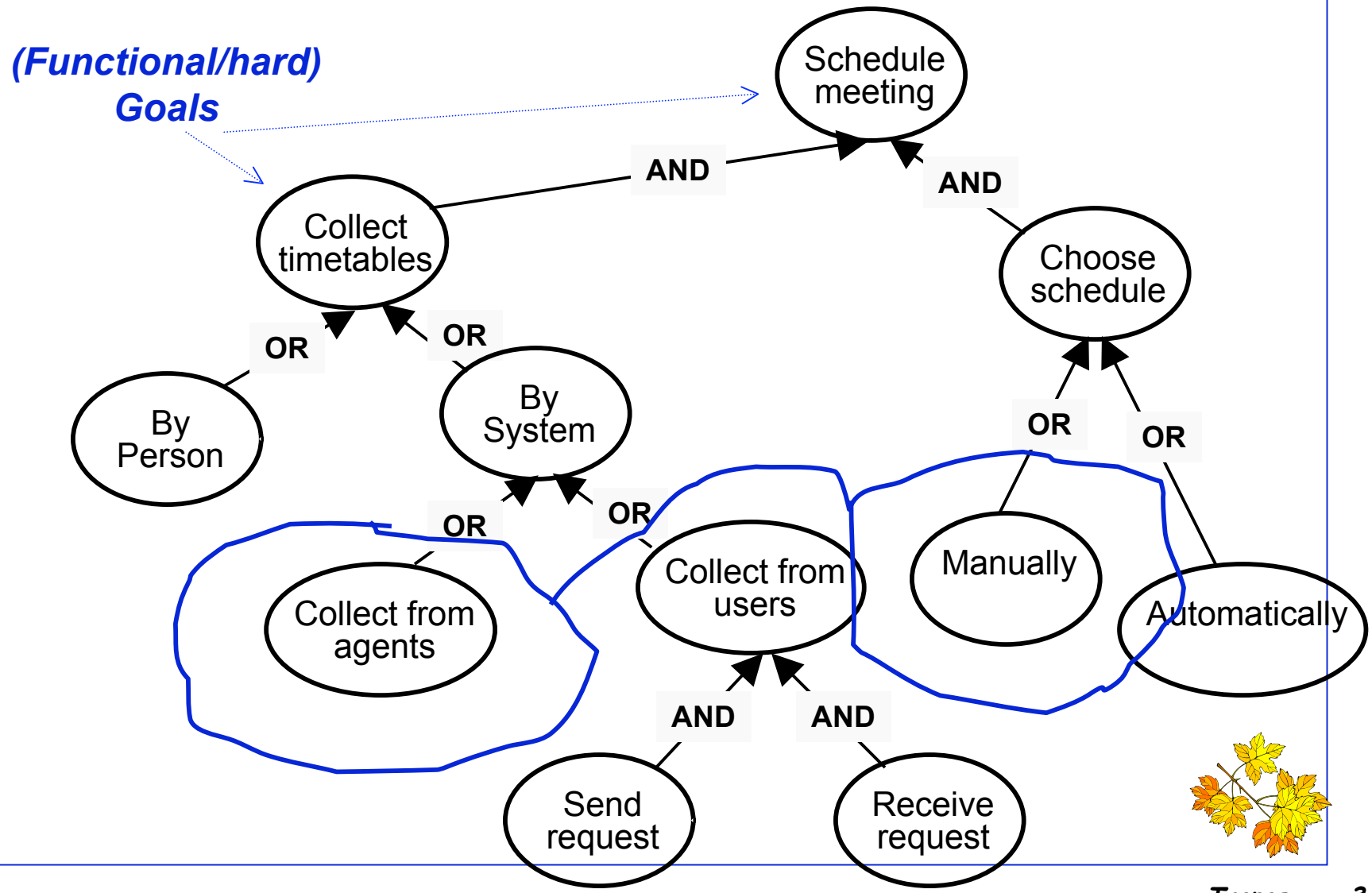
Agent-Oriented Software Engineering with Tropos

Modeling Stakeholders and their Goals -- i*
Agent-Oriented Software Engineering
Tropos Development Phases
Formal Analysis of Tropos Models
The Tropos Project

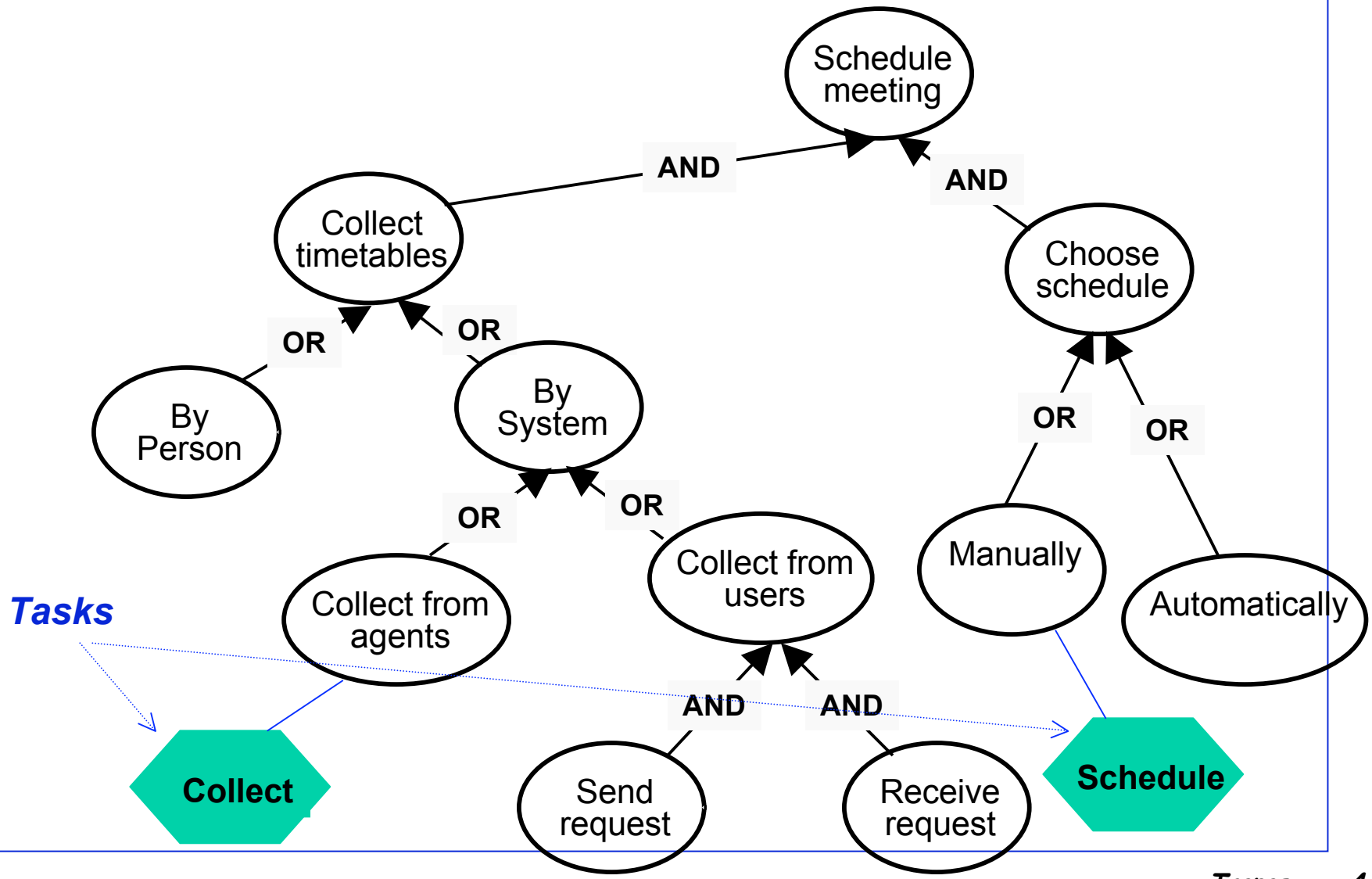
Goal-Oriented Requirements Engineering (~1993)

- Goal-oriented analysis focuses on early requirements, when problems are identified, and alternative solutions are explored and evaluated.
- During goal-oriented analysis, we start with initial stakeholder goals such as "Fulfill every book request", or "Schedule meeting" and keep refining them until we have reduced them to alternative collections of functional requirements each of which can satisfy the initial goals.
- Initial goals may be contradictory, so the analysis must facilitate the discovery of tradeoffs and the search of the full space of alternatives, rather than a subset.

Goal Analysis Leads to Alternatives

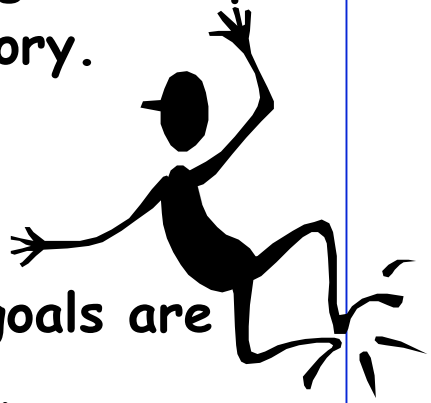


Alternatives Lead to Designs/Plans



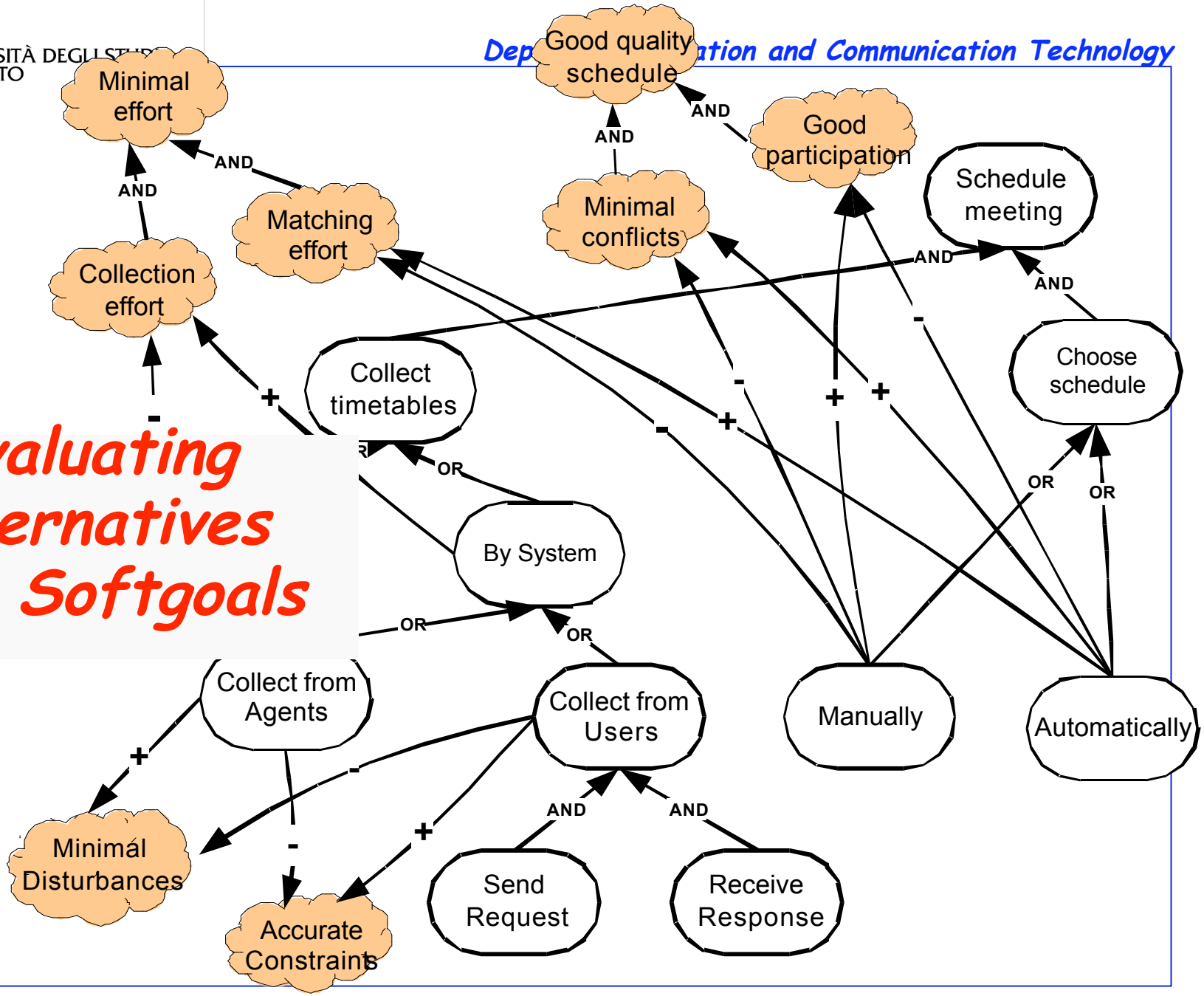
Softgoals

- Functional goals, such as "Schedule meeting" are well defined in the sense that they admit a formal definition.
- Non-functional goals, such "higher profits", "higher customer satisfaction" or "easily maintainable system" specify *qualities* a socio-technical system should adhere to.
- Such qualities usually admit no generally agreed upon definition, are inter-related and often contradictory.
- Such qualities are represented as *softgoals*.
- Softgoals can be thought as "fuzzy goals" with no clear-cut criteria for satisfaction; hence softgoals are *satisficed*, rather than satisfied (NFR framework, [Mylopoulos92], [Chung93]).



Softgoals for Representing Non-Functional Requirements

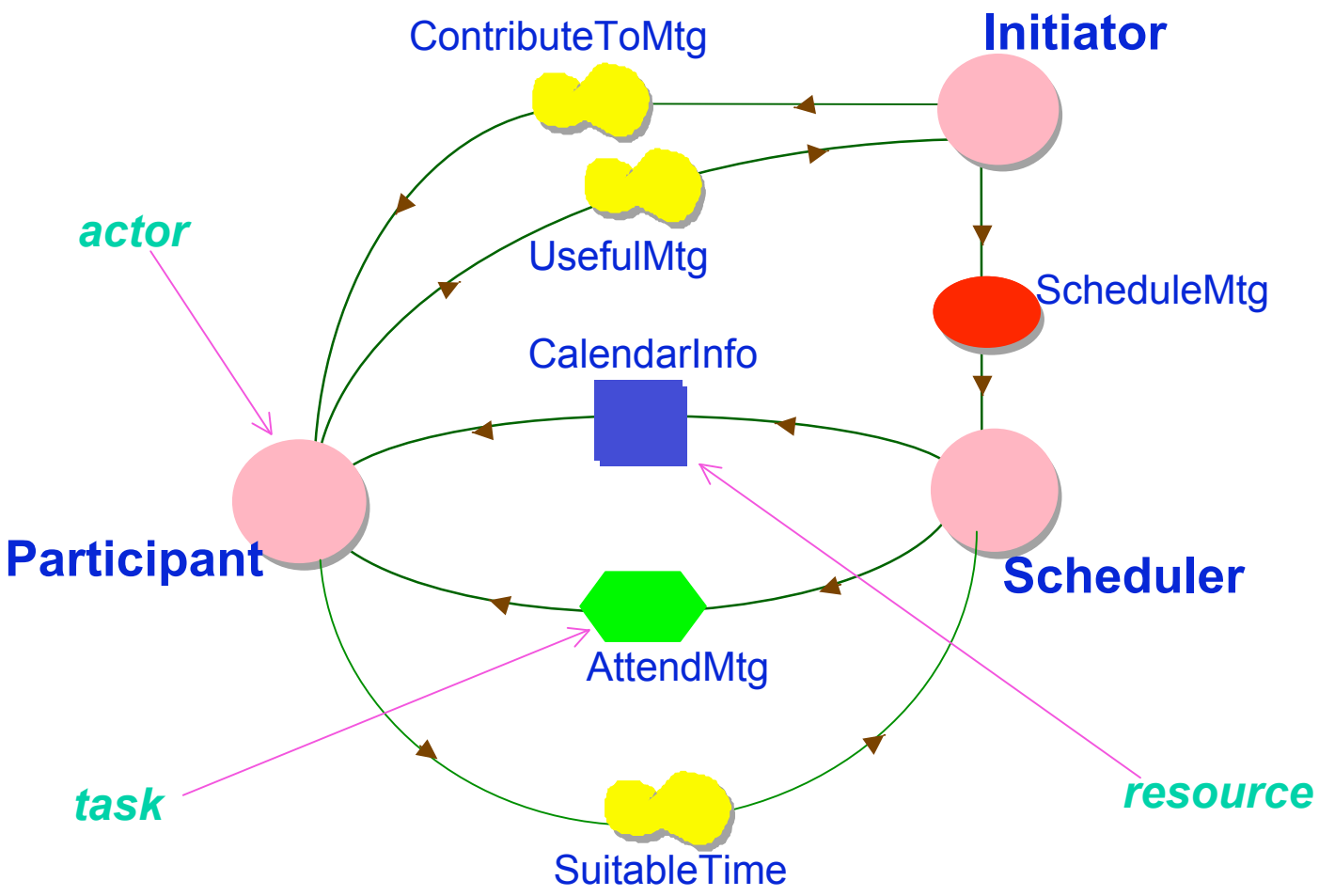
Evaluating Alternatives with Softgoals



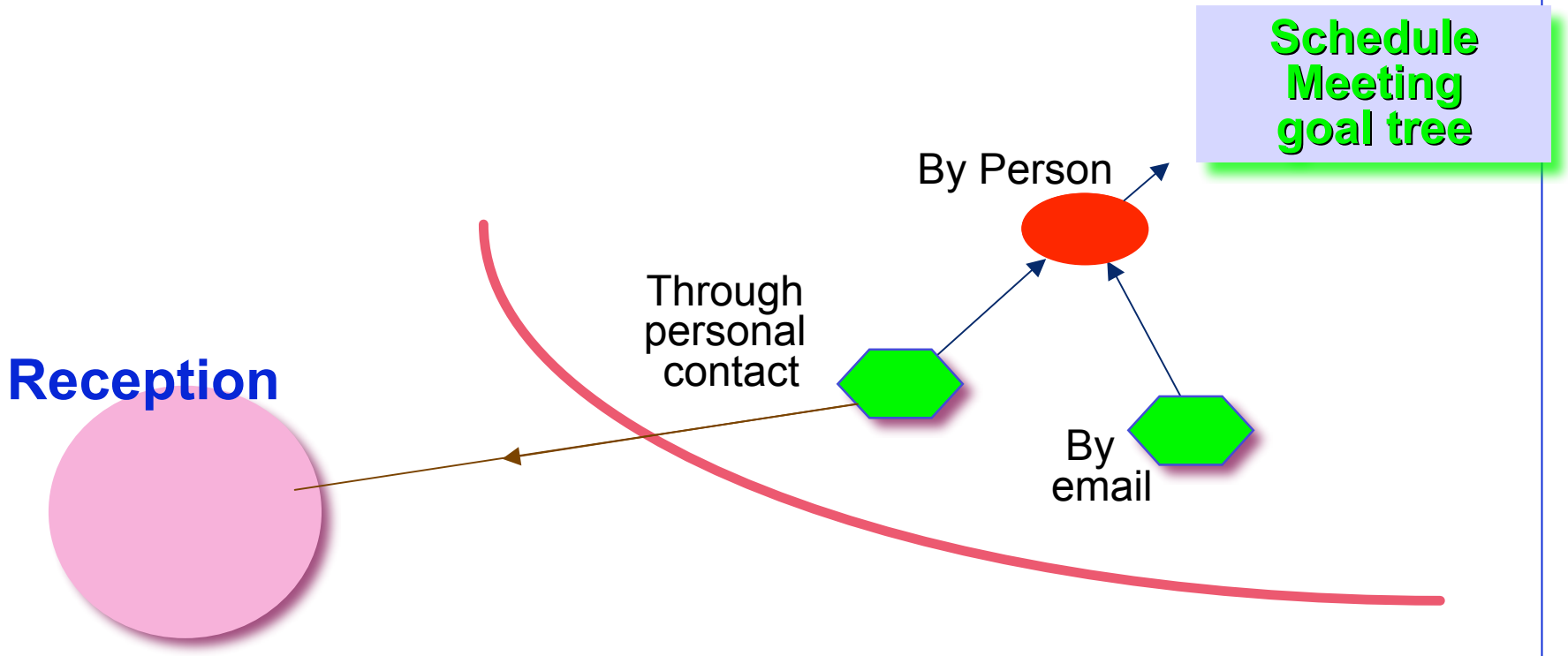
Stakeholders and Their Goals

- In KAOS, goals are global objectives for the system-to-be.
- In *i** [Yu93], goals are desired by *actors* and are *delegated* to other actors for fulfillment.
- In this framework then, early requirements involve identifying stakeholders and their goals, analyzing these goals, delegating them to other actors etc.
- The result of this process consists of *actor dependency* and *actor rationale* models.

