# Lecture V
# Agent-Oriented Software Engineering

**John Mylopoulos**
**University of Ottawa**

**Federal University of Pernambuco (UFPE),**
**Recife, November 20, 2019**

# …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, which can be analyzed for consistency and can be refined into a more specialized theory (***Mathematical perspective***)
- 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***)

# 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, web services, pervasive and/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, negotiation, and exception handling become essential features for such software components.

☛ ***... agents!***

# Agent-oriented software engineering

- Many researchers have been working on it for ~20 years.
- Research on the topic :
  - 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].
- We proposed the Tropos methodology for building agent-oriented software; the methodology supports ***requirements analysis***, as well as ***design***.

# What is an agent?

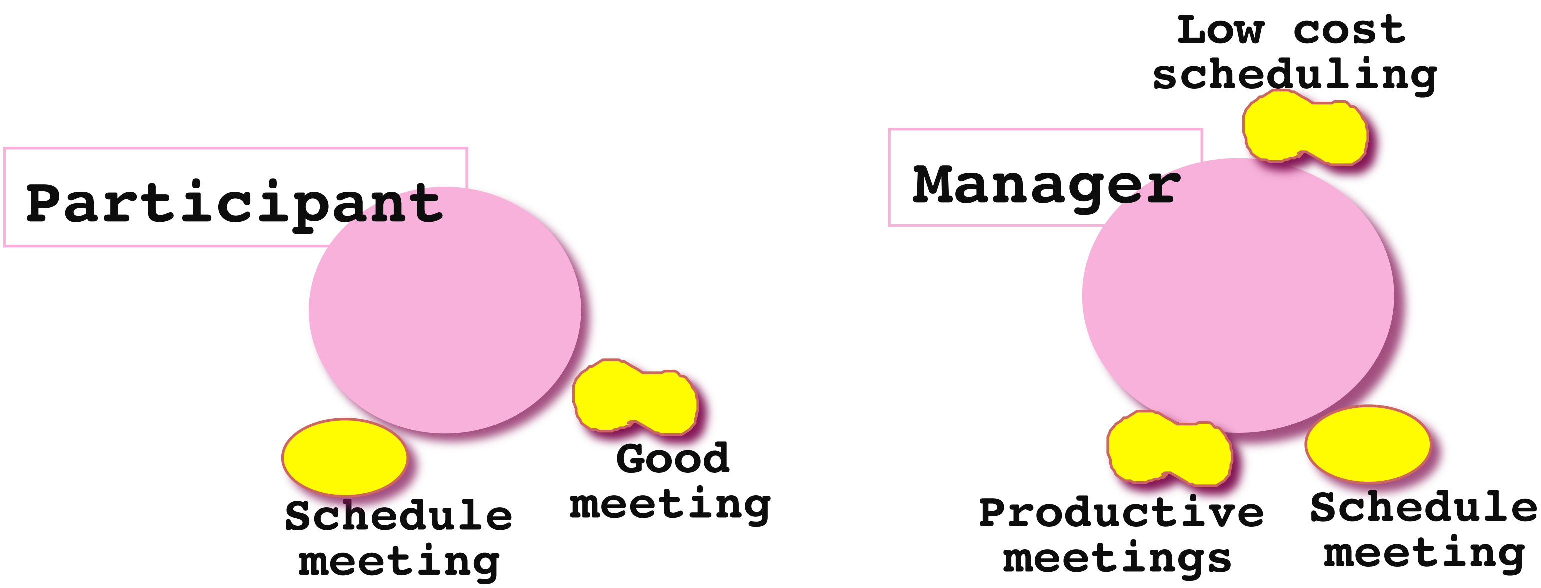
- A person, an organization, certain kinds of software.
- A ***software agent*** has ***beliefs***, goals (***desires***), ***intentions,*** hence they have a BDI architecture.
- Agents are situated, autonomous, flexible, and social.
- But note: human/organizational agents can't be ***prescribed***,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

