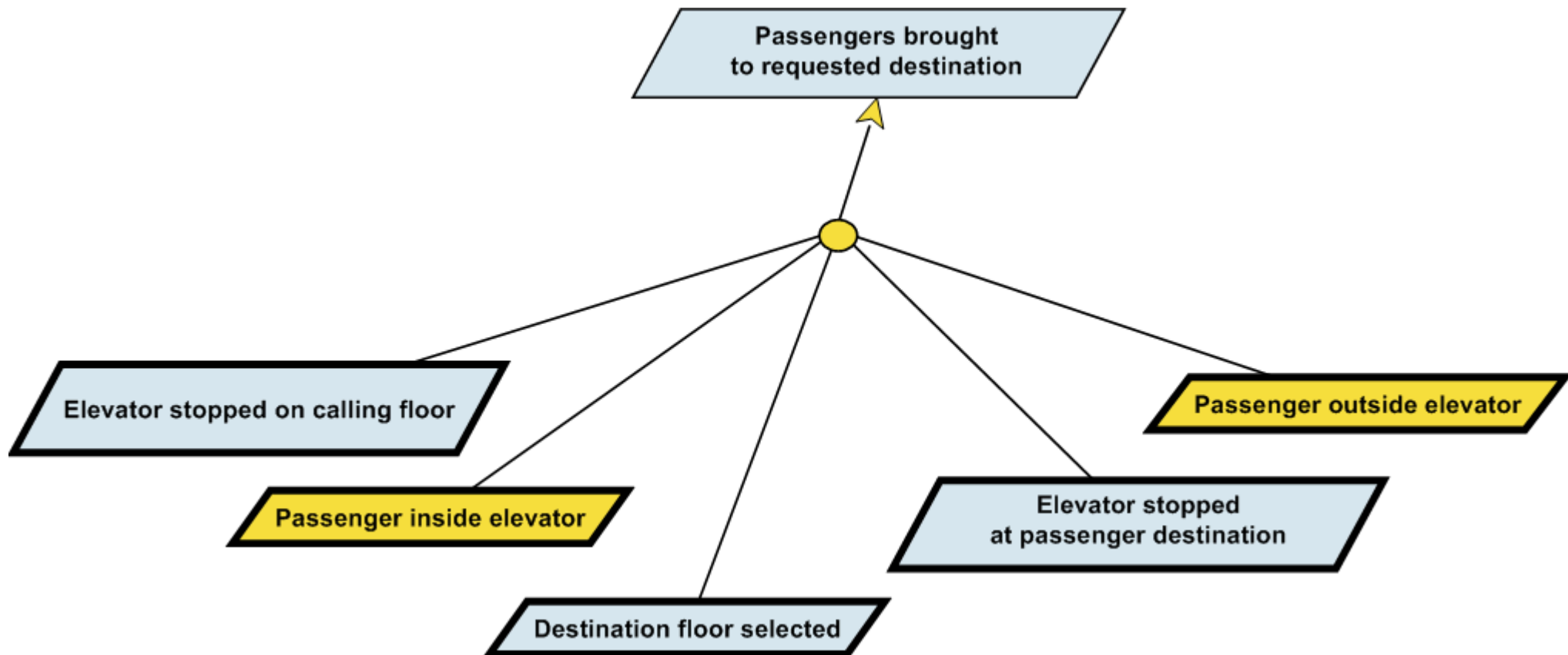


The design having a bidirectional button panel on each floor seems to be the best compromise.

Based on the bidirectional button panel design, we can now refine the goal « Button depressed »:

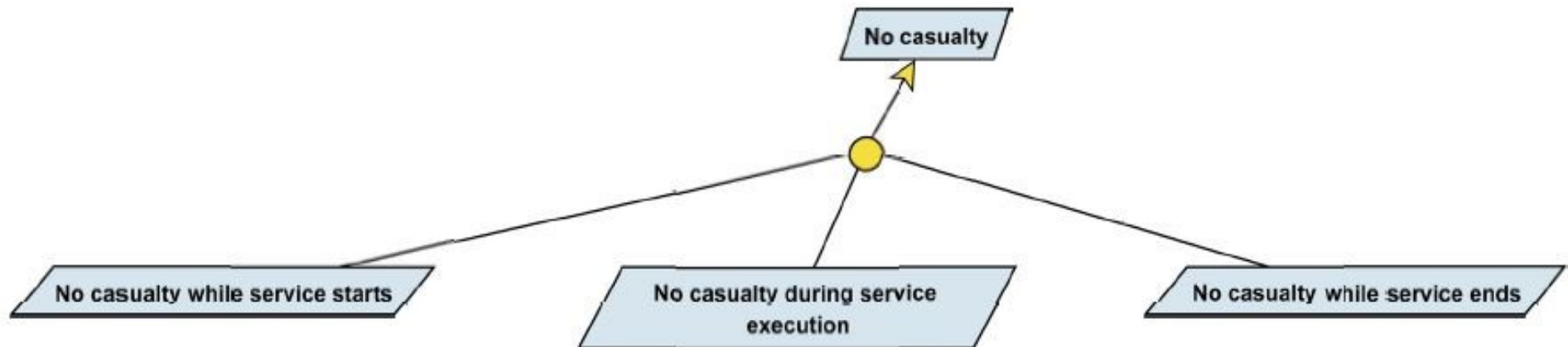


Let's now refine the goal « Passengers brought to requested destination », applying the milestone tactics:



To refine the goal « No casualty », we use a generic pattern that also use a milestone-driven refinement tactic:

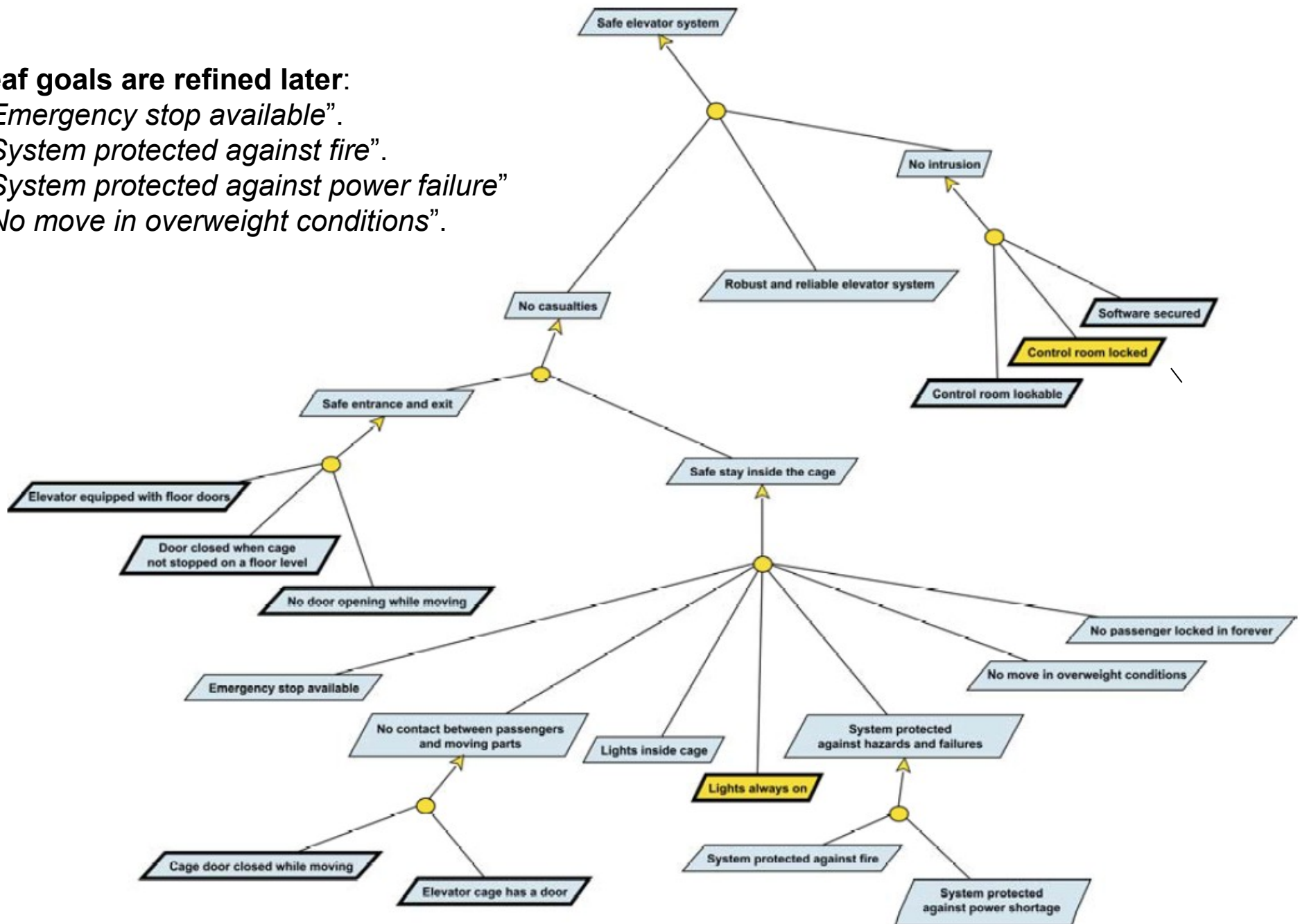
Generic goal « No casualty »

