

UML for Agent-Oriented Software Development: The Tropos Proposal

**John Mylopoulos
University of Toronto**

**UML'01 Conference
Toronto, October 3-5, 2001**



Why Agent-Oriented Software?

- Next generation software engineering will have to support open, dynamic architectures where components can accomplish tasks in a variety of operating environments.
- Consider application areas such as eBusiness, application service provision, pervasive or P2P computing.
- These all call for software components that find and compose services dynamically, establish/drop partnerships with other components and operate under a broad range of conditions.
- Learning, planning, communication and negotiation become essential features for such software components.

... agents!

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!**

Why Worry About Human/Organizational Agents?

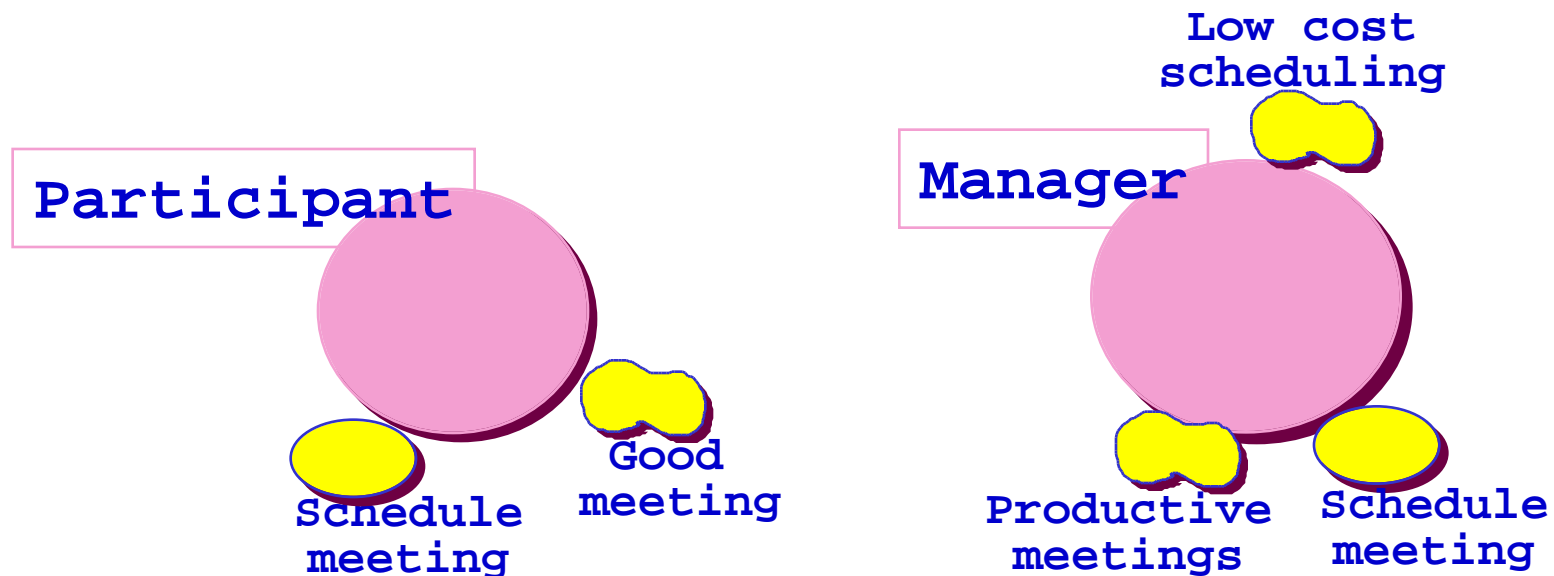
- Because their goals lead to software requirements, and these influence the design of a software system.
- Note the role of human/organizational agents in OOA, e.g., use cases.
- Also note the role of agents in up-and-coming requirements engineering techniques such as KAOS [Dardenne93].
- In KAOS, requirements analysis begins with a set of goals; these are analysed/decomposed to simpler goals which either lead to software requirements, or are delegated to external agents.

