Towards an Agent-Oriented Software Development Methodology

John Mylopoulos
University of Toronto

University of Trento,
June 15, 2000

Abstract

Software Development has traditionally been implementation-driven in the sense that the programming paradigm of the day (structured programming, object-oriented programming) dictated the design and requirements analysis techniques widely used (structured analysis and design, object-oriented analysis and design respectively).

We speculate on what a software development methodology might look like if it was founded on early requirements analysis concepts and techniques. For our purposes, we adopt *i** [Yu94] as modeling framework. *i** supports concepts such as those of actor, agent, position and role, also resource, task and goal dependencies among actors. The presentation suggests elements of late requirements analysis, architectural and detailed design through examples, and notes a number of areas where such a methodology might break new ground with respect to traditional software development techniques, as well as agent-oriented programming.

Preamble

- We have conducted research on Conceptual Modelling for 25 years. Much of that research was carried out within the context of two long-term projects:
  - Taxis [Mylopoulos80] -- a design language for information systems;
  - Telos [Mylopoulos90] -- a language for modelling requirements, design, implementation information, and more.
- This research was conducted by many colleagues, including Alex Borgida, Matthias Jarke, Yannis Vassiliou, Sol Greenspan, Phil Bernstein and others.
- We are embarking on a third long-term project on Conceptual Modelling, Tropos. This presentation offers the motivation and the objectives of the Tropos project.

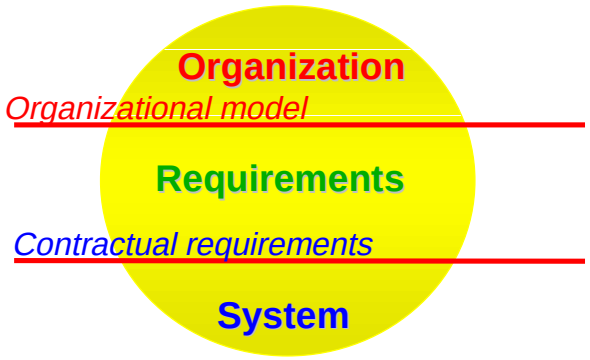
Software Development Techniques

- Software development techniques offer concepts, tools and methods for building software systems.
- Traditionally, such techniques have been implementation-driven.
- This means that the programming paradigm of the day dictated the design and requirements paradigms.
- So, structured programming led to structured design and structured (requirements) analysis, while object-oriented programming led to object-oriented design and analysis.
- Aligning the paradigms used for requirements, design and implementation makes perfect sense. But why start with an implementation paradigm?

What would requirements-driven software development look like??

Early vs Late Requirements

- We need to distinguish between an early phase of requirements analysis, when the analyst is trying to understand an organizational setting, from late phases when the analyst formulates a solution



Early vs Late Requirements

- Early requirements amount to the definition of a search space ("scoping") and a search among alternatives within that space.
- Late requirements amount to refining, disambiguating and completing the description of the chosen alternative.
- **Structured** and **object-oriented analyses** are OK for late requirements.
- **Goal-oriented analysis** is more appropriate for early requirements analysis because it focuses on the definition and exploration of a space of alternatives

Goal-Oriented Analysis

- Goal-oriented analysis focuses on early requirements phases, when alternatives are being explored and evaluated.
- During goal-oriented analysis, we start with goals such as
  - "Higher profits",
  - "Faster time-to-market",
  - "Schedule meeting",
  - "Easily maintainable system",
  - "Good performance" etc.

  and keep decomposing them until we have reduced them to alternative collections of design decisions each of which can satisfy the initial goals.
- Initial goals may be organization- or system-oriented; they may also be conflicting, so the analysis must facilitate the discovery of tradeoffs and the search of the full space of alternatives, rather than a subset.

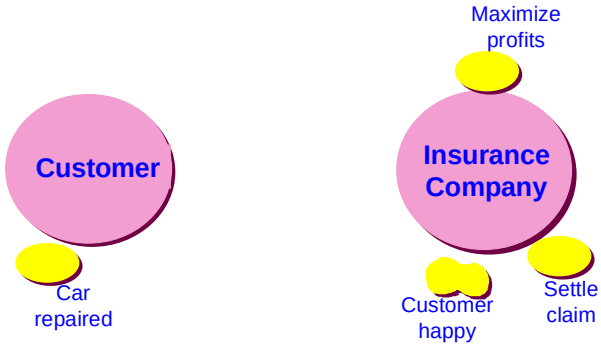
Goal-Oriented Analysis is not New!