# The Tropos Methodology

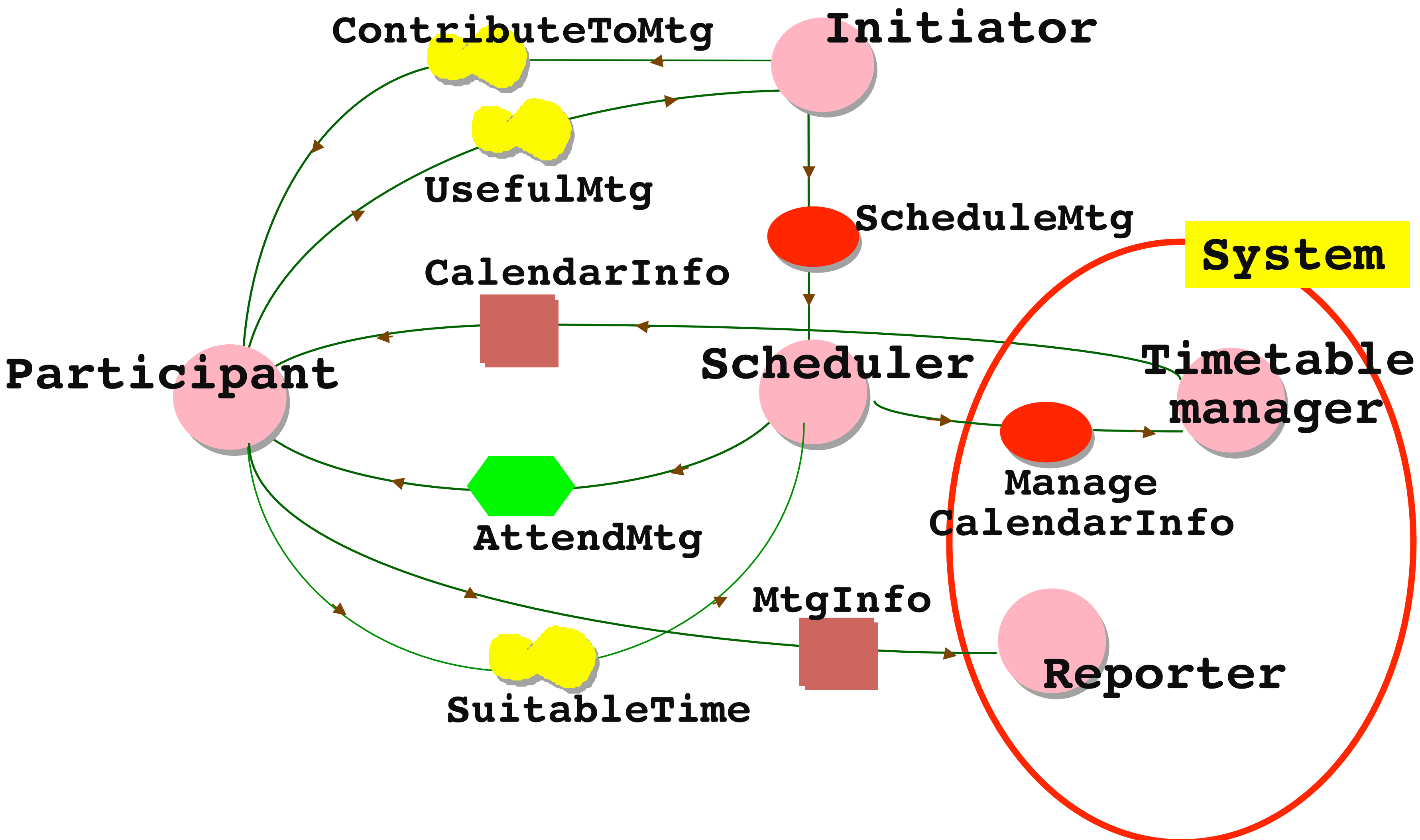
- We propose a set of primitive concepts, as in *i**, and a methodology for agent-oriented requirements analysis and design.
- We want to cover four phases of software development:
  - ✓ ***Early requirements*** -- identifies stakeholders and their goals;
  - ✓ ***Late requirements*** -- introduce system 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 req'nts: Stakeholder and their goals