The Tropos Methodology

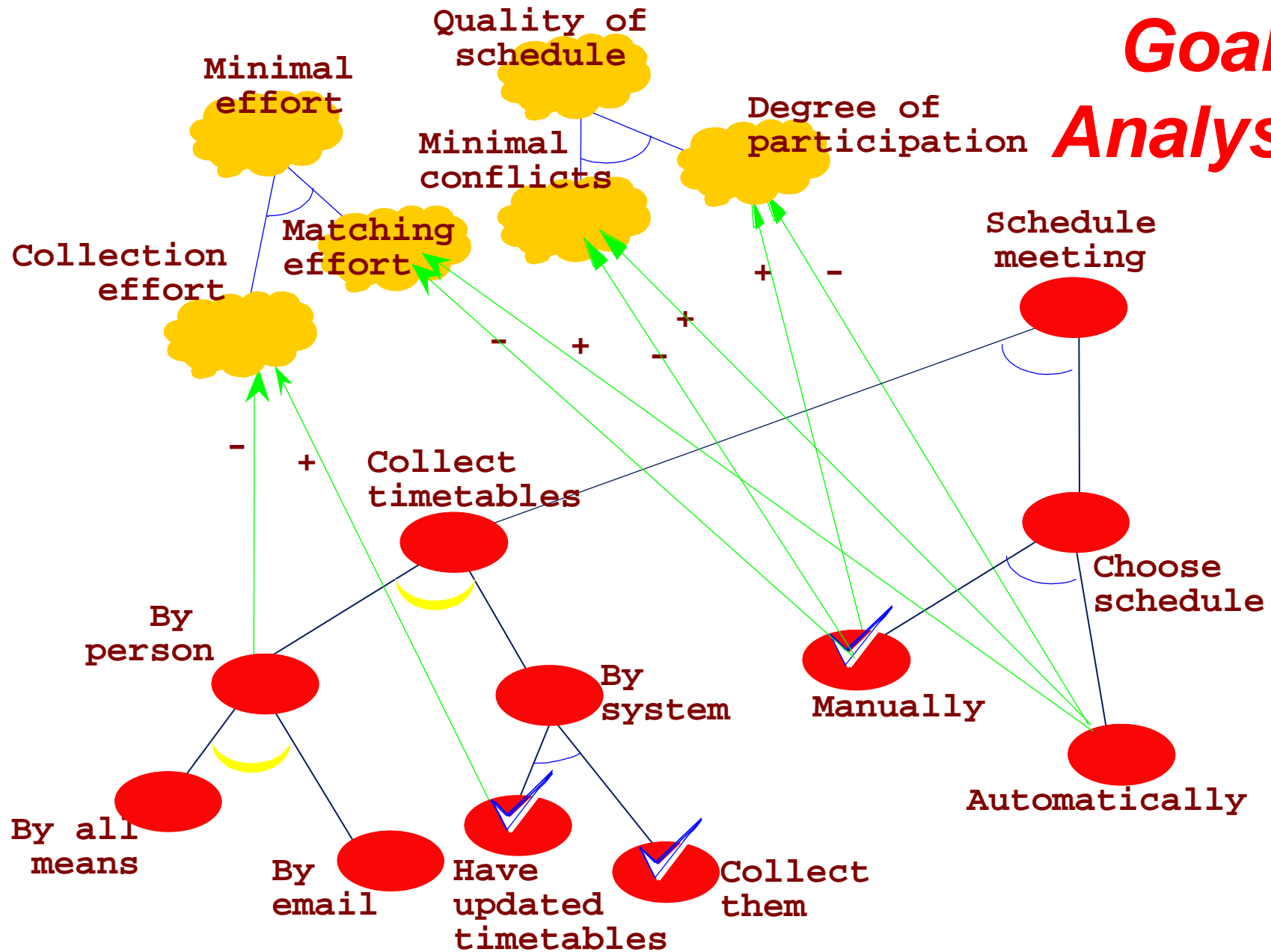
- We propose a set of primitive concepts and a methodology for agent-oriented requirements analysis and design. We adopt i^* [Yu95] as a modeling framework.
- Actors = Agents \cup Positions \cup Roles.
- We want to cover four phases of software development:
 - ✓ **Early requirements** -- identifies stakeholders 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: Actors and their Goals