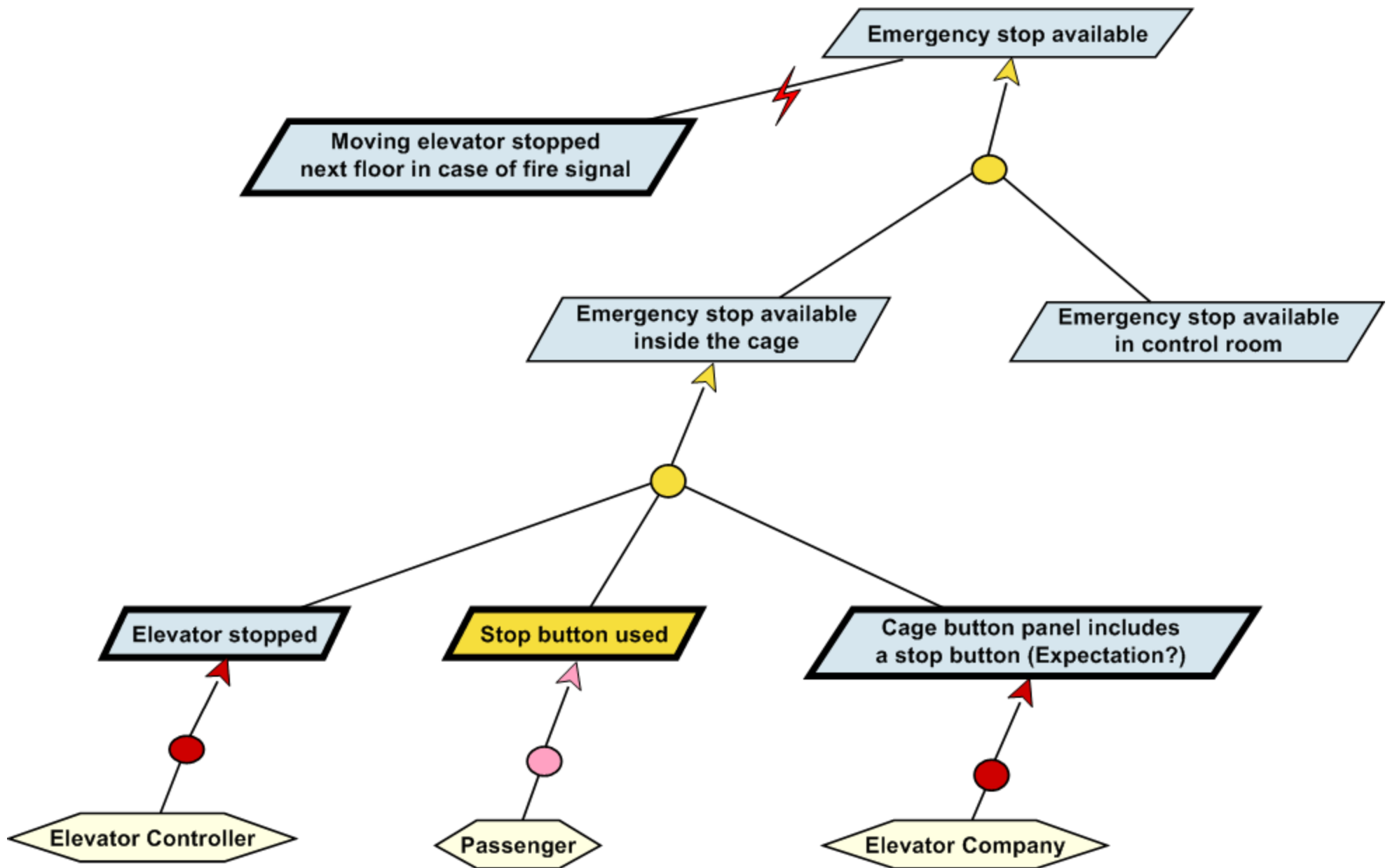


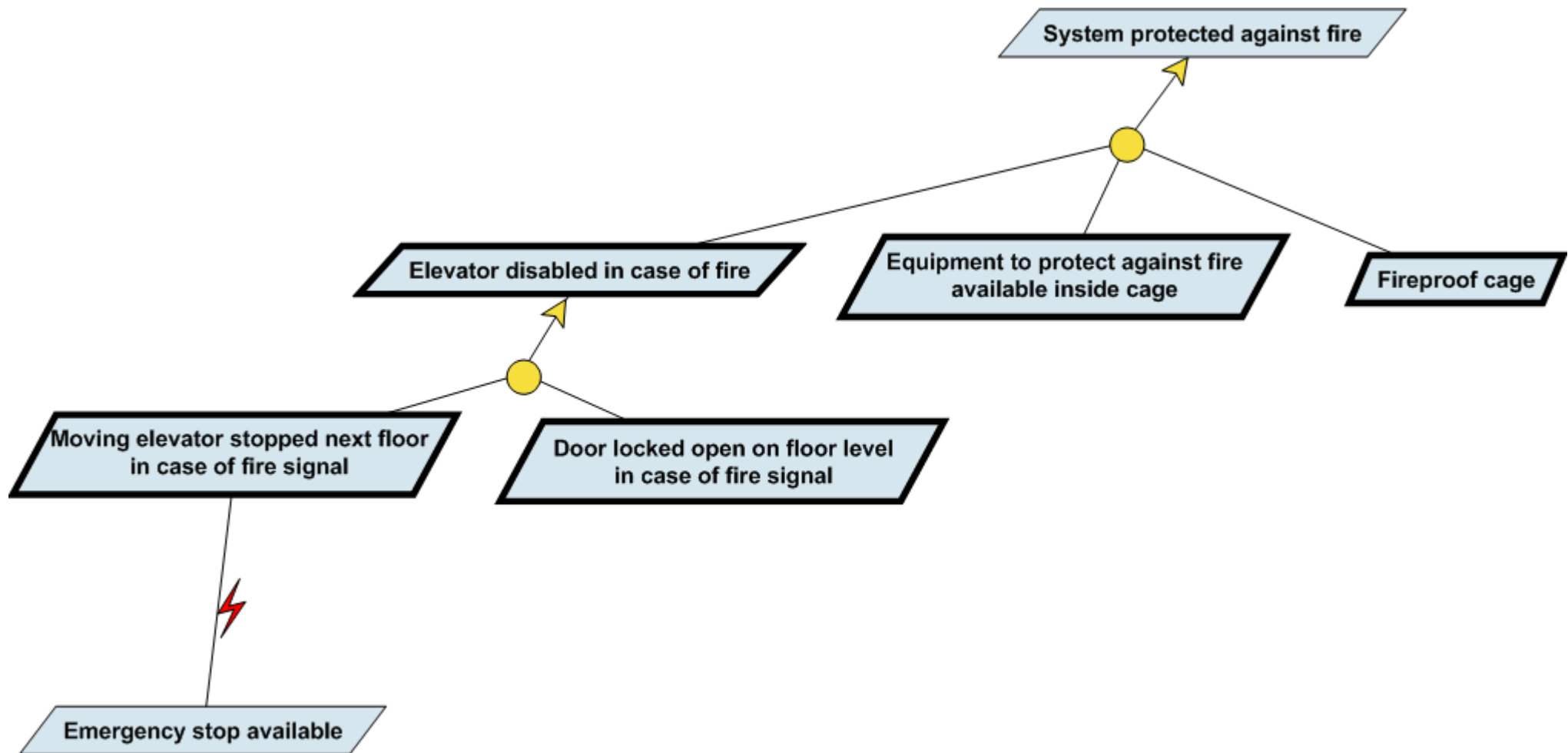
The following figure shows how this pattern has been used, and how the « No intrusion » goal has been decomposed.

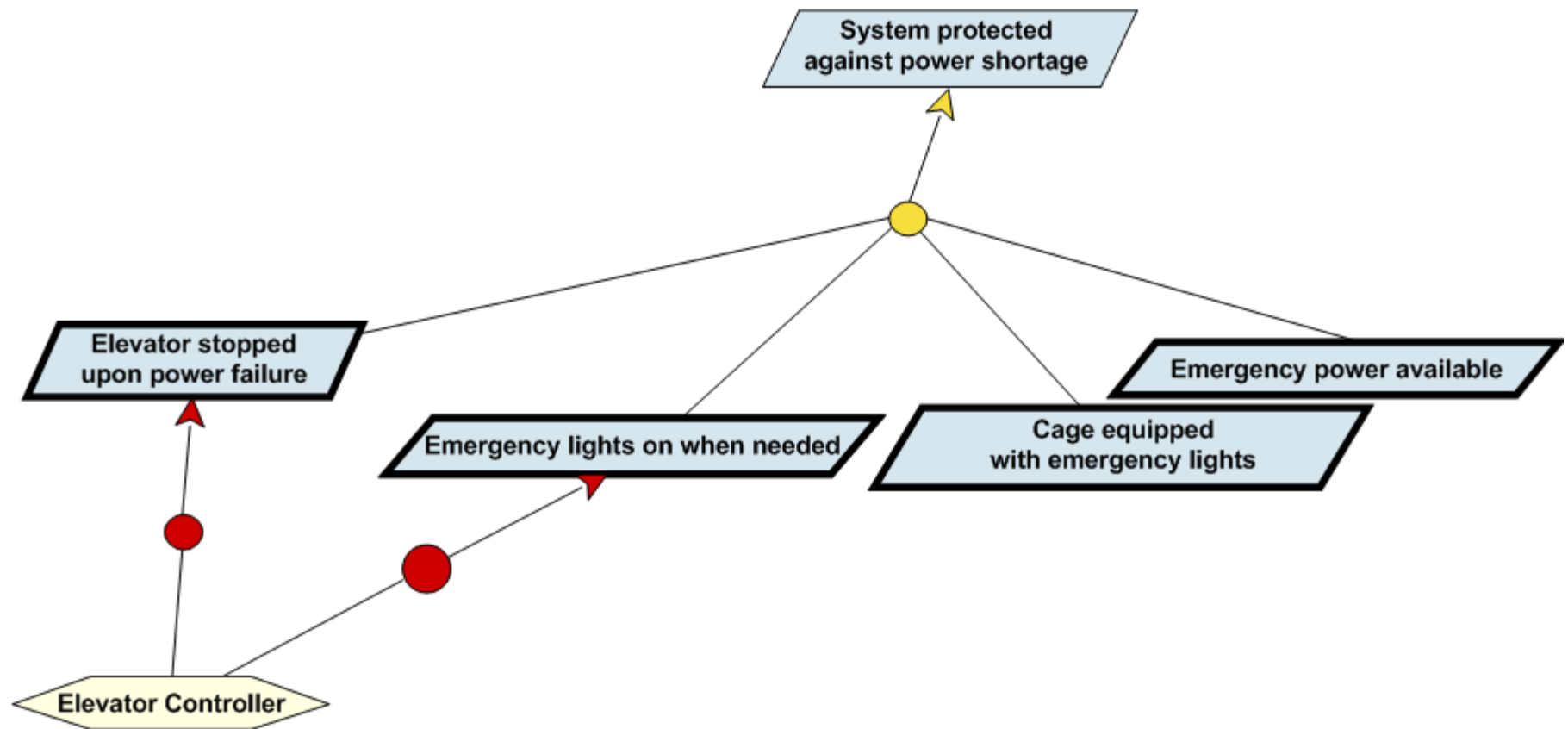
Leaf goals are refined later:

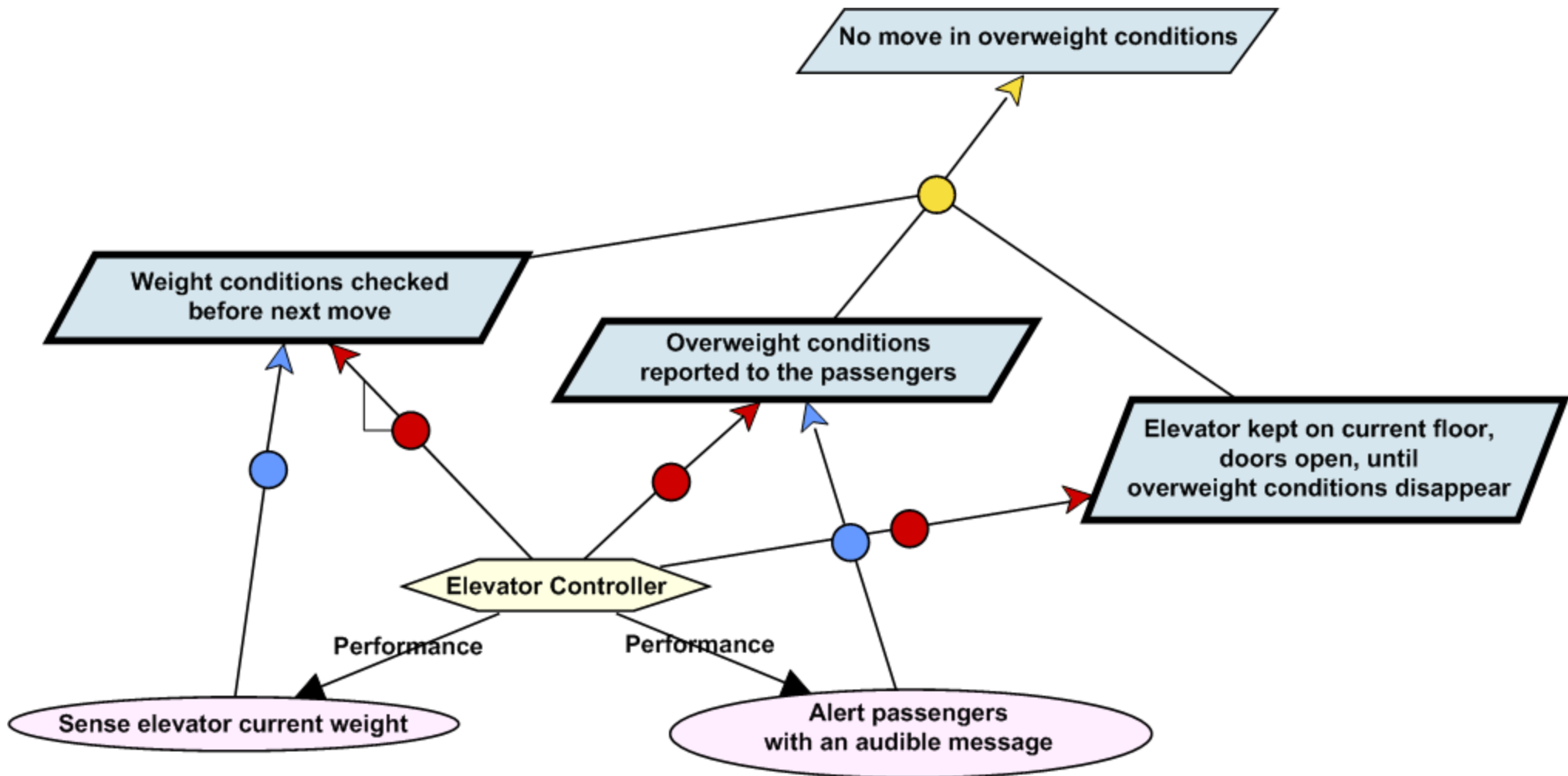
- “Emergency stop available”.
- “System protected against fire”.
- “System protected against power failure”
- “No move in overweight conditions”.