An Actor Dependency Model

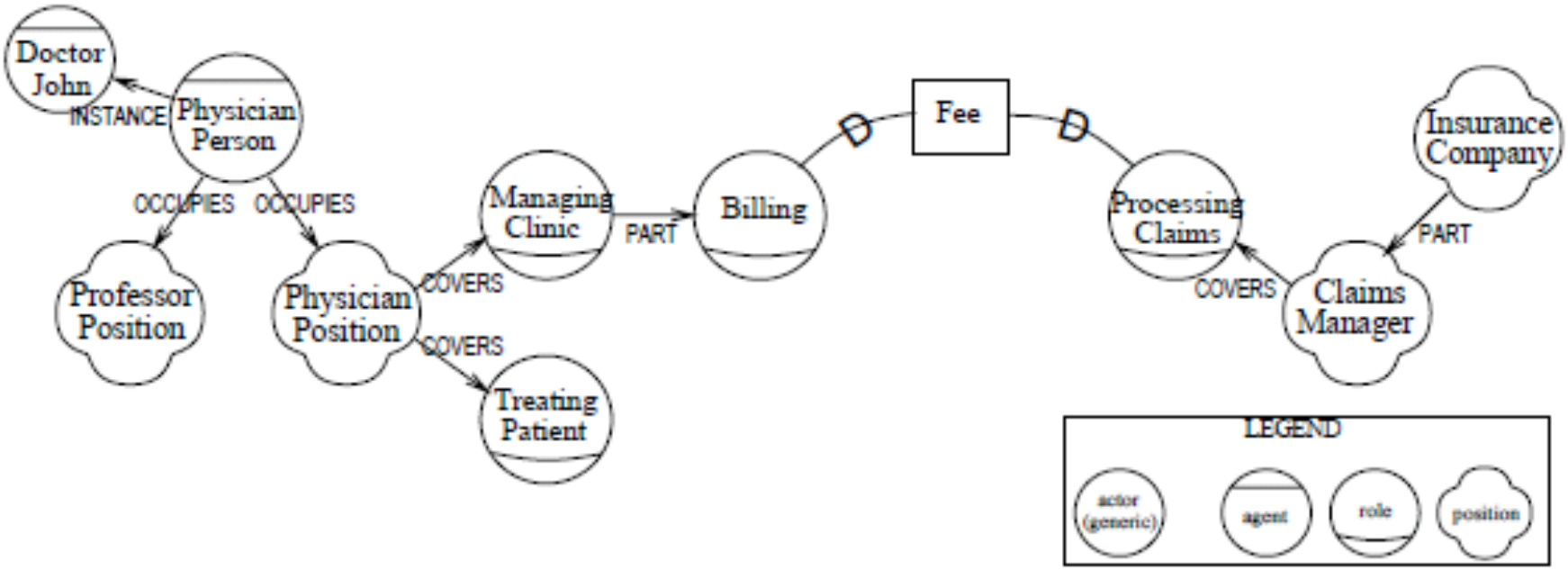


An Actor Rationale Model



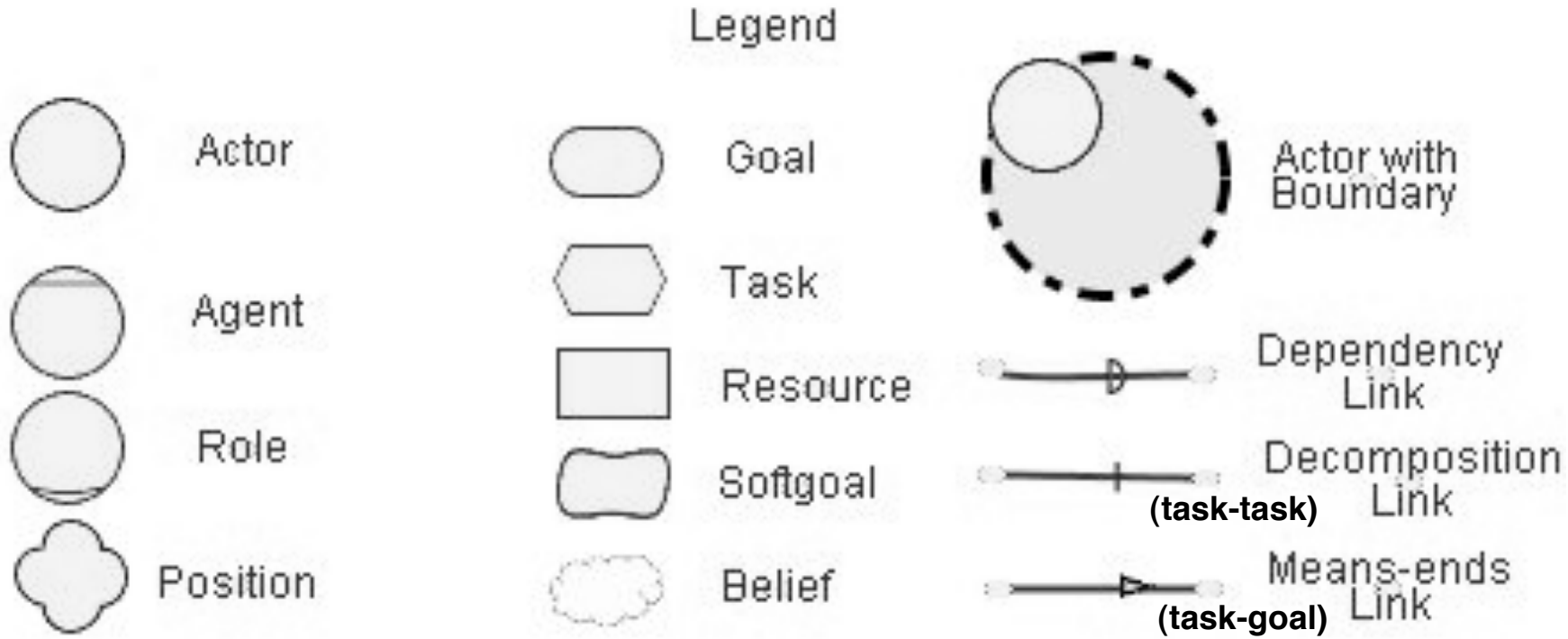
Actor dependencies are intentional: One actor *wants* something, another is *willing* and *able* to deliver.

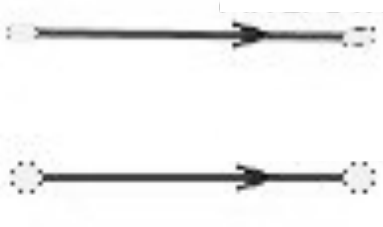
Agents, Roles, Positions



- Agents are concrete actors (people, organizations, systems)
- Roles are abstract actors, are played by agents
- Positions cover several roles, are occupied by agents

Modeling Primitives





AND/OR/++/--/+/-

isA/in/plays/occupies/covers/partOf

Contribution Link

Actor Association Link

Goals in Software Design

- KAOS, the NFR proposal, as well as *i** advocate the use of goals in designing software.
- KAOS uses goals to go from organizational objectives to functional requirements.
- NFR uses them to represent and analyze non-functional requirements. Non-functional requirements lead to criteria for evaluate functional alternatives (… AND functional requirements).
- *i** relates goals to the actors who want them and keeps track of delegations.

...An Idea...

- Software Engineering methodologies have traditionally come about in a "late-to-early" phase (or, "downstream-to-upstream") fashion.
- In particular, Structured Programming preceded (and influenced!) Structured Analysis and Design; likewise, Object-Oriented Programming preceded Object-Oriented Design and Analysis.
- In both cases, programming concepts were projected upstream to dictate how designs and requirements are to be conceived.

What would happen if we projected requirements concepts downstream to define software designs and even implementations?

What is Software?

- An engineering artifact, designed, tested and deployed using engineering methods; rely heavily on testing and inspection for validation (*Engineering perspective*)
- A mathematical abstraction, a theory that can be analyzed for consistency and can be refined into a more specialized theory (*Mathematical perspective*)

...but more recently...

- A non-human agent, with its own personality and behavior, defined by its past history and structural makeup (*CogSci perspective*)
- A social structure of software agents, who communicate, negotiate, collaborate and cooperate to fulfil their goals (*Social perspective*)

These two perspectives will grow in importance -- in practice, but also SE research!

Why Agent-Oriented Software?

- Next generation software will consist of open, dynamic architectures where components can accomplish tasks in a variety of operating environments.
- Consider application areas such as eBusiness, web services, pervasive and/or P2P computing.
- These call for components that find and compose services dynamically, establish/drop partnerships with others and operate under a broad range of conditions.
- Learning, planning, communicating, negotiating, and exception handling become essential features for such software components…

☛ ... agents!

Agent-Oriented Software Engineering

- Many researchers working on it.
- Research on the topic generally comes in two flavours:
  - ✓ Extend UML to support agent communication, negotiation etc. (e.g., [Bauer99, Odell00]);
  - ✓ Extend current agent programming platforms (e.g., JACK) to support not just programming but also design activities [Jennings00].
- Tropos is requirements-oriented, adopts concepts from early RE and applies them to other software development phases.

What is an Agent?

- A person, an organization, certain kinds of software.
- An *agent* has *beliefs*, goals (*desires*), *intentions*.
- Agents are situated, autonomous, flexible, and social.
- But note: human/organizational agents can't be *prescribed*, they can only be *partially described.*
- Software agents, on the other hand, have to be completely specified during implementation.
- Beliefs correspond to (object) state, intentions constitute a run-time concept. For design-time, the interesting new concept agents have that objects don't have is...

☛ ***...goals!***

The Tropos Methodology

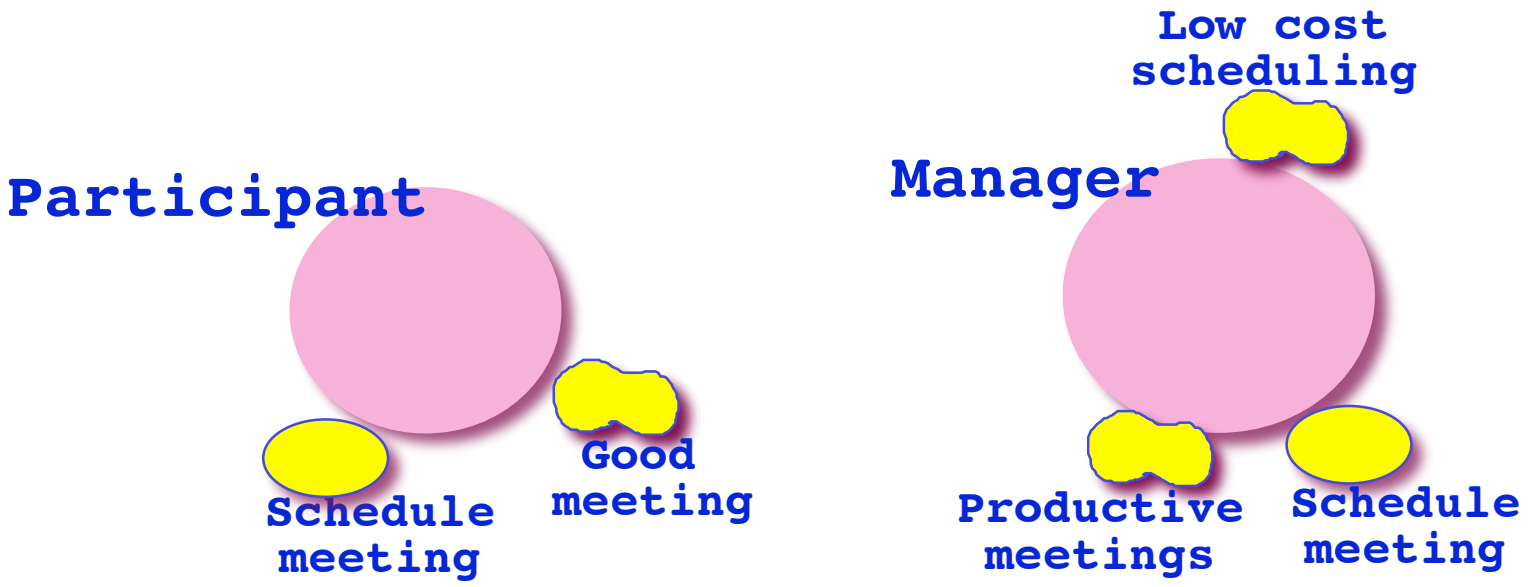
- We propose a set of primitive concepts and a methodology for agent-oriented requirements analysis and design.
- We want to cover four phases of software development:
  - ✓ *Early requirements* -- identifies stakeholders and their goals;
  - ✓ *Late requirements* -- introduce system as another actor which can accommodate some of these goals;
  - ✓ *Architectural design* -- more system actors are added and are assigned responsibilities;
  - ✓ *Detailed design* -- completes the specification of system actors.

Early Requirements?

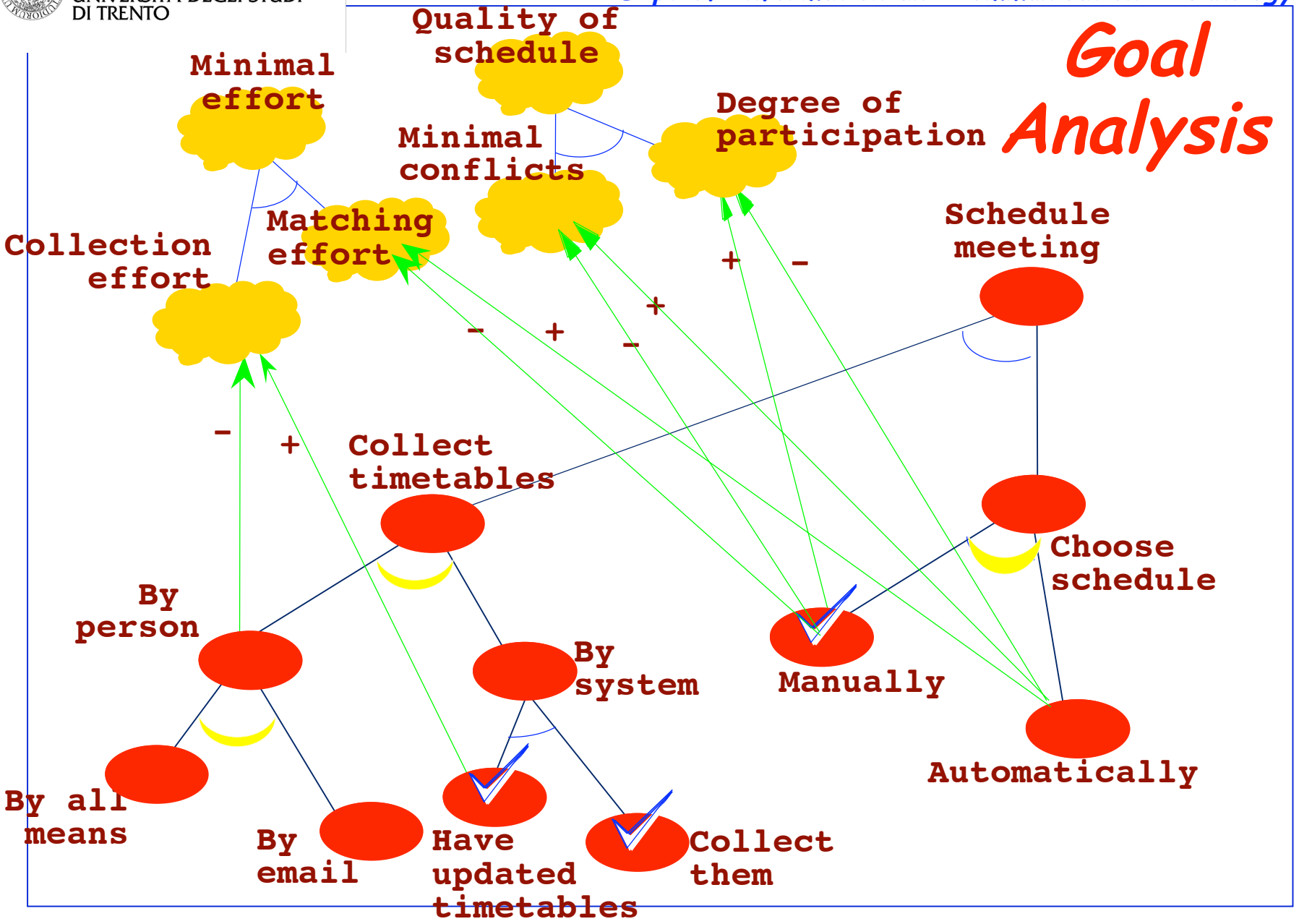
- The organizational environment of a software system can be conceptualized as a set of business processes, actors and/or strategic/tactical goals.
- The KAOS project defines the state-of-the-art on modeling early requirements in terms of goals; also offers well-developed analysis techniques and tools for generating late requirements.
- We focus on *actors* and *goals*. In particular, we adopt the *i** framework of Eric Yu [Yu95].
- ActorClass = AgentClass ∪ PositionClass ∪ RoleClass.