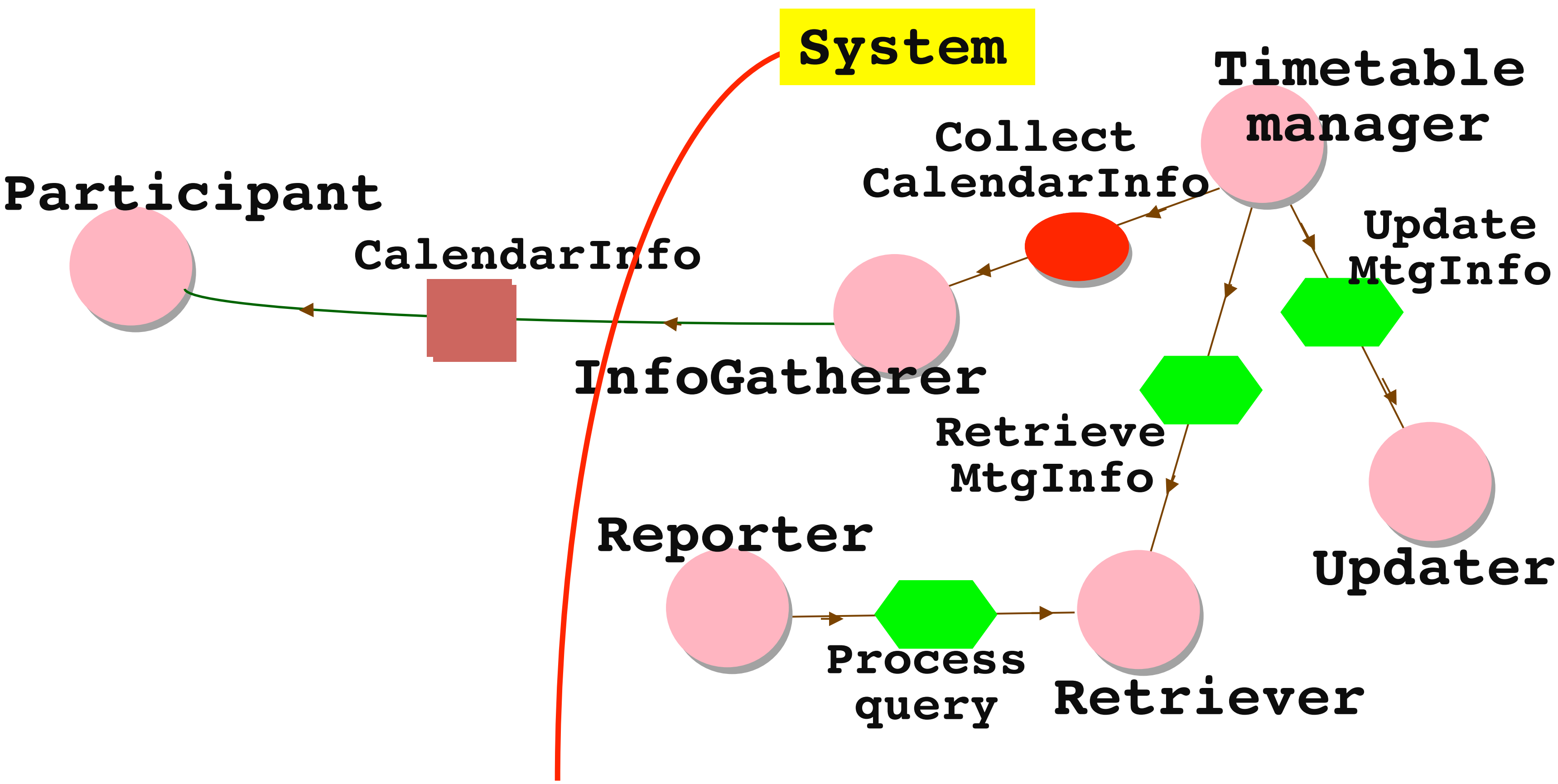
A social setting consists of actors, each having ***goals*** (and/or ***softgoals***) to be fulfilled.



