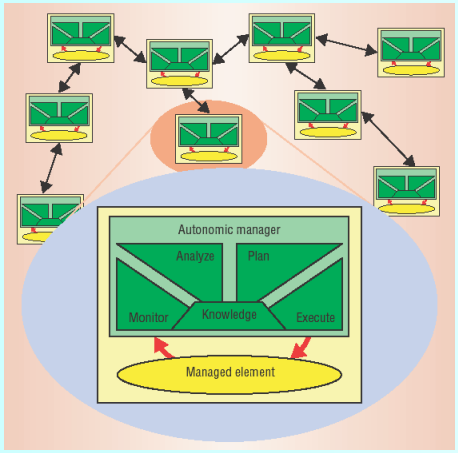
DEAS 2005

DESIGN AND EVOLUTION OF AUTONOMIC APPLICATION SOFTWARE

Towards Requirements-Driven Autonomic Systems Design

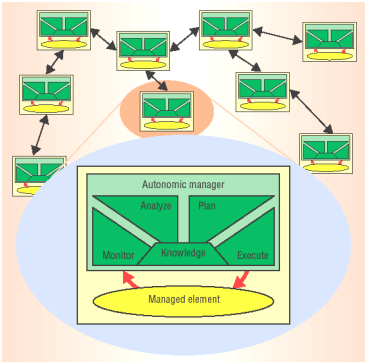


Alexei Lapouchnian
Sotirios Liaskos
John Mylopoulos
Yijun Yu

Agenda

1. Autonomic computing
2. Goal-oriented requirements engineering
3. High-variability designs
4. Towards autonomic computing systems
    1. From goals to autonomic systems
    2. A hierarchical autonomic architecture
    3. Goal model-based autonomic behaviors
5. Conclusion

1. Autonomic computing



Requirements for AC systems

- *Why* autonomic computing in software engineering?
  - Moore's law: hardware speed doubles every 18 months
  - Lehman's evolution law: software complexity is increasing
  - Moore's law versus Lehman's law => increasing software maintenance cost comparing to the hardware cost

J. Kephart and D.M. Chess. "The vision of autonomic computing". IEEE Computer Journal. 36(1):41-50. 2003.

Requirements of AC systems

Three basic ways to build such systems

- Designed to support all possible behaviors (our proposal)
- Delegating tasks to external agents (agent-oriented)
- To let evolution decides the better fits (evolutionary)

To model all possible behaviors, we need to analyze the requirements of the AC system to be built

An autonomic system requires:

- Adaptive: evolves like biology systems
- Intelligent: reasons like expert systems
- Self-awareness: senses the environment and self
  what's going on, what's wrong, what's better, what to change …
- Self-conduct: plans without intervention
  what to do, how to change, …

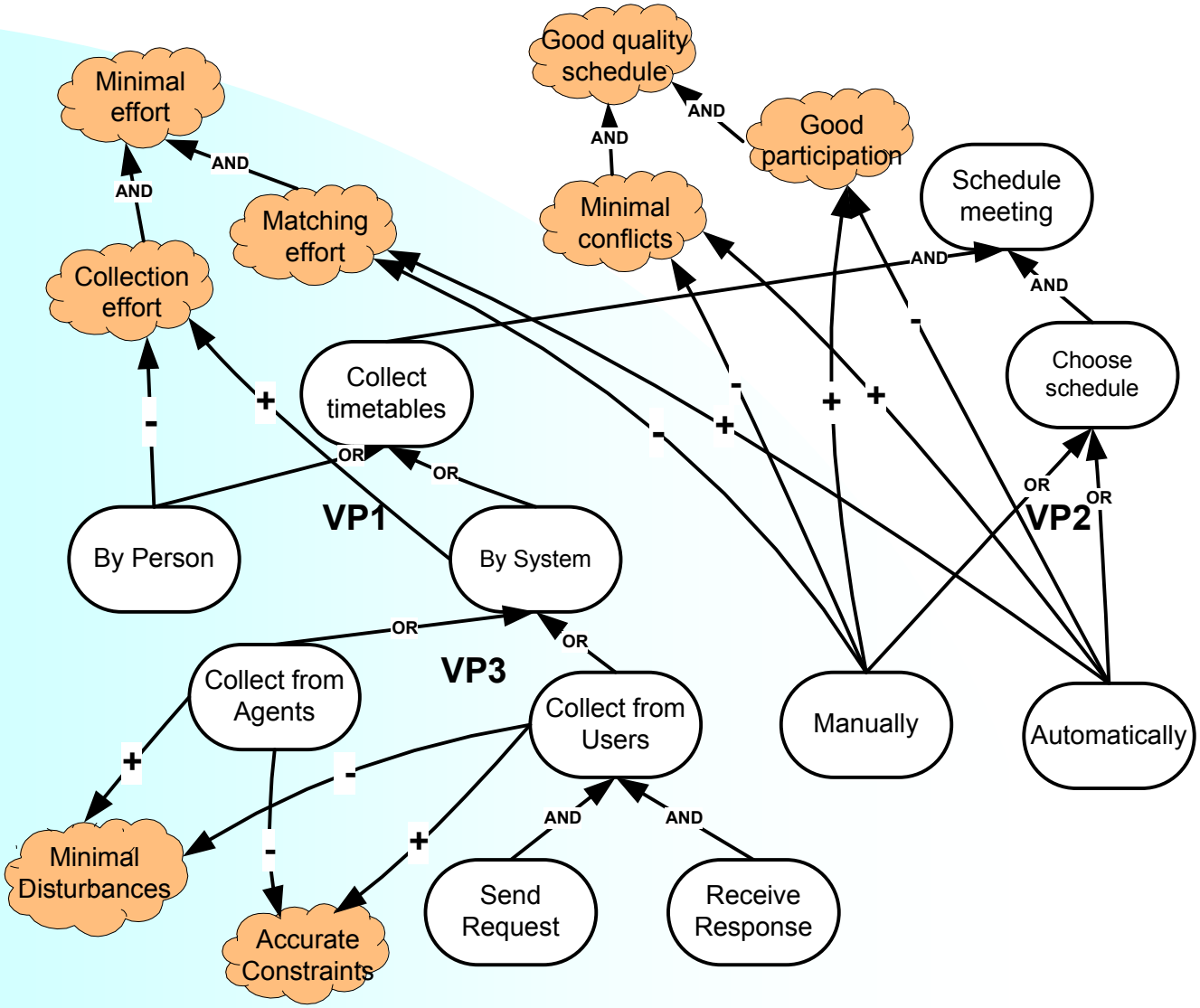
2. Goal-oriented RE

- Before designing a system, analyze the goals (intentions) of stakeholders
  - *divide and conquer*: ask *Why* and *How* to establish solutions to the problem
  - The solutions are result of AND/OR decomposition of the problem
- Functional requirements are hard goals
- Non-functional requirements (NFR) are soft goals

  NFRs are used to model quality attributes associated with metrics
- The result of GORE process is a *goal model: the causal dependencies among requirements*

A. van Lamsweerde. “From systems goals to software architectures”. FSE. 2004.
L. Chung, B.A. Nixon, E. Yu, J. Mylopoulos. *Non-functional requirements in software engineering*. Kluwer Academic Press. 1999.