Early Requirements: Stakeholders and their Goals

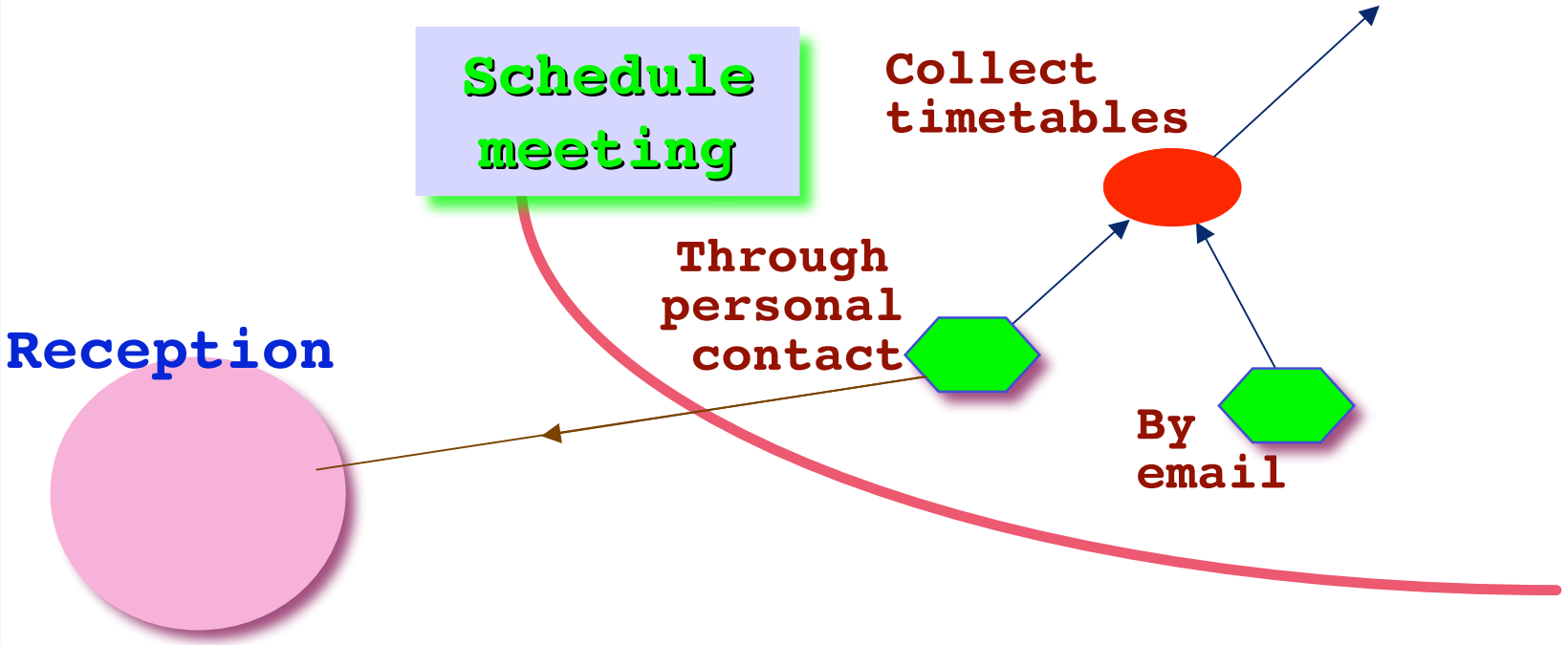
A social setting consists of actors, each having *goals* (and/or *softgoals*) to be fulfilled.



Goal Analysis

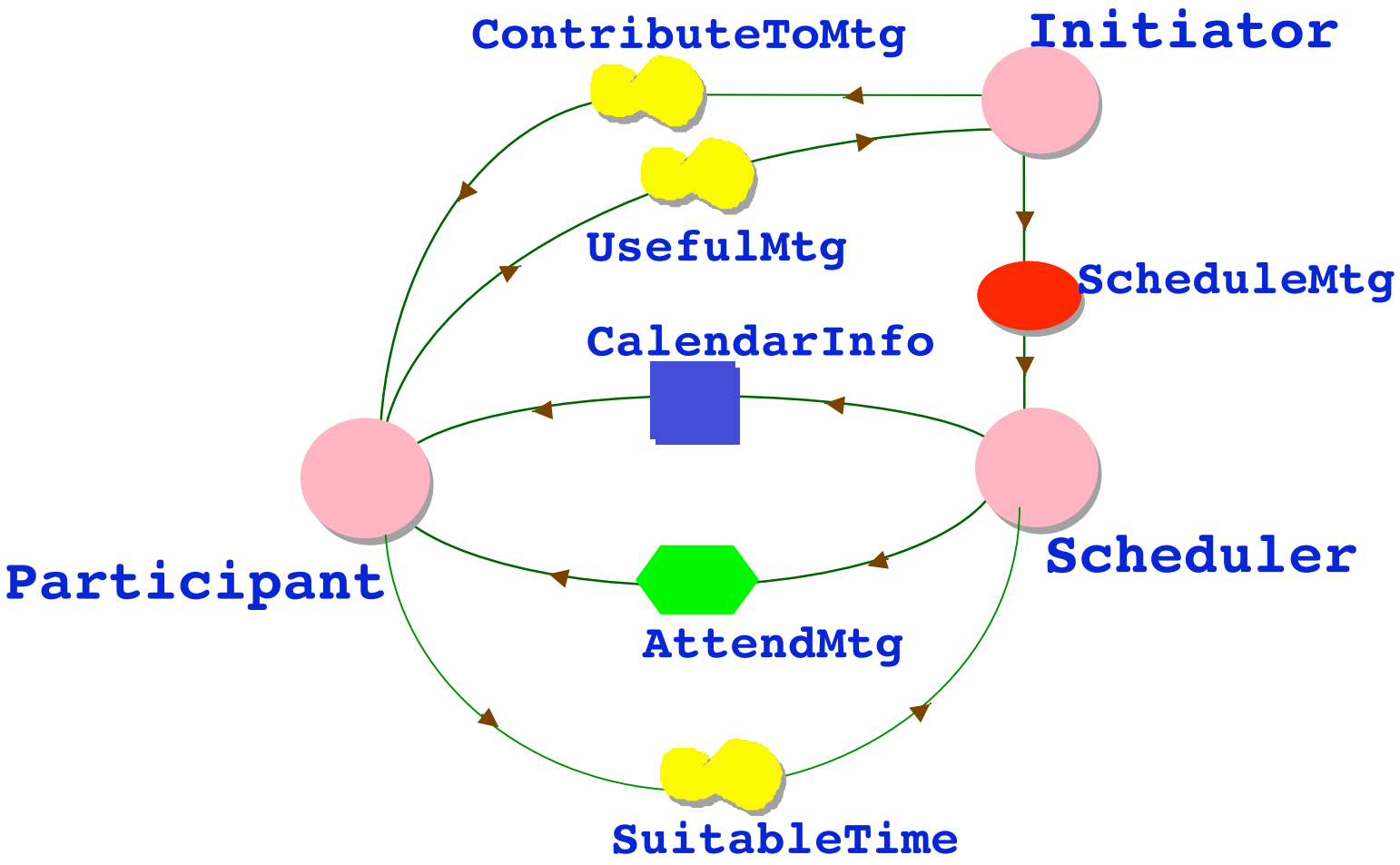


Actor Dependencies



Actor dependencies are intentional: One actor *wants* something, another is *willing* and *able* to deliver.

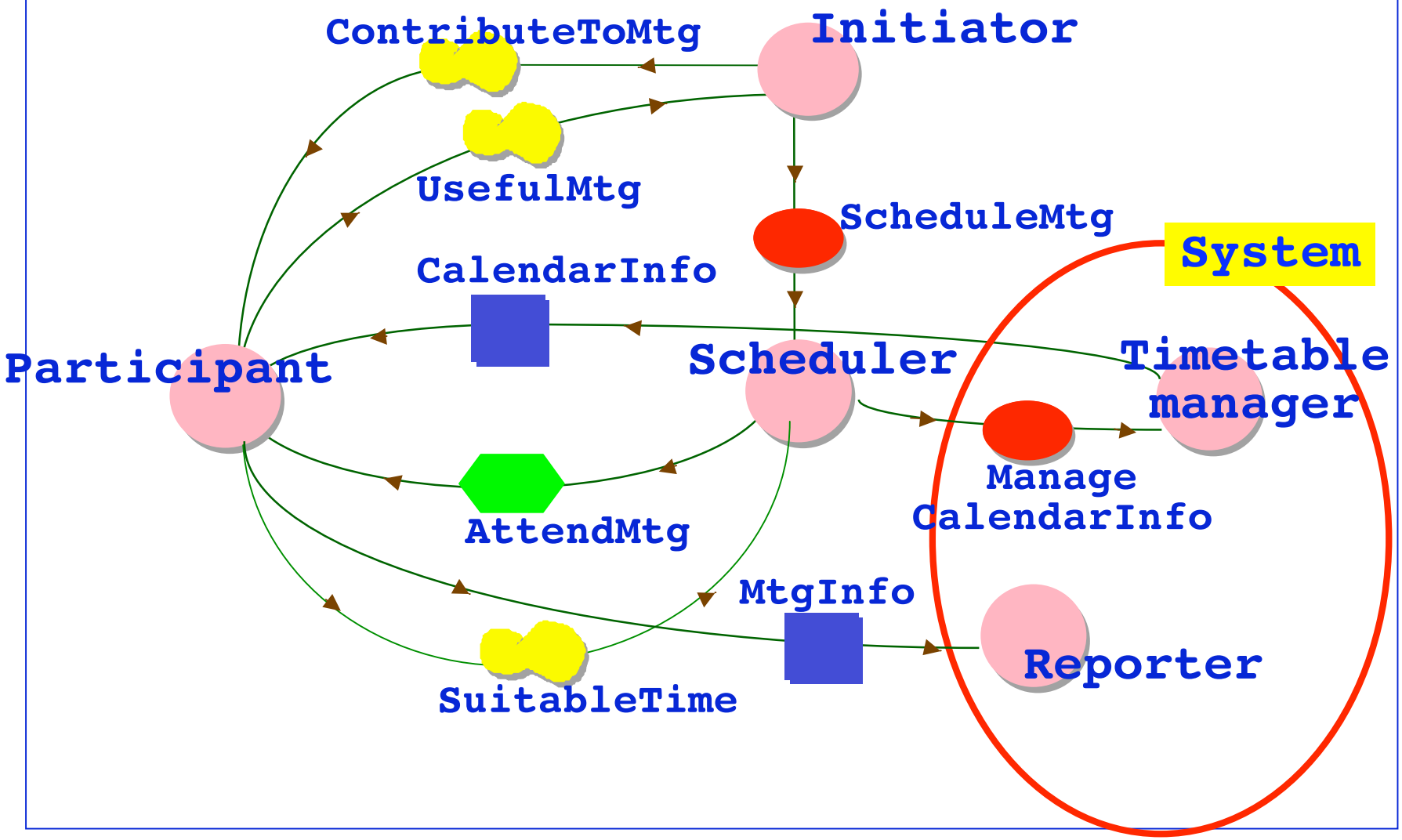
Actor Dependency Models



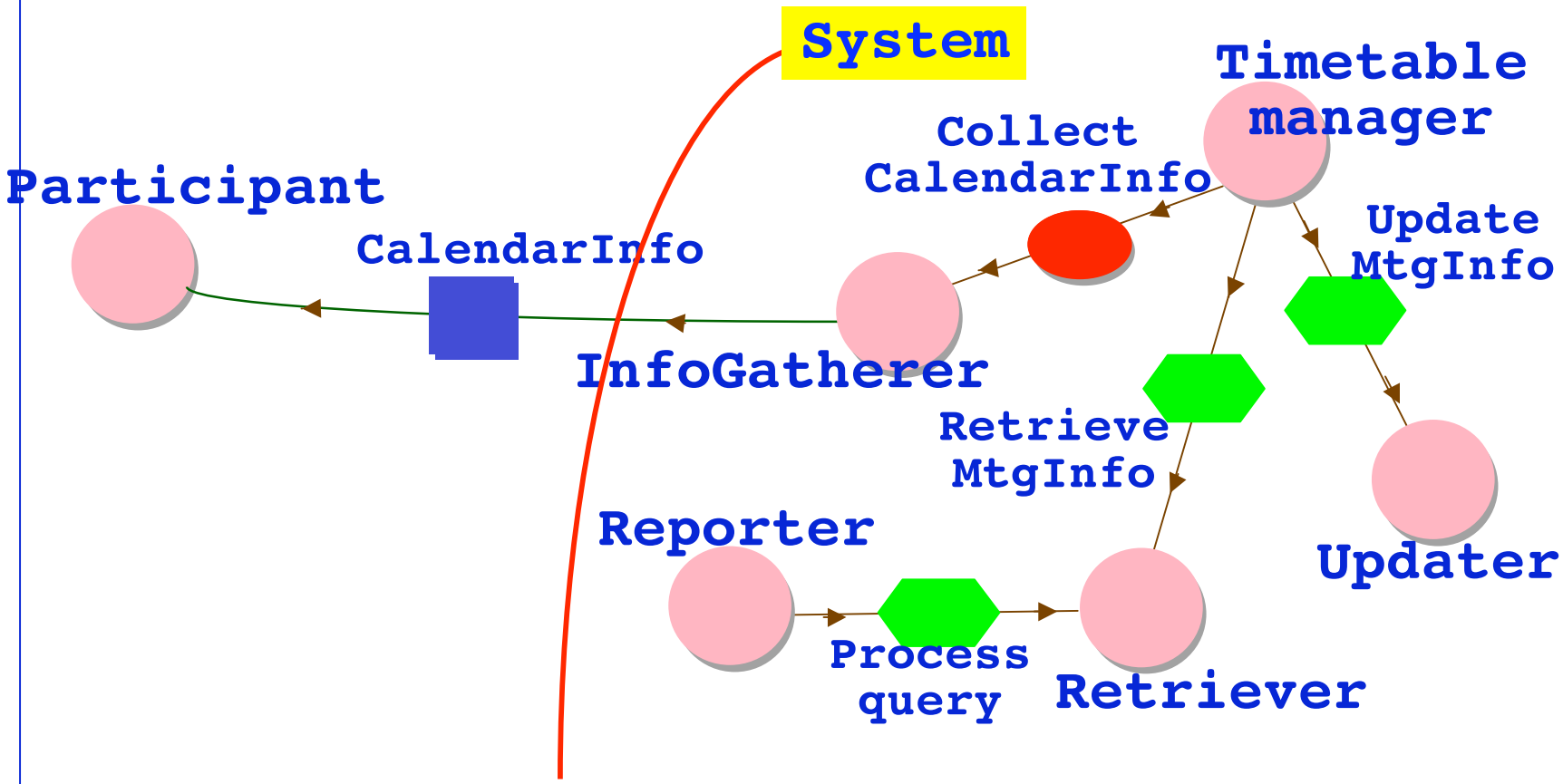
Using These Concepts

- During early requirements, these concepts are used to model external stakeholders (people, organizations, existing systems), their relevant goals and inter-dependencies.
- During late requirements, the system-to-be enters the picture as one or a few actors participating in *i** models.
- During architectural design, the actors being modelled are all system actors.
- During detailed design, we are not adding more actors and/or dependencies; instead, we focus on fully specifying all elements of the models we have developed.

Late Requirements with i*



Software Architectures with i*



...Yet Another Software Development Process

- Initialization: Identify stakeholder actors and their goals;
- Step: For each new goal:
  - ✓ adopt it;
  - ✓ delegate it to an existing actor;
  - ✓ delegate it to a new actor;
  - ✓ decompose it into new subgoals;
  - ✓ declare the goal "denied".
- Termination condition: All initial goals have been fulfilled, assuming all actors deliver on their commitments.

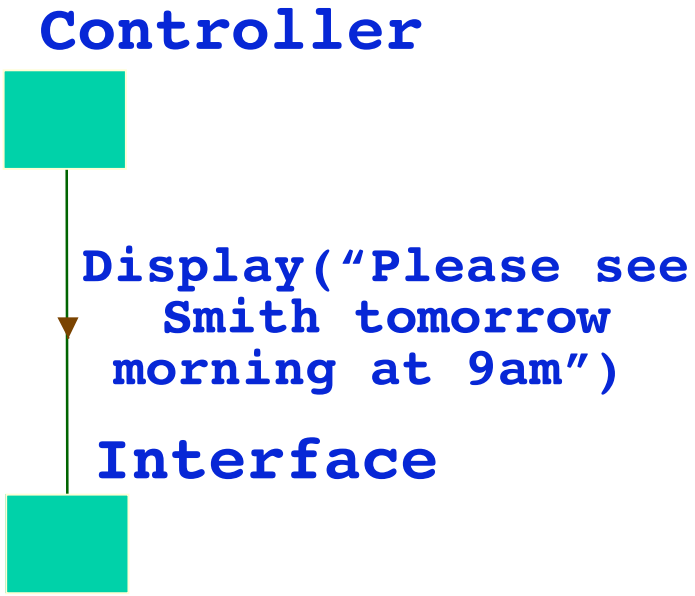
What is Different?

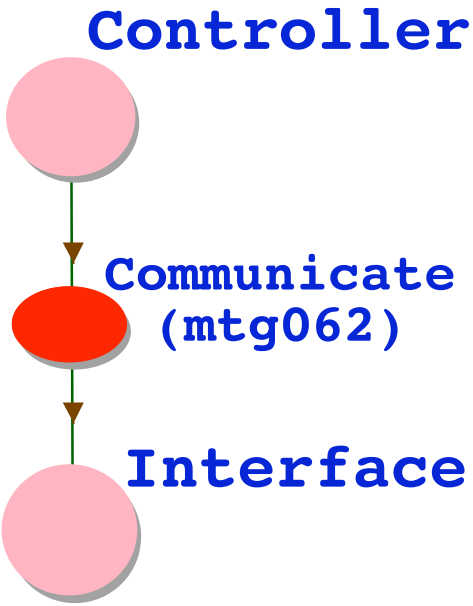
- Goal refinement extends functional decomposition techniques, in the sense that it explores alternatives.
- Actor dependency graphs extend object interaction diagrams in that dependencies are *intentional*, need to be monitored, may be discarded, and can be established at design- or run-time.
- In general, an actor architecture is open and dynamic; evolves through negotiation, matchmaking and like-minded mechanisms.
- The distinction between design and run-time is blurred.
- So is the boundary between a system and its environment (software or otherwise.)

Why is this Better (…Sometimes…)

- Traditionally, goals (and softgoals) are operationalized and/or metricized before late requirements.
- This means that a solution to a goal is frozen into a software design early on and the designer has to work within the confines of that solution.
- This won't do in situations where the operational environment of a system, including its stakeholders, keeps changing.
- This won't do either for software that needs to accommodate a broad range of users, with different cultural, educational and linguistic backgrounds, or users with special needs.

