Agent-Oriented Software Engineering and Tropos

Agent-Oriented Software Engineering
Early Requirements and i*
Modeling with i*
Formal Tropos
Analyzing Formal Tropos Models
The Tropos Project

...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***)

...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 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 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].*
- *We are developing a methodology for building agent-oriented software which supports **requirements analysis**, as well as **design**.*

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, --> 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 eventually either lead to software requirements, or are delegated to external agents.

The Tropos Methodology

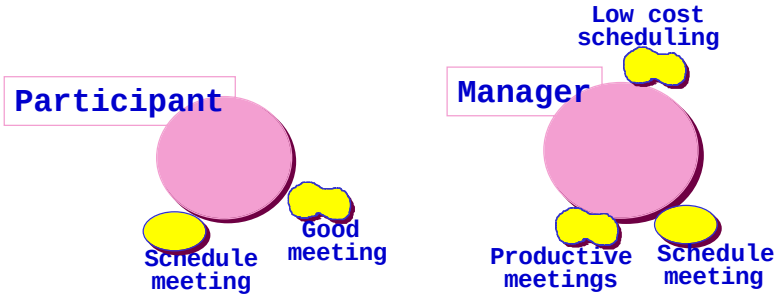
- We propose a set of primitive concepts and a methodology for agent-oriented requirements analysis and design.
- We want to cover four phases of software development:
  - ✓ ***Early requirements*** -- identifies stakeholders and their goals;
  - ✓ ***Late requirements*** -- introduce system as another 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 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].
- Actors = Agents ∪ Positions ∪ Roles.

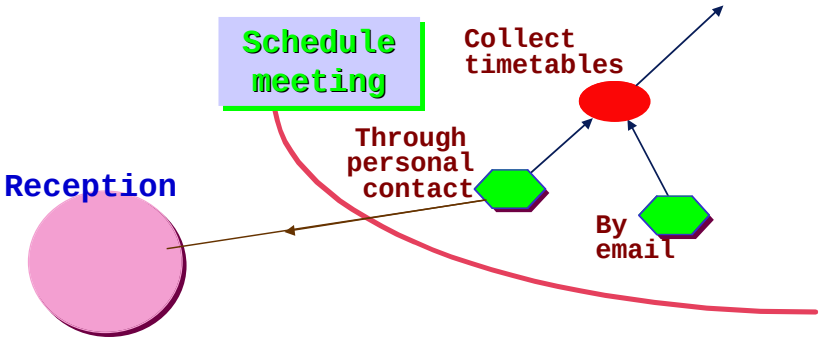
Early Requirements: External Actors and their Goals