Meeting scheduler example



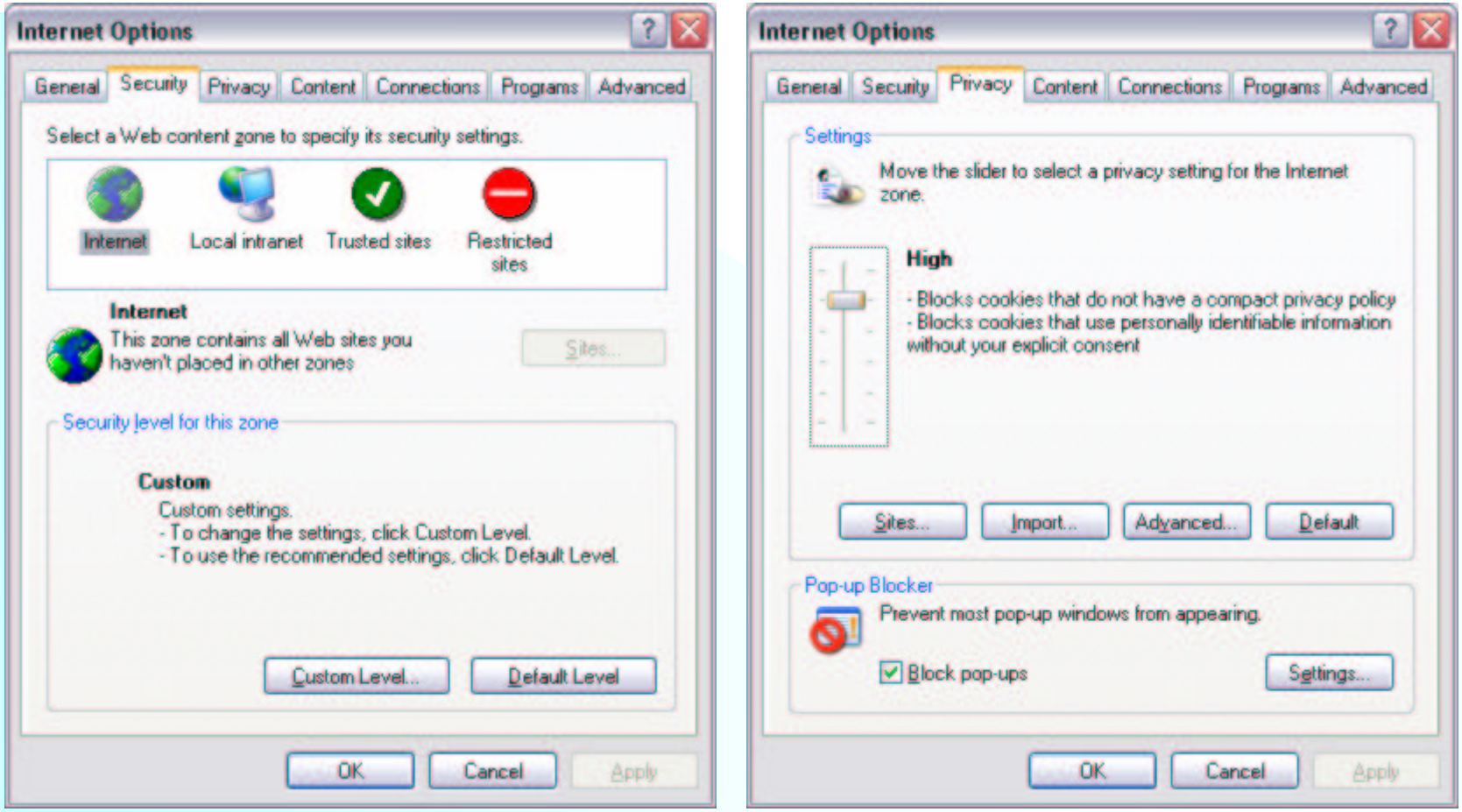
3. Variability: more than one ways to attain a goal

- A key difference between autonomic systems design and traditional design: *can the system adapt its behavior to the environmental changes?*
- Alternative ways to solve a problem are captured by the OR goal decompositions
- Alternatives in the goal model captures the variability in the problem domain
  - Variability in problem domain must be reflected in the solution domain as alternative configurations, behaviors and structures
  - The selection criteria for THE solution can be guided by softgoals in the goal model

Toward high-variability designs

- Variability in problem domain must be reflected in the solution domain as alternative *configurations*, *behaviors, structures, concerns, etc.*
  1. *Configure* variability*:* feature models in the product-line family software
  2. *Behavioral* variability*:* transitional systems typically statecharts
  3. *Structural* variability*:* components compositions patterns in software architectures
  4. *Concerns* variability: aspect-oriented compositions
  5. … and so on so forth …