The Tale of Two Designs



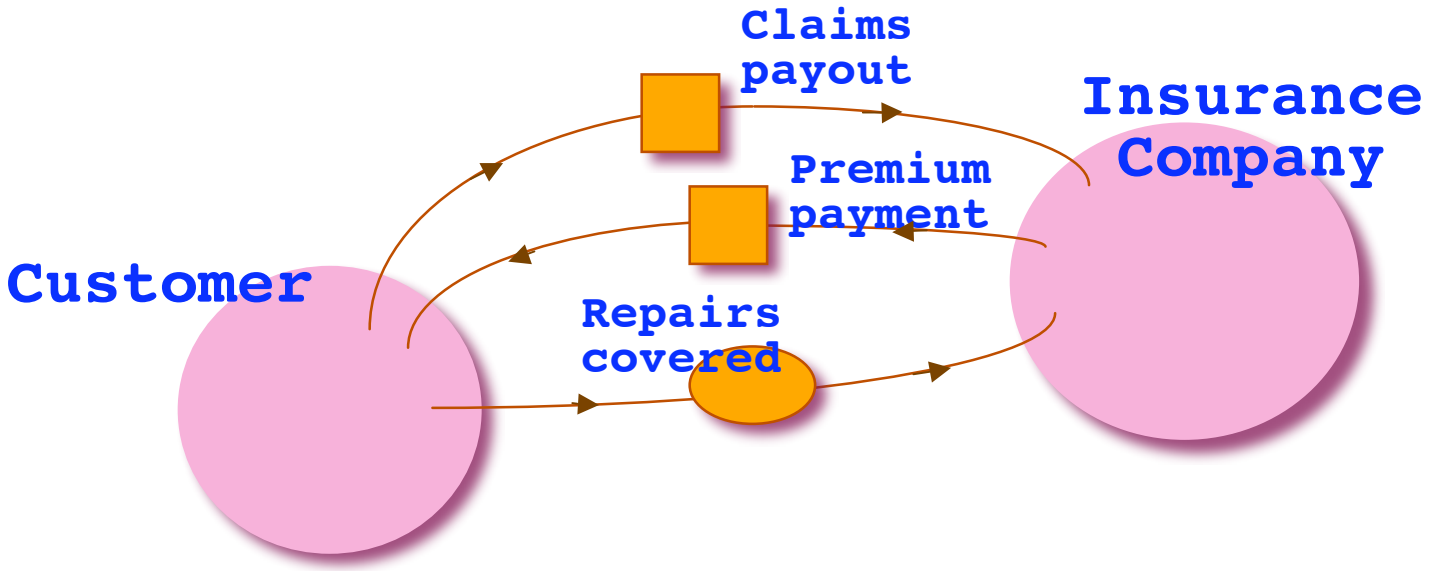


Analysing Models

- Models in Software Engineering are used primarily for human communication.
- Building these models is not enough! Large models can be hard to understand, or take seriously!
- We need analysis techniques which offer evidence that a model makes sense:
  - ✓ *Simulation* through model checking, to explore the properties of goals, entities, etc. over their lifetime;
  - ✓ *Goal analysis* which determine the fulfillment of a goal, given information about related goals;
  - ✓ *Social analysis* which looks at viability, workability,… for a configuration of social dependencies.

Formal Tropos

- Each concept in a Tropos diagram can be defined formally, in terms of a temporal logic inspired by KAOS.
- Actors, goals, actions, entities, relationships are described statically and dynamically.



A Formal Tropos Example

```
Entity Claim
    Has claimId: Number, insP: InsPolicy,
      claimDate, date: Date, details: Text
    Necessary date before insP.expDate
    Necessary (∀x)(Claim(x) ∧ ●¬Claim(x) ⇒
      ¬RunsOK(x.insP.car))
end Claim

Action MakeRepair
    Performed by BodyShop
    Refines RepairCar
    Input cl : Claim
    Pre ¬RunsOK(cl.insP.car)
    Post RunsOK(cl.insP.car)...
```

A Goal Dependency Example

```
GoalDependency CoverRepairs
   Mode       Fulfil
   Depender Customer
   Dependee InsuranceCo
   Has cl: Claim
   Defined   /*   the   amount   paid   out   by the
   insurance company covers repair costs */
end CoverRepairs
```

Model Checking for Tropos

- Goal: Apply model checking to richer models than those that have been tried before.
- Approach
  - ✓ Definition of an automatic translation from Formal Tropos specifications to the input language of the nuSMV model checker [Cimatti99].
  - ✓ Verification of temporal and logical properties of state representations of finite Tropos models.
  - ✓ Discovery of interesting scenarios that represent counterexamples to properties not satisfied by the specifications.
  - ✓ Model simulation using the *T-tool*.

Mapping Tropos to nuSMV

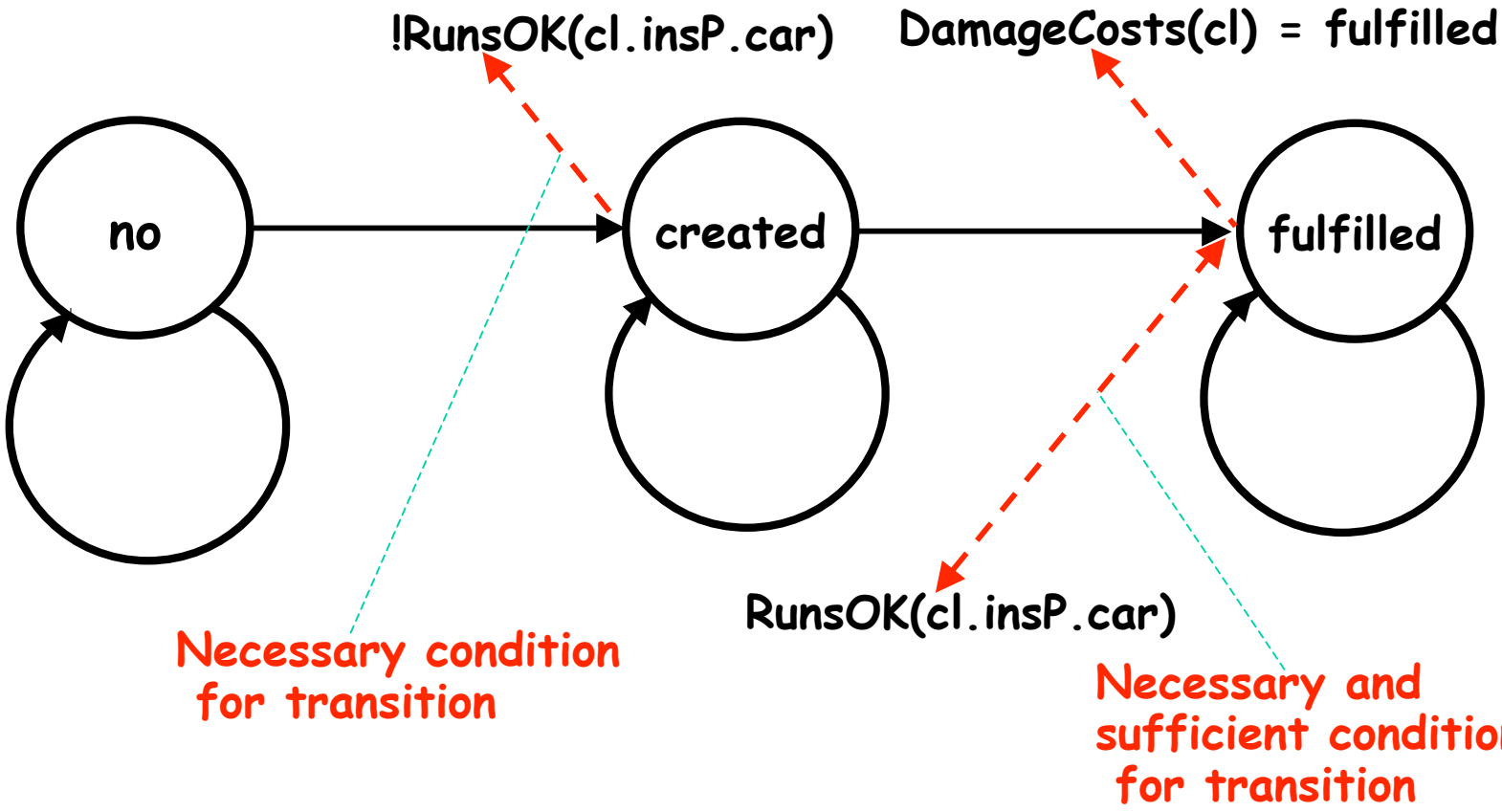
- The language supported by a model checker includes variables that can take one of a finite number of values. Also, constraints on the allowable transitions from one value to another.
- How do we map Formal Tropos to nuSMV?
  - ✓ Each goal instance is represented by a variable that can take values "no", "created", "fulfilled"; these represent the possible states of a goal instance.
  - ✓ Each action is represented by a Boolean variable that is true only at the time instance when the action occurs.

Translation for CoverRepairs

```
VAR  CoverRepairs : {no, created, fulfilled}
INIT  CoverRepairs = no
TRANS   CoverRepairs = no -> (next(CoverRepairs) = no |
   next(CoverRepairs) = created)
TRANS   CoverRepairs = created -> (next(CoverRepairs) =
   created | next(CoverRepairs) = fulfilled)
TRANS   CoverRepairs = fulfilled -> next(CoverRepairs) = fulfilled
TRANS    CoverRepairs = no -> next(CoverRepairs = created ->
   !RunOK)
TRANS   CoverRepairs = created -> next(CoverRepairs = fulfilled
   -> DamageCosts = fulfilled)
TRANS   CoverRepairs = created -> next(CoverRepairs = fulfilled
   <-> RunsOK)
```

From nuSMV Specs to FSMs

- Finite State Machine for CoverRepairs(cl)



Model Checking

- A model consists of a finite set of FSMs, each representing an instance of a class in the Tropos model (goal, dependency, entity, …), or a propositional variable (e.g., RunsOK(cl.insP.car)).
- A simulation considers all possible simulations of these FSMs, taking into account inter-FSM constraints.
- Even though the space of possible simulations is infinite, only a finite (but usually large!) number of these matters.

An Interesting Property

LTLSPEC F[CoverRepairs(cl) = fulfilled -> MakeRepair(cl.insP.car)]

"If/when sometime in the future CoverRepairs(cl) is fulfilled, then (at that time) MakeRepairs(cl.insP.car) is true"

This property does not hold for the model. A counterexample is:

Variable	t_1	t_2	t_3	t_4
RunsOK	false	false	true	true
DamageCosts	no	no	created	fulfilled
CoverRepairs	no	created	created	fulfilled
MakeRepair	false	false	false	false

A Fix

Add to the definition of the entity class Car

…

Necessary

$\neg RunsOK(self) \wedge \neg MakeRepair(self) \Rightarrow \bullet \neg RunsOK(self)$

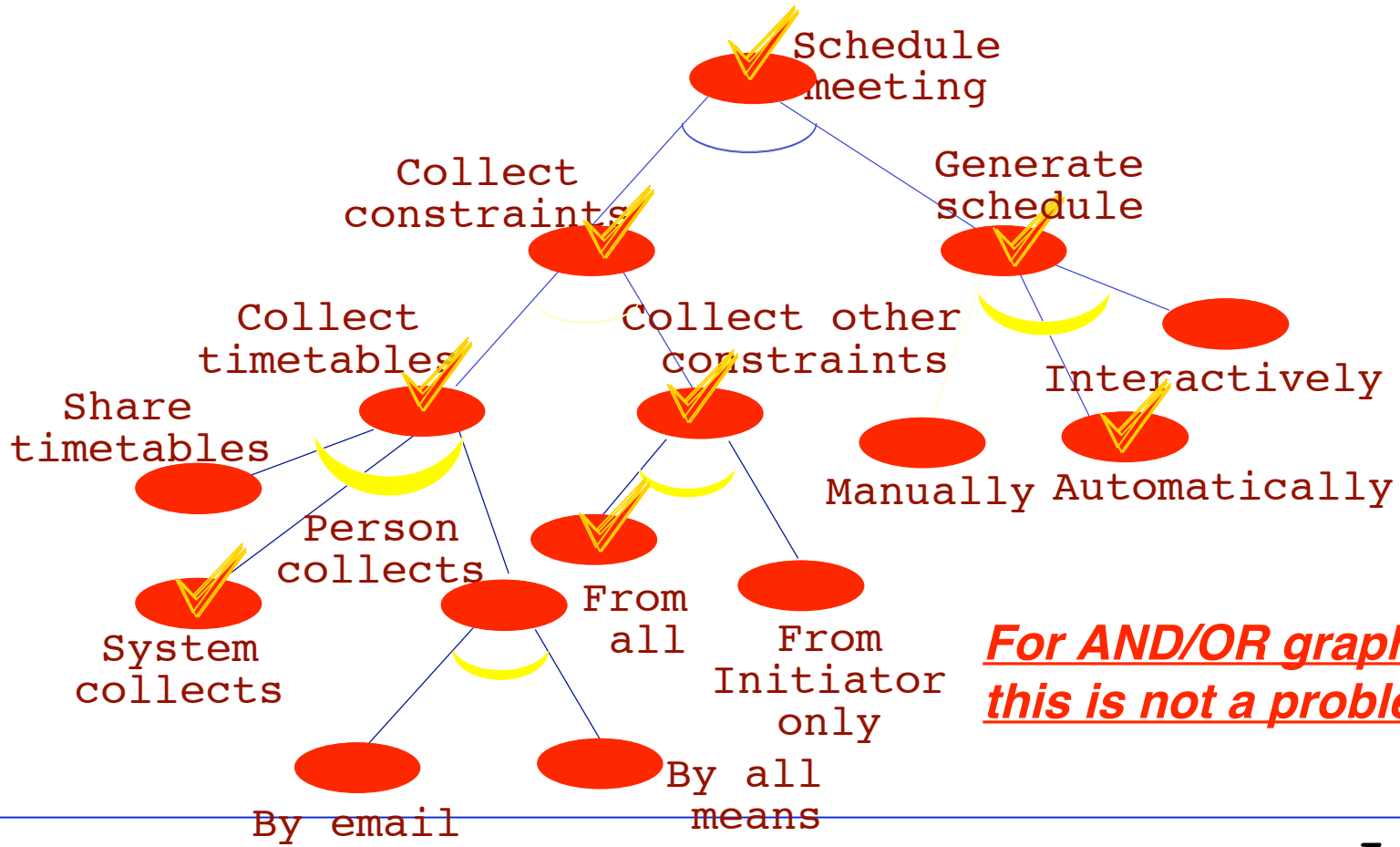
…

Experiments with the T-Tool

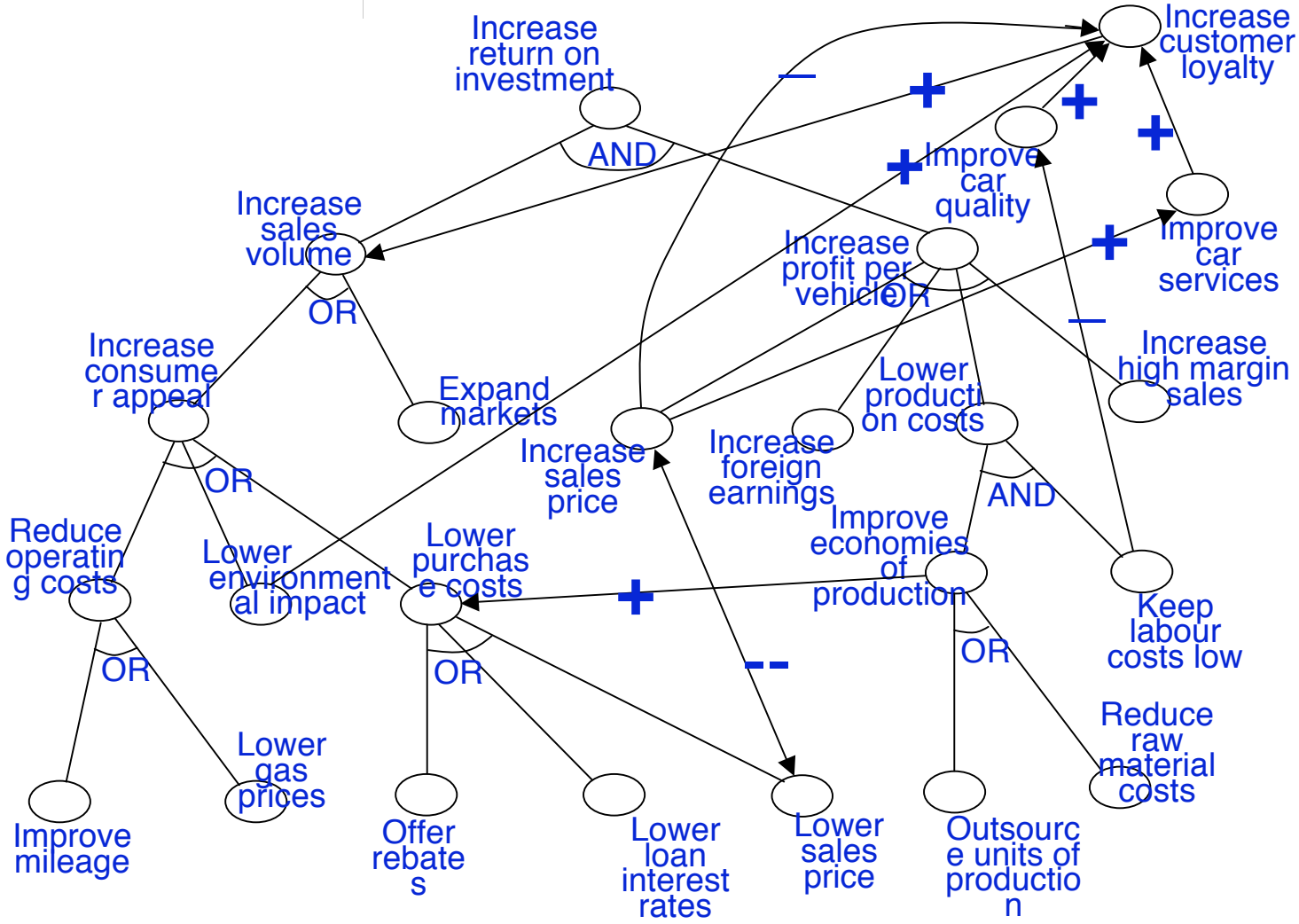
- Tropos models are infinite; to make model checking work, we pick finite submodels (e.g., 1, 2,... instances per class, for any one property we want to prove and see if it leads to counter-examples.
- How do we pick submodels? How do we know when to stop?
- Experiments to demonstrate the scalability of the approach.

Goal Analysis

- Given a goal graph with some goals satisfied/denied, we want to draw inferences about other goals of the graph.



For AND/OR graphs, this is not a problem



How do we evaluate this graph?

Analyzing Softgoal Graphs

- Given a goal graph structure, the following label propagation procedure determines the status of each goal node. A goal is labelled
  - ✓ satisficed (S) - if it is satisficeable and not deniable
  - ✓ denied (D) - if deniable and not satisficeable
  - ✓ conflicting (C) - if both deniable and satisficeable
  - ✓ undetermined (U) - if neither
- Labelling procedure iterates over two basic steps:

  I. For each goal, compute the label of each satisficed outgoing link; these labels can be one of four mentioned earlier, plus U-, U+ and ?