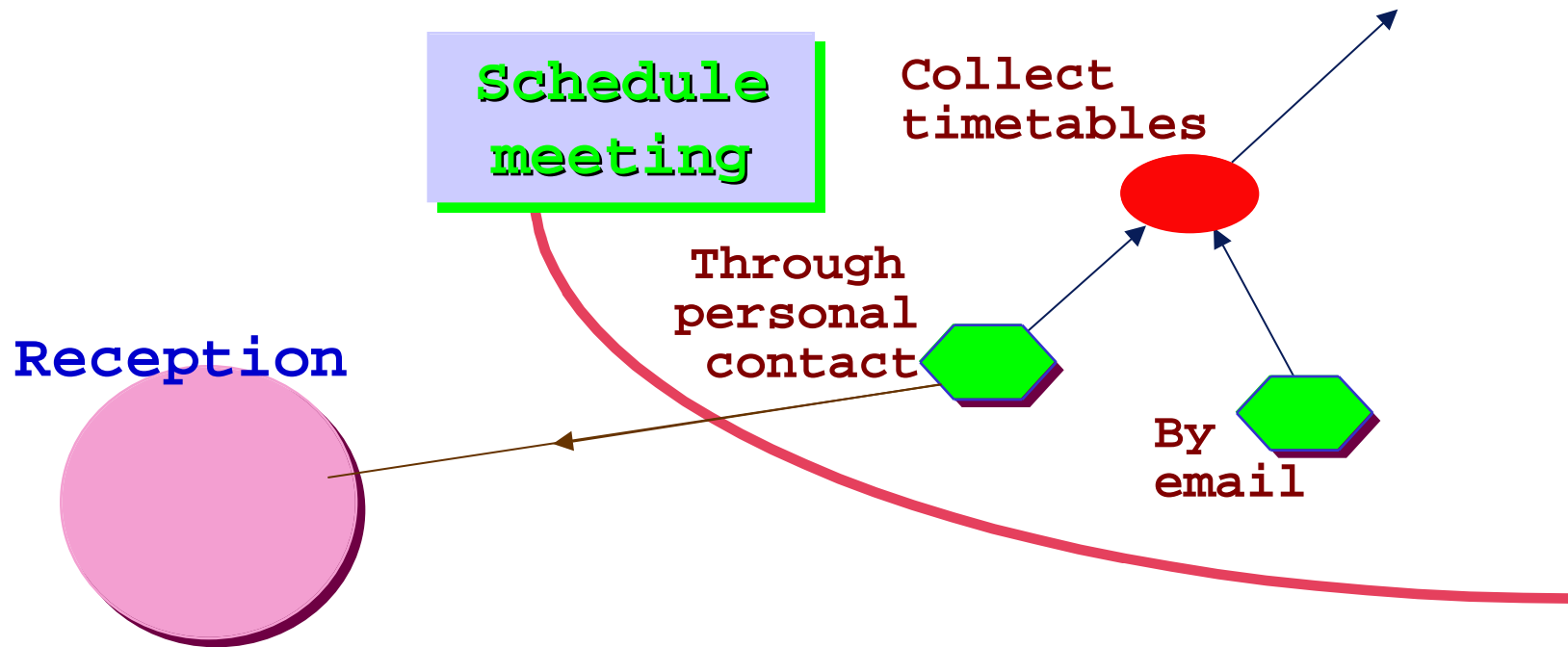
A social setting consists of actors, each having **goals** (and/or **softgoals**) to be fulfilled.