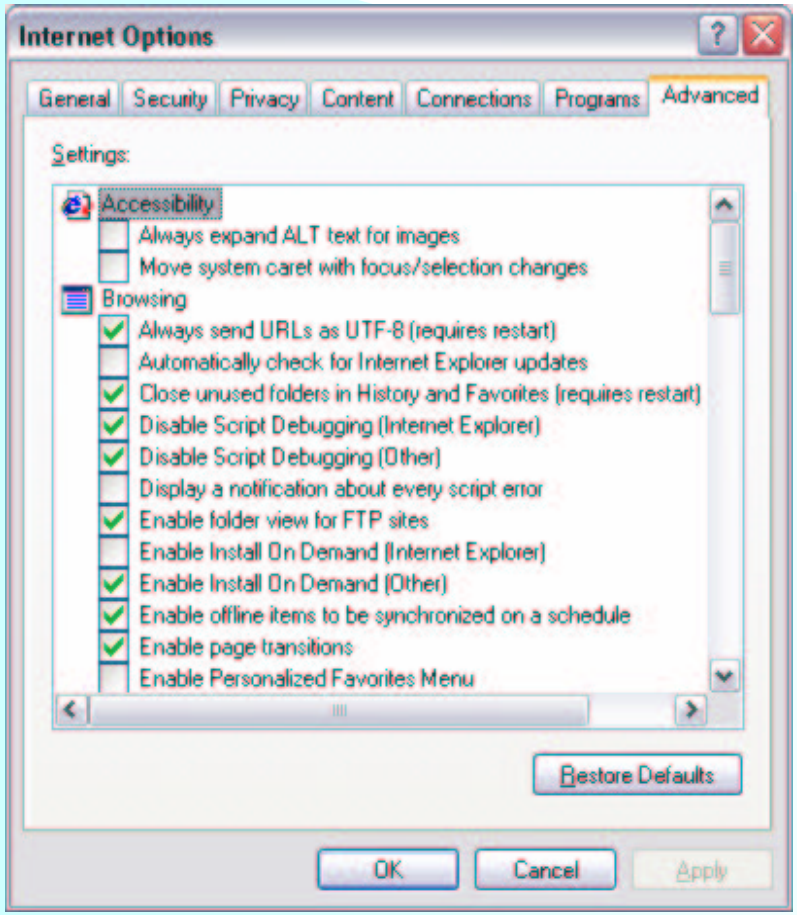
Configuration variability

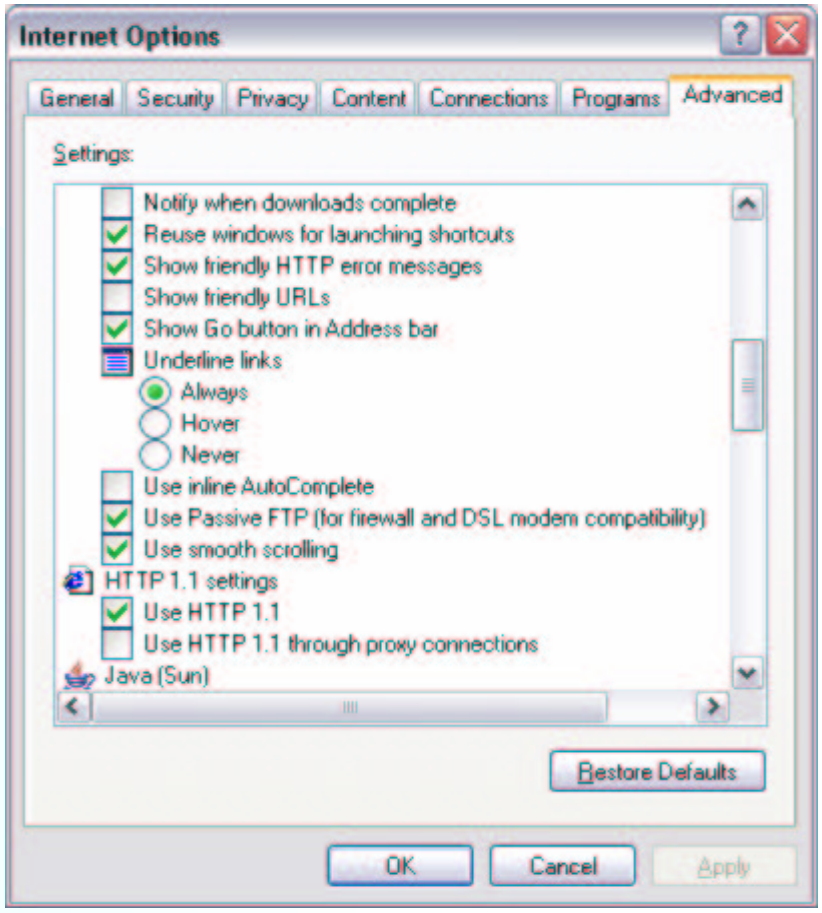


B. Hui et al. “Requirements analysis for customizable software: goals-skills-preferences farmework”, RE’03.

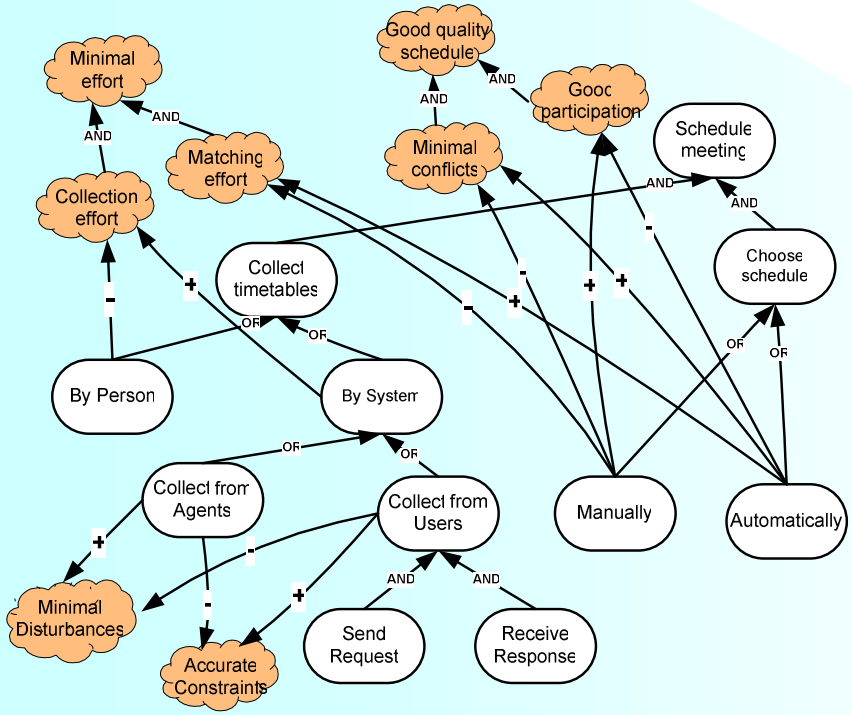
S.Liaskos et al. “Configuring common personal software: a requirements-driven approach”. To appear, RE’05.

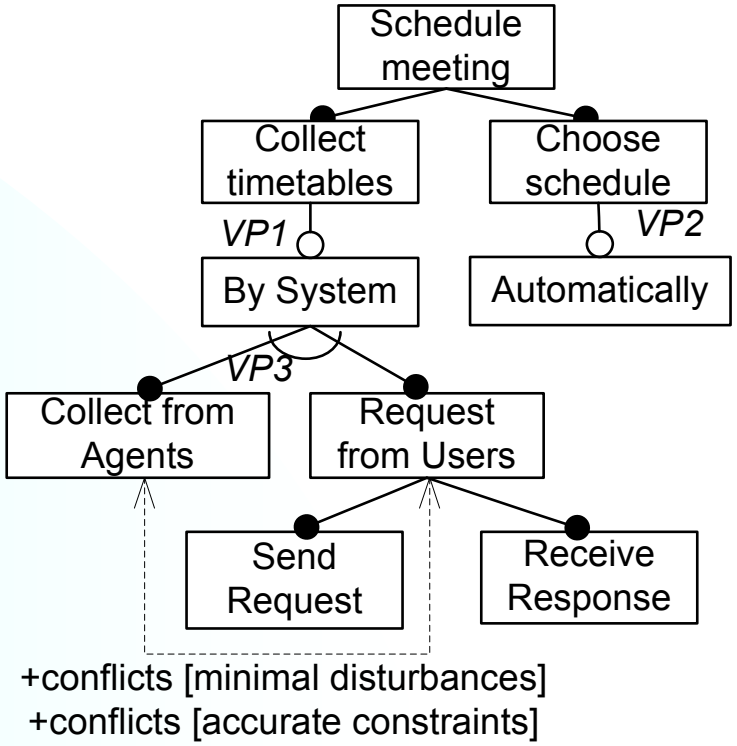
you may not want to configure in more details