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



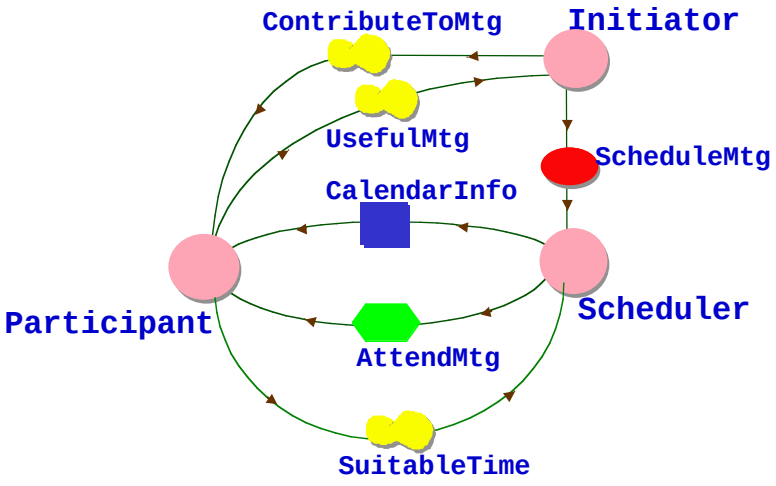
Goal Analysis

Actor Dependencies



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

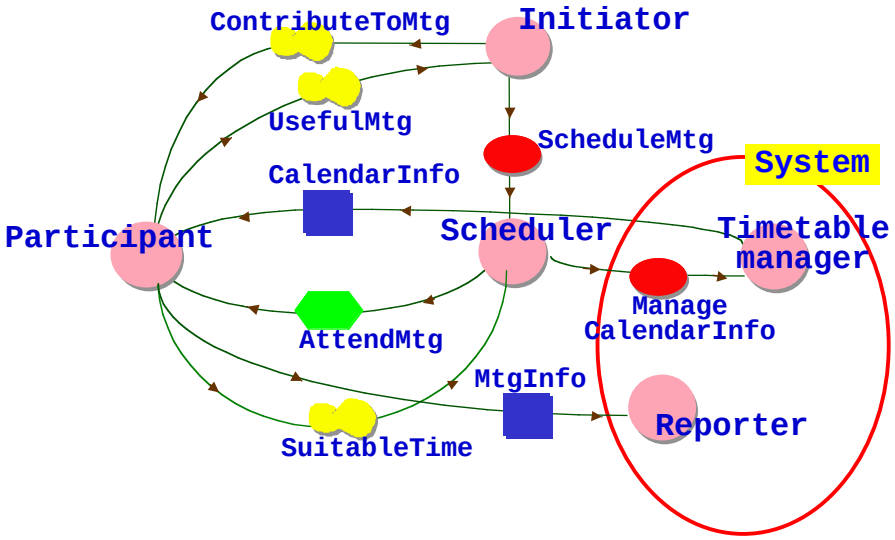
Actor Dependency Models



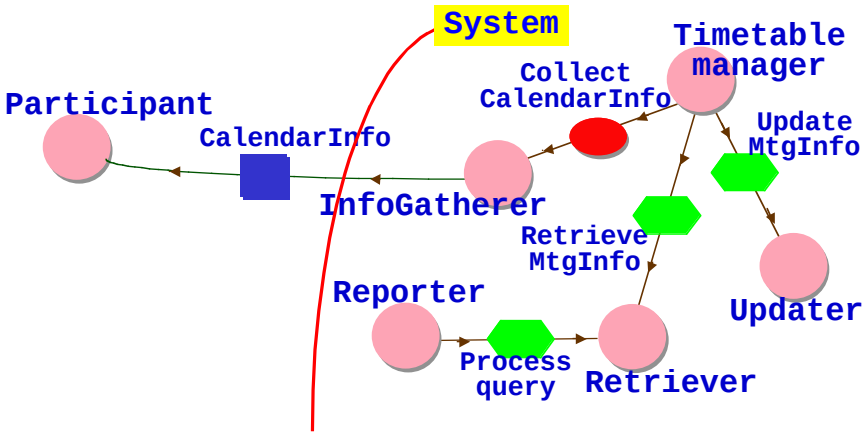
Using These Concepts

- 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.*

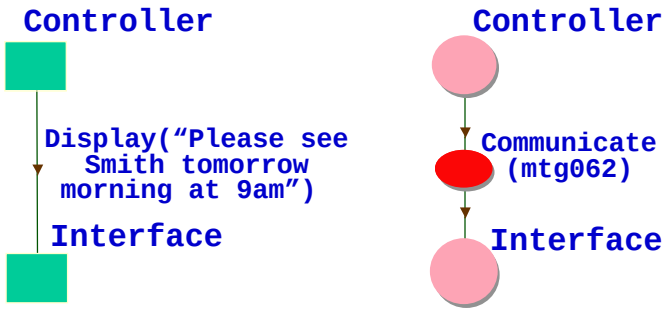
What is Different?

- Goal refinement extends functional decomposition techniques, in the sense that it explores alternatives.
- Actor dependency graphs extend object interaction diagrams in that 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…)

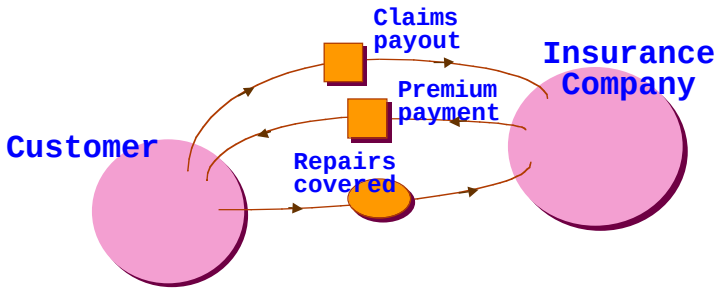
- Traditionally, goals (and softgoals) are operationalized and/or metricized before late requirements.
- This means that a solution to a goal is frozen into a software design early on and the designer has to work within the confines of that solution.
- This won't do in situations where the operational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

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 (∀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
```

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 configuration 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.

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)
```

An Interesting Property

LTLSPEC F(CoverRepairs = fulfilled) -> F(MakeRepair)

"If sometime in the future CoverRepairs is fulfilled, then sometime in the future MakeRepairs is carried out too"