# Late Requirements with i*



# Software architectures with i*



# The Tropos development process

- **Initialization**: Identify stakeholder actors and their goals, place them in ***S*** and ***G*** respectively;
- **Step**: For each goal g in ***G*** wanted by a in ***S***:
  - ✓ Actor a adopts g;
  - ✓ Actor a delegates g to an existing actor in ***S***;
  - ✓ Actor a delegates it to a new actor a'; a' is added to ***S***;
  - ✓ Refine g into new subgoals g1, …, gn; add these to ***G***;
  - ✓ Declare goal g "denied".
- **Termination condition**: All initial goals have been fulfilled, assuming all actors deliver on their commitments.

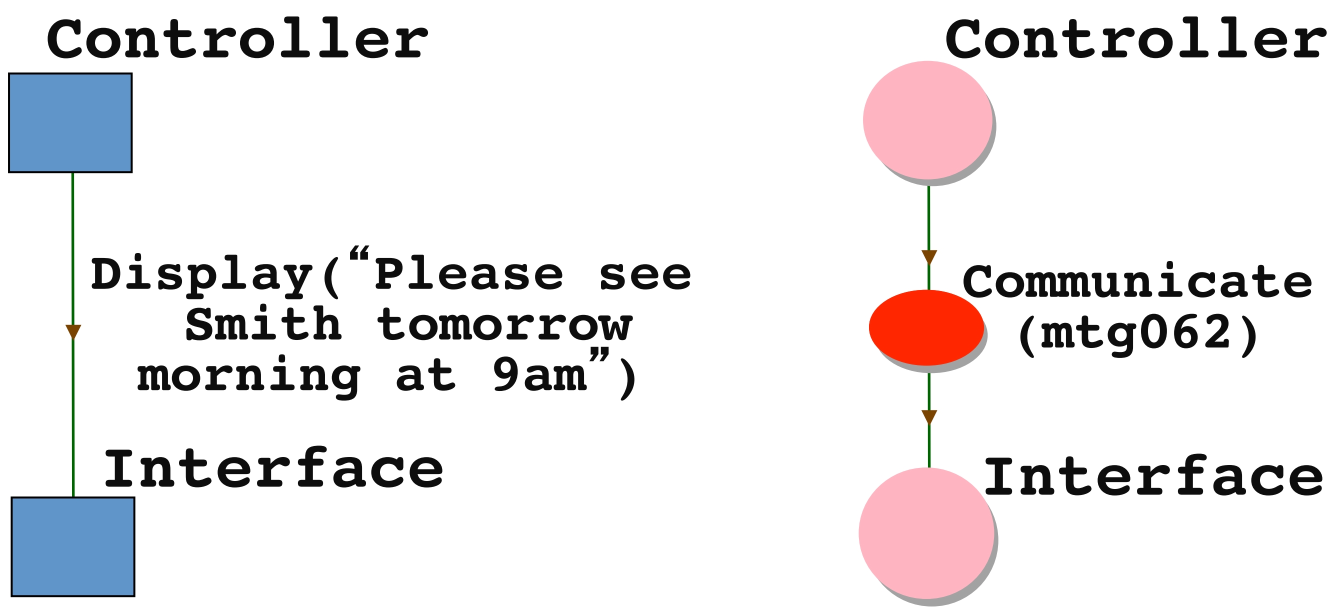
# Tropos compared to OO techniques

- 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 …)