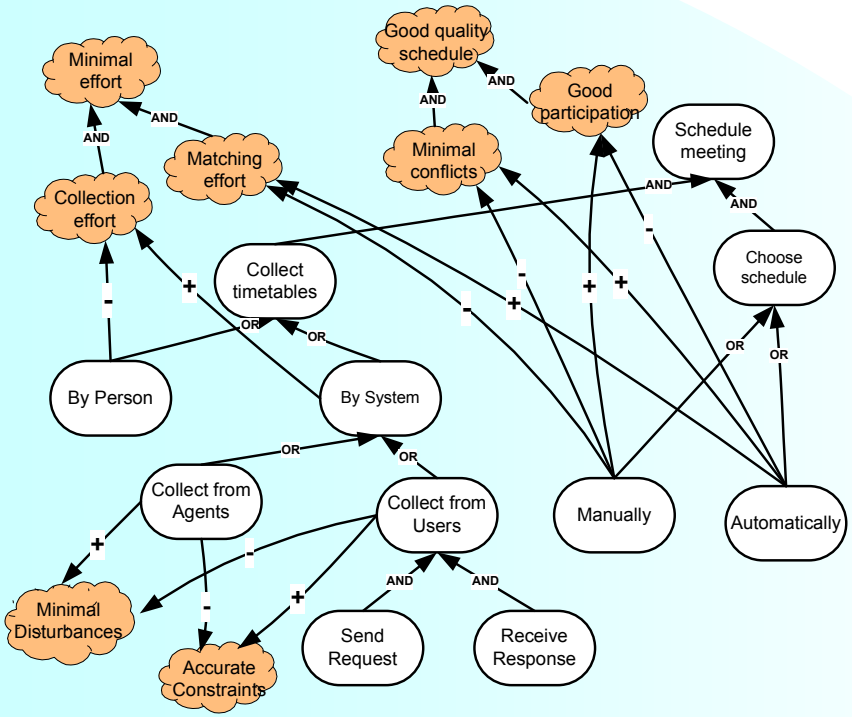


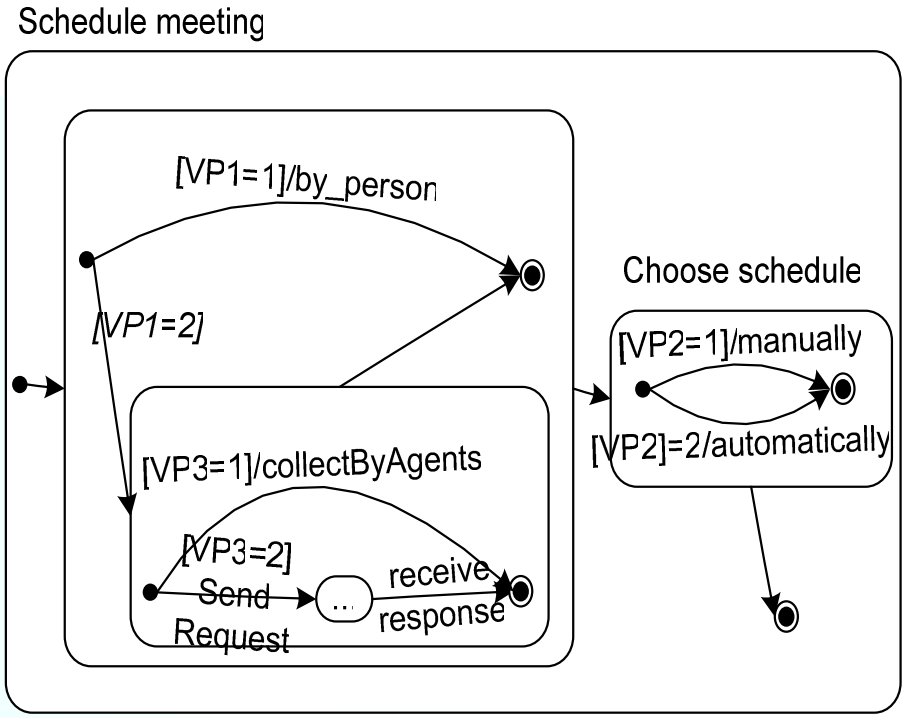
3.1 Converting into feature model



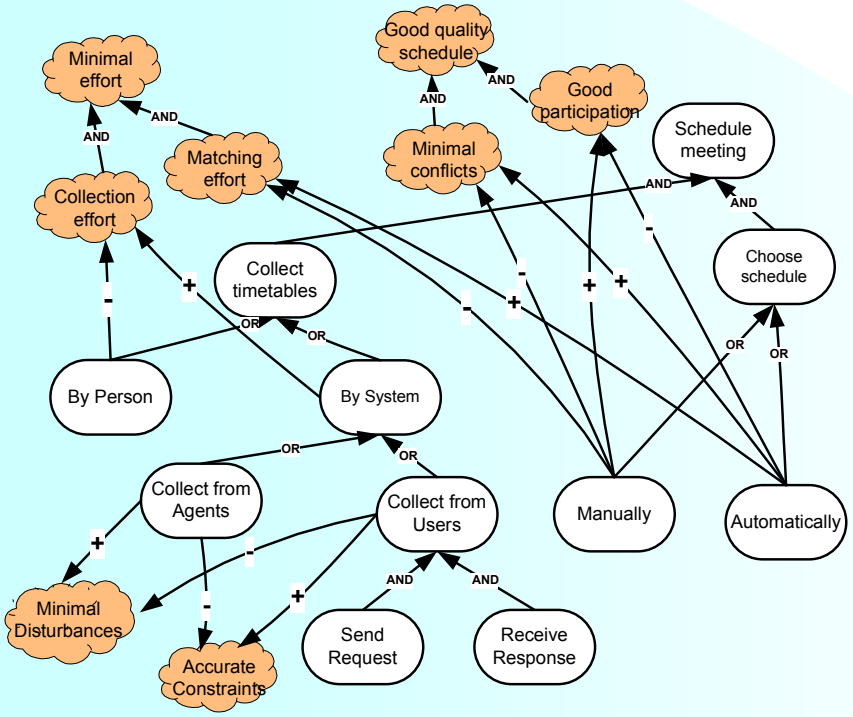


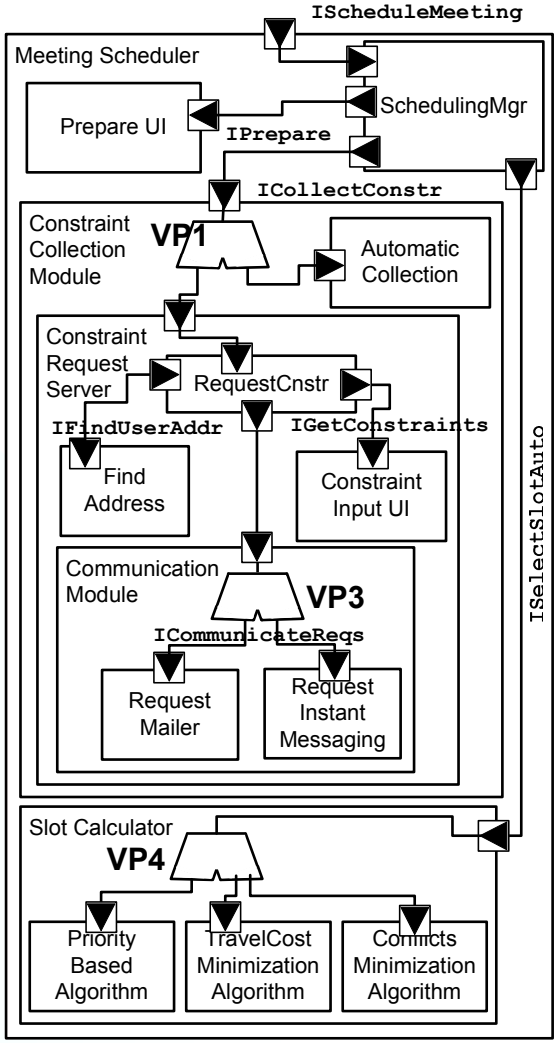
3.2 Converting into statecharts





3.3 Converting into ADL





Light-weight enrichments

- Goal models in the AND/OR graph form need to be enriched with design-specific information to transform into a design
- Keep it simple stupid: such enrichments are light-weight: minimal information to derive the design
- Keeping the traceability among the variabilities is crucial to the design of AC systems

Enriching the goal models

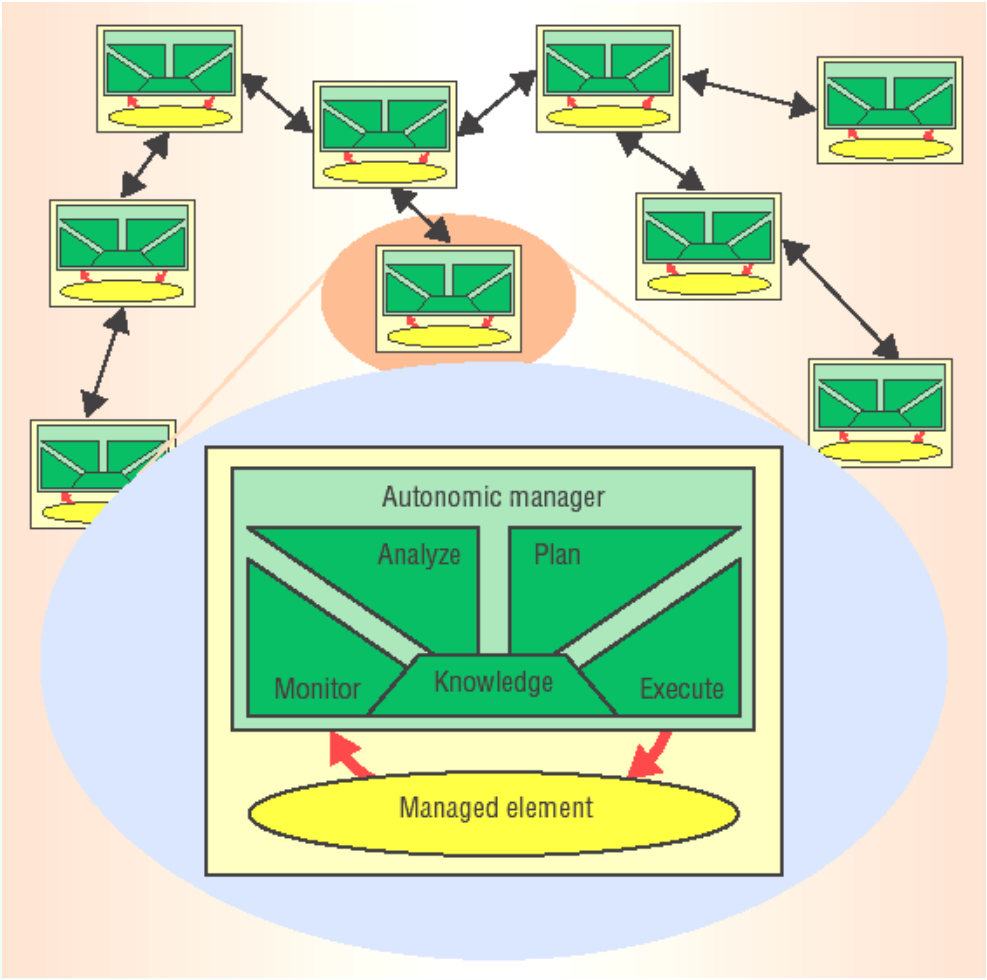
1. Feature models?
   - Mandatory or Options: System/Non-System boundary
   - Inclusive or Alternatives: OR/XOR
2. Statecharts?