  II. The labels accumulated for a single proposition are combined into one of the four labels (S, D, C, U)

How Are Labels Propagated?

$label_{source}$	link type				
	sub	sup	-sub	-sup	und
S	U+	S	U-	U	U
D	D	U-	S	U+	U
C	?	?	?	?	U
U	U	U	U	U	U

Assuming AND(G0, {G1,...Gn}), label(G0) = min(label(Gi))

Assuming OR(G0, {G1,...Gn}), label(G0) = max(label(Gi))

where S≥ U, C ≥ D.

Combination of labels assigned to a single proposition is done on the basis of label(P) = min(labels(P)), where U ≥ S, D ≥ C

Critique

- It is not clear that this algorithm terminates; it is possible that the label of a node will keep changing after each step of the label propagation algorithm.
- The label combination rules seem ad hoc and with no justification (other than a feeble "they seem intuitive").
- It is unclear what goal relationships such as '+', '-' really mean.
- Can we come up with a goal analysis algorithm which (a) terminates, (b) is computationally tractable, and (c) is semantically well-founded?

A Qualitative Goal Model

- We use S(atisfied), D(enied) and don't assume that they are logically exclusive (remember, goals may be contradictory!)
- We offer several axioms for every goal relationship.

$\forall g1,g2,g3[AND(\{g1,g2\},g3) \Rightarrow ((S(g1) \wedge S(g2)) \Rightarrow S(g3))]$

$\forall g1,g2,g3[OR(\{g1,g2\},g3) \Rightarrow ((S(g1) \vee S(g2)) \Rightarrow S(g3))]$

$\forall g1,g2[++(g1,g2) \Rightarrow (S(g1) \Rightarrow S(g2))]$

$\forall g1,g2[+(g1,g2) \Rightarrow \exists g[(g \neq g2 \wedge S(g) \wedge S(g1)) \Rightarrow S(g2)]]$

$\forall g1,g2,g3[AND(\{g1,g2\},g3) \Rightarrow ((D(g1) \vee D(g2)) \Rightarrow D(g3))]$

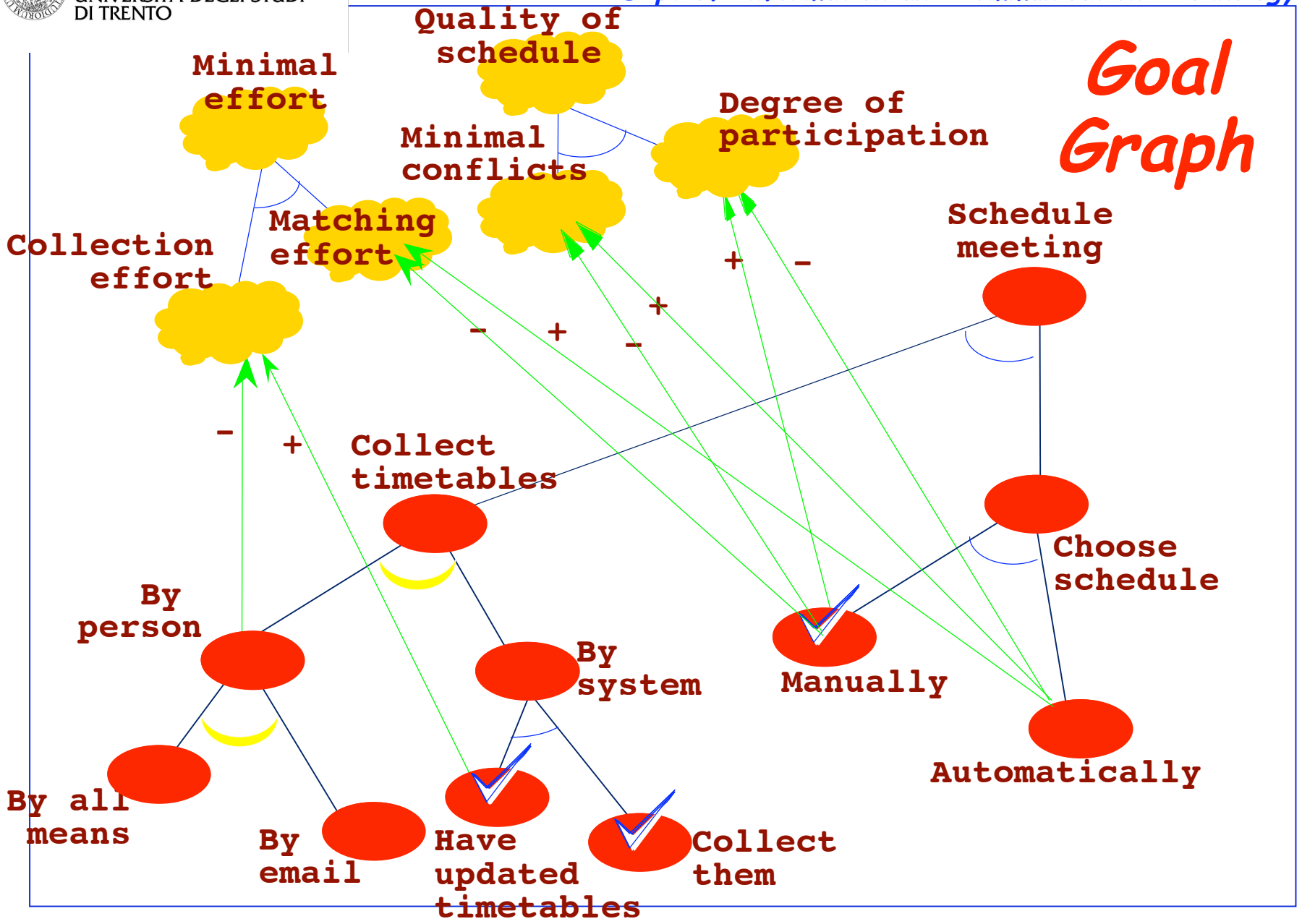
$\forall g1,g2,g3[OR(\{g1,g2\},g3) \Rightarrow ((D(g1) \wedge D(g2)) \Rightarrow D(g3))]$

$\forall g1,g2[++(g1,g2) \Rightarrow (D(g1) \Rightarrow D(g2))]$

$\forall g1,g2[+(g1,g2) \Rightarrow \exists g[(g \neq g2 \wedge (D(g) \wedge D(g1))) \Rightarrow D(g2)]]$

...more axioms for predicate D, goal relationships --, -...

Goal Graph


Quality of schedule
Minimal effort
Minimal conflicts
Degree of participation
Matching effort
Collection effort
Schedule meeting
Collect timetables
Choose schedule
By person
By system
Manually
Automatically
By all means
By email
Have updated timetables
Collect them

Qualitative Goal Analysis

- Given a goal graph, we can instantiate these axioms into a collection of propositional Horn clauses, e.g.,

$\forall g1,g2,g3[\mathtt{AND}(\{g1,g2\},g3) \Rightarrow ((\mathtt{S}(g1) \wedge \mathtt{S}(g2)) \Rightarrow \mathtt{S}(g3))]$

$\mathtt{==>}\ (\mathtt{S(collectTbl)} \wedge \mathtt{S(chooseSchl)}) \Rightarrow \mathtt{S(scheduleMtg)}$

- We are also given some S and D labels for some goals, e.g., S(haveUpdatedTbl)
- There is an O(N) proof procedure which will generate all inferences from these axioms. Our proof procedure works as a label propagation algorithm.
- We have also developed algorithms to accommodate probabilities and criticalities for goals.

Quantitative Goal Analysis

- Now, S and D have different meaning:

  `S(g,p)` -- "probability that `g` is satisfied is at least `p`"

  `D(g,p)` -- "probability that `g` is denied is at least `p`"

- For example,
  - ✓ "The probability that a meeting will be scheduled is .95"
  - ✓ "The probability that an ambulance will arrive within 15 minutes is .9"

Axiomatization and Results

- The axioms become

$\forall g1,g2,g3[AND(\{g1,g2\},g3) \Rightarrow ((S(g1,p1) \wedge S(g2,p2)) \Rightarrow S(g3,p1*p2))]$

$\forall g1,g2,g3[OR(\{g1,g2\},g3) \Rightarrow ((S(g1,p1) \wedge S(g2,p2)) \Rightarrow S(g3,p1 \oplus p2))]$

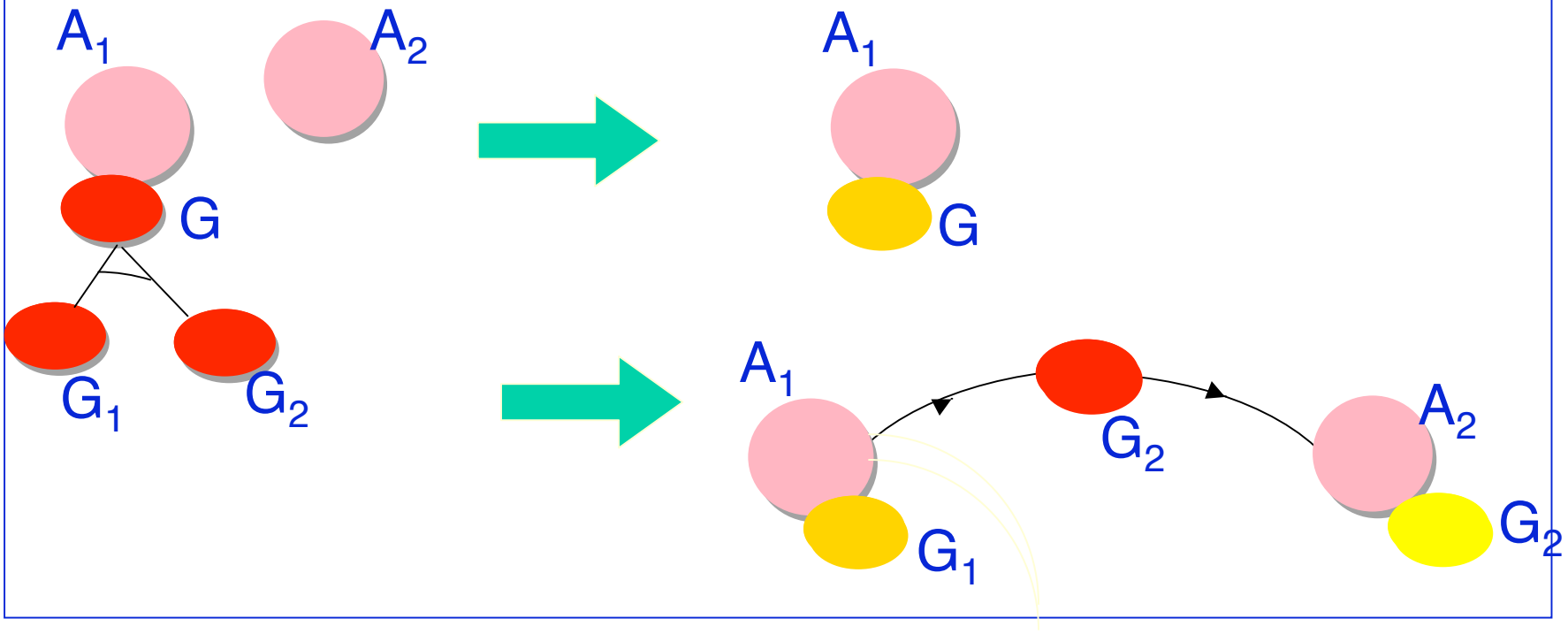
$\forall g1,g2[+(g1,g2,p) \Rightarrow [S(g1,p1)) \Rightarrow S(g2,p*p1)]]$

p1+p2-p1*p2

- We have a label propagation algorithm which is sound and complete wrt this axiomatization, and converges to the right values.
- There are other ways to embellish this axiomatization in order to account for amount of evidence, multiple sources of evidence,...

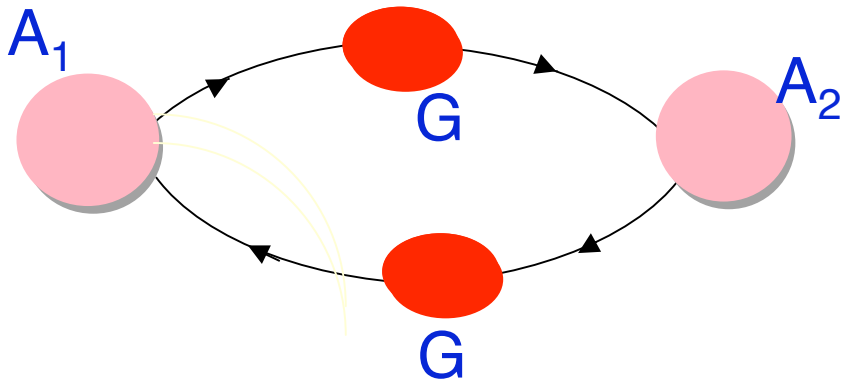
Dependency Graph Analysis

- Given a set of actors, each with associate root goals, and a goal graph for each root goal, find a dependency graph which fulfills all root goals



Well-Formed Dependency Graphs

- Some dependency graphs don't make sense...

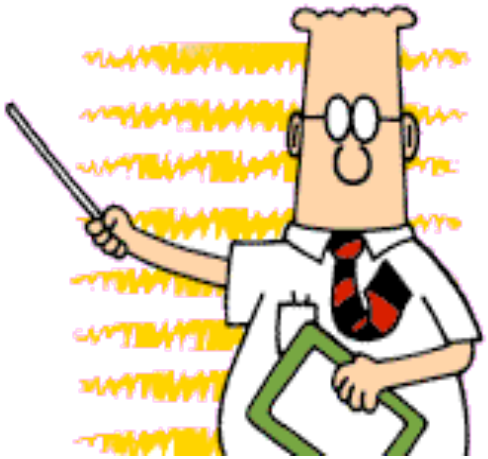


- What is a "good" dependency graph assuming that we are interested in:
  - ✓ minimizing dependence;
  - ✓ distributing work;
  - ✓ Network stability.
- PhD thesis by Volha Bryl (Trento).

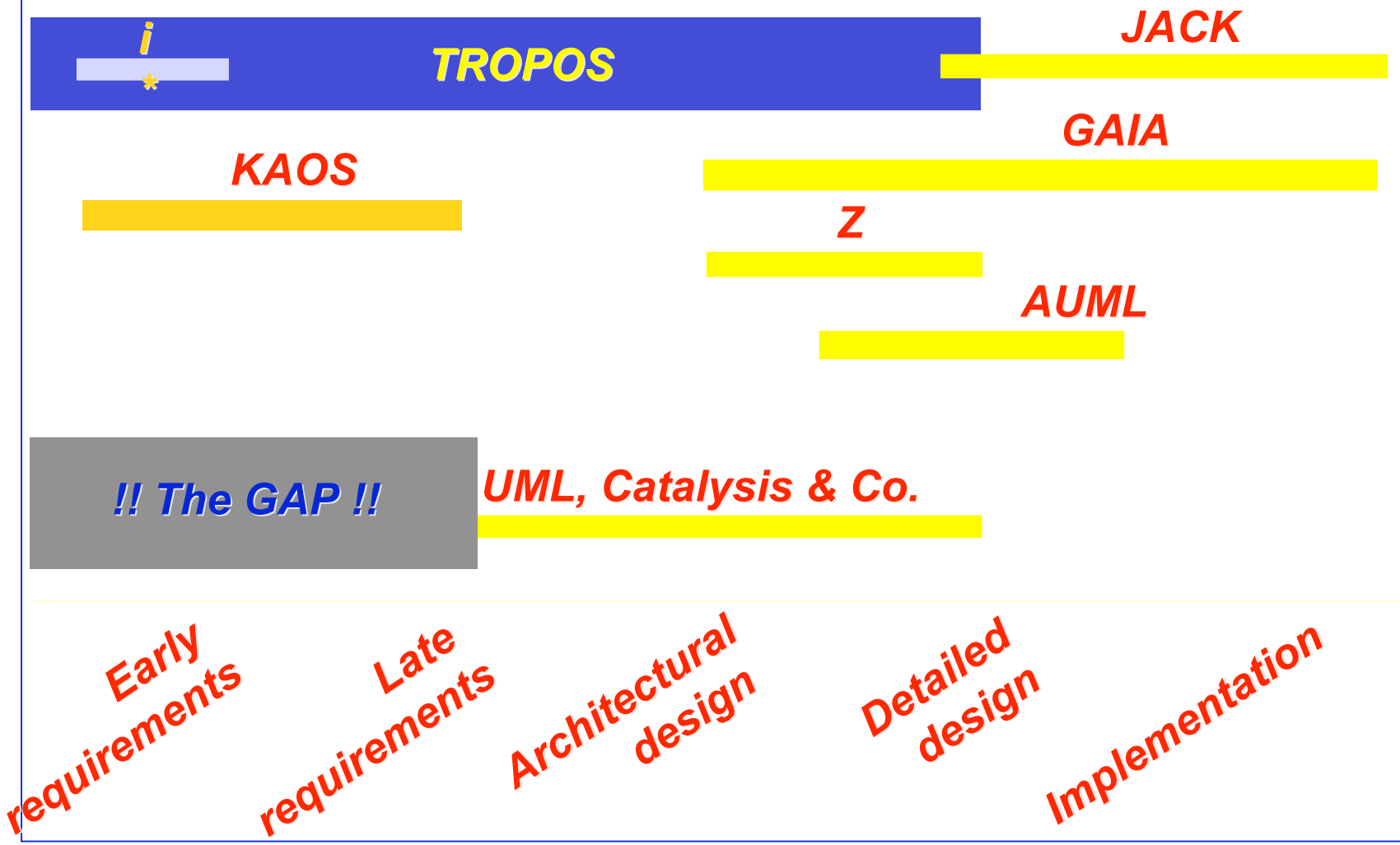
Other Threads

- [Security] Extend Tropos to support concepts of ownership, permission and trust; this leads to models where you can check whether every actor has the permissions she needs to carry out her obligations [RE'05].
- PhD thesis by Nicola Zannone (Trento).
- [Risk Management] Extend the DDP risk management framework [Feather05] to allow hierarchical goal/requirement and risk decompositions.
- PhD thesis by Yudis Asnar (Trento).