- Specification of composite systems -- [Feather87]
- Goal-oriented elaboration of requirements -- ALBERT [Dubois94]
- Goal-oriented requirements acquisition -- KAOS [Dardenne93]
- Knowledge representation and reasoning in the design of composite systems -- Critter [Fickas92]
- Goal-oriented requirements analysis -- Potts, Anton
- *i** and the Non-Functional Requirements framework -- Yu, Chung
- NATURE -- [Jarke93]
- F3 -- [Bubenko93]

...and many others…

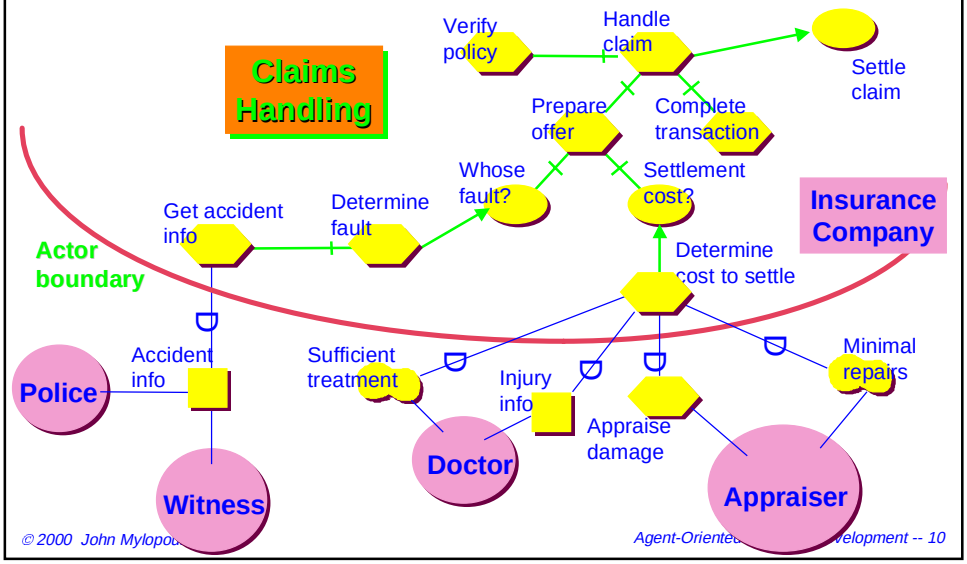