   If one knows data dependencies among leaf goals, control is either sequential (;) or parallel (||).
3. Software architecture?

   Data bindings for inputs and outputs (corresponds to goal *topics*)

4. Towards AC systems

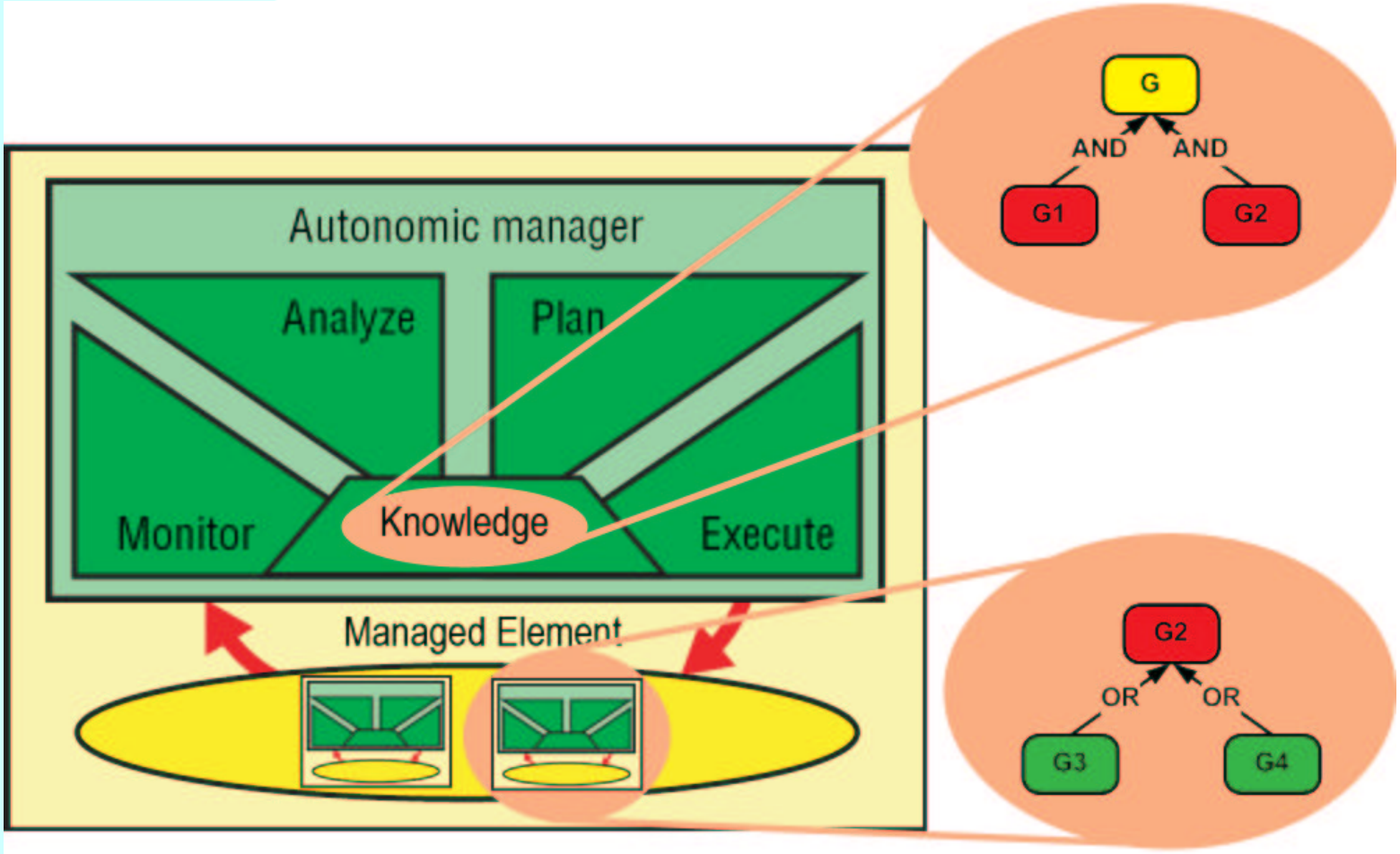
- Goal models + high-variability design patterns => AC systems
- An autonomic manager element (MAPE)
  - Knowledge
  - Monitor
  - Analyze
  - Plan
  - Execute



4.1 Goal models as the core knowledge

- Autonomic elements are full-fledged intelligent agents; Goal models represent the requirements of agents
- Goal models help autonomic elements:
  - ***Monitor***: goals-qualities-metrics (GQM) framework
  - ***Analyze***: NFR and GSP frameworks -- ranking alternatives through softgoals (quality criteria)
  - ***Plan***: Designing all alternatives and switching among them
  - ***Execute***: high-variability design translations from enriched goal models
- Thus, properly enriched goal models are the core knowledge for autonomic element

4.2 Hierarchical Autonomic Architecture



Advantages of the HAA

- Reduces complexity of AC systems
  - Propagating high-level concerns to low-level concerns as guiding policies
    => self-configuring
  - Propagating low-level metrics to high-level attention only when necessary (localize the changes)
    => self-managing
  - Monitoring costs of alternatives allows fast switching among them to reduce cost
    => self-tuning
  - Monitoring the satisfaction of alternatives allows fast detecting failures to invoke corrective tasks (modeled as delta-alternatives)
    => self-healing
- Supporting mechanisms:
  Top-down and bottom-up qualitative and quantitative goal reasoning algorithms

4.3 Goal-model-based autonomic behaviors

What do self-* behaviors mean for us in the MAPE feedback loops

- *Self-configuration* and reconfiguration
  Choosing the better configurations with respect to historical statistical data
- *Self-optimizing* (self-tuning)
  Monitoring NFR quality metrics with respect to the analytical data
- *Self-repairing* (self-healing)

  Monitoring FR with respect to the analytical data

5. Conclusion and future work

- Stakeholder requirements goals are the core knowledge of autonomic elements in order to make AC *applications* adapt for the better in users' perspectives
- Autonomic systems cost less at run-time with high-variability designs
- A hierarchical autonomic architecture is proposed to reduces the design-time complexity of AC systems
- Different autonomic behaviors are interpreted as different applications of goal models: goal monitoring, goal reasoning and goal-to-design translations

Future work:

- Implement the MAPE feedback loop with goal models
- Apply the goal-model based autonomic elements to AC software systems

Questions

Backup slides

- References
- Enrichments
- Translating patterns

References (1)