Beware!!!
When designing software organizational systems, try to avoid the pitfalls of human organizational ones…



Related Work



The Tropos Project

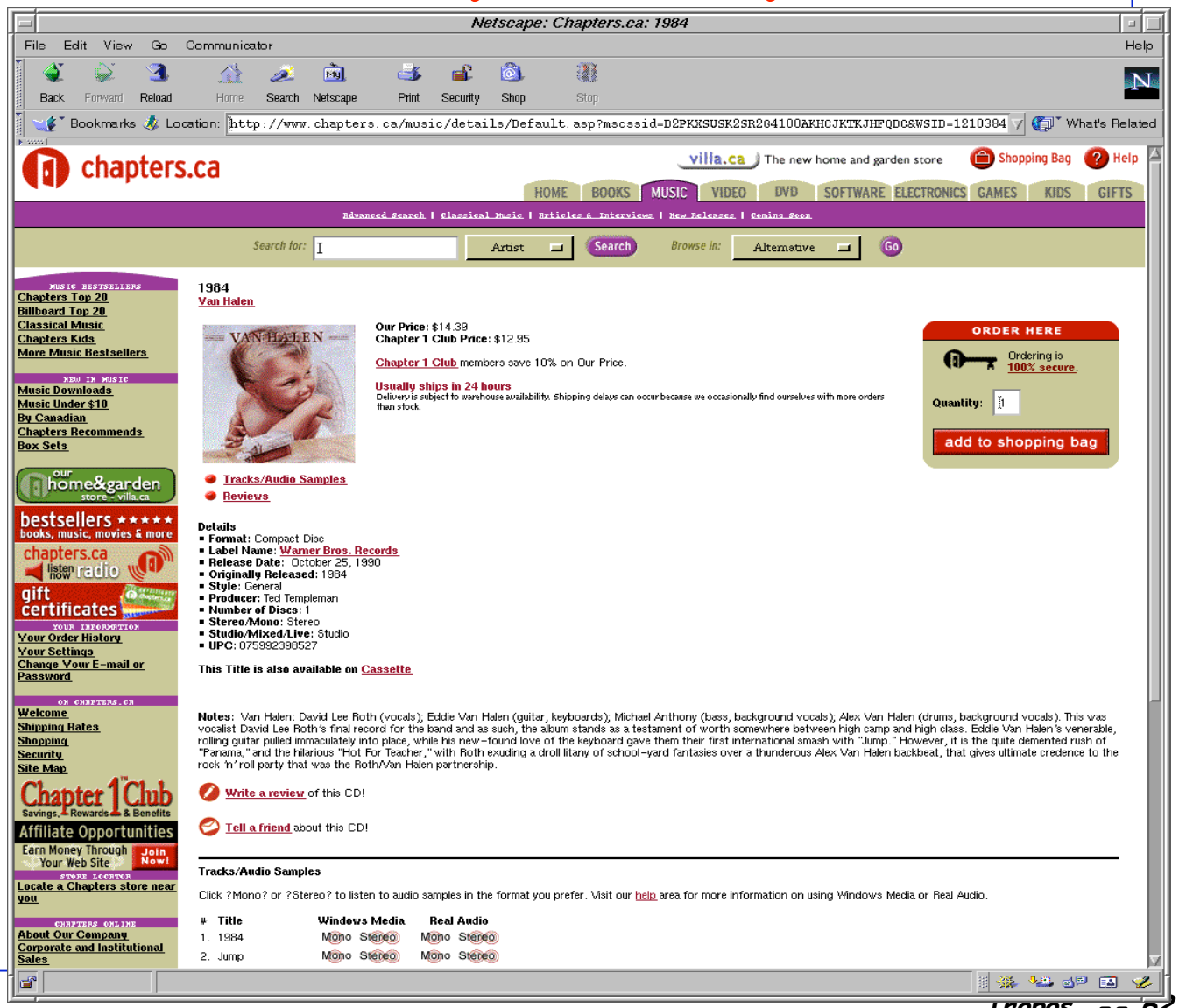
- Project was launched in April 2000.
- Participating teams includes:
  - ✓ UToronto (Canada): Eric Yu, Alexei Lapouchnian, Sotirios Liaskos, Yijun Yu, Yiqiao Wang, Neil Ernst;
  - ✓ UTrento/IRST (Italy): Anna Perini, Angelo Susi, Loris Penserini, Paolo Giorgini, Fabio Massacci, Roberto Sebastiani, Nicola Zannone, Yudis Asnar, Volha Bryl, Paolo Traverso,...;
  - ✓ Elsewhere: Jaelson Castro (Brazil), Matthias Jarke (Germany), Manuel Kolp (Belgium), Julio Leite (Brazil), Gerhard Lakemeyer (Germany), Lin Liu (China);
- Publications and other information about the project can be found at http://www.troposproject.org.

Conclusions

- We have proposed a set of concepts and sketched a methodology that can support Agent-Oriented Software Development.
- Agent-Oriented software development is an up-and-coming paradigm because of an ever-growing demand for customizable, robust and open software systems that truly meet the needs and intentions of their stakeholders.
- This is a long-term project, and much remains to be done.

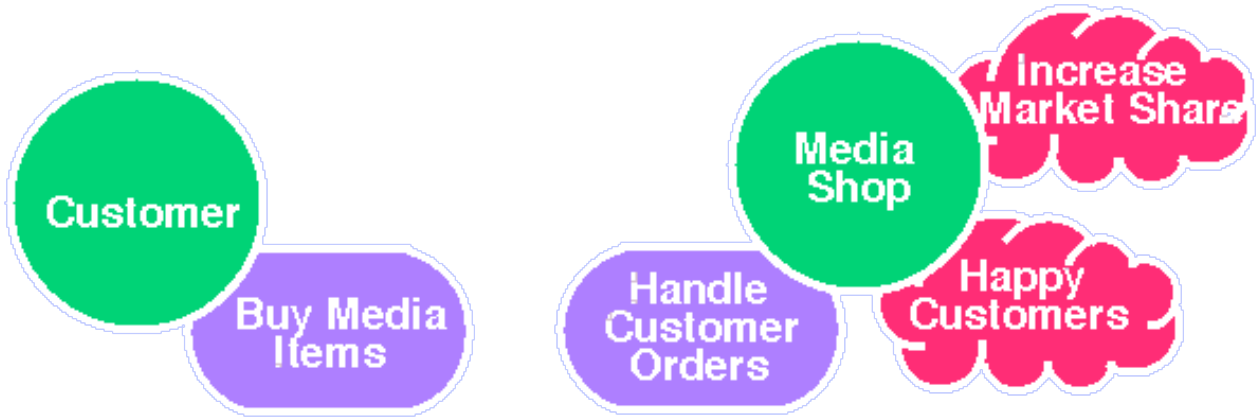
The Media Shop Example

- Media taxonomy
  - ✓ on-line catalog
  - ✓ DBMS
- E-Shopping Cart
  - ✓ Check In
  - ✓ Buying
  - ✓ Check Out
- Search Engine
  - ✓ catalog browser
  - ✓ Keywords
  - ✓ full-text
- Secure
  - ✓ $ transactions
  - ✓ orders
- Multimedia
  - ✓ description
  - ✓ samples



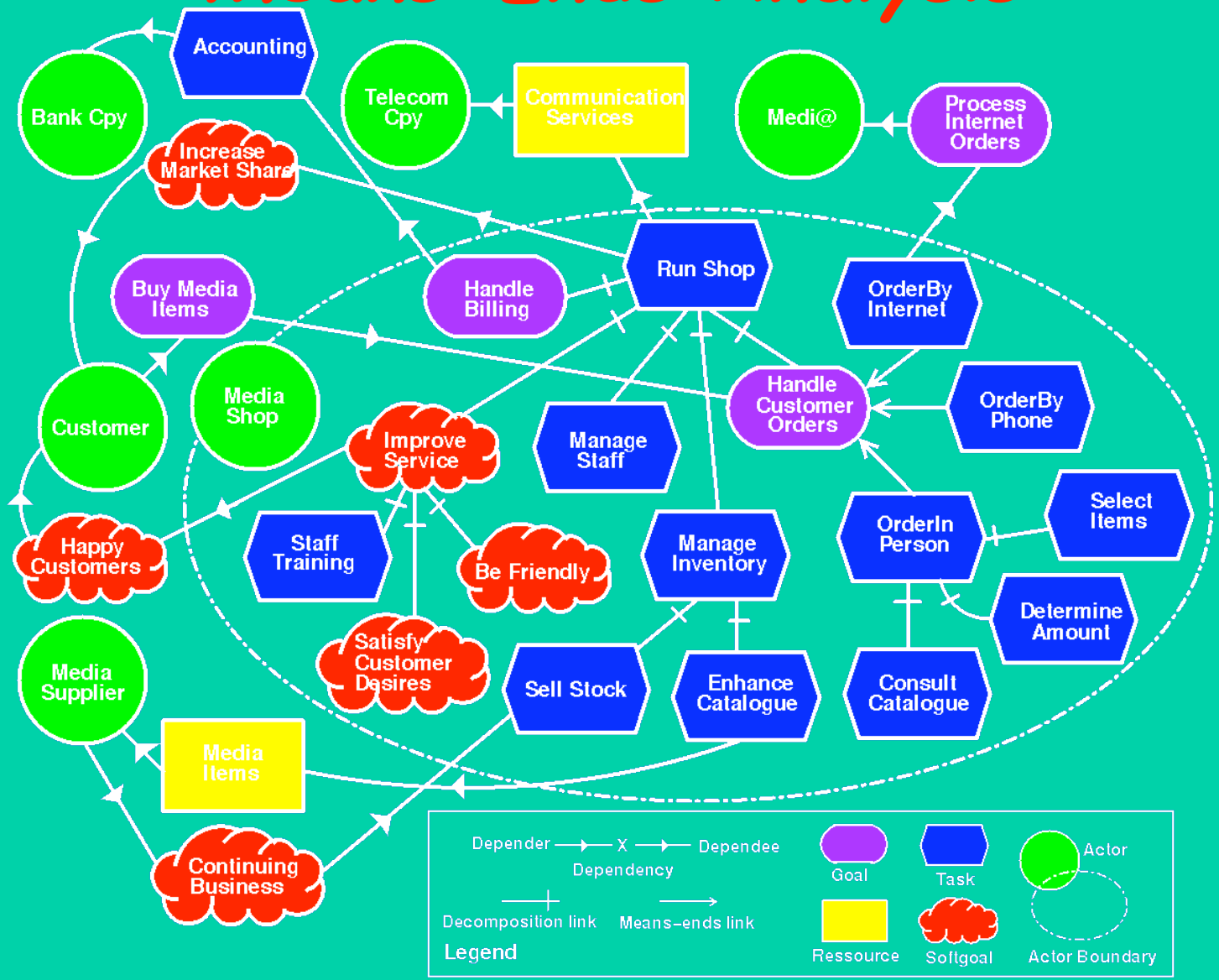
Early Requirements Analysis

- Understand the *organizational* setting, produce an *organizational model* with relevant *actors* and their respective *goals*.