We'll work with i*

The i* Framework

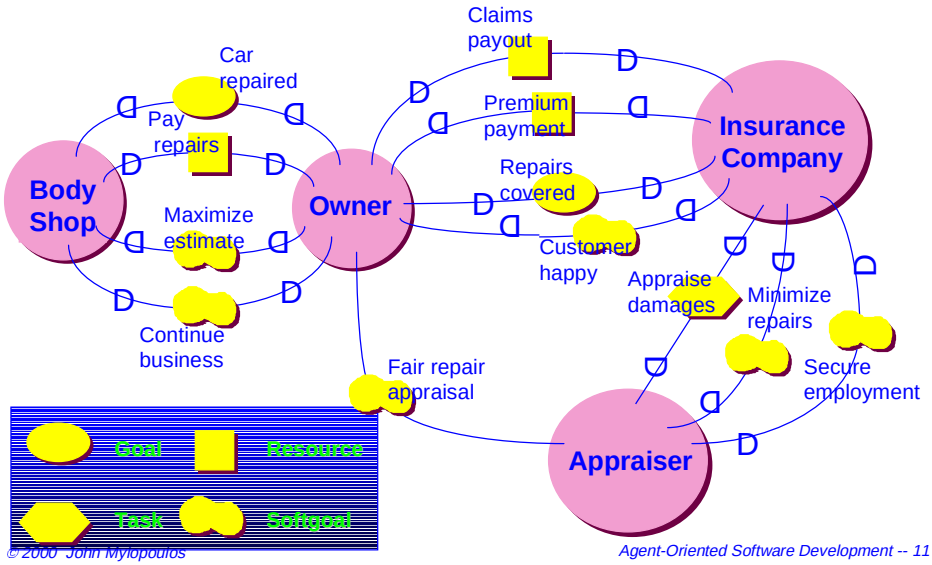


Goals are relative, fulfillment is collaborative

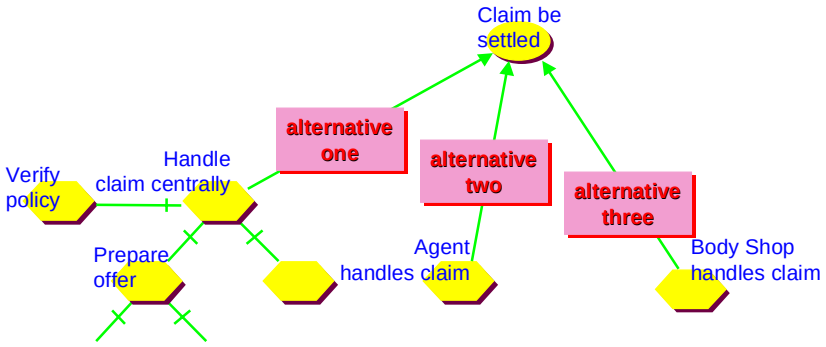
Means-Ends Analysis



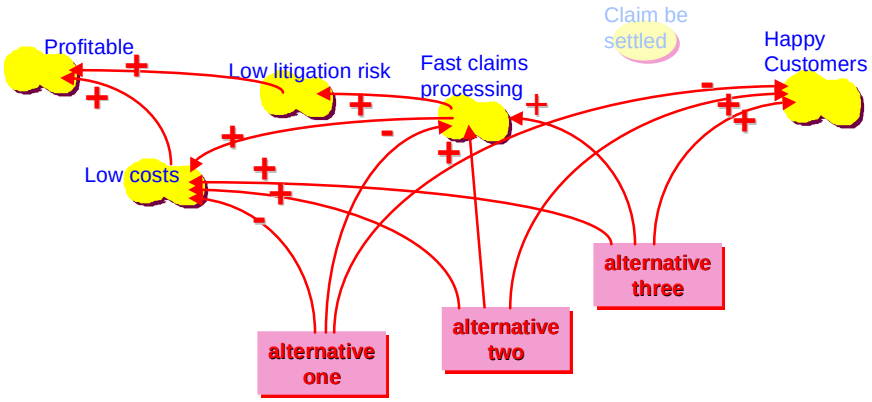
Strategic Dependency Models



Functional Alternatives

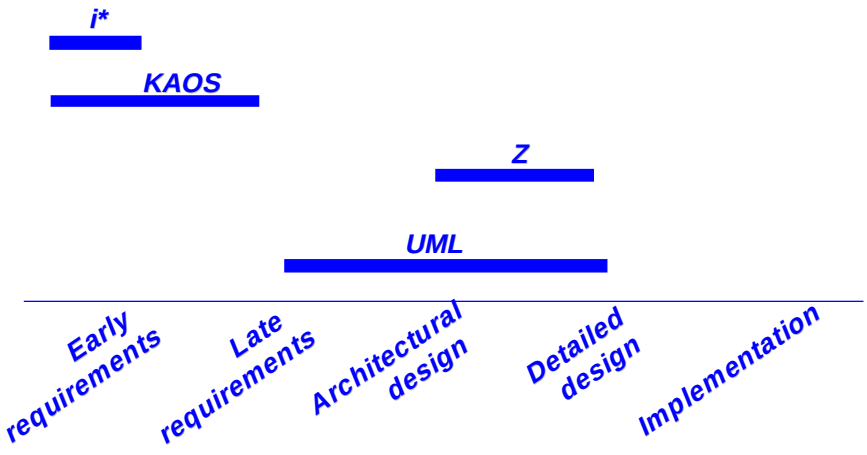


Non-Functional Rationale for Choosing Among Alternatives

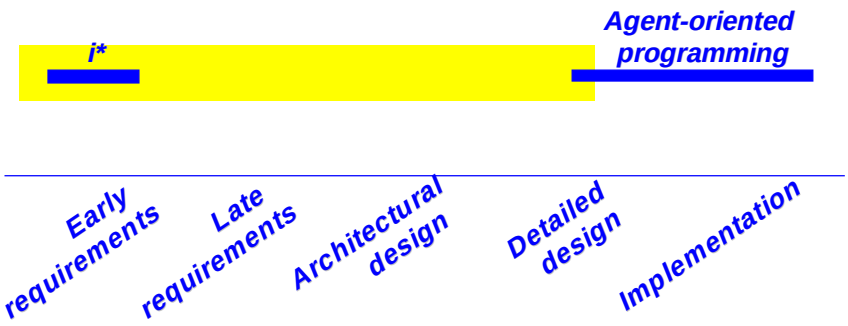


[Chung93]

Where Are We??