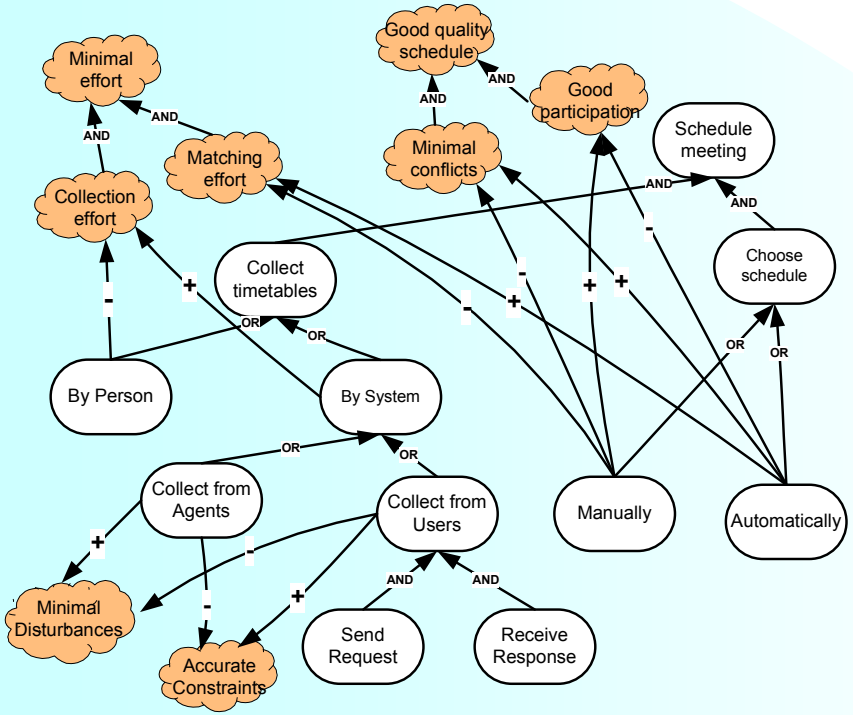
- **Autonomic Computing systems**
  - J. Kephart and D.M. Chess. “The vision of autonomic computing”. IEEE Computer Journal. 36(1):41-50. 2003.
- **goal-oriented requirements engineering**
  - A. van Lamsweerde. “From systems goals to software architectures”. FSE. 2004.
  - L. Chung, B.A. Nixon, E. Yu, J. Mylopoulos. *Non-functional requirements in software engineering*. Kluwer Academic Press. 1999.
- **goal-oriented software configuration**
  - B. Hui et al. “Requirements analysis for customizable software: goals-skills-preferences farmework”, RE’03.
  - S.Liaskos et al. “Configuring common personal software: a requirements-driven approach”. To appear, RE’05.
- **goal-oriented software tuning**
  - Y.Yu et al. “Software refactoring guided by multiple soft-goals”, REFACE@WCRE’03.
- **quality-driven software reengineering**
  - Ladan Tahvildari, Kostas Kontogiannis, John Mylopoulos: “Quality-driven software re-engineering”. Journal of Systems and Software 66(3): 225-239 (2003)
- **quality-based software reuse**
  - J.C.Leite, et al. “Quality-based software reuse”, CAiSE’05.
- **reverse engineering goal models**
  - Y.Yu, et al. “From goals to aspects: discovering aspects from requirements goal models”. RE’04.
  - Y.Yu et al. “Reverse engineering goals from source code”, to appear, RE’05.
  - S.Liaskos et al. “Configuring common personal software: a requirements-driven approach”. To appear, RE’05.

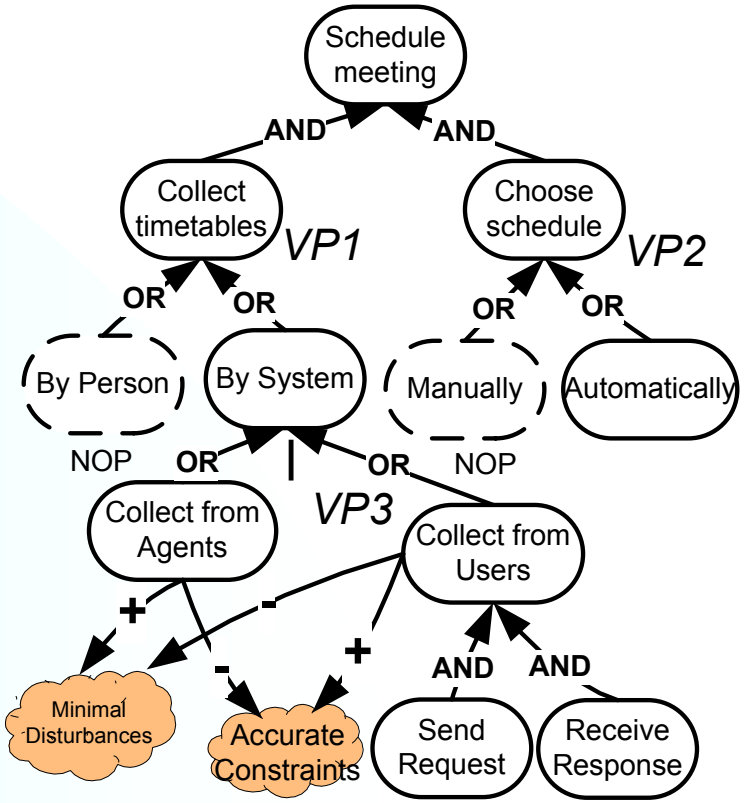
References (2)

On High-variability software design

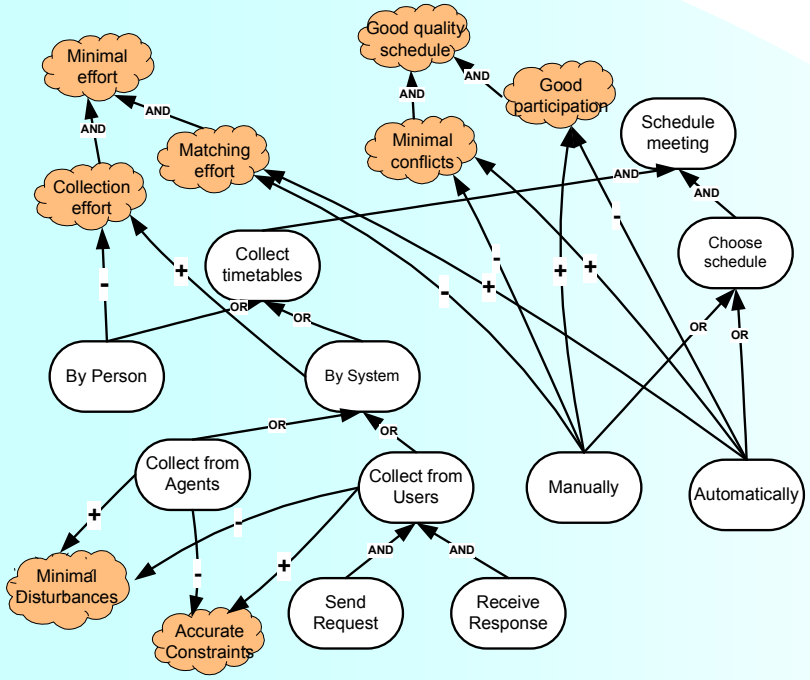
- **Feature model and product-line family**:
  - K.C. Kang et al. “Feature-oriented domain analysis (FODA) feasibility study”, SEI. 1990.
  - K. Czarnecki et al. *Generative Programming: Methods, Tools, and Applications*. Addison-Wesley, 2000.
  - D. Batory et al. “*Scaling Stepwise Refinements*”. ICSE 2003.
- **Statecharts**
  - D. Harel. “The STATEMATE Semantics of Statecharts”. TOSEM 5(4):293—333.
- **Software architectures and ADL**
  - L. Bass et al. *Software Architecture in Practice*, 2nd Ed. Addison-Wesley, 1998.
- **Aspect-oriented programming**
  - G. Kiczales. “Aspect-oriented programming”. EOOP. 1997.
  - C. Zhang et al. “Just-in-time middleware configuration using aspects”. AOSD’05.