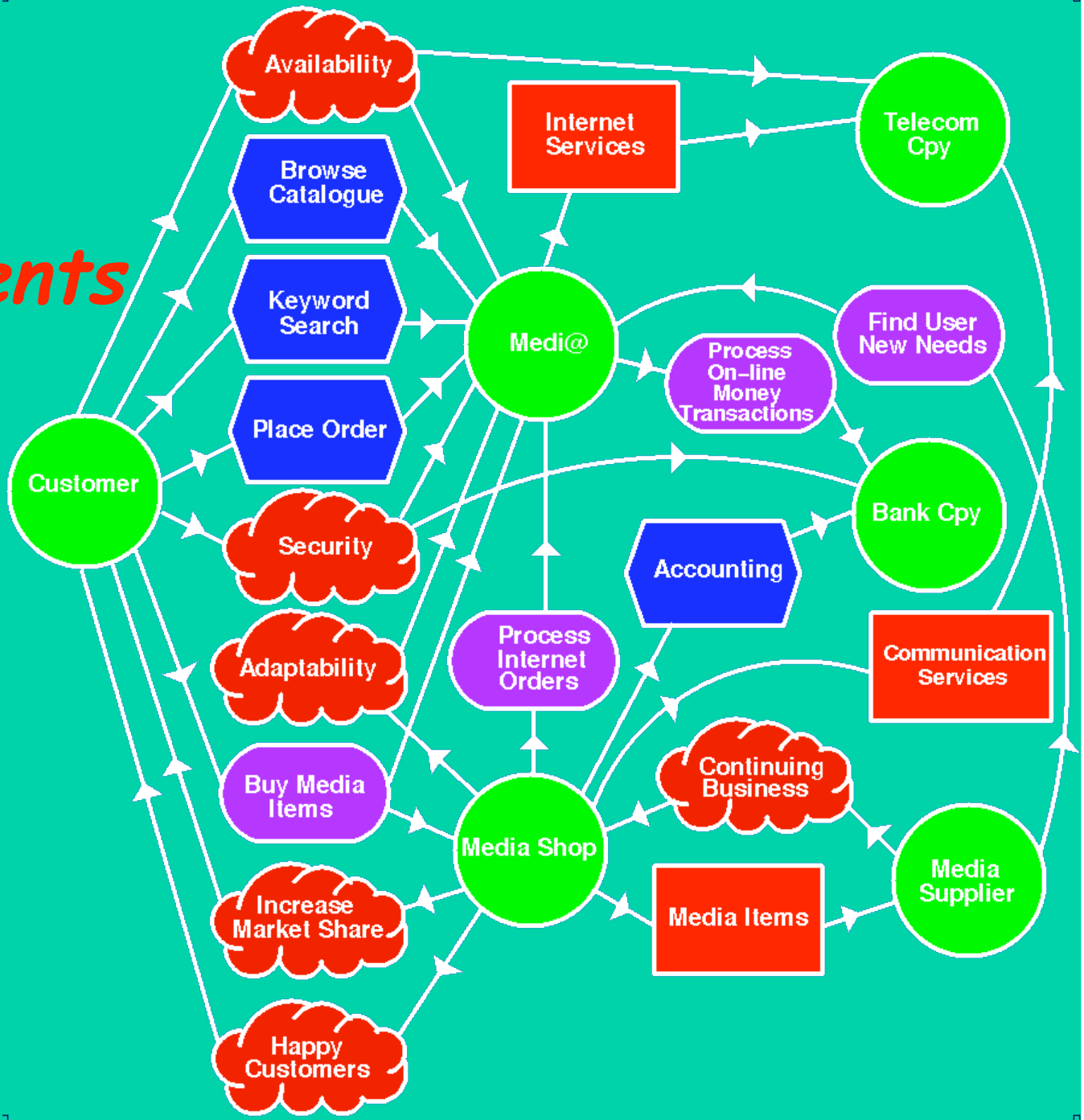


Goals are relative, fulfillment is collaborative

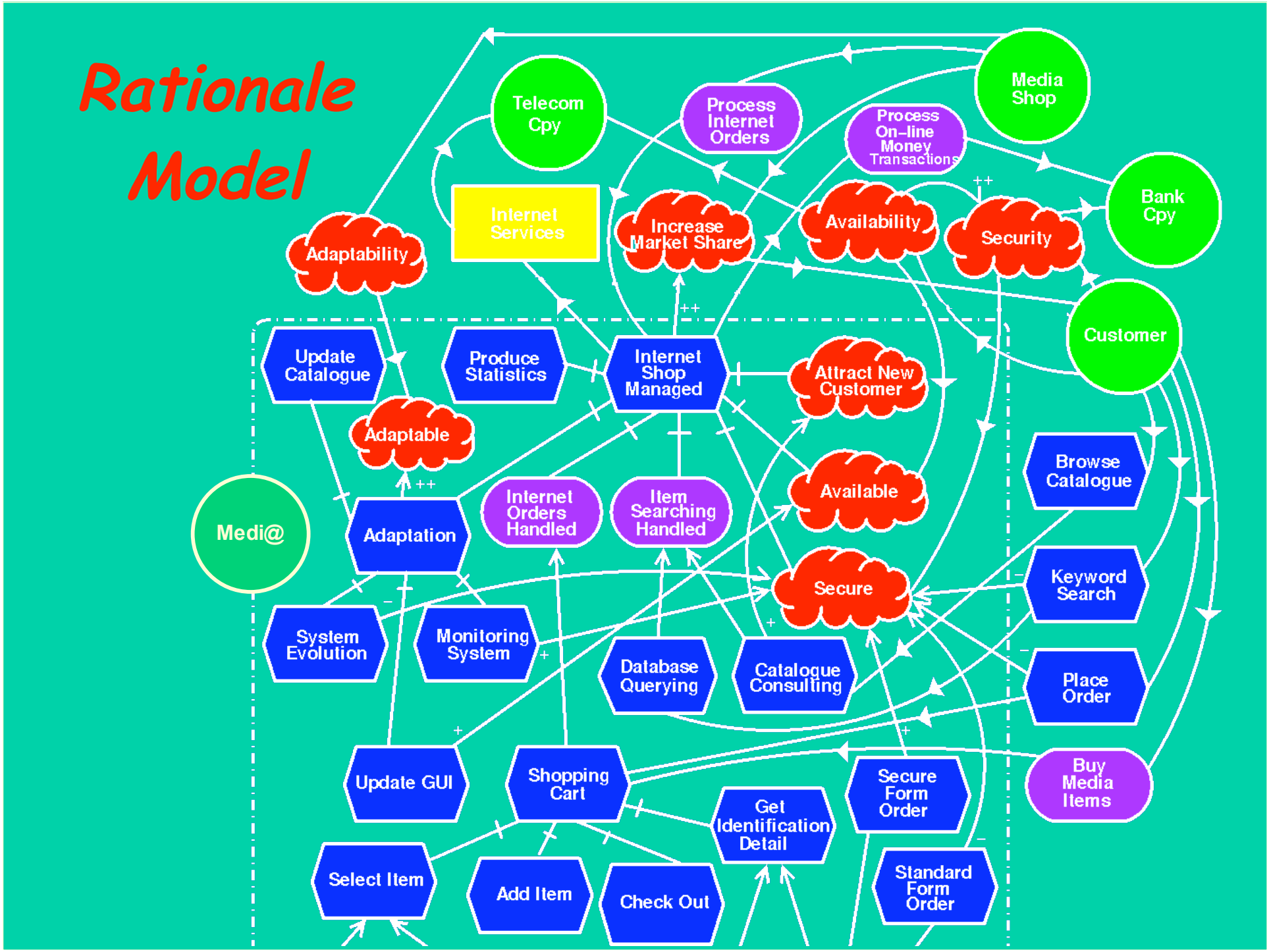
Means-Ends Analysis



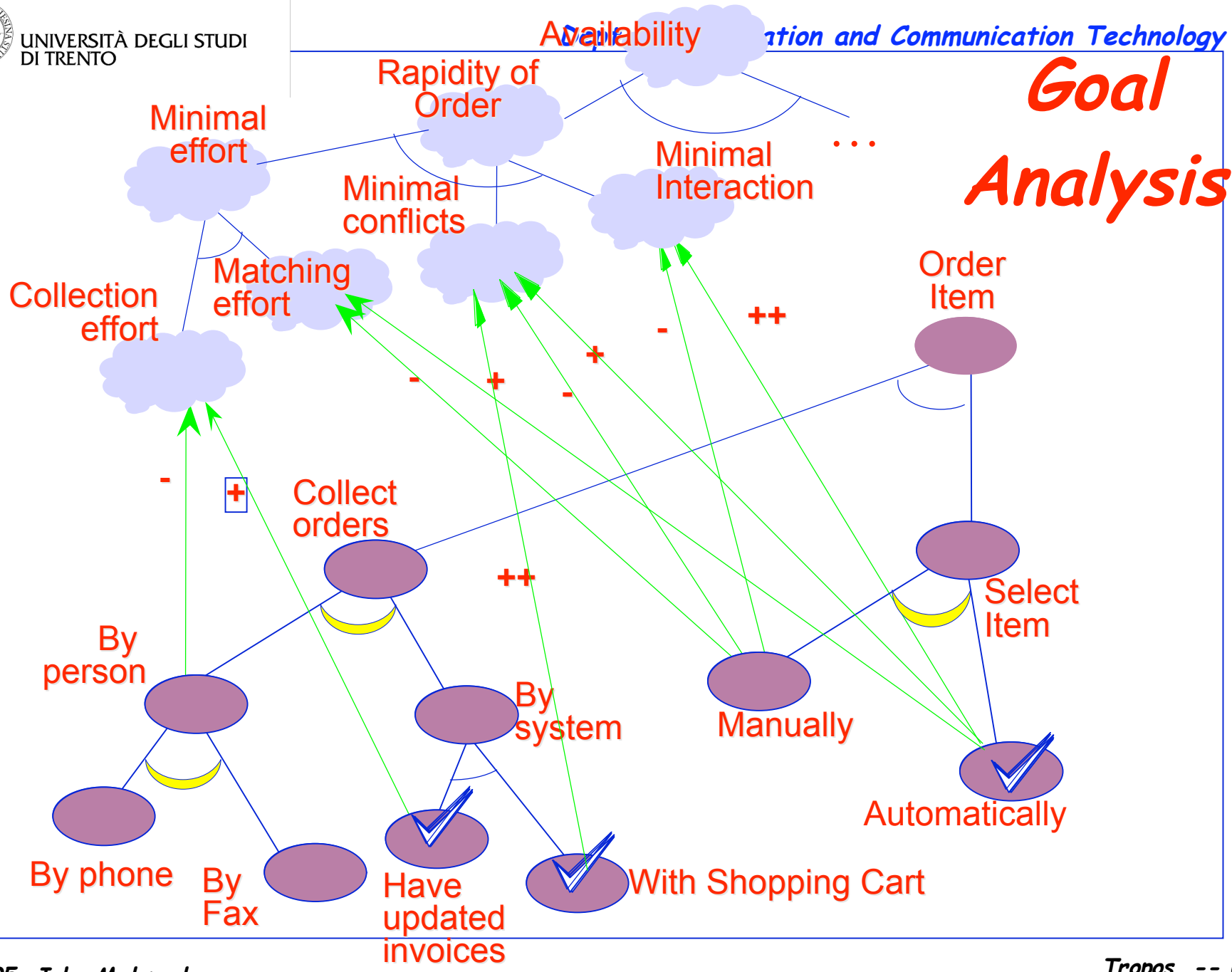
Late Requirements



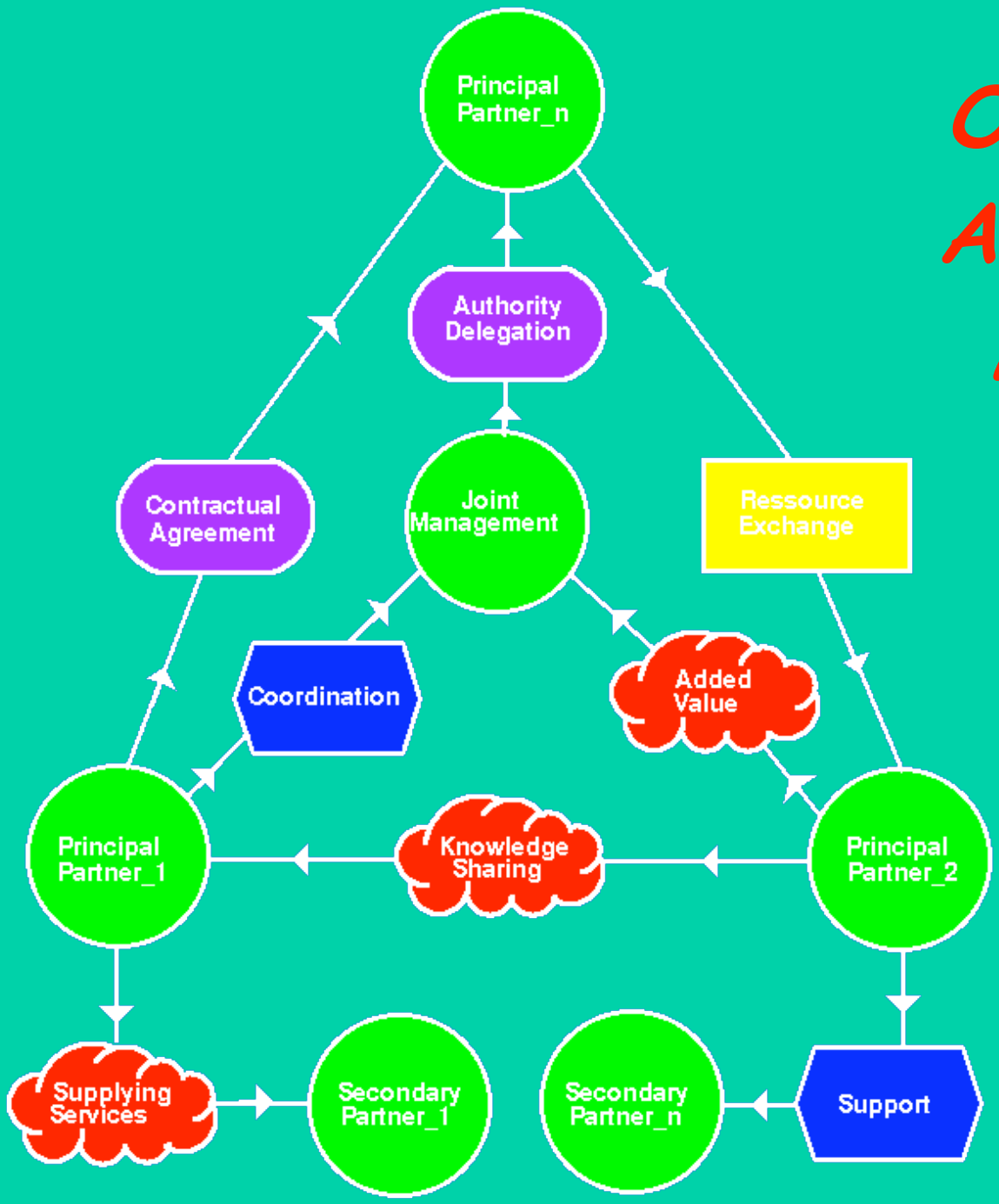
Rationale Model



Goal Analysis

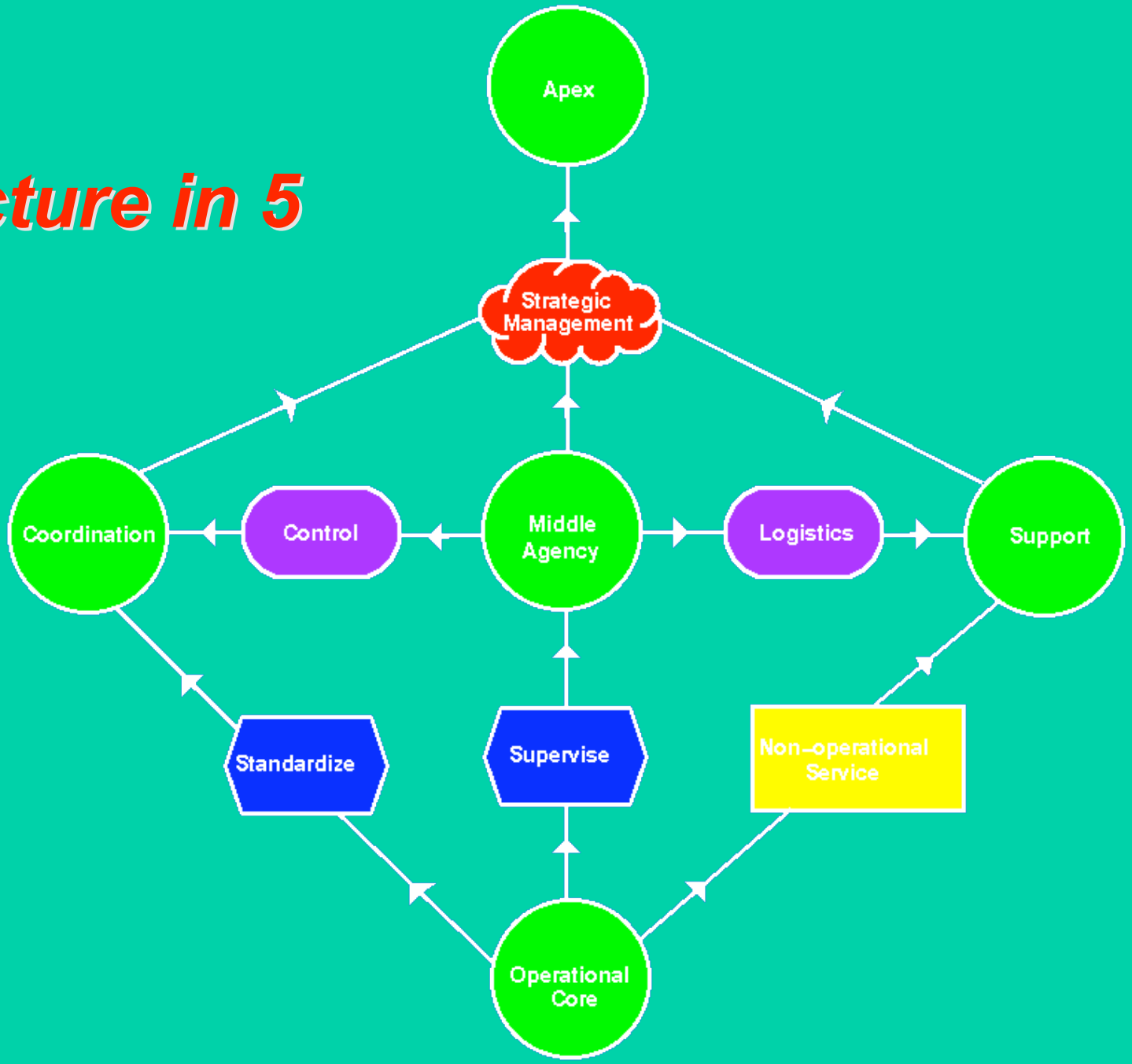


Organizational Architectures: Macro Level



Joint Venture

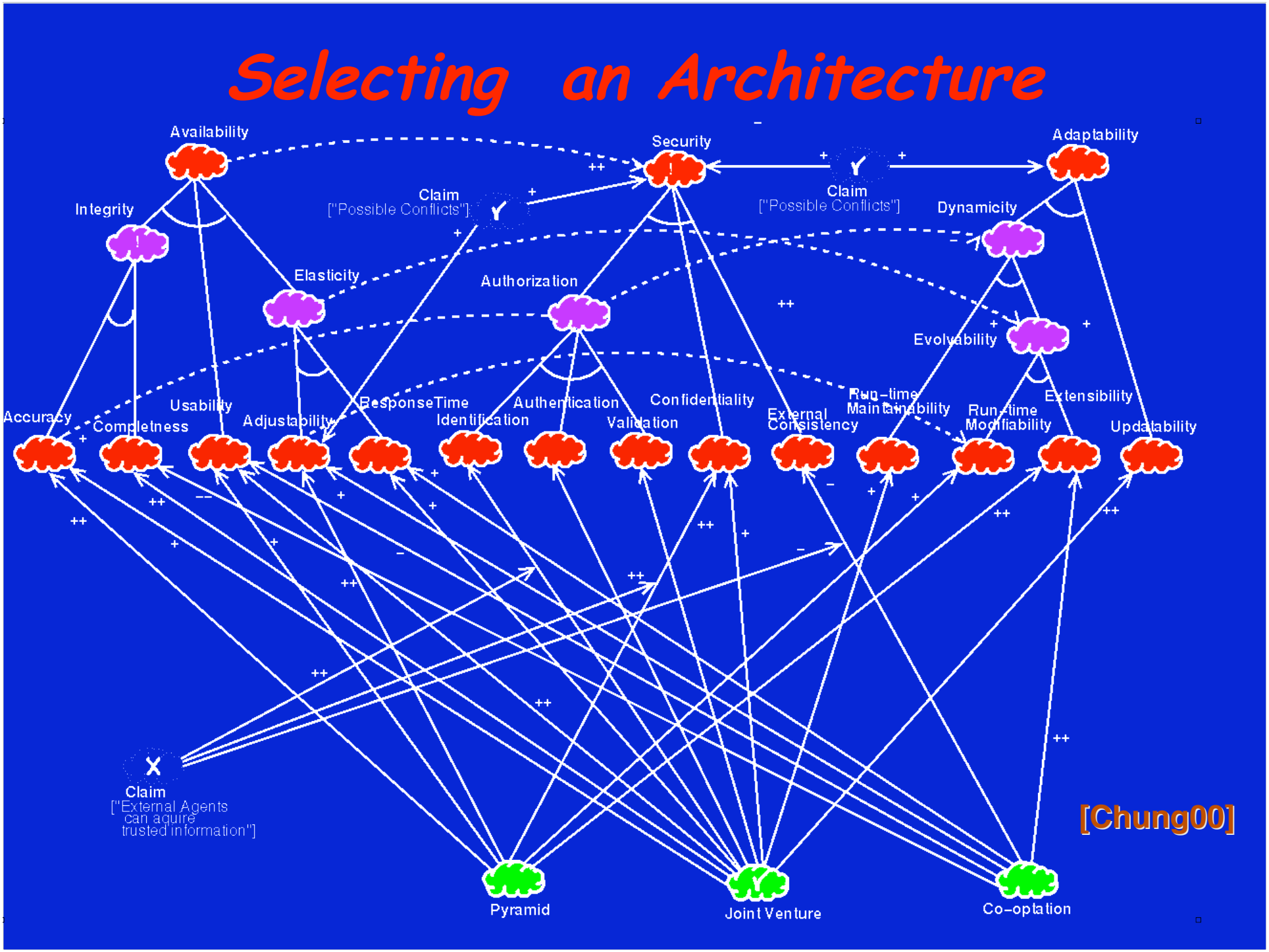
Structure in 5

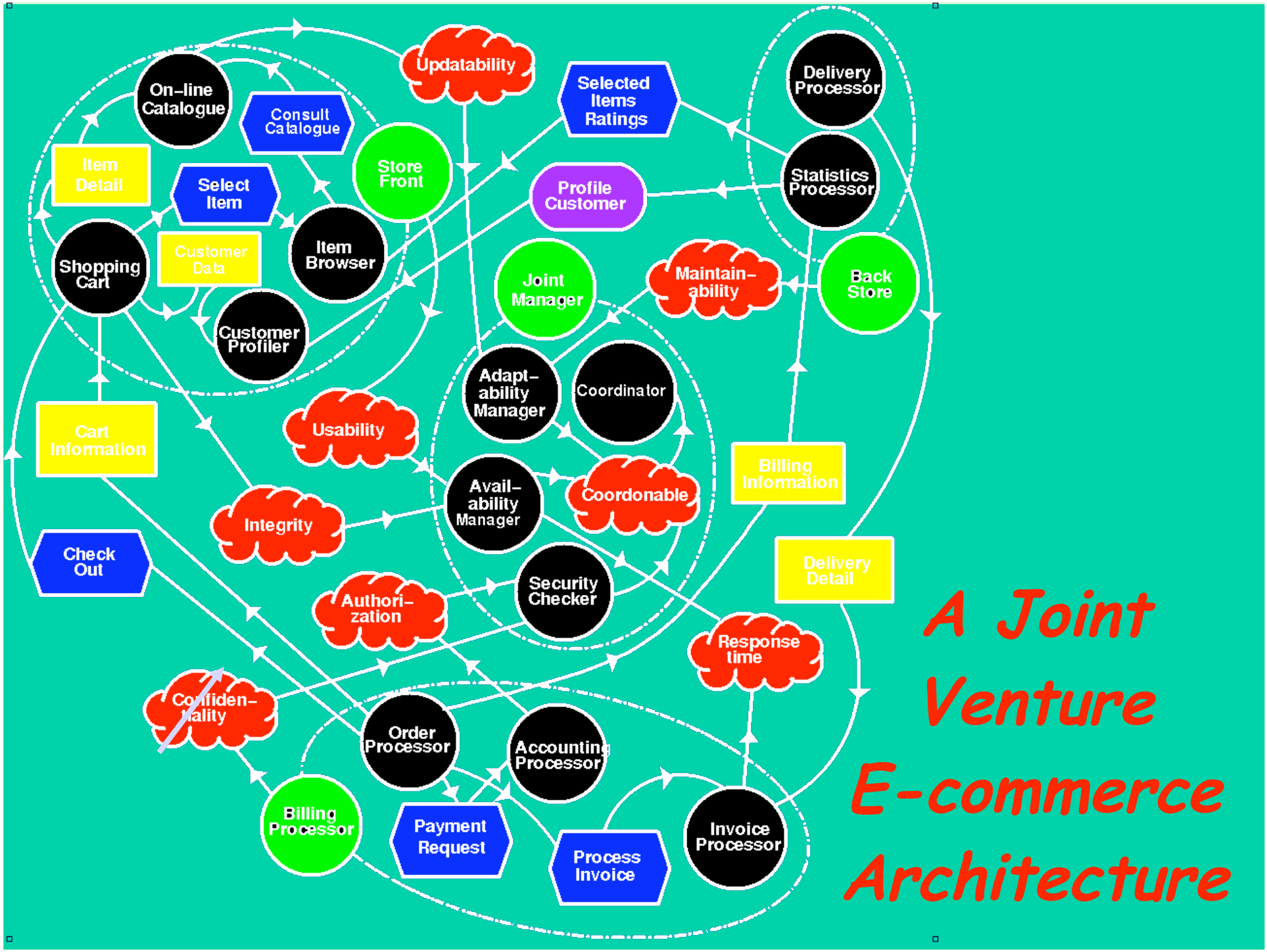


Choosing an Architecture

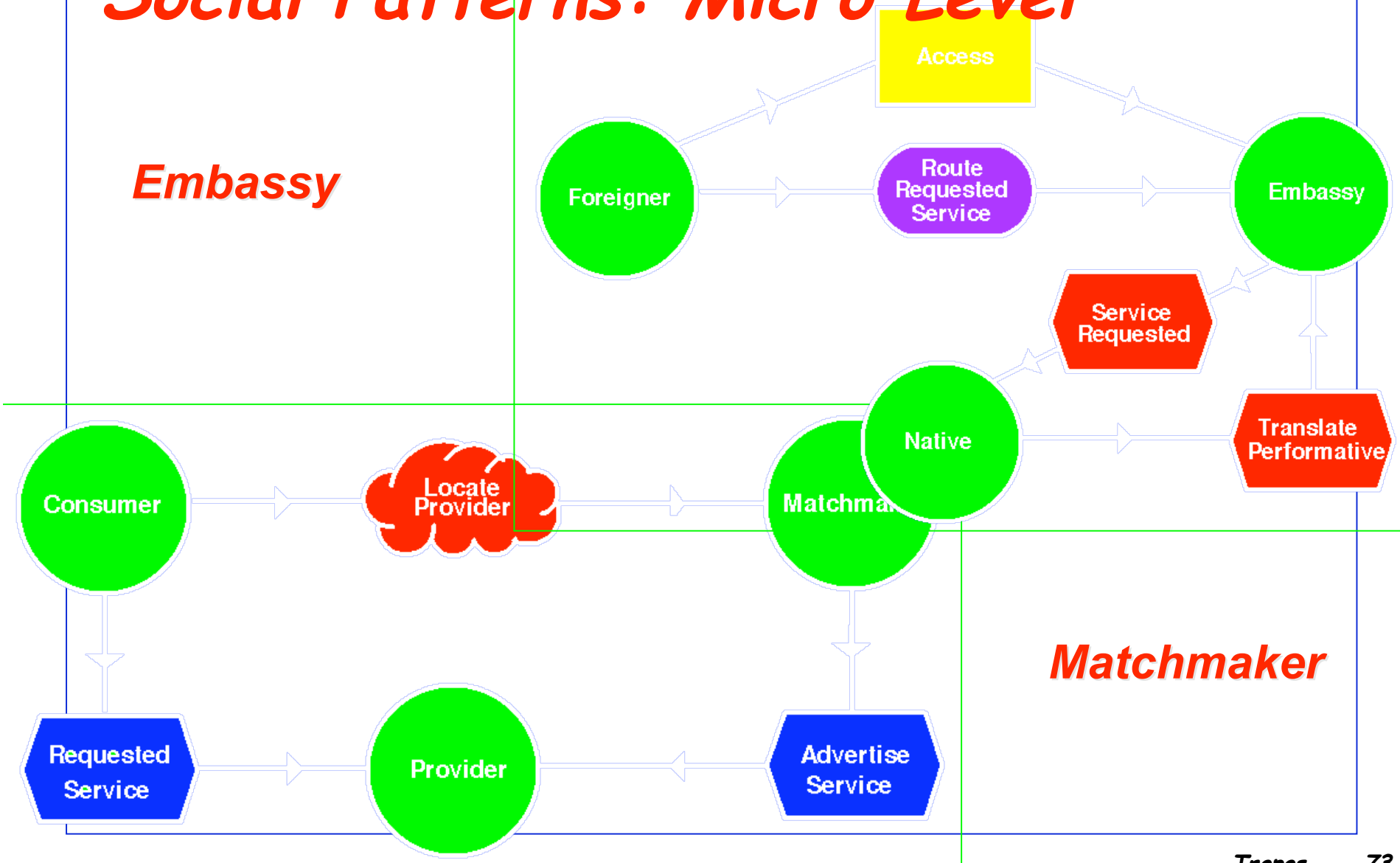
Correlation Catalog	Predict.	Secur.	Adapt.	Cooperat.	Compet.	Availab.	Failabil.	Modul.	Aggreg.
Flat Structure	BREAK	BREAK	MAKE			HELP	HELP	MAKE	HURT
Structure-in-5	HELP	HELP		HELP	HURT	HELP		MAKE	MAKE
Pyramid	MAKE	MAKE	HELP	MAKE	BREAK	HELP	BREAK	HURT	
Join Venture	HELP	HELP	MAKE	HELP	HURT	MAKE		HELP	MAKE
Bidding	BREAK	BREAK	MAKE	HURT	MAKE	HURT	BREAK	MAKE	
Takeover	MAKE	MAKE	HURT	MAKE	BREAK	HELP		HELP	HELP
Arm's-Length	HURT	BREAK	HELP	HURT	MAKE	BREAK	MAKE	HELP	
Hierarch Cont.			HELP	HELP	HELP	HELP		HELP	HELP
Vert. Integ.	HELP	HELP	HURT	HELP	HURT	HELP	BREAK	BREAK	BREAK
Co-optation	HURT	HURT	MAKE	MAKE	HELP	HURT	HELP		

Selecting an Architecture

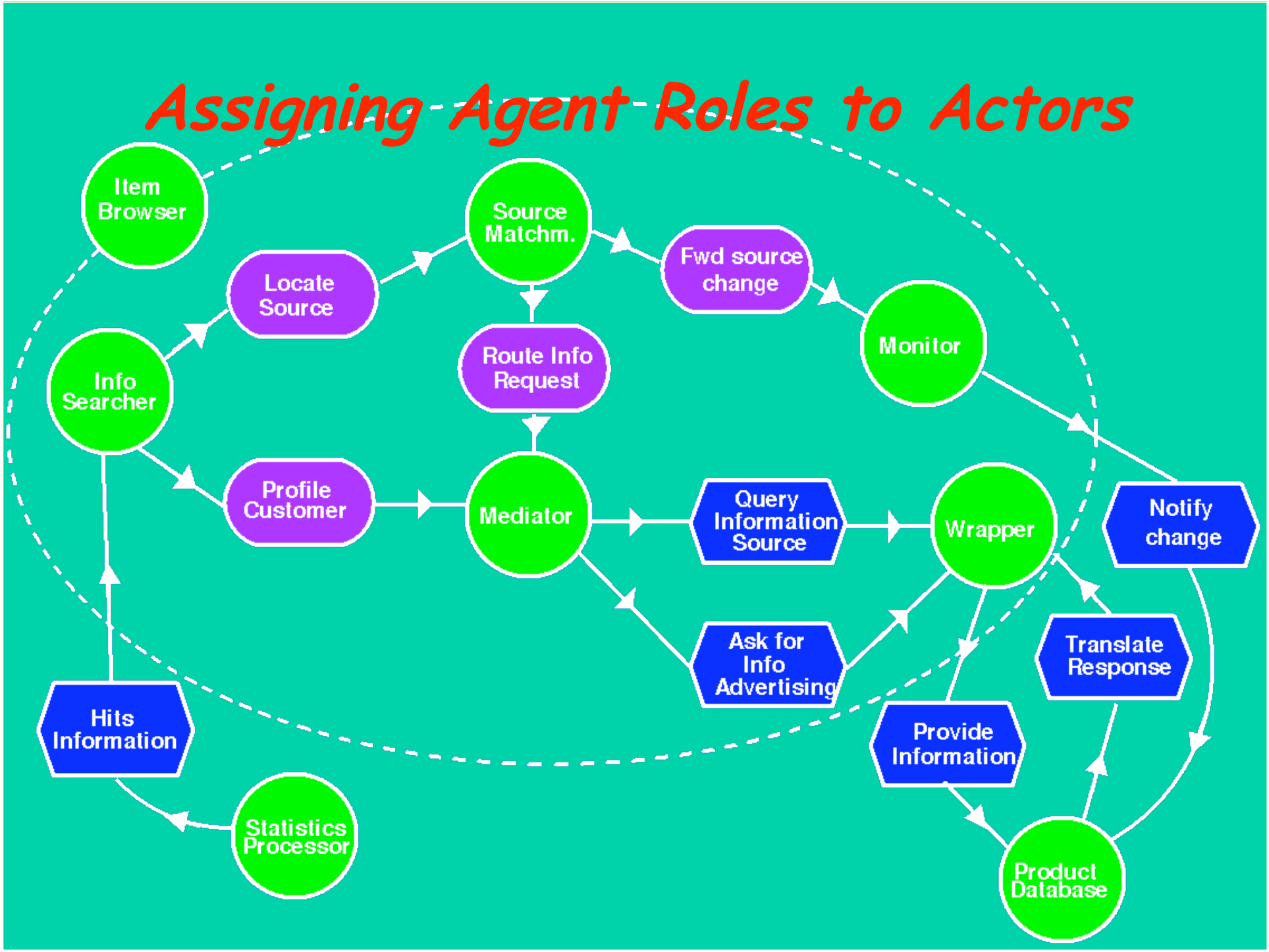



Updatability
On-line Catalogue
Consult Catalogue
Selected Items Ratings
Delivery Processor
Item Detail
Store Front
Select Item
Profile Customer
Statistics Processor
Shopping Cart
Customer Data
Item Browser
Joint Manager
Maintain-ability
Back Store
Customer Profiler
Adapt-ability Manager
Coordinator
Cart Information
Usability
Billing Information
Avail-ability Manager
Coordonable
Integrity
Check Out
Delivery Detail
Security Checker
Authori-zation
Response time
Confiden-tiality
Order Processor
Accounting Processor
Billing Processor
Payment Request
Process Invoice
Invoice Processor
A Joint Venture
E-commerce
Architecture

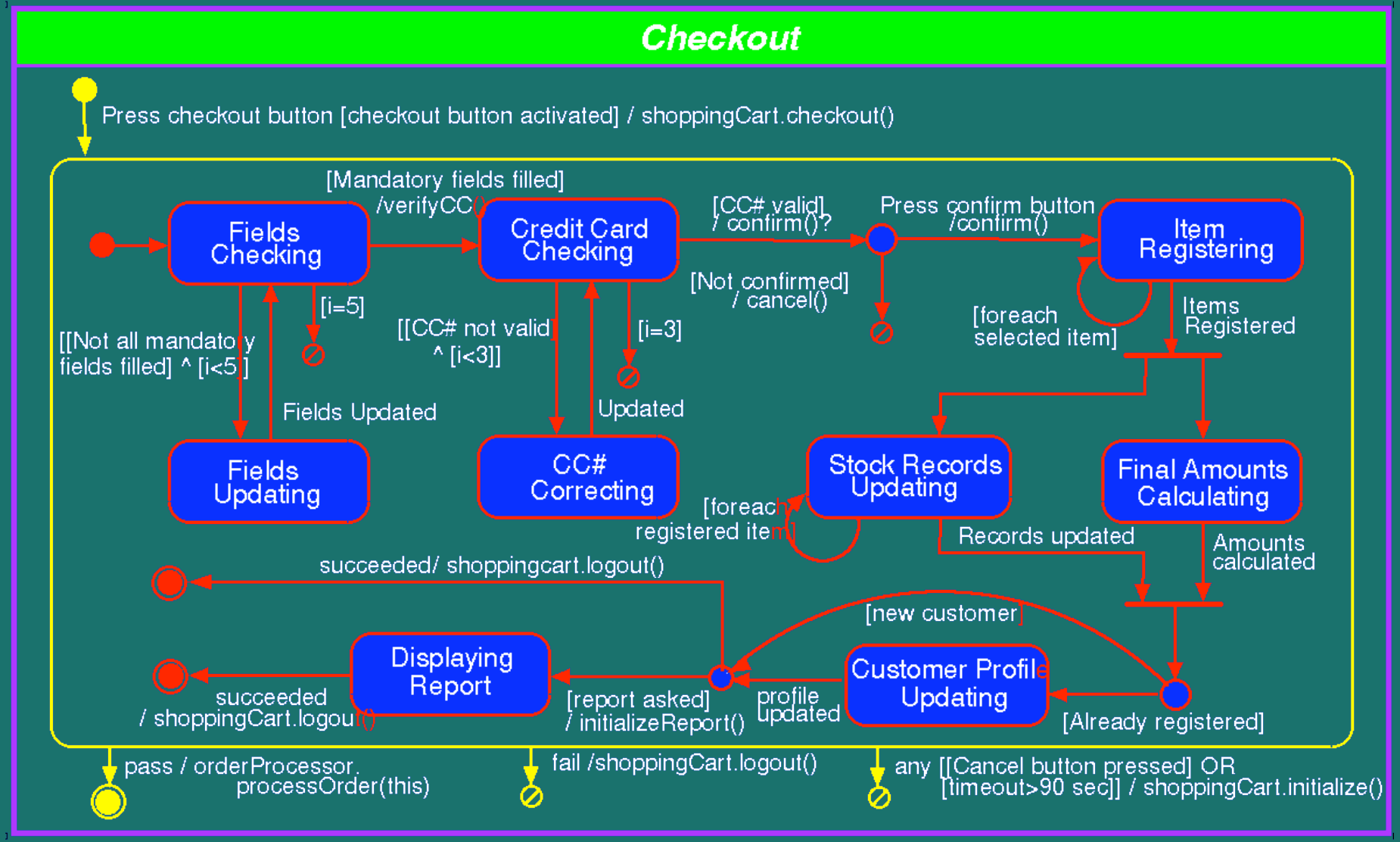
Social Patterns: Micro Level

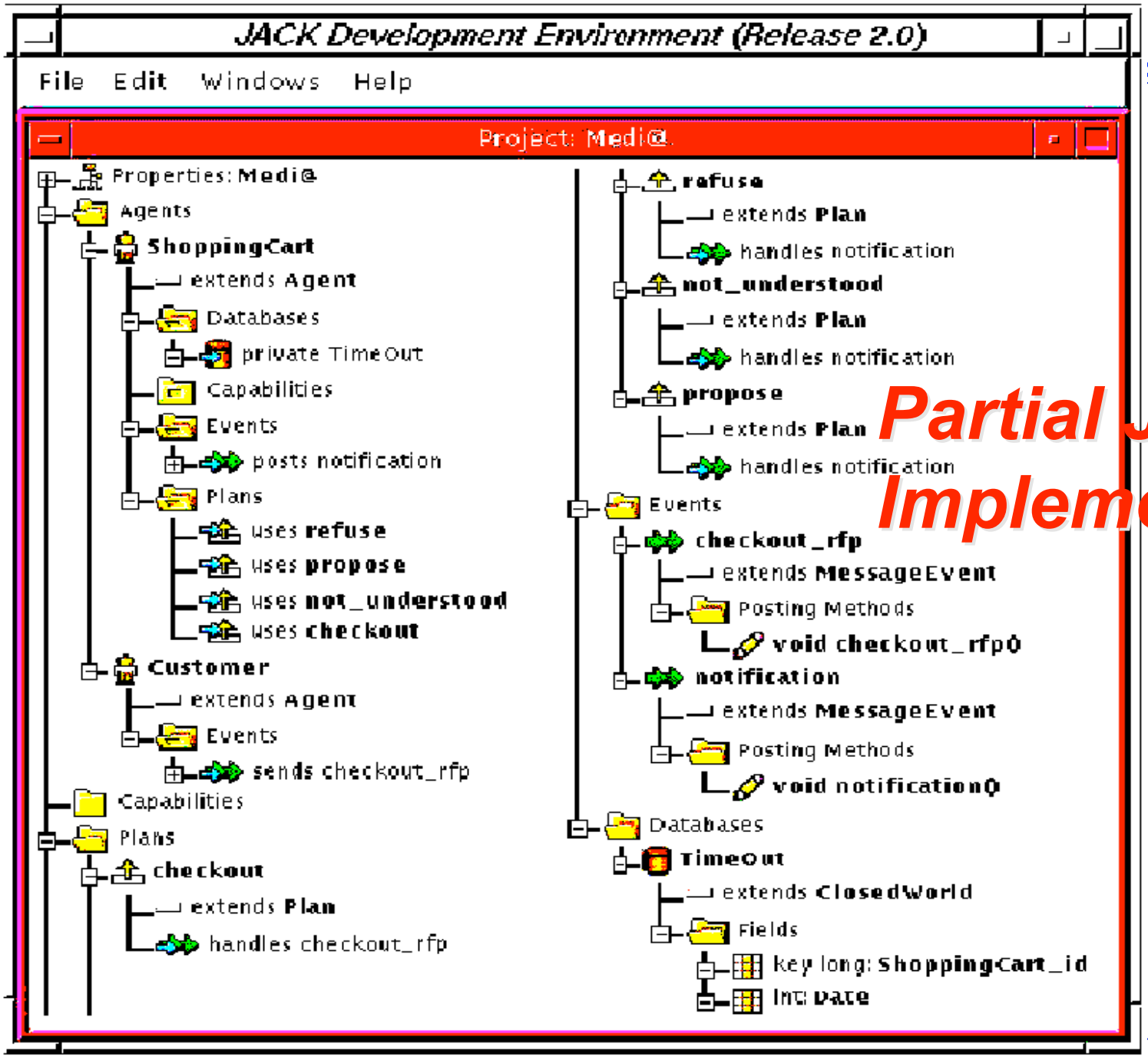


Assigning Agent Roles to Actors



Detailed Design





Partial JACK Implementation

References

- [Bauer99] Bauer, B., *Extending UML for the Specification of Agent Interaction Protocols.* OMG document ad/99-12-03.
- [Castro02] Castro, J., Kolp, M., Mylopoulos, J., "Towards Requirements-Driven Software Development Methodology: The Tropos Project," *Information Systems 27(2)*, Pergamon Press, June 2002, 365-389.