Goal Analysis

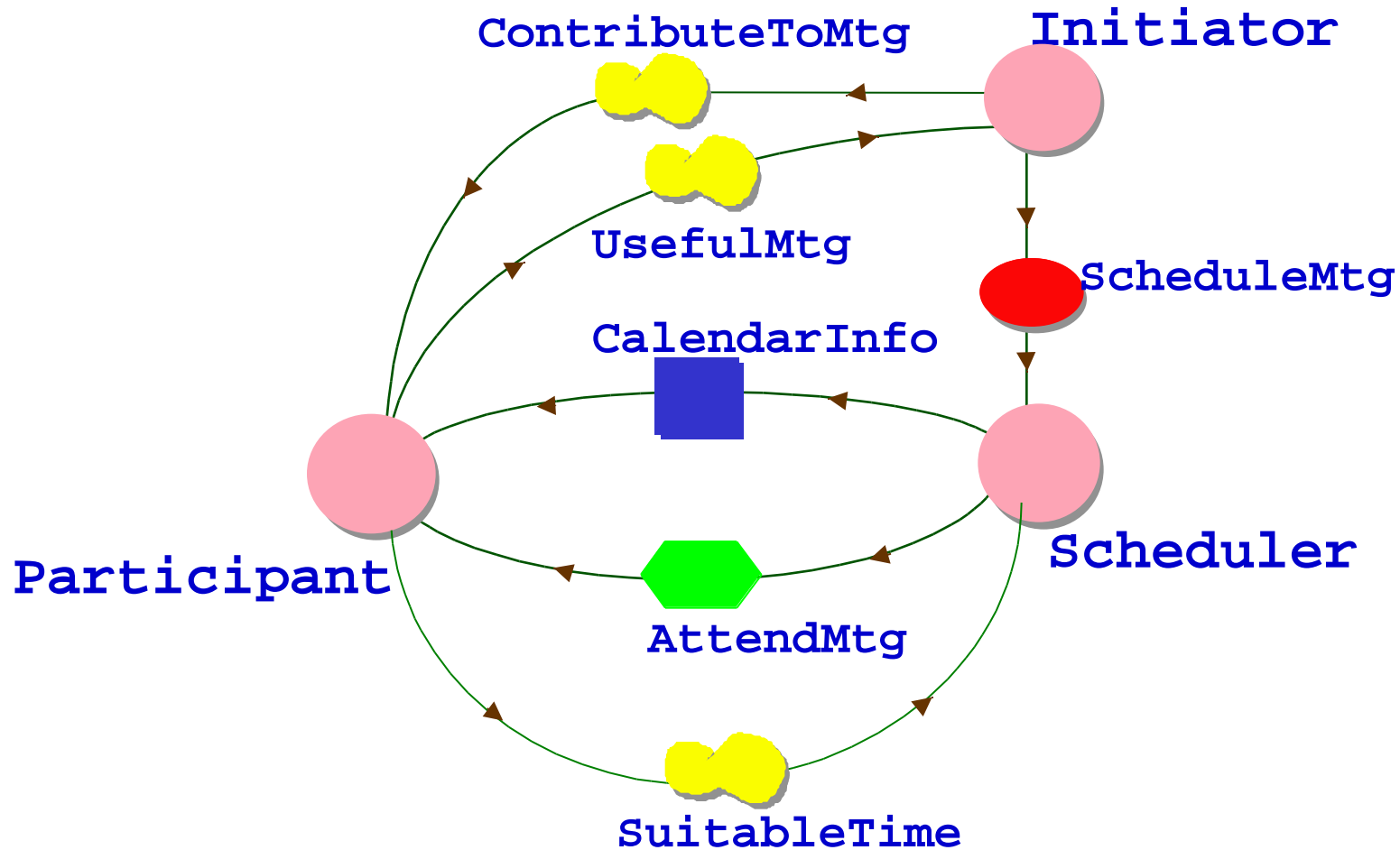


Actor Dependencies



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

Actor Dependency Models



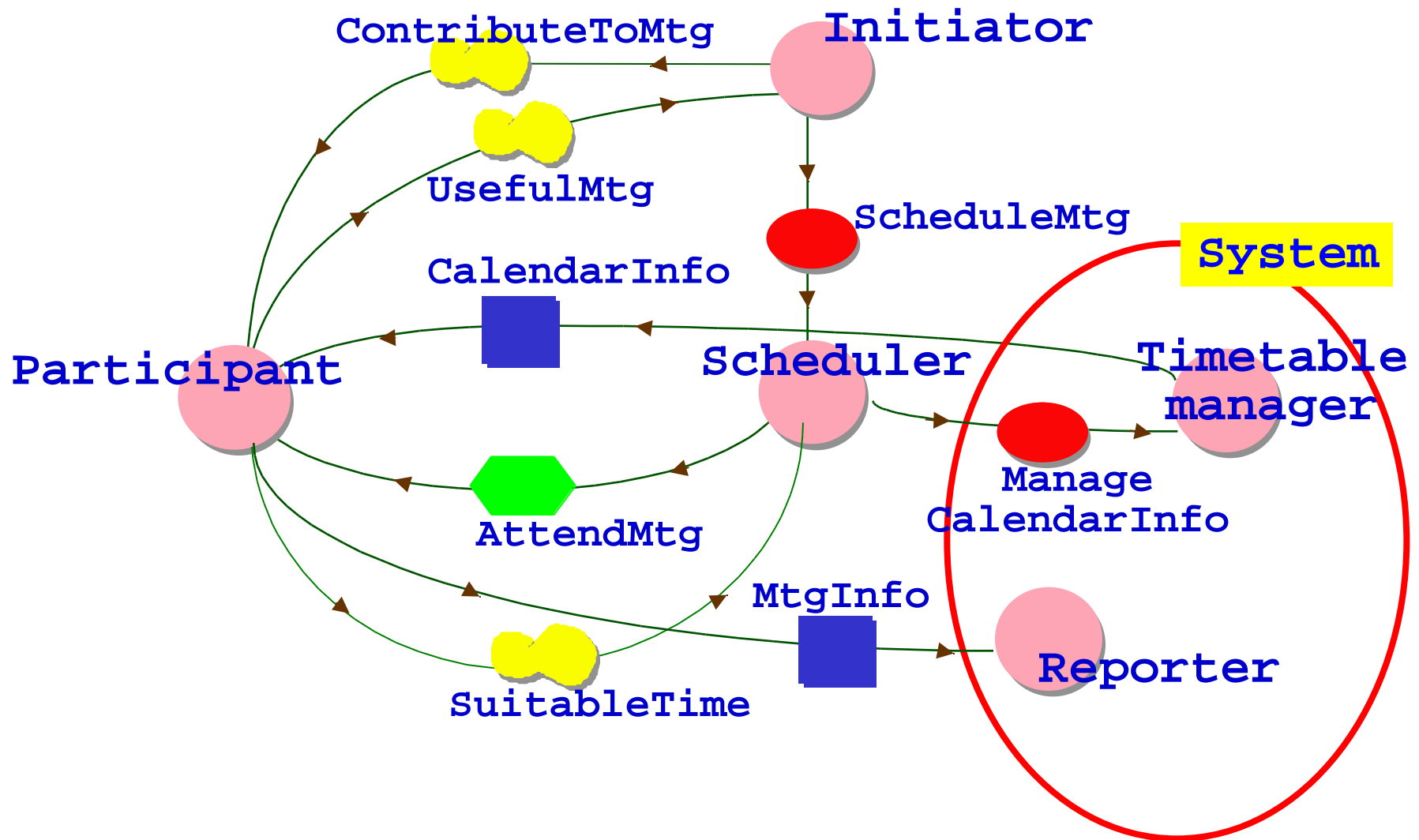
Tasks vs Goals

- Tasks are processes actors perform to fulfill goals.
- In general, there will be many alternative tasks (possibly by different actors) for fulfilling a goal.
- When actors are assigned goals, they are (or, ought to be) able to fulfil them by carrying out one or more tasks, and/or through delegation to other actors.
- A delegated goal may not be achievable by the dependee actor (who is supposed to achieve it); in this case, the depender actor has to look for an alternative solution.