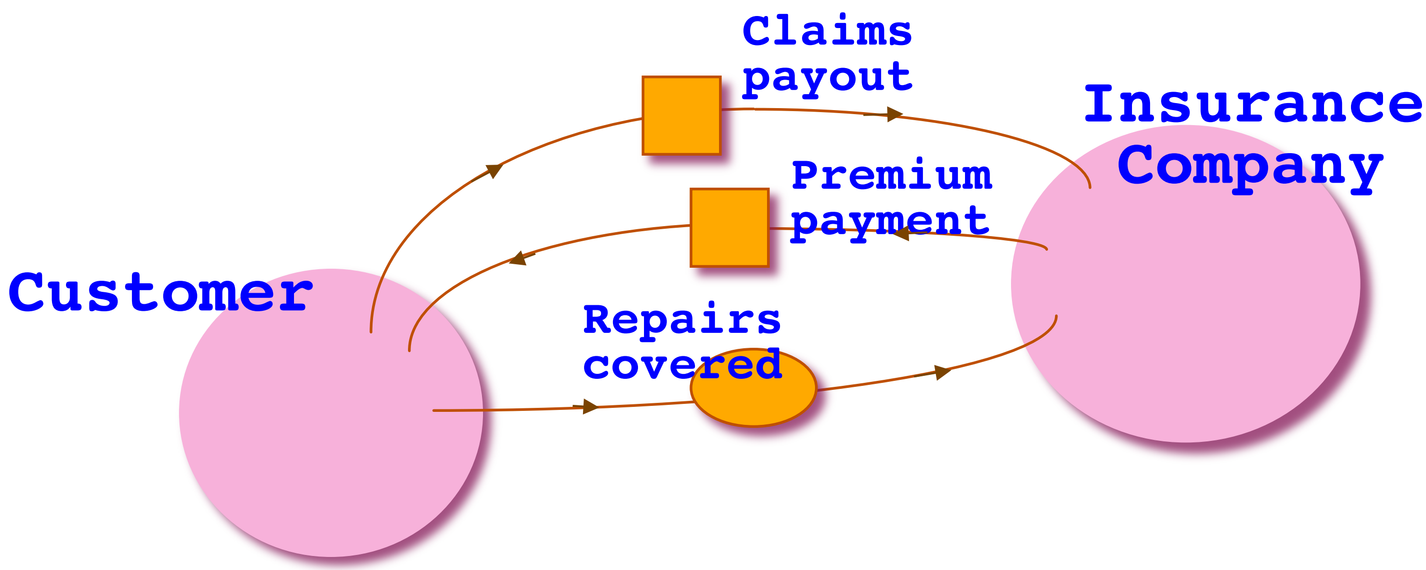
- Traditionally, goals (and softgoals) are operationalized and/or metricized before late requirements.
- This means that a solution to a goal is frozen into a software design early on and the designer has to work within the confines of that solution.
- This won't do in situations where the operational environment of a system, including its stakeholders, keeps changing.
- 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 leading to ever-changing requirements.

# The tale of two designs



# 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** $(\forall x)(\mathrm{Claim}(x) \land \bullet\neg \mathrm{Claim}(x) \Longrightarrow \neg \mathrm{RunsOK}(x.\mathrm{insP.car}))$
**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** Fulfill

**Depender** Customer

**Dependee** InsuranceCo

**Has** cl: Claim

**Defined** /* the amount paid out by the insurance company covers repair costs */

**end** CoverRepairs

# Analysing Tropos models

- Models in SE are used for analysis 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 configuration of social dependencies.

# Model checking for Tropos

- Define 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.