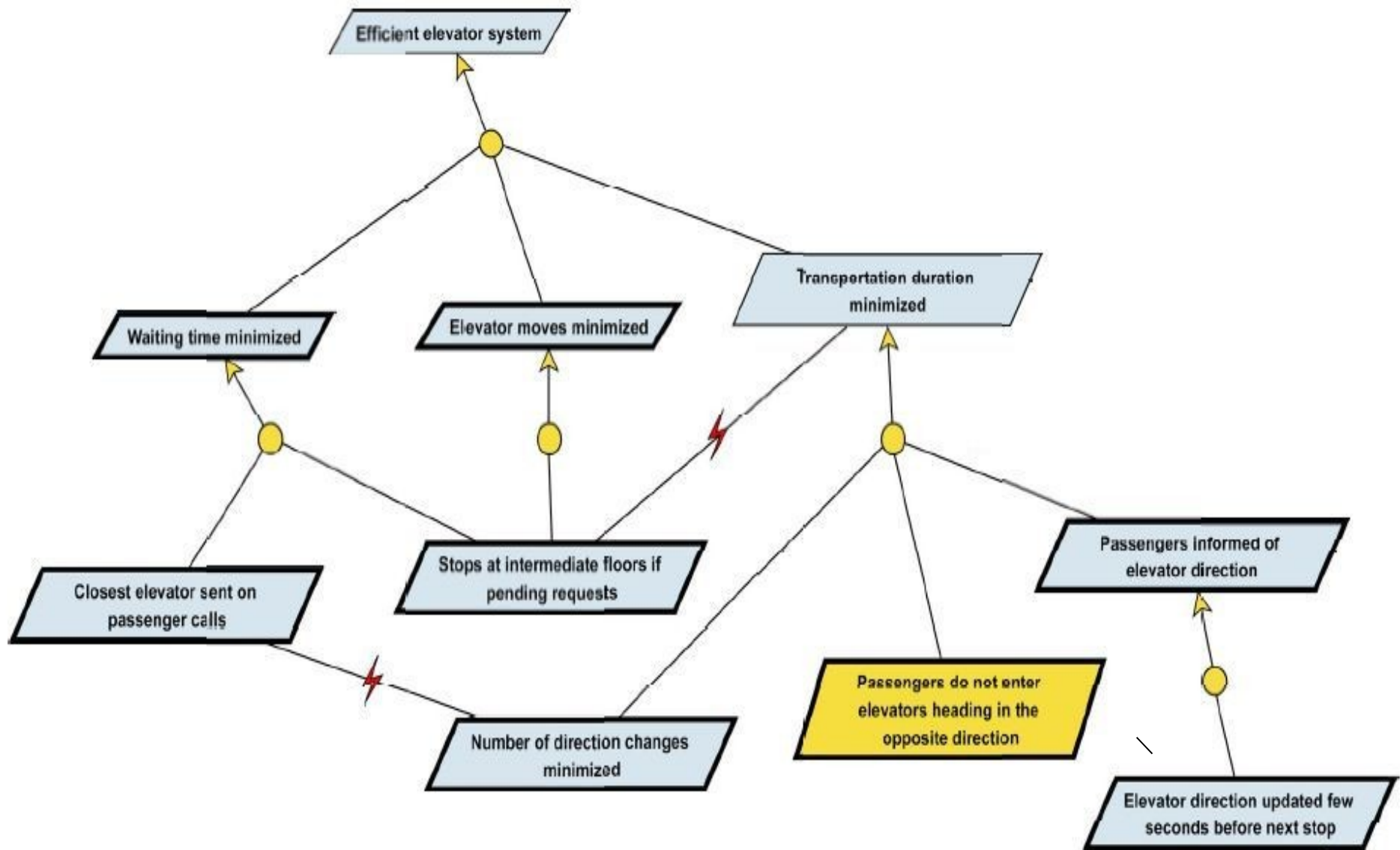




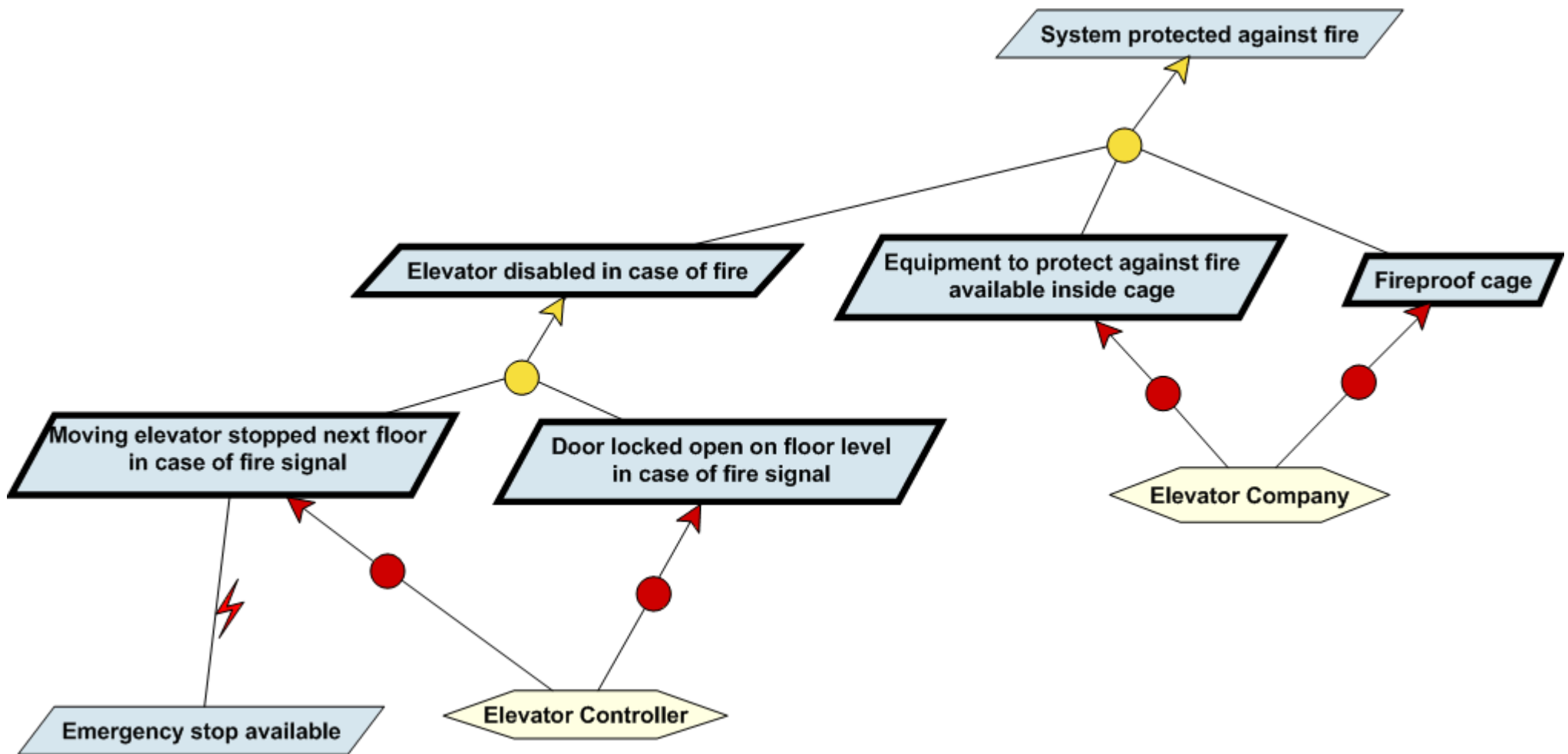


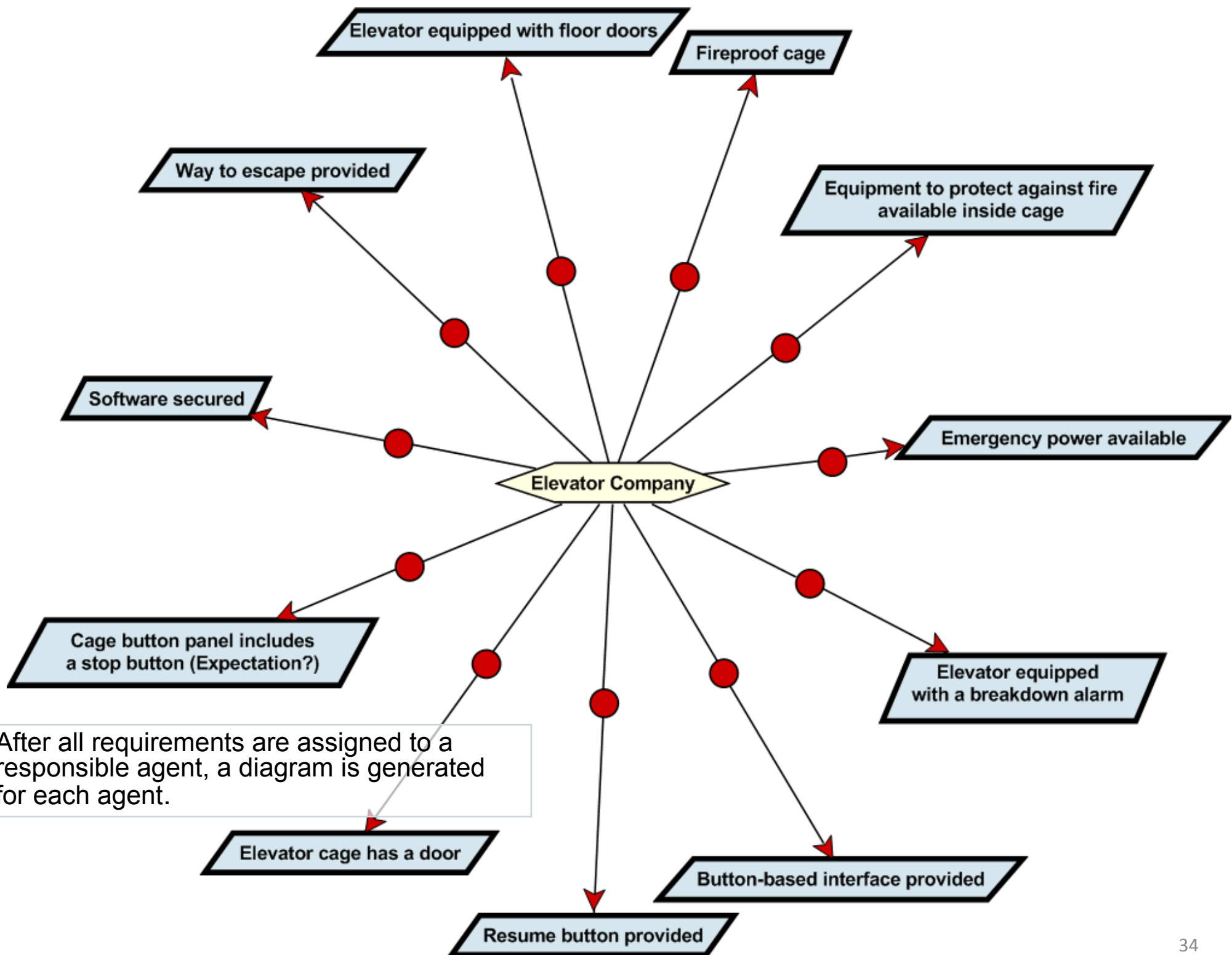


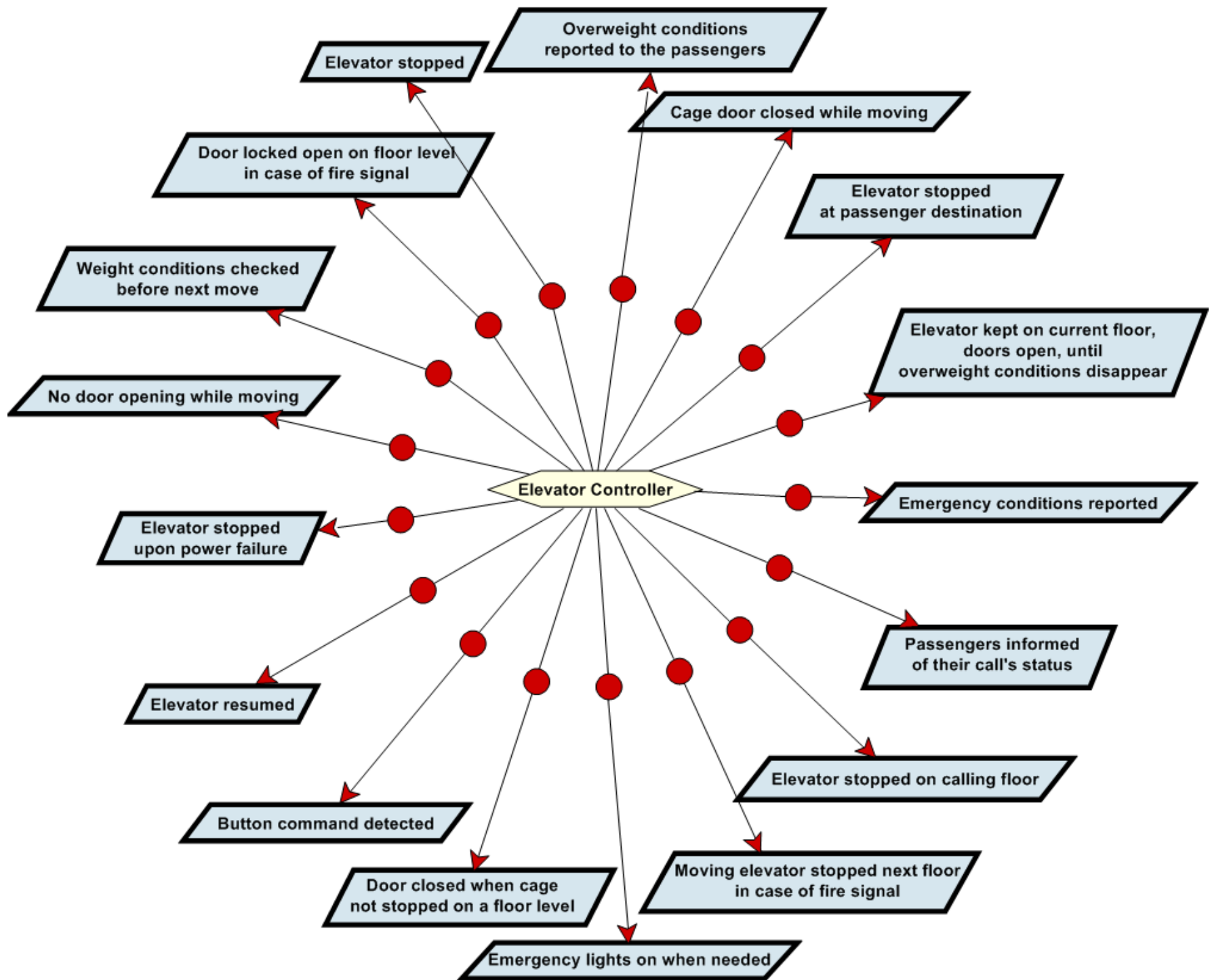
First shot of what an efficient elevator system should be:



« System protected against fire » with responsibilities:







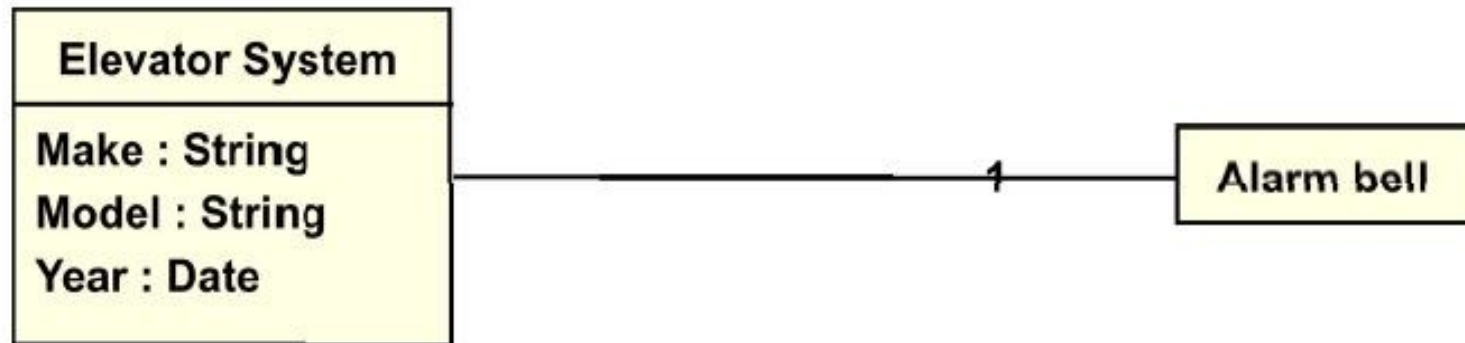
*The **Object model** is used to define and document the concepts of the application domain that are relevant with respect to the known requirements and to provide static constraints on the operational system that will satisfy the requirements.*

Part of the Object model are objects pertaining to the stakeholder's domain and other objects for expressing requirements or constraints on the operational model.

*There are three types of objects: **entities**, **agents**, **relationships**.*