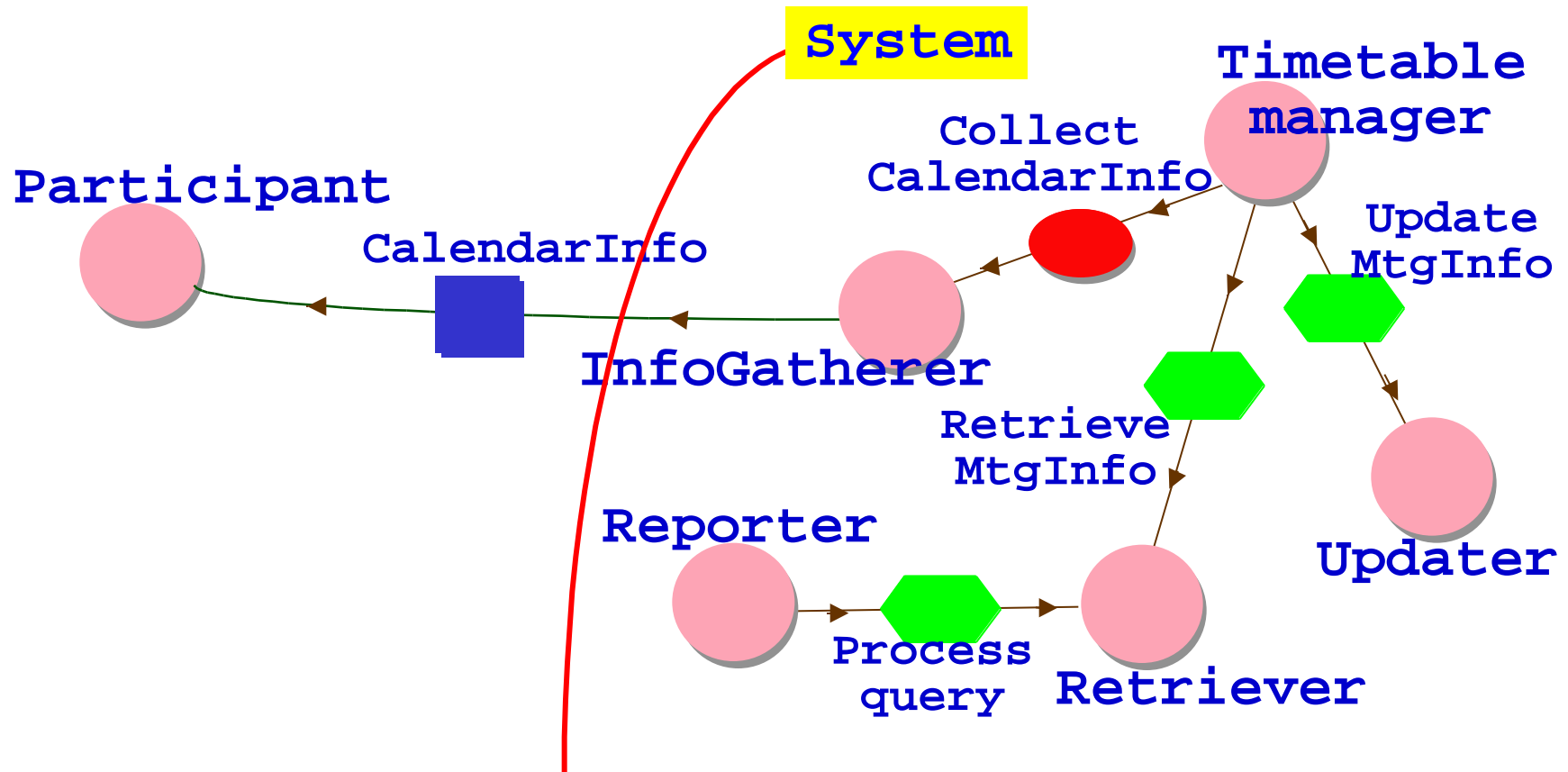
Using These Concepts

- 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**



What is Different?

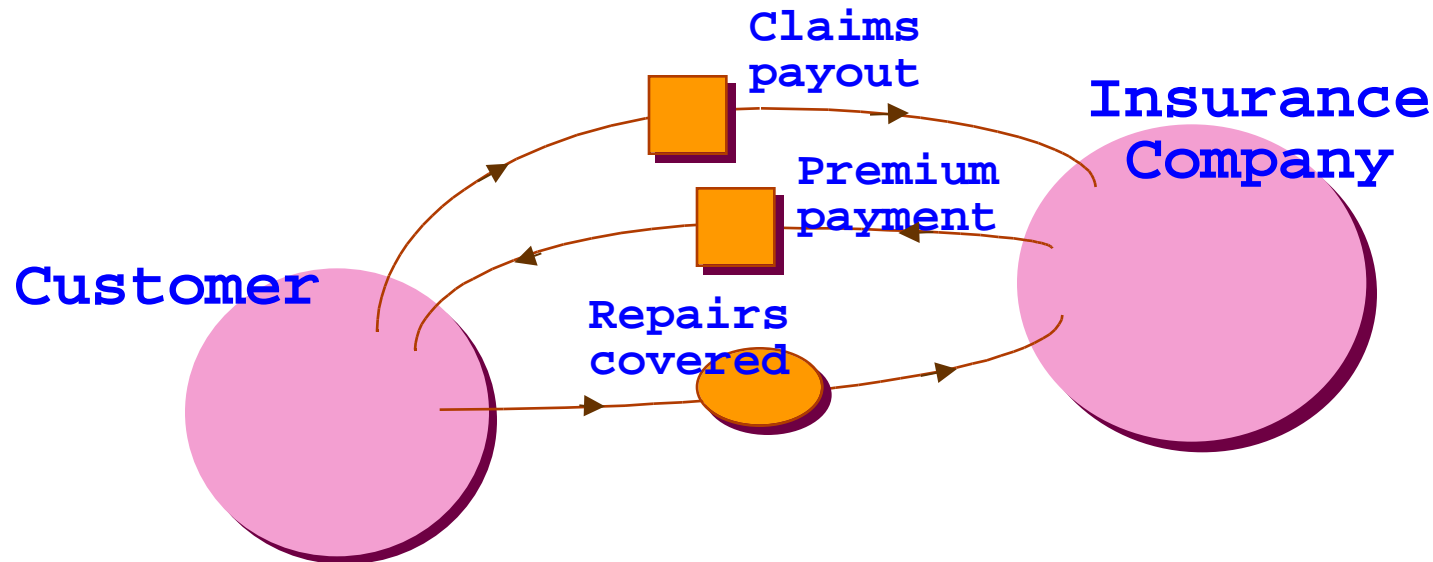
- Goal refinement extends functional decomposition techniques, in the sense that it explores alternatives.
- Actor dependency graphs extend object interaction diagrams in that a dependency is *intentional*, needs to be monitored, may be discarded, and can be established at design- or run-time.
- In general, 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.