Where Do We Want To Be??

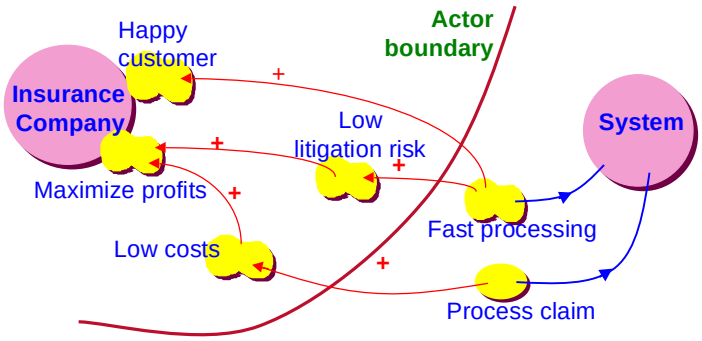


Guiding Principle: Push concepts as far down as possible (…and see what happens!)

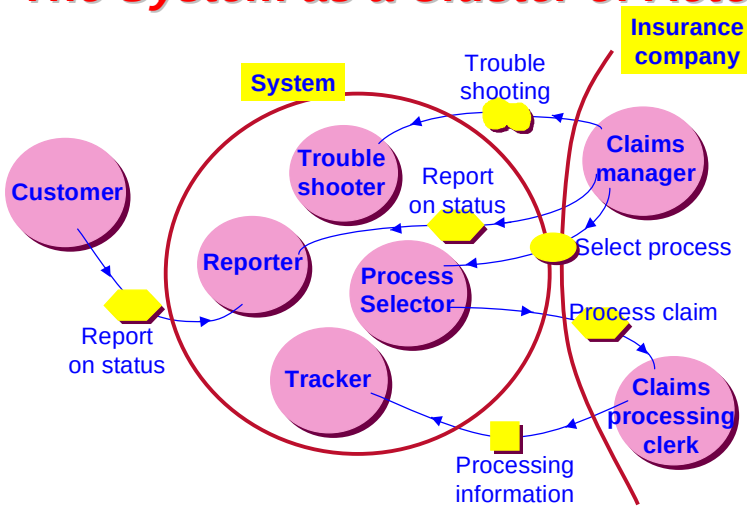
Late Requirements with i*

- The system is now represented as one or more actors which participate in a strategic dependency model.
- Resource, task and softgoal dependencies correspond naturally to functional and non-functional requirements.
- Leaving (some) goal dependencies between software system actors and other actors is a novelty. Traditionally, functional goals are "operationalized" during late requirements, and quality softgoals are either operationalized or "metricized".
- Leaving goal dependencies with system actors as dependees makes sense whenever there is a foreseeable need for flexibility in the performance of a task on the part of the system.

The System as an Actor

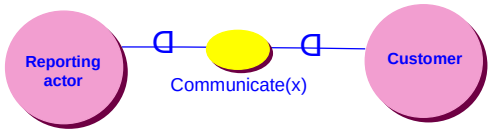


The System as a Cluster of Actors



Removing Goals Early On Leads to Fragile Software Systems!

- Consider a goal laid out during early requirements "communicate(x,y)".
- Conventionally, such goals are "operationalized" during late requirements into "constraints" for the system-to-be, such as having a user interface, also supporting a dialogue during which information x is communicated to person y.
- Such "operationalizations" lead to fragile systems; …what if y doesn't engage in dialogue with the system?… y doesn't understand the message?… the system crashes during the dialogue?… etc.