- ✓ **Entities**: independant (needn't refer to other objects) and passive (can't perform operations) objects; may have attributes whose values define a set of states the entity can transition to
- ✓ **Agents**: independant and active objects; they can perform operations that usually imply state transitions on entities (ex: elevator company, passenger, elevator controler)
- ✓ **Associations**: dependant and passive objects that may have attributes (ex: "**At**" that links a cage and a floor)

Ex: entity « Alarm bell » as a component of the « Elevator »:

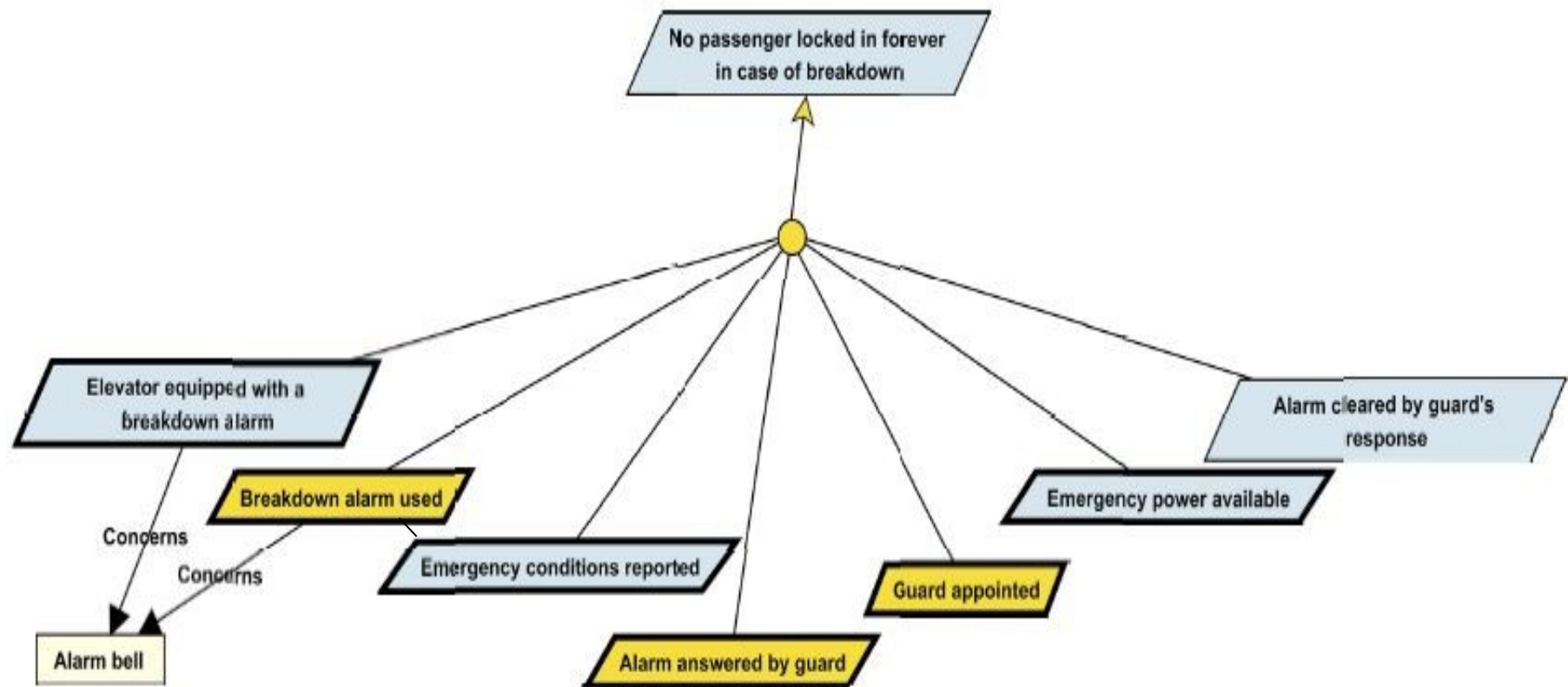


Object identification is driven by the goal definition process. Most goals's short and long definitions refer to domain objects being modelled and documented.