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) ● \neg Claim(x)
 \neg RunsOK(x.insP.car))

end Claim

Action MakeRepair

Performed by BodyShop

Refines RepairCar

Input cl : Claim

Pre \neg 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 RepairsCovered

Analysing Models

- Models are used primarily for human communication
- But, this 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 configuraion of social dependencies.

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 properties of state representations of finite Tropos models.
 - ✓ Discovery of interesting scenarios that represent counterexamples to properties not satisfied by the specifications.
 - ✓ Model simulation.

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)

Goal Analysis

- Need to formalize the different types of goal relationships (AND, OR, +, -, etc.) and offer a (tractable) proof procedure.
- 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.

$g1, g2, g3 [AND(\{g1, g2\}, g3) \quad (\$(g1) \ S(g2)) \ S(g3)]$

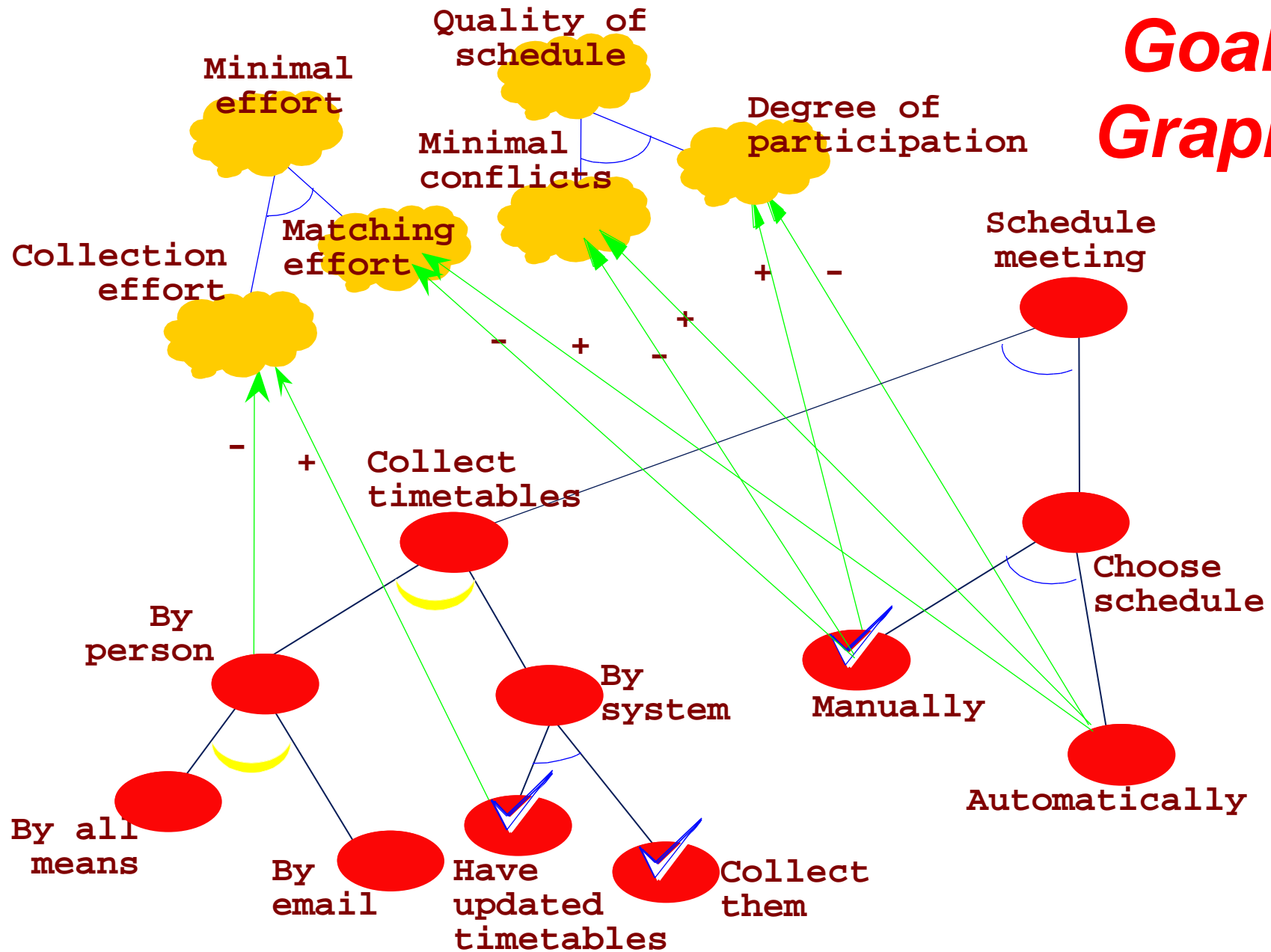
$g1, g2, g3 [OR(\{g1, g2\}, g3) \quad ((S(g1) \ S(g2)) \ S(g3))]$