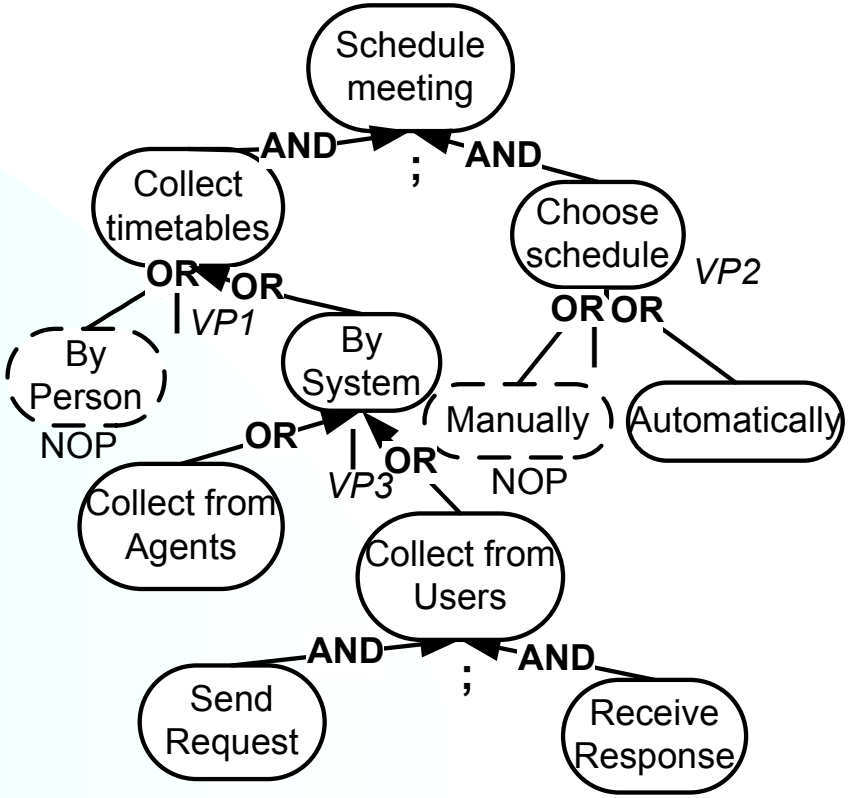
3.1b For configuring variability



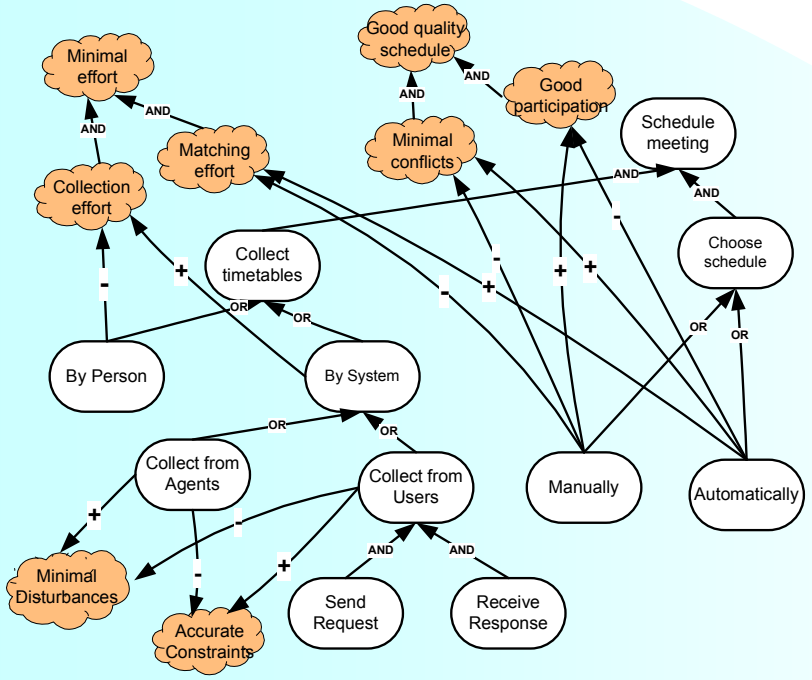


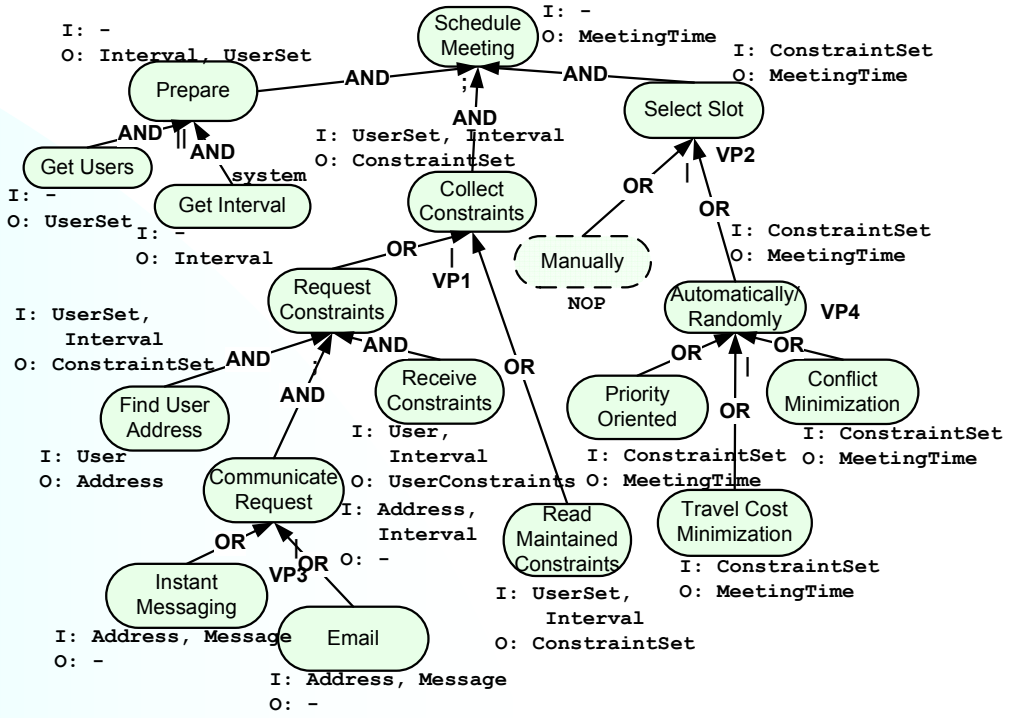
3.2b For behavioral variability



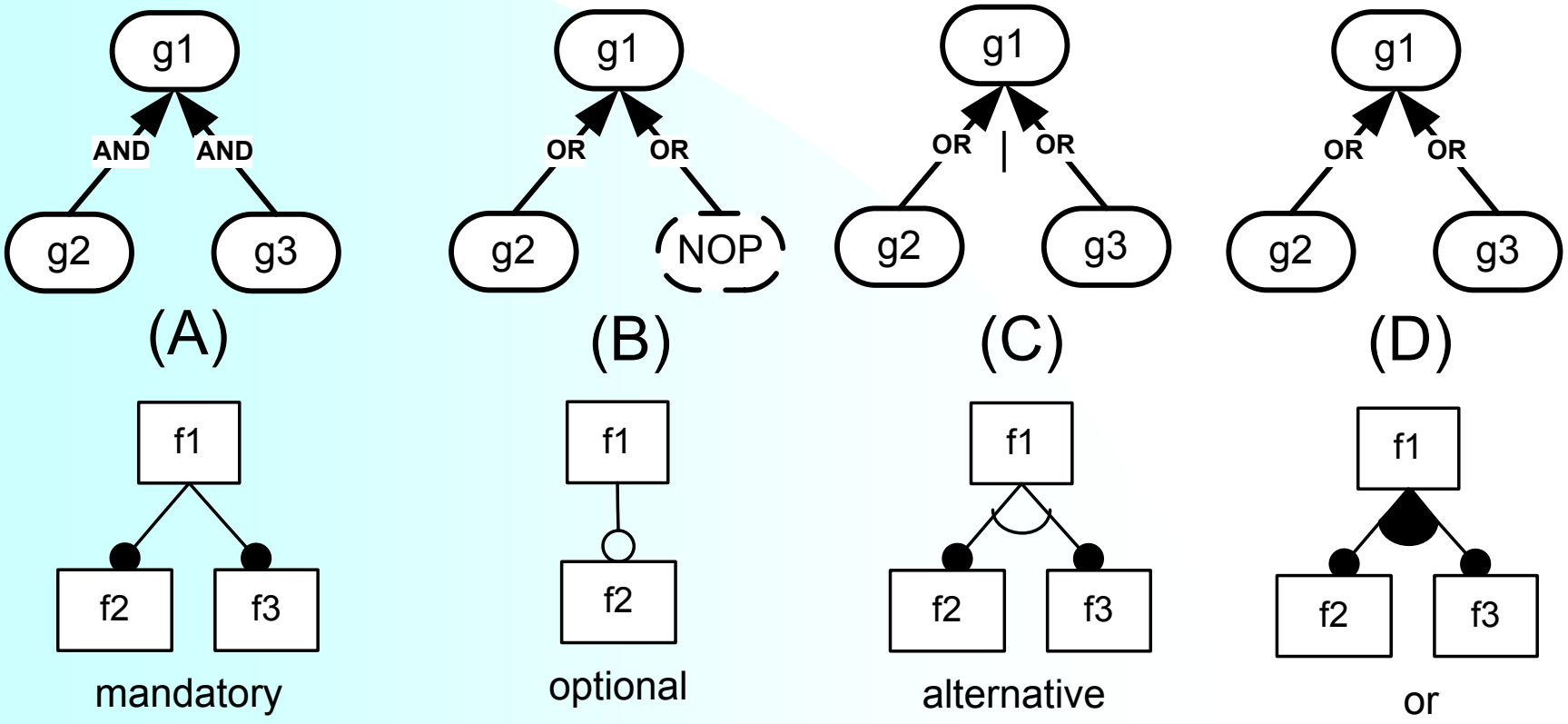


3.3b For structural variability

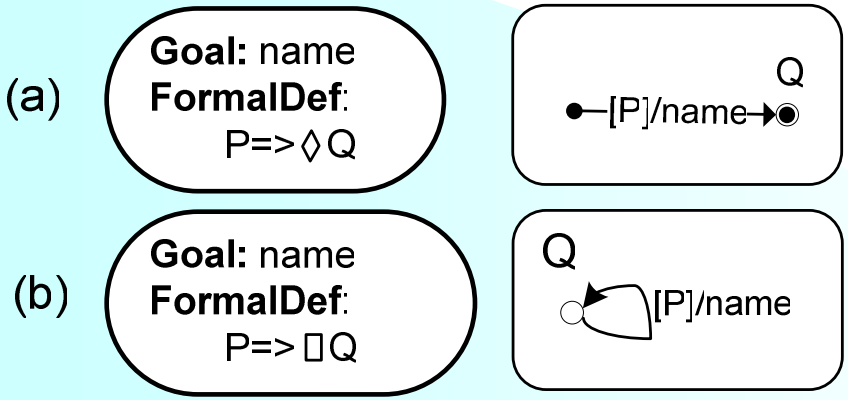




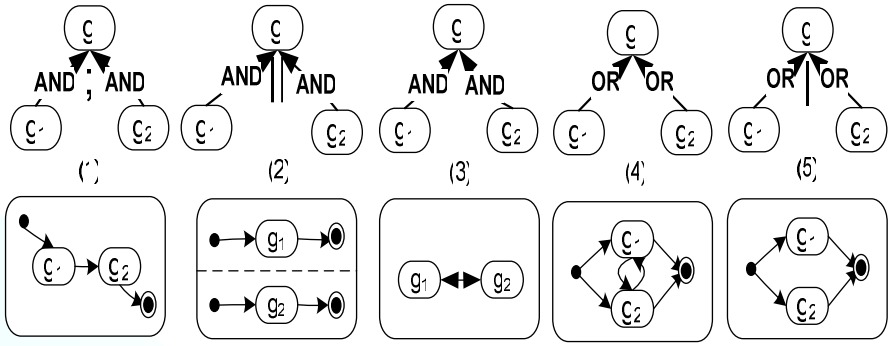
3.1c From goal to feature



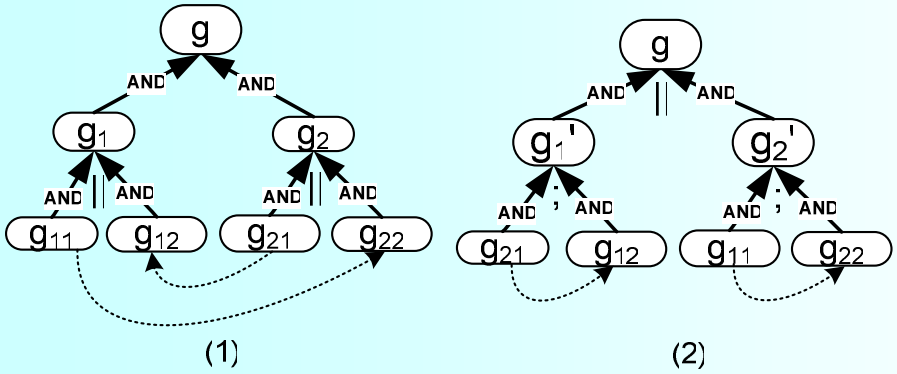
3.2c From goal to state



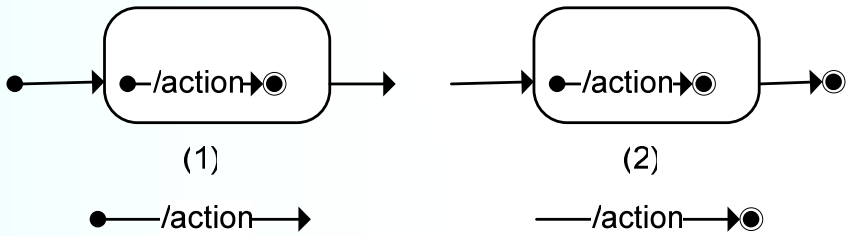
1. Defining states