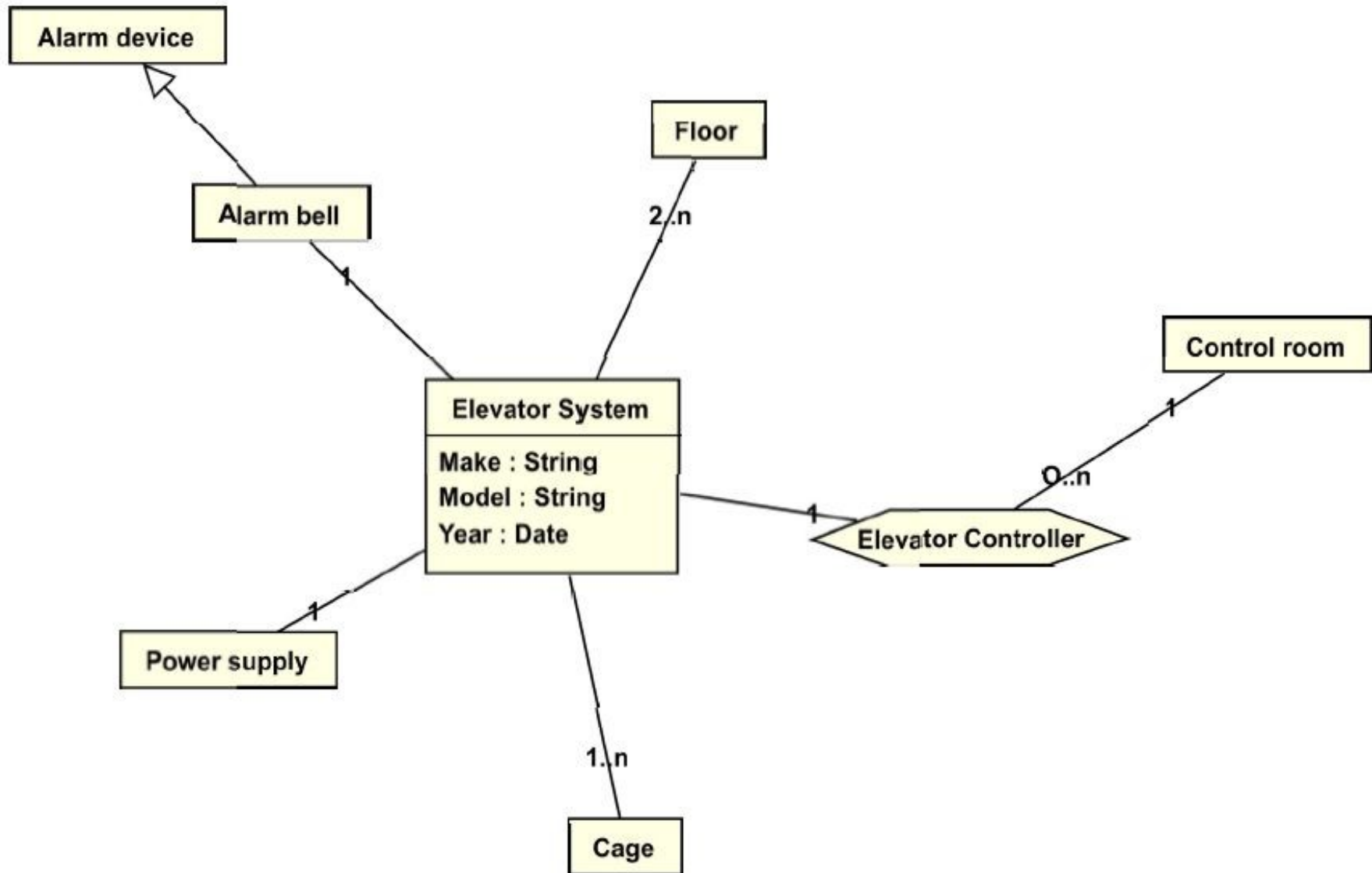
All modelled objects shall have their own entries in the glossary section of the requirements document. During review of the object model, stakeholders will formally agree on a common vocabulary.

Another way to identify new objects consists in looking at the requirements and discover the system components they are necessary for satisfying them.

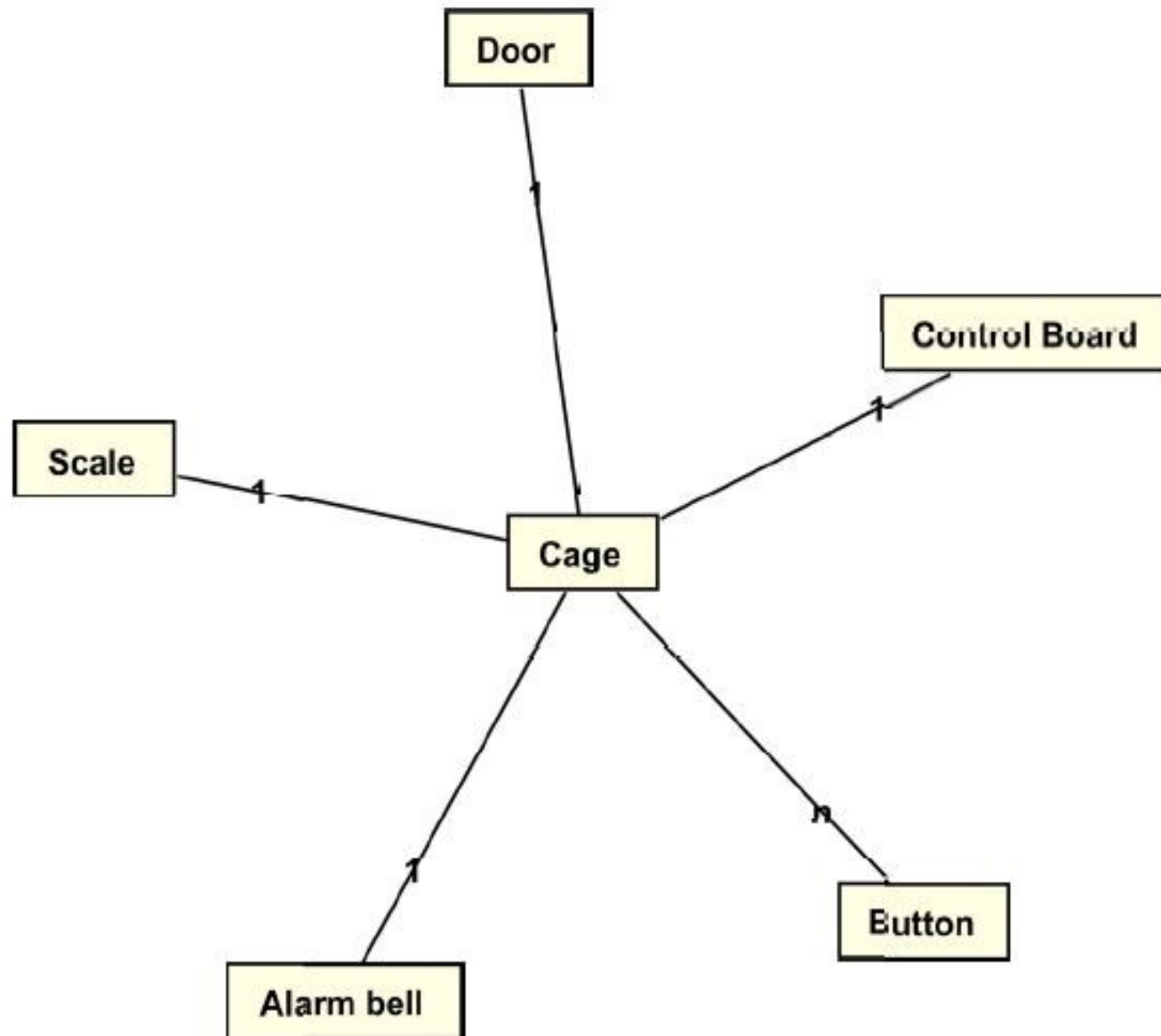
Objects may be represented in goal diagrams; the *concerns* relationship is used to link a requirement to the objects that are needed for it to be satisfied. Identifying those objects shall further restrict the space of solutions that can be proposed by the future system provider. Ex:



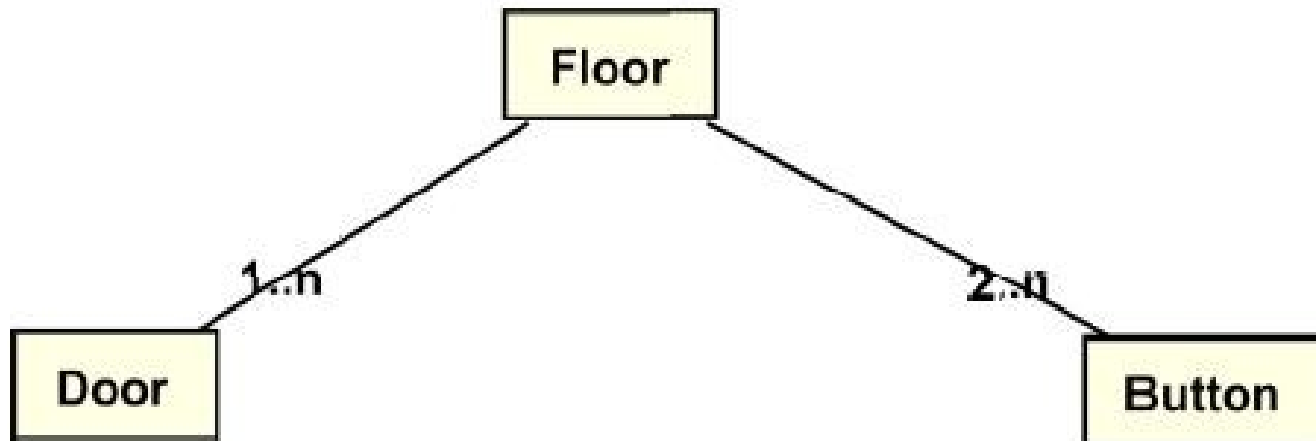
Object « Elevator system »:



Object « Cage »:



Object « Floor »:



The KAOS model is compliant with UML class diagrams in that KAOS entities correspond to UML classes; and KAOS associations correspond to UML binary association links or n-ary association classes. Inheritance is available to all types of objects (including associations).

Objects can be qualified with attributes.

The **Operation model** describes all the behaviors that agents need to fulfill their requirements.

Behaviors are expressed in term of **operations** performed by agents.

Ex: press button, enter, exit (perform by passengers) ; open doors, close doors, move up, move down (perform by elevators).

Those operations work on **objects**: they can create objects, trigger object state transitions and activate other operations (by sending an **event**).

Operations are justified by the goal they operationalize: an operation without a goal means there're missing goals, while a requirement without operations is just wishful thinking.

Operations identification sources:

- stakeholders, that typically describe processes rather than goals during interviews; the analyst then asks specific questions in order to identify the reasons behind the existing processes and hence unveil the goals that justify these processes;
- existing requirements: operations explain how they have to be realized.