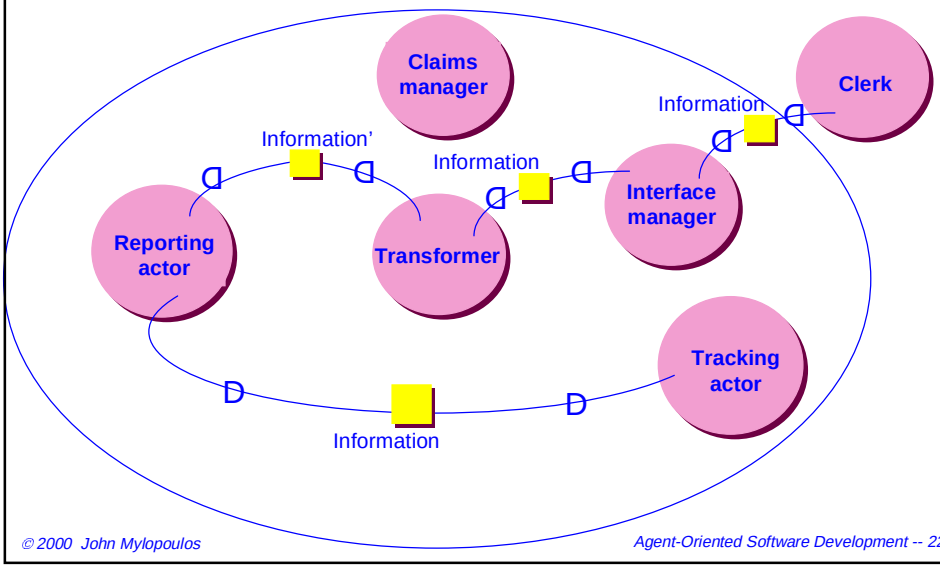
Leaving System Goals Enhances System Flexibility

- Leaving the communication goal as part of the late requirements spec, or even the design means that the system-to-be will be designed with several alternative strategies for satisfying the goal, including getting help from outside.
- The goal may be dealt with during architectural design, by including a variety of user interface modes, or it may be dealt with at run-time by having a "communication" agent which specializes in communicating with people; such an agent may or may not have a planning capability.

Architectural Design with i*

- Now we focus on actors that are part of the system.
- Add "helper" actors, to help existing (system) actors meet their obligations. This leads to conventional hierarchical architectures,
- Add actors that are delegated obligations; here one obligation (e.g., delivering some resource) is broken down to several obligations.
- Decide whether actors will be assigned a (software) agent, a position or a role. This assignment determines how tighly coupled an actor is to the agent that fulfills its obligations.

Architectural Design with i*



Detailed Design

- The intentions of all actors, I.e., the process they use to fulfil their obligations, also timing and other constraints are specified using diagrams (a la UML or other) and formal specification techniques.
- As with detailed design for other techniques, the idea here is to specify completely the behaviour of each actor.

Detailed Design Using Tropos

Entity Claim
Has claimId:Number, insP:InsPolicy,
claimDate, accDate:Date, details:Text
Invariant after(insP.expDate, accDate)
Invariant $(\forall x)(Claim(x) \wedge \bullet\neg Claim(x) \Rightarrow \neg RunOK(x.insP.car))$
end Claim

Entity InsPolicy
Has insP#:Number, expDate:Date, car:Car, insured:
Customer,
policyT:Type, premium:Amount
end InsPolicy

More Detailed Design

Action MakeClaim
Input insPolicy [**Arg**:insP], Date [**Arg**:accD], Text [**Arg**:det]
Output Claim [**Arg**:claim]
Precondition ¬RunOK(insP.car))
/* This was specified earlier under Claim, so it's redundant */
Postcondition claim.insP = insP ∧ claim.accDate = accD ∧
claim.details = det

Actor Customer
Has name:Name, addr:Address, phone#:Number,
rating:RatingType
Capable of MakeClaim, Pay **end** Customer

Implementing Agent-Oriented Software

- Agents are implemented using some agent-oriented implementation platform (e.g., JACK). Such platforms offer a generic agent architecture (e.g., BDI), belief base management facilities, and more.
- If there are dangling goal dependencies, I.e., goal dependencies for which no one has undertaken the responsibility to fulfill, build into the responsible agent skills for meeting these goals.
  E.g., a communication goal might be met through repeated email, asking a third party to communicate etc.
- If there are dangling softgoal dependencies, build into the responsible agent skills for addressing such softgoals.
  E.g., a security agent would have a number of ways of meeting security goals

A Multi-Perspective View of Software

- We are working towards an agent-oriented software development methodology, founded on the key concepts of actor, goal, (goal, task, resource) dependency, etc.
- Software is viewed from four perspectives:
  - **Organizational** -- who are the relevant actors, what do they want? What are their obligations? …capabilities??
  - **Intentional** -- what are the relevant goals and how they interrelate? How are they being met? … by whom??
  - **Process-oriented** -- what are the relevant business/computer processes? Who is responsible for what?
  - **Object-oriented** -- relevant objects
- We have focused on organizational and intentional perspectives because they are novel. For the others we propose to use UML-type modelling techniques.