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

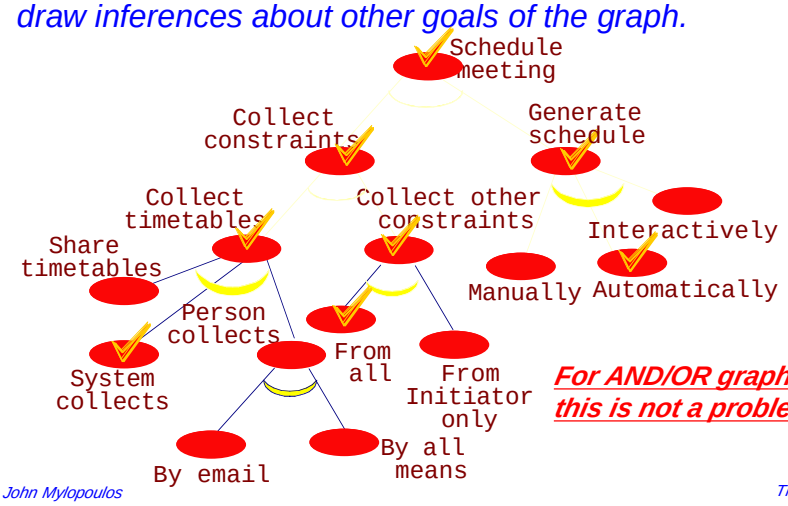
Variable	t_1	t_2	t_3	t_4
RunsOK	false	false	true	true
DamageCosts	no	no	created	fulfilled
CoverRepairs	no	created	created	fulfilled
MakeRepair	false	false	false	false

On-Going Work

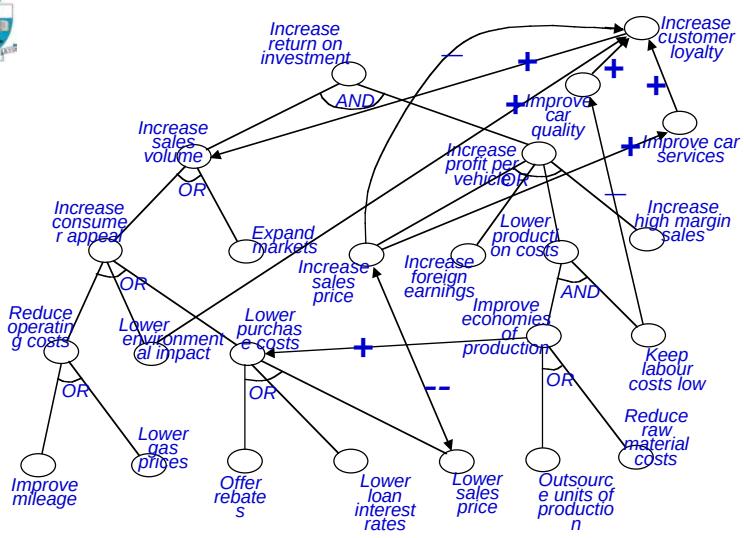
- *Tropos models are infinite; to make model checking work, we pick finite submodels (e.g., 1, 2,... instances per class, for any one property we want to prove and see if it leads to counter-examples.*
- *How do we pick submodels? How do we know when to stop?*
- *Experiments to demonstrate the scalability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

Goal Analysis

- *Suppose we are given a goal graph, and we are told that some of its goals are satisfied/denied. We would like 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