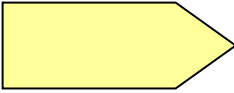
Requirements can be operationalized by:

- some object(s)
- some agent behavior(s)
- a combinaison of both

They are respectively requirements that describe static, dynamic, and both properties on the system:

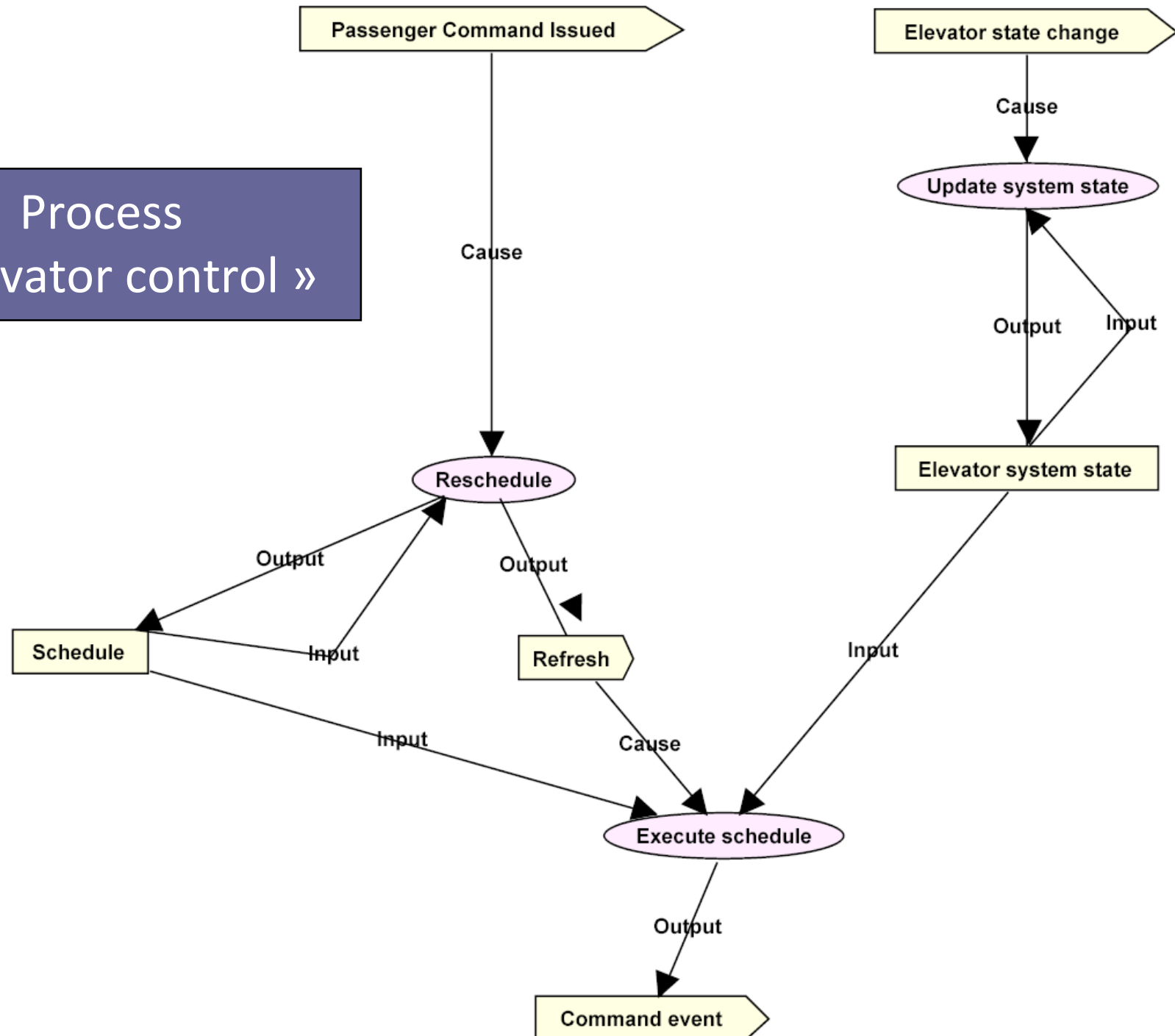
- the requirement « Elevator equiped with floor doors » will be operationalized by an object « Floor door »
- requirements that describe dynamic system properties are operationalized by operations
- the expectation « Stop button used » will be operationalized with an operation « Push button » and the « Button » entity

In a process, operations are represented as ovals, concerned objects are connected to the operations by means of **input** and **output** links.

Events are represented as: . Events may be external or produced by operations (through an output link). They may start (cause) or stop operations.

An operation model (or process) typically composes operations performed by one or several agents to achieve a requirement.

Process « Elevator control »



Completeness criterion 3 : to be complete, a process diagram must specify

- (i) the agents who perform the operations
- (ii) the input and output data for each operation.

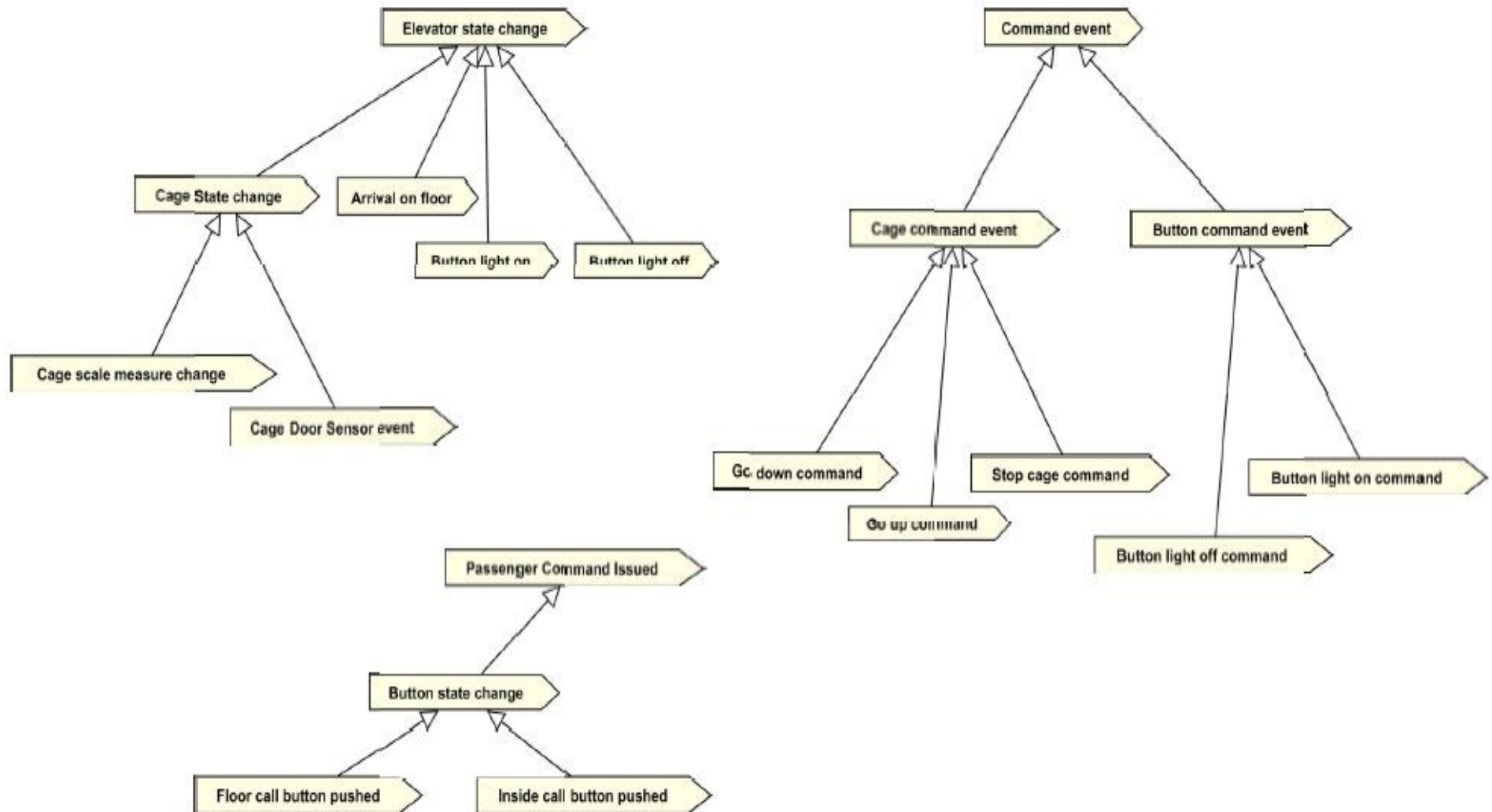
Completeness criterion 4 : to be complete, a process diagram must specify when operations are to be executed.

Completeness criterion 5 : all operations are to be justified by the existence of some requirements (through the use of operationalization links).

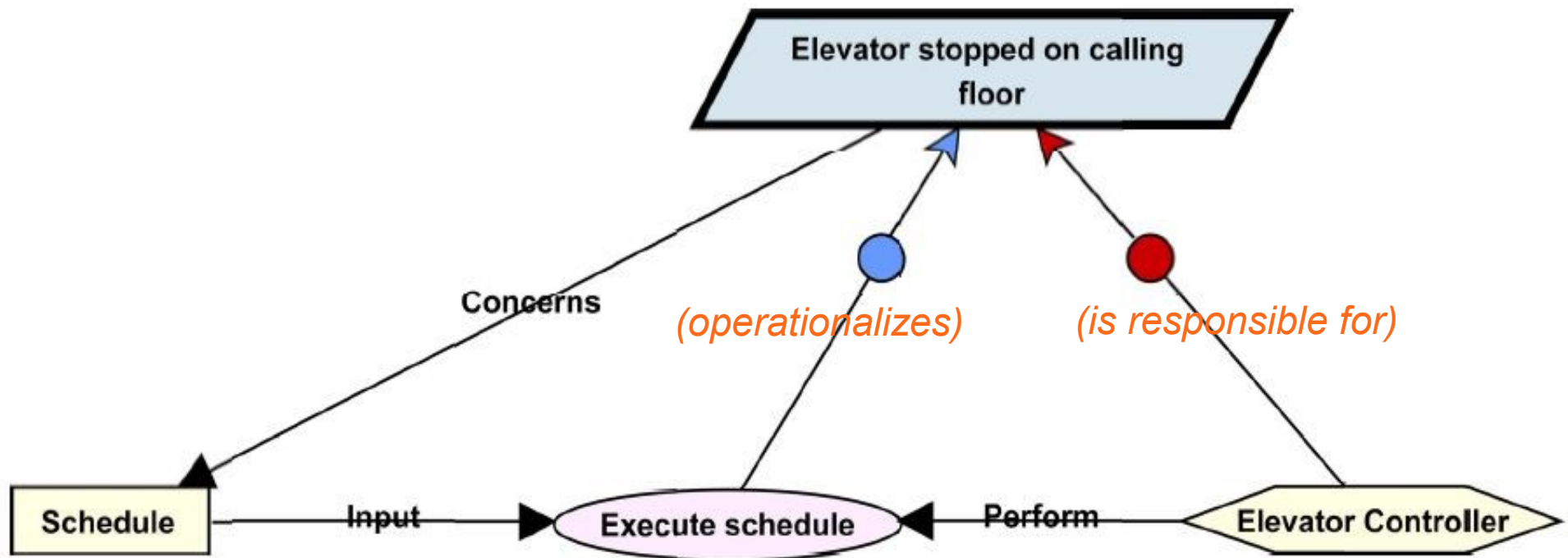
Operations can be **triggered explicitly** by an event. If no event is specified, the operation will be triggered **implicitly** when all the data needed in input are available (data flow).

Requirements left unoperationalized are called « open requirements »; solution providers have the complete freedom to address the requirement as they want. The less open requirements are left in the model, the more precise the requirements document will be by eliminating possible implementations.