Tropos

- A research project whose aim is to develop a software development methodology for agent-oriented systems.
- The list of participants includes Eric Yu (University of Toronto), Yves Lesperance (York University), Jaelson Castro University of Pernambuco, Brazil) and Manuel Kolp (University of Toronto); also Alex Borgida (Rutgers University), Matthias Jarke and Gerhard Lakemeyer (Technical University of Aachen), Fausto Giunchiglia, Paolo Bresciani, Paolo Giorgini, and Anna Perini.
- The concepts of *i** will be embedded in a modeling framework which also supports generalization, aggregation, classification and contexts. Some elements of UML will be adopted as well for modeling object and process perspectives.

Conclusions

From a Software Engineering perspective, this proposal, however speculative, has advantages:

- Leads to more flexible, robust and open software architectures;
- Offers a coherent framework which encompasses all phases of software development, from early requirements to implementation
- Is consistent with the next generation programming paradigm, I.e., agent-oriented programming.
- This paradigm is already gaining a foothold in key application areas, such as telecommunications, electronic commerce and web-based systems.

...More Conclusions

As well, from an Agent-Based Systems perspective the proposal

- Suggests a comprehensive methodology for building agent-oriented software;
- Offers a design dimension along which one decides how to accommodate tradeoffs among qualities such as flexibility, robustness, and performance.
- Some preliminary work on development methodologies for agent-oriented software systems can be found in [Wooldridge99], [Singh99].

...BUT,...the research has just begun!

(Note: These slides are available in www.science.unitn.it/~jm)

References

- [Bubenko93] Bubenko, J., "Extending the Scope of Information Modelling", Proceedings Fourth International Workshop on the Deductive Approach to Information Systems and Databases, Costa Brava, 1993.
- [Chung93] Chung, L., *Representing and Using Non-Functional Requirements: A Process-Oriented Approach*, PhD thesis, Department of Computer Science, University of Toronto, June 1993.
- [Dardenne93] Dardenne, A., van Lamsweerde, A. and Fickas, S.,"Goal-Directed Requirements Acquisition", in *The Science of Computer Programming 20*, 1993.
- [Feather87] Feather, M., "Language Support for the Specification and Development of Composite Systems", *ACM Transactions on Programming Languages and Systems 9(2)*, 1987.
- [Fickas92] Fickas, S. and Helm, R., "Knowledge Representation and Reasoming in the Design of Composite Systems", *IEEE Transactions on Software Engineering 18(6)*, June 1992.
- [Greenspan84] Greenspan, S., *Requirements Modelling: A Knowledge Representation Approach to to Software Requirements Definition*, PhD thesis, Department of Computer Science, University of Toronto, 1994.

References (cont'd)

- [Singh99] Singh, M., "Synthesizing Requirements for Heterogeneous Autonomous Agents", *Autonomous Agents and Multi-Agent Systems*, Kluwer, (to appear).
- [Wickler99] Wickler, G., "Intelligent Agent Platform and Environment -- I-APE," ITC-IRST lecture slides, IRST, Trento, Italy, 1999.
- [Wooldridge99] Wooldridge, M., Jennings, N., Kinny, D., "A Methodology for Agent-Oriented Analysis and Design", Proceedings ACM Conference on Autonomous Agents (Autonomous Agents'99), Seattle, May 1999.
- [Yu94] Yu, E., *Modelling Strategic Relationships for Process Reengineering*, PhD thesis, Department of Computer Science, University of Toronto, December 1994.
- [Yu95] Yu, E., Du Bois, P., Dubois, E. and Mylopoulos, J., "From Organization Models to System Requirements: A 'Cooperative Agents' Approach", Proceedings Third International Conference on Cooperative Information Systems, Vienna, May 1995.

Actor Assignments