- [Chung00] Chung, L., Nixon, B., Yu, E., Mylopoulos, J., *Non-Functional Requirements in Software Engineering*, Kluwer Publishing, 2000.
- [Dardenne93] Dardenne, A., van Lamsweerde, A. and Fickas, S., "Goal-directed Requirements Acquisition", *Science of Computer Programming, 20*, 1993.
- [Fuxman01a] .Fuxman, A., Pistore, M., Mylopoulos, J. and Traverso, P., "Model Checking Early Requirements Specifications in Tropos", Proceedings Fifth International IEEE Symposium on Requirements Engineering, Toronto, August 2001.
- [Fuxman01b] Fuxman,A., Giorgini, P., Kolp, M., Mylopoulos, J., "Information Systems as Social Organizations", Proceedings International Conference on Formal Ontologies for Information Systems, Ogunquit Maine, October 2001.

...More References

- [Iglesias98] Iglesias, C., Garrijo, M. and Gonzalez, J., "A Survey of Agent-Oriented Methodologies", *Proceedings of the 5th International Workshop on Intelligent Agents: Agent Theories, Architectures, and Languages* (ATAL-98), Paris, France, July 1998.
- [Jennings00] Jennings, N. "On Agent-Based Software Engineering", *Artificial Intelligence 117*, 2000.
- [Mylopoulos92] .Mylopoulos, J., Chung, L. and Nixon, B., "Representing and Using Non-Functional Requirements: A Process-Oriented Approach," *IEEE Transactions on Software Engineering 18(6)*, June 1992, 483-497.
- [Odell00] Odell, J., Van Dyke Parunak, H. and Bernhard, B., "Representing Agent Interaction Protocols in UML", *Proceedings 1st International Workshop on Agent-Oriented Software Engineering* (AOSE00), Limerick, June 2000.
- [Wooldridge00] Wooldridge, M., Jennings, N., and Kinny, D., "The Gaia Methodology for Agent-Oriented Analysis and Design," *Journal of Autonomous Agents and Multi-Agent Systems, 3(3)*, 2000, 285–312.
- [Yu95] Yu, E., *Modelling Strategic Relationships for Process Reengineering*, Ph.D. thesis, Department of Computer Science, University of Toronto, 1995.
- [Zambonelli00] Zambonelli, F., Jennings, N., Omicini, A., and Wooldridge, M., "Agent-Oriented Software Engineering for Internet Applications," in Omicini, A., Zambonelli, F., Klusch, M., and Tolks-Dorf R., (editors), *Coordination of Internet Agents: Models, Technologies, and Applications*, Springer-Verlag LNCS, 2000, 326–346.