Partial classification of the events of the process:

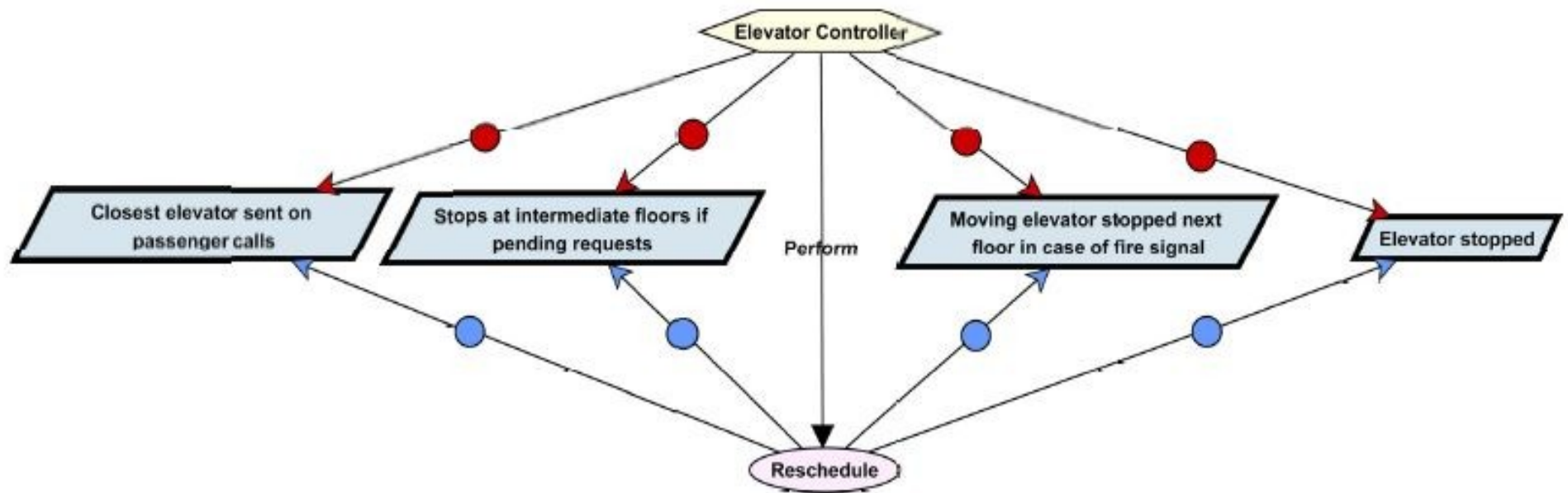


Typical instance of a process pattern KAOS analysts have to reproduce in all projects:



A requirement is said to be "closed" when this triangular relationship **Responsability-Operationalization-Performance** has been established.

Operationalization diagram for the *Reschedule* operation:



Dealing with **obstacles**.

Obstacles are situations in which a goal, a requirement or an expectation is violated. The obstacle is said to "obstruct" the goal, requirement or expectation.

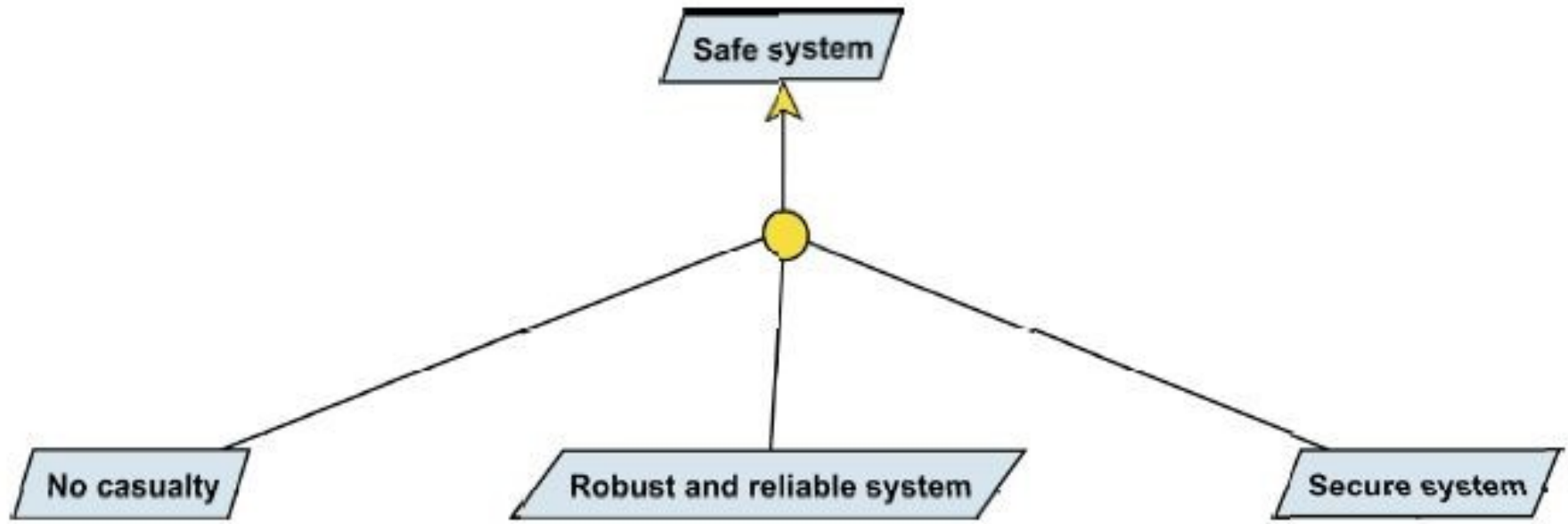
Dealing with obstacles is very important for safety-critical systems: it allows analysts to identify and address circumstances at requirements engineering time (instead of at programming or running time) in order to produce for instance robust requirements or new requirements to avoid or reduce impacts of obstacles. The result will be a more reliable software.

Obstacle analysis with KAOS is a goal-oriented activity. It begins with exploring the goal model and with negating each goal in turn.

As a goal G is refined into goals G_1, \dots, G_n if and only if G_1, \dots, G_n all together imply G , we know that if goal G is violated, it is because at least one of its subgoals is violated.

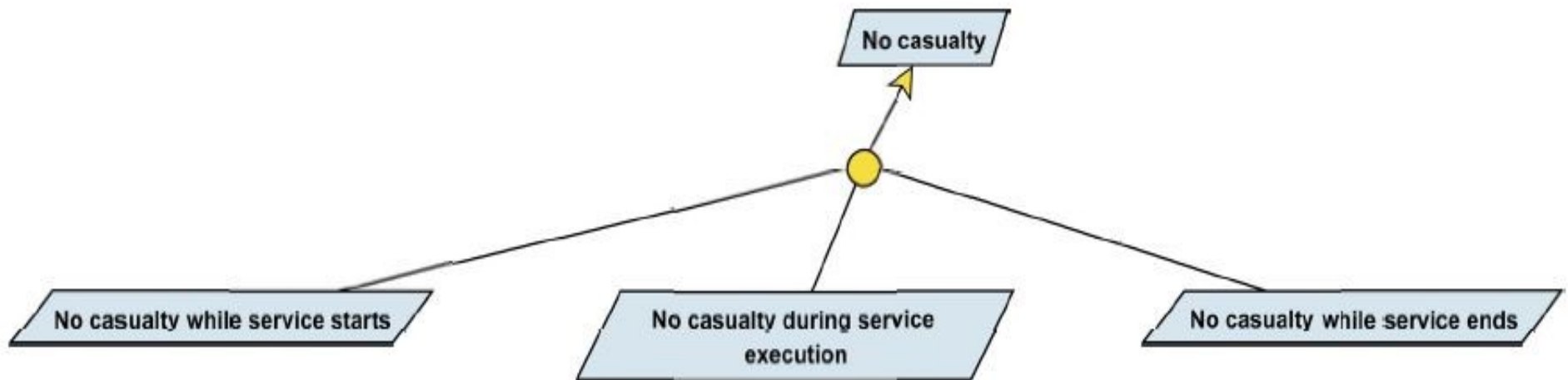
Each leaf obstacle is then reviewed with domain experts to study whether it is worth considering it in the obstacle analysis. It allows the analyst to prune the obstacle space and to focus on the most relevant obstacles.

Application to:



- ❖ casualty occurs during system use
- ❖ some system component breaks down
- ❖ somebody succeeds in hacking into the system.

Other application:



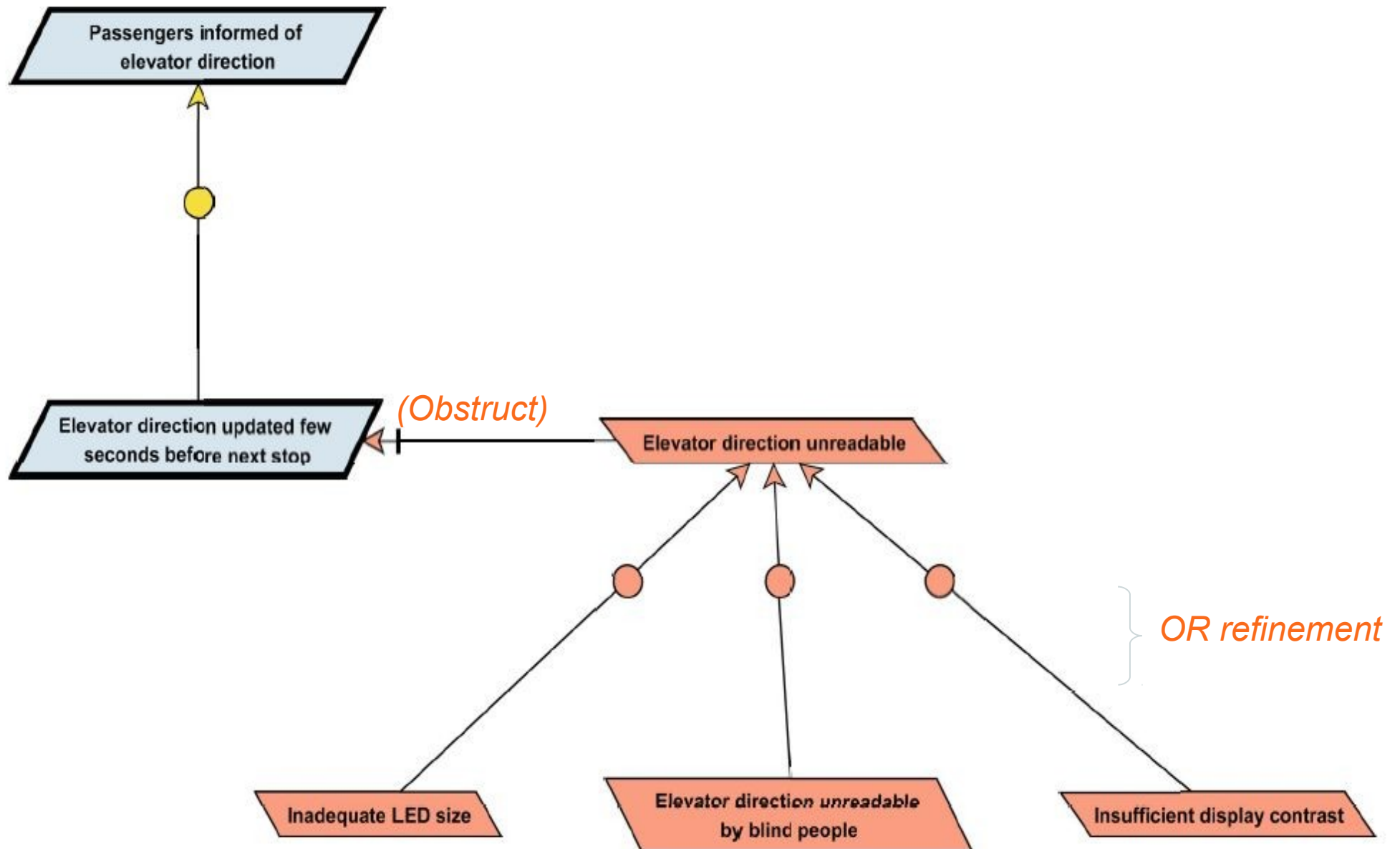
- ❖ casualty occurs at service start
- ❖ casualty occurs during service
- ❖ casualty occurs at service end.