# Mapping Tropos to nuSMV

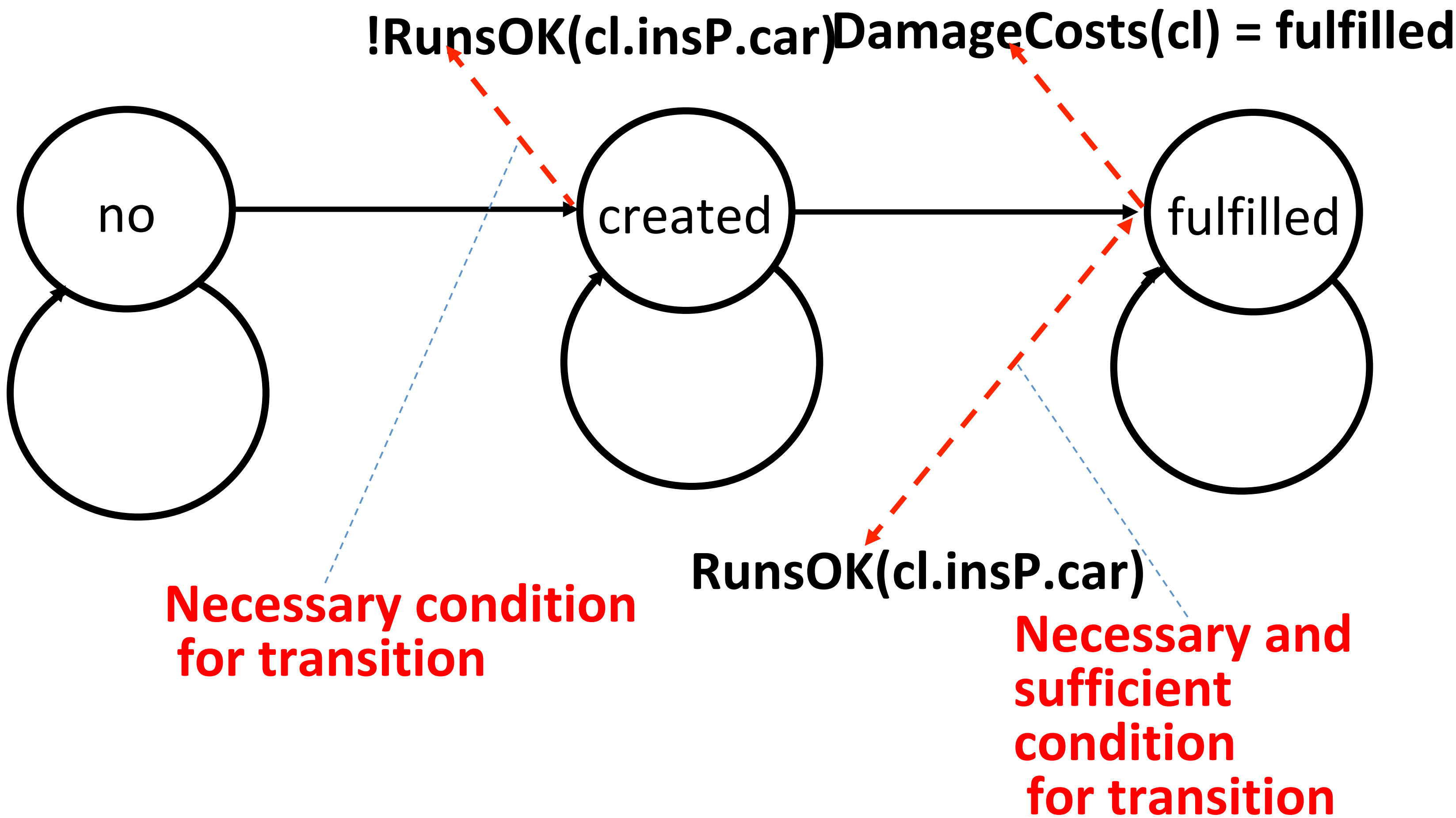
- The language supported by a model checker includes variables that can take one of a finite number of values. Also, constraints on the allowable transitions from one value to another.
- How do we map Formal Tropos to nuSMV?
  - ✓ Each goal instance is represented by a variable that can take values “no”, “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) $\Rightarrow$ ❍$\neg$RunsOK(self)