3. Transforming hierarchies

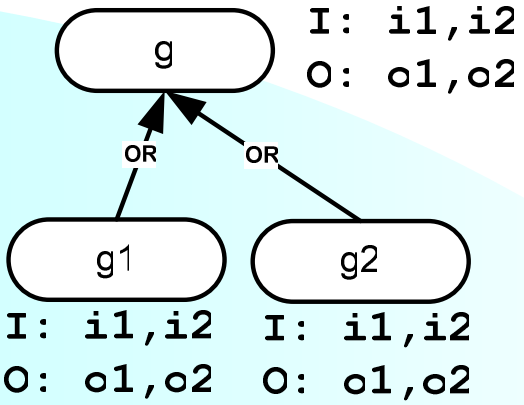


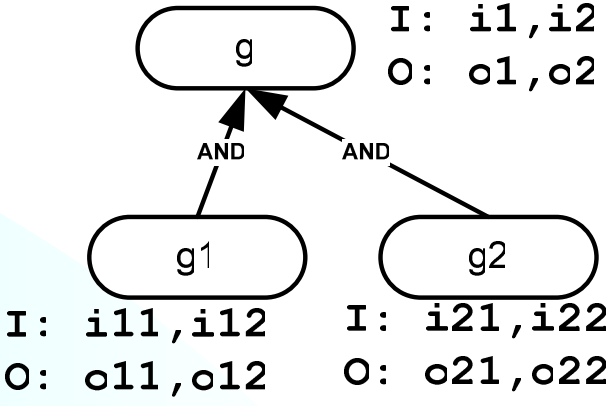
2. Treating dependencies

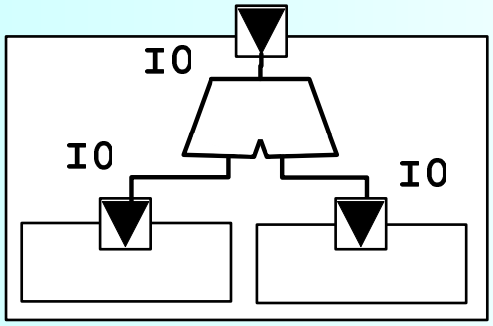


4. Simplifying leaf statecharts

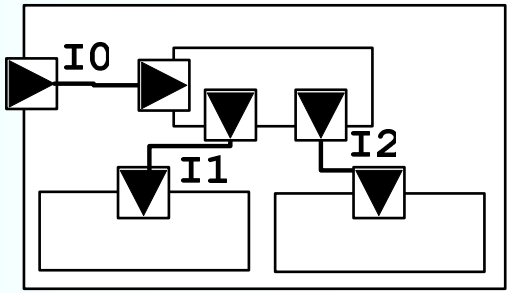
3.3c From goal to interfaces








```
interface type I0{
G(IN i1, IN i2
   OUT o1, OUT o2);
}
```




```
interface type I0{
 G(IN i1, IN i2, OUT o1, OUT o2);
}
Interface type I1 {
 G1(IN i11, IN i12, OUT o11, OUT o12);
}
Interface type I2 {
 G2(IN i21, IN i22, OUT o21, OUT o22);
}
```