Next, conditions for the obstacle to be raised are looked for. Typical ways used to identify obstacles start by considering component failures or people behaving in an unexpected way, maliciously or because of some people's capability limitation (disable people, children, ...).

Ex: an obstacle to the goal "*Elevator direction updated few seconds before next stop*" occurs when blind people want to use the elevator system as they can't read the indications.

Obstacles are linked to the goals they obstruct (**obstruction** link). Obstacles are refined the same way as goals, but **while goals are generally 'AND-refined', obstacles are most often 'OR-refined'**.

Obstacle to « Passengers informed of elevator direction »:



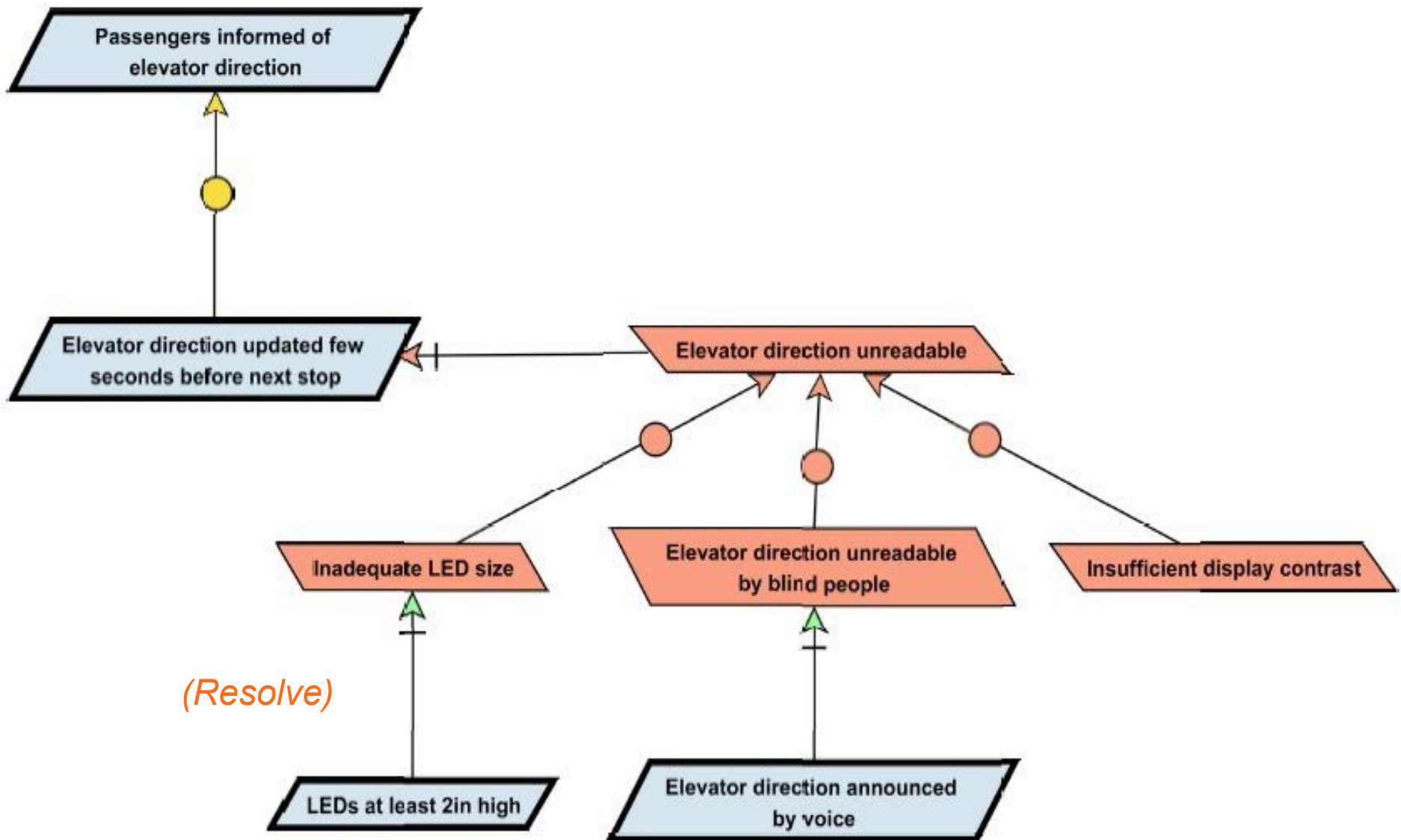
Having identified the conditions for it to occur, the analyst can fix the obstacle in several ways. A first way consists in defining new requirements that prevent the obstacle from occurring.

In the preceding example, one can add a new requirement: "Elevator direction announced by voice upon floor arrival".

New requirements are linked to the obstacle they resolve by a **resolution** link.

Other requirements for blind people could be "Floor level announced by voice upon floor arrival" and "Call buttons in Braille".

Obstacle resolution for « Passengers informed of elevator direction »:

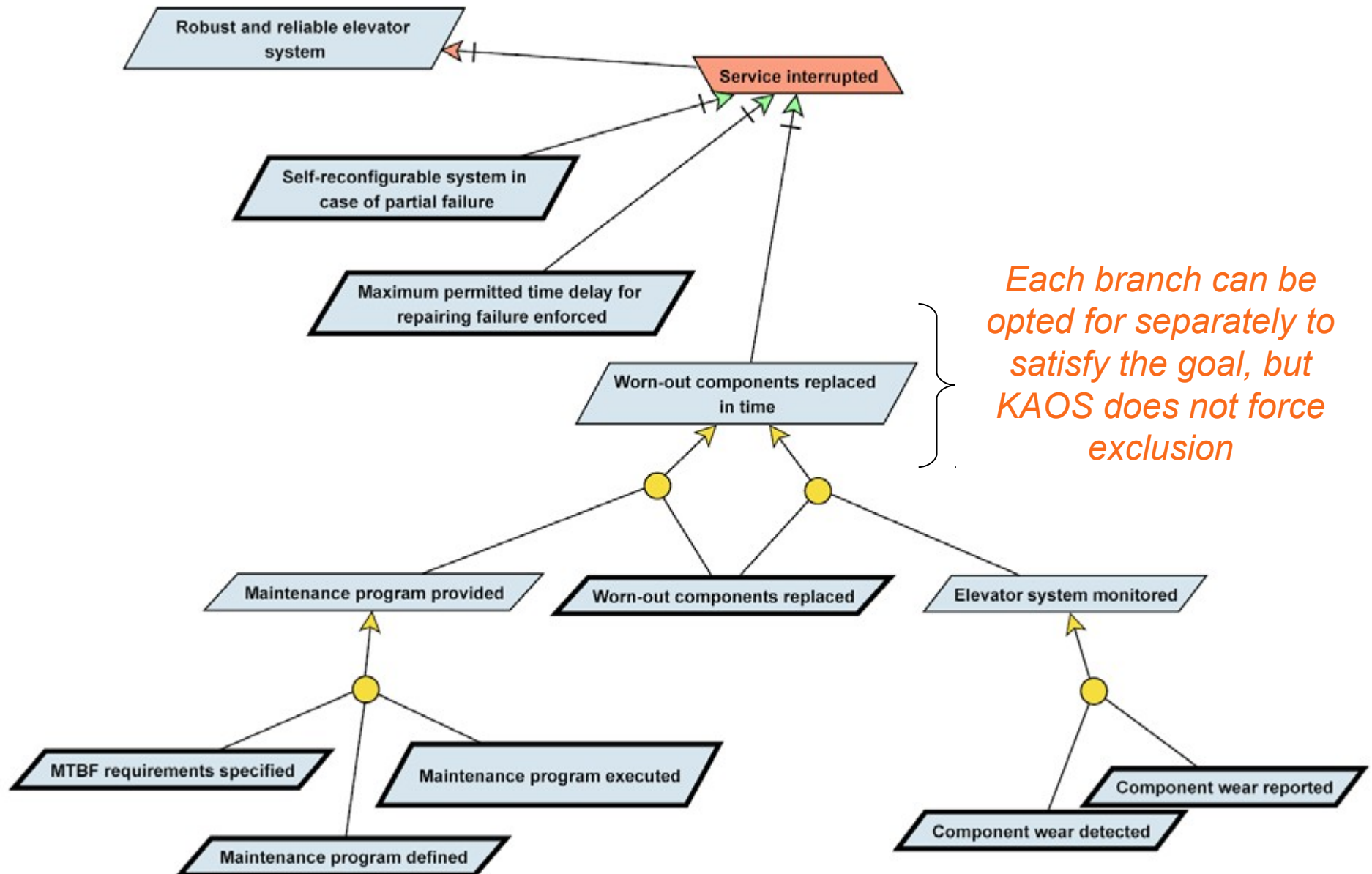


An important source of obstacles concerns system (and components) **reliability** and **robustness**. Reliability and robustness requirements aim at preventing failures to occur. One can for instance introduce requirements in terms of Mean Time Before Failure (MTBF), or in terms of a maintenance program to replace components systematically at the right time, and so on.

A second way to fix an obstacle is **to restore** the **obstructed goal** once the obstacle occurs. For instance, if an elevator failure puts the system out of order, a restoration requirement can prescribe the maximum delay within which the system must be repaired.

- A third way to fix an obstacle consists in **minimizing its effects**.
 - Suppose the building is equipped with two elevator cages, one serving floors 1 to 10, the other one floors 10 to 20. If one cage goes out of order, a requirement should prescribe that the system shall be self-reconfigured so that the other cage provides service to all floors (from 1 to 20).
- It's also possible to fix obstacles by modifying the KAOS model in different ways, for instance by substituting an agent with a more "capable" one, by weakening the obstructed goal so that the obstacle is no longer relevant or by substituting a goal to another one.

Possible refinement of the goal « Robust and reliable elevator system »:



The structure of the **requirements document** generated from a KAOS model is borrowed from the IEEE 830-1998 standard for software requirements specification :

Table of contents

1. Introduction

1.1 Document purpose

1.2 System purpose

1.3 Definitions, acronyms, and abbreviations → glossary derived from the KAOS model

1.4 References

1.5 Overview

2. Overall description

2.1 System perspective

2.2 User requirements → a subsection for each goal diagram and a list of the requirements

2.3 User characteristics

2.4 Constraints

2.5 Assumptions and dependencies → assumptions & obstacles not

2.6 Apportioning of requirements dealt with

3. System requirements

3.1 System architecture

→ decomposition of the system by

KAOS agents (responsability model)

3.2 Object model

→ diagrams of the KAOS object model

3.3 Operation model

→ diagrams of the KAOS operation model

A major benefit of KAOS is that it provides a bidirectional traceability between the problem description and the expected solution description.

As systems require frequent changes, it is vital to keep the requirements document up to date with respect to the current state of development.

Requirements documents with KAOS tend to be more complete.

Completeness must be understood by looking at the **five completeness criteria**: a complete KAOS model leaves no space for wishful thinking (a goal not refined), requirements with no responsible, unjustified operations, and operations for which we ignore who will execute what and when.

Completeness of the goal model also relies on 3 points :

- ✓ the quality of the requirements definition phase (interviewing stakeholders, if operated properly, leads to a good goal covering rate)
- ✓ the validation meetings during which stakeholders review consolidated goal diagrams (conflicts are addressed, forgotten goals have a new chance to come up)
- ✓ refinement techniques that guarantee completeness and consistency (generic refinement patterns that require formal notations for safety-critical applications).