$g1, g2 [++(g1, g2) \quad \$(g1) \ S(g2)]$

$g1, g2 [+ (g1, g2) \quad g [(g \ g2 \ S(g) \ S(g1)) \ S(g2)]]$

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

Goal Graph



Goal Analysis (cont'd)

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

$g1, g2, g3 [AND(\{g1, g2\}, g3) \quad (\$(g1) \ S(g2)) \ S(g3))]$

$\implies (S(\text{collectTbl}) \ S(\text{chooseSchl})) \ S(\text{scheduleMtg})$

- We are also given some S and D labels for some goals, e.g.,
 $S(\text{haveUpdatedTbl})$
- 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 are currently extending this algorithm to accommodate probabilities and criticalities for goals.

Tropos Proposal and UML

- Tropos offers a set of concepts and a methodology for developing agent-oriented software.
- These concepts can be accommodated within UML in terms of new stereotypes (see proceedings).
- One can also use UML/AUML diagrams and other techniques from the multi-agent system area to specify coordination and other details during detailed design (see proceedings).

Related Proposals

- Goal-Oriented Analysis (GOA) has been researched in Requirements Engineering since the '80s; unlike other proposals, *i** goals are ***relative*** and fulfillment is ***cooperative***; also other GOA frameworks focus on the transformation from early to late requirements.
- AUML is a recent proposal for agent modeling; its emphasis is on agent coordination+communication.
- UML has been recently extended to include goals for business modeling applications; *i** is a mature modeling framework, has been formalized in a variety of settings and comes with a methodology for doing actor and goal analysis.

Tropos

- Project started in April 2000.

<http://www.cs.toronto.edu/km/tropos>

- The team of participating researchers includes
 - ✓ UToronto (Canada): Fernandez Damian, Ariel Fuxman, Daniel Gross, Manuel Kolp, Linda Liu, Eric Yu;
 - ✓ UTrento/IRST (Italy): Paolo Bresciani, Paolo Giorgini, Fausto Giunchiglia, Eleonora Nicchiarelli, Anna Perini, Marco Pistore, Roberto Sebastiani, Paolo Traverso;
 - ✓ TUAachen (Germany): Matthias Jarke, Gerhard Lakemeyer.
 - ✓ FUPernambuco (Brazil): Jaelson Castro

Conclusions

- Agent-Oriented software development is an up-and-coming paradigm because of an ever-growing demand for generic, robust and open software systems.
- We have proposed a set of concepts and sketched a methodology which can support this paradigm.
- We have also sketched how such a proposal can be accommodated within UML and benefit from recent UML extensions.

...and a Postscript...

What is Software?

- An engineering artifact, designed, tested and deployed using engineering methods, which rely heavily on testing and inspection for validation (*Engineering perspective*)
- A mathematical abstraction, a theory, which 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 perspectives
will grow in importance
-- in practice, but also SE research!***

References

- [Bauer99] Bauer, B., *Extending UML for the Specification of Agent Interaction Protocols*. OMG document ad/99-12-03.
- [Dardenne93] Dardenne, A., van Lamsweerde, A. and Fickas, S., “Goal-directed Requirements Acquisition”, *Science of Computer Programming*, 20, 1993.
- [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. R., “On Agent-Based Software Engineering”, *Artificial Intelligence*, 117, 2000.
- [Odell00] Odell, J., Van Dyke Parunak, H. and Bernhard, B., “Representing Agent Interaction Protocols in UML”, *Proceedings of the First International Workshop on Agent-Oriented Software Engineering (AOSE-2000)*, Limerick, June 2000.
- [Yu95] Yu, E., *Modelling Strategic Relationships for Process Reengineering*, Ph.D. thesis, Department of Computer Science, University of Toronto, 1995.