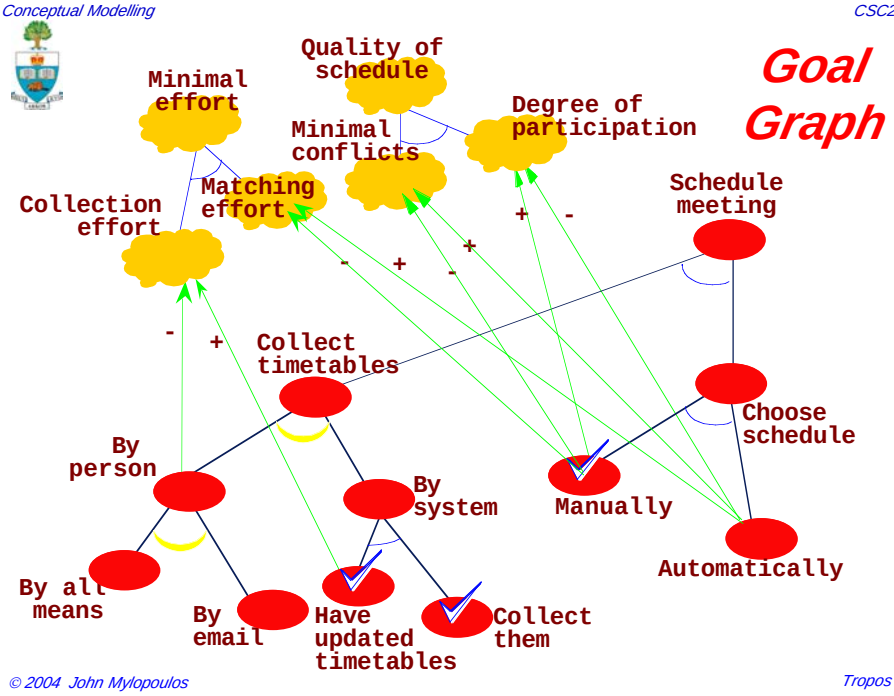
- *It is not clear that this algorithm terminates; it is possible that the label of a node will keep changing after each step of the label propagation algorithm.*
- *The label combination rules seem ad hoc and with no justification (other than a feeble "they seem intuitive").*
- *It is unclear what goal 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 --, -...



Qualitative Goal Analysis

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

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

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

- *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))]`

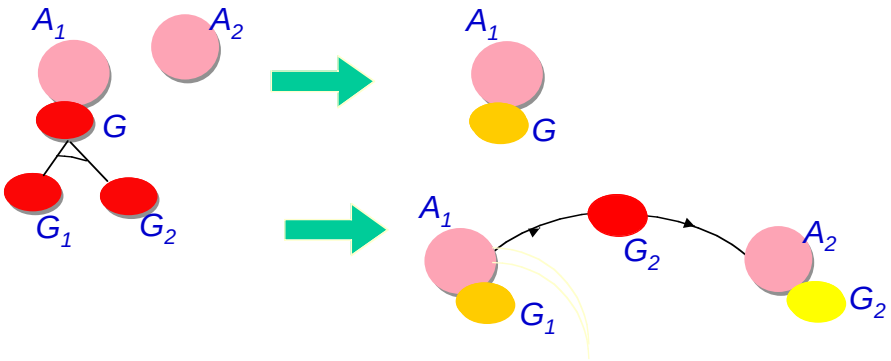
$\forall$`g1,g2,g3[OR({g1,g2},g3)` $\Rightarrow$
`((S(g1,p1)`$\wedge$`S(g2,p2))`$\Rightarrow$ `S(g3,p1`$\oplus$`p2))]`

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

- *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 embelish this axiomatization in order to account for amount of evidence, multiple sources of evidence,...*

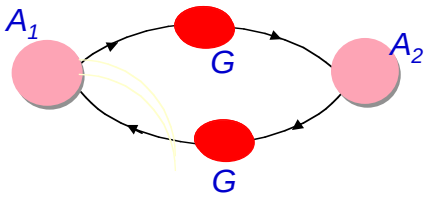
Dependency Graph Analysis

- 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

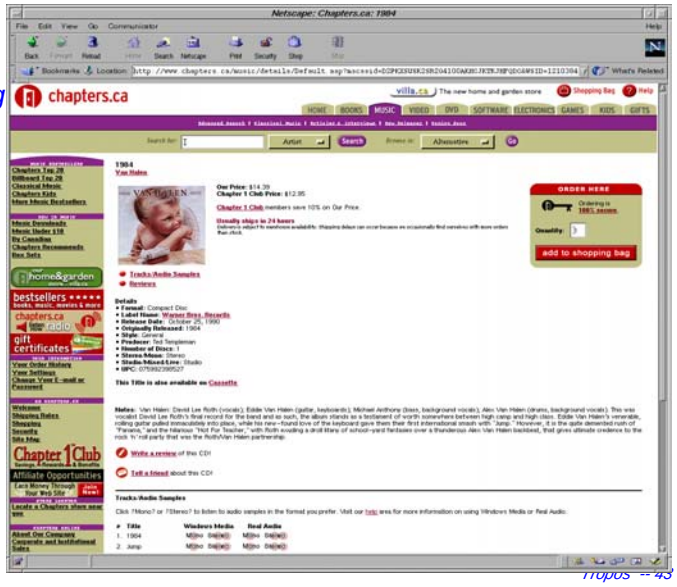
- Some dependency graphs don't make sense...



- What is a "good" dependency graph assuming that we are interested in:
  - ✓ minimizing dependence;
  - ✓ distributing work.
- Algorithms for transforming dependency graphs.