…

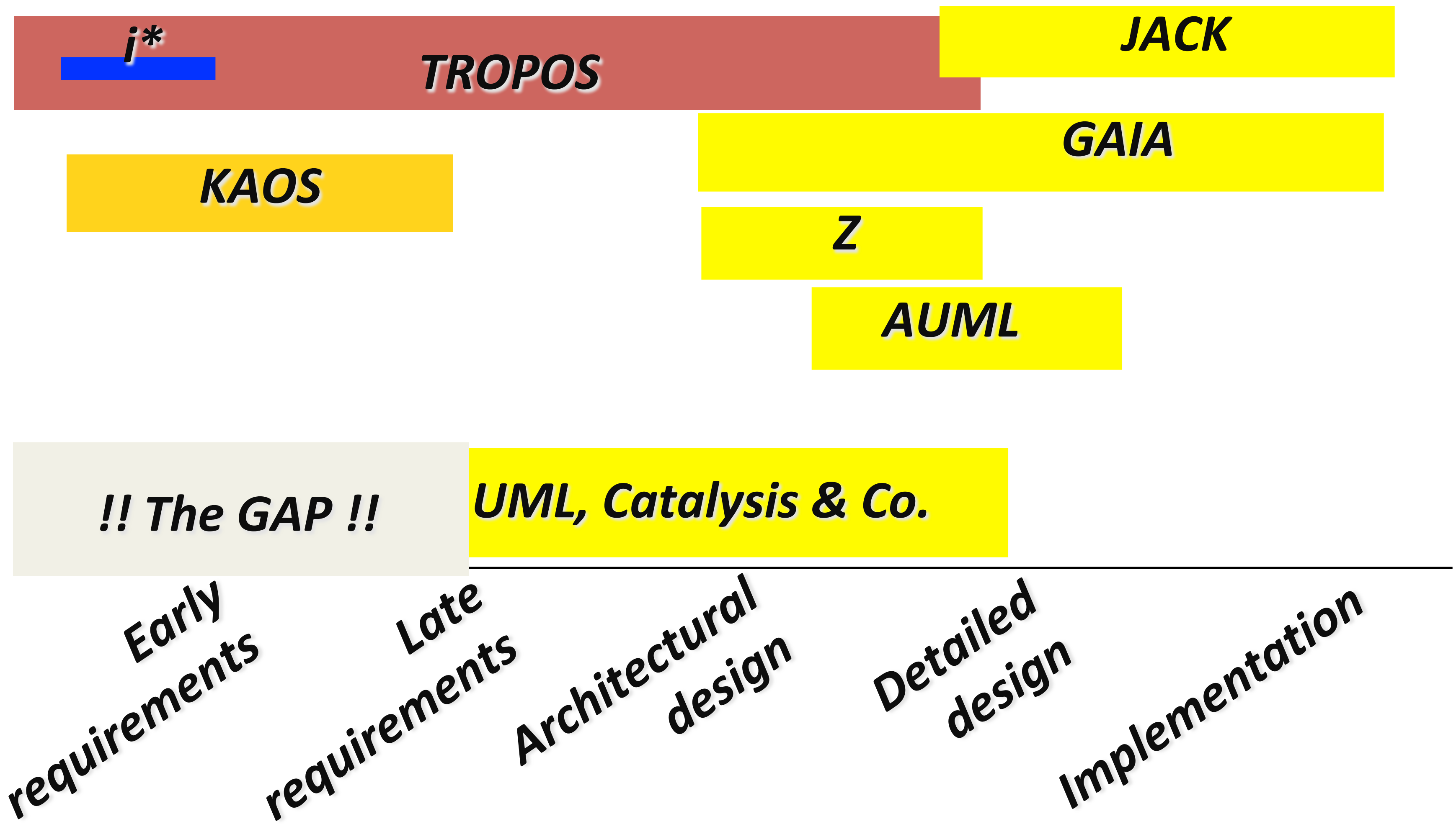
# Experiments with the T-Tool

- Tropos models can have an unbounded number of instances; to make model checking work, we pick increasingly larger models (e.g., 1, 2,... instances per class) and check whether a property we want to prove leads to counter-examples.
- How do we pick models? How do we know when to stop?
- Experiments to demonstrate the scalability of the approach.

# Other threads of research

- **[Security]** Extend Tropos to support ‘ownership’, ‘permission’ and ‘trust’; this leads to models where you can check that every actor has the permissions she needs to carry out her obligations [Zannone05] ➔ PhD thesis by Nicola Zannone (Trento, 2007).
- **[Risk Management]** Extend the risk management framework [Feather05] to allow goal-based risk analysis ➔ PhD thesis by Yudis Asnar (Trento, 2009).

# Related work



# Conclusions

- We have proposed a set of concepts and sketched a methodology that together support Agent-Oriented Software Development.
- Agent-Oriented Software Development has been an up-and-coming paradigm for more than 20 years, thanks to the rise and ever-growing demand for social software.
- This is a long-term project, and much remains to be done.

## 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.
- [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.

# References (cont'd)


- [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.