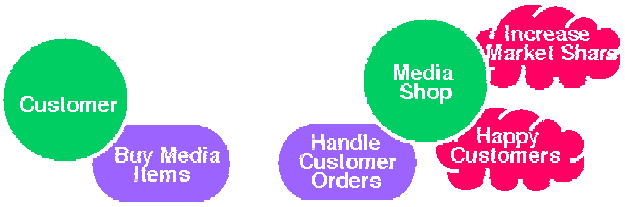
The Media Shop Example

- 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



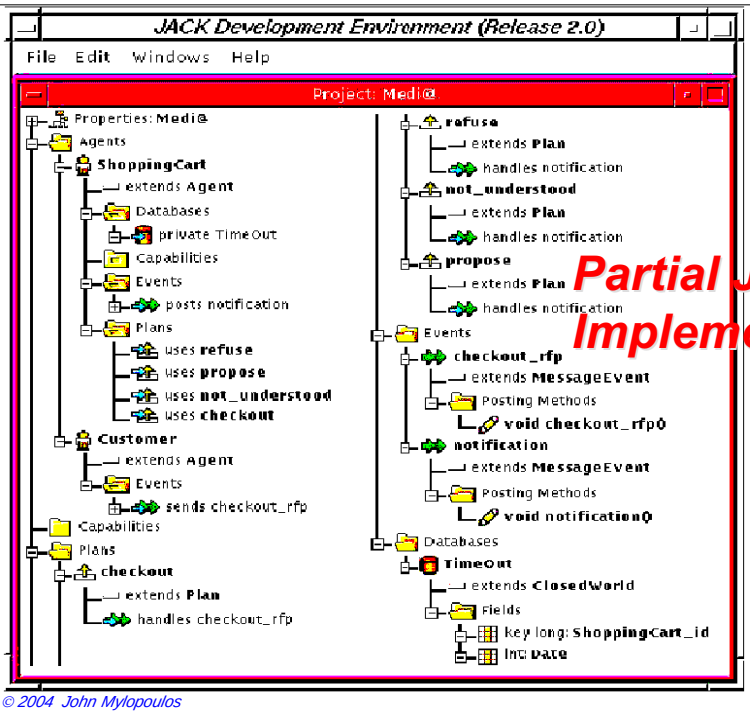
Early Requirements Analysis

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



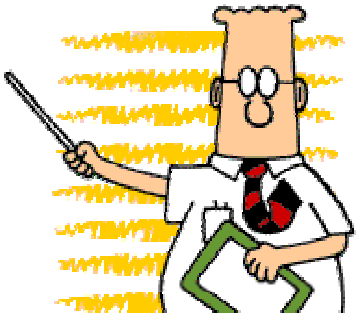
Goals are relative, fulfillment is collaborative

...Missing Slides...

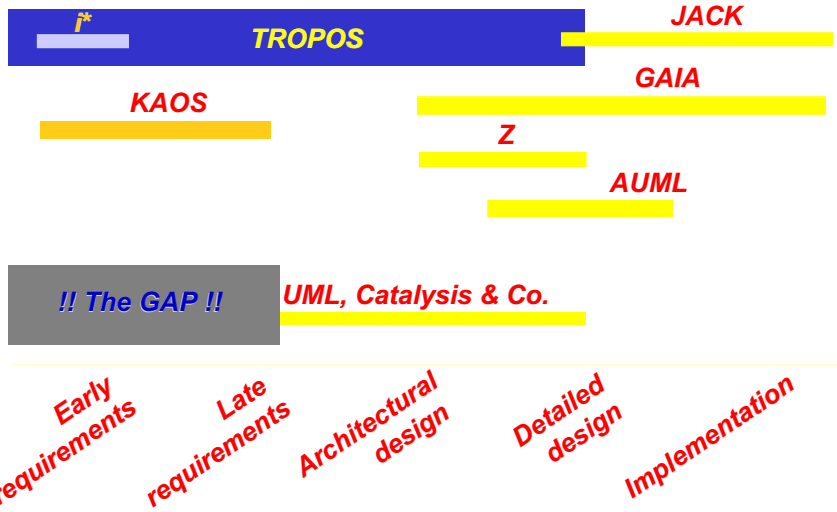


Partial JACK Implementation

Beware!!!
When we design software organizational systems, we must avoid the pitfalls of human organizational ones…



Related Work



Tropos

- Project started in April 2000.

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

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

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.

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.

