Paolo Giorgini
Dipartimento Ingegneria e Scienza dell’Informazione
University of Trento - Italy
http://www.dit.unitn.it/~pgiorgio

Agent-Oriented Software Engineering course
Laurea Specialistica in Informatica
A.A. 2009-2010
Outline

- The Tropos methodology
  - Motivation and objectives
  - Modeling language
  - Modeling activities
  - Phases and process
  - TAOM4E
  - Patterns
  - Formal Tropos
Tropos Objectives

- Define a software development methodology for agent-oriented software systems
- (In other words) Define a UML-type language and methodology for agent-oriented software, but cover a broader spectrum of the software development process
- Specifically, the Tropos methodology is intended to cover early requirements, late requirements, architectural design and detailed design; also to interface with agent programming platforms
Tropos in Perspective

Filling the gap

Agent-oriented programming

KAOS

UML

A knowledge level methodology
Tropos Approach (1)

- Tropos adopts a requirements-driven software development approach, exploiting goal analysis and actor dependencies analysis techniques.
- i* (by E. Yu) as foundation for the modeling framework.
- We model the what and the how, as well as the why.
- i* does offer a small set of well-worked out concepts for modeling social setting … but i* was intended for early requirements modeling and analysis, so it needs to be revised and extended to cover the scope of the Tropos methodology.
Approach (2) - knowledge level

- The notion of agent and goal is used along all the phases
  - the key elements and dependencies describing the organizational setting can be used to justify and motivate each design and implementation choice

- With respect to research methodology, Tropos aspires to make progress on three fronts: formal methods, tools and development process.
A Model-Driven development process

- Early Requirements
- Late Requirements
- Architectural Design
- Detailed Design
- Implementation

capability level [Penserini et al., 2006]

knowledge level
The Tropos conceptual framework

Modeling language
- Formal language
- Graphical language

Tropos Models

Analysis Techniques
Tropos language: basic concepts

- **Actor**
  - Intentional entity: role, position, agent (human or software)

- **Goal (softgoal)**
  - Strategic interest of an actor

- **Task**
  - Particular course of action that can be executed in order to satisfy a goal

- **Resource**
  - Physical or informational entity (without intentionality)

- **Social dependency (between two actors)**
  - One actor depends on another to accomplish a goal, execute a task, or deliver a resource
Actor: an intentional entity

Actor Meta-model
Actor: graphical representation

Paolo is a Person
Paolo occupies the position of Professor
The position Professor covers the roles
• Researcher
• Advisor
• Teacher

----> Paolo plays the role of Researcher, Advisor, Teacher
Goal and softgoal

- A Goal is always associated to an actor

Goal
- Hard goal: we have a clear criteria to know when and whether the goal is satisfied
- Soft Goal: not clearly defined and it is not possible to define a criteria of satisfiability (often used for qualities)
Task -- Plan

- A task is a particular course of action
  - Different levels of abstraction
    - Plan with atomic and/or composite actions
    - Business Process description
- A task is used to satisfy goals
  - always associated to one or more goals
Resource

- A non intentional entity
  - Physical (e.g., printer, car, desk, …)
  - Informative (e.g., AOSE lecture notes, web site, …)
- A resource can be used to
  - achieve goals
  - Perform tasks
- Can be produced by
  - achieving goals
  - executing tasks
- A resource can be shared
  - by agents for
    - Achieving goals and executing tasks
Social dependency

- Always between two actors
  - One actor depends on another to accomplish a goal, execute a task, or deliver a resource

- Content of the dependency
  - Goal
  - Task
  - Resource

- An example of goal dependency
  1. Paolo needs to book and hotel in Malaga
  2. Antonio is able to book the hotel
  3. Paolo ask Antonio to book the Hotel
  4. Antonio agrees to book the hotel for Paolo
  5. Paolo depends on Antonio to book the Hotel in Malaga
Social dependency: graphical rep.

Goal dependency

Softgoal dependency

Task dependency

Resource dependency
Tropos relations

- Decomposition
  - AND decomposition
  - OR decomposition
  - A goal can be decomposed in subgoals
  - A task can be decomposed in subtasks

- Means-ends
  - A task (mean) can be used to achieve a goal (end)

- Contribution
  - A goal/task/softgoal can contributes to the satisfaction of a softgoal
    - +, ++, -, -- (we will see more on this in the next lectures)

- Resource need
  - A task/goal needs a resource

- Resource production
  - A task/goal produces a resource
Tropos models

- Three main type of models
  - Actor Model
    - Goal Model
    - Plan Model
    - Capability Model (only for design)
  - Dependency Model
  - Mixed model
    - dependency+actor model
Actor model

- Allows to model a single actor
- Represent the local view
- Process
  1. Goal modeling
     - Main (top) goals and softgoals are identified
     - Goals AND/OR-decomposed
  2. Task modeling
     - For each leaf goal, means tasks are identified
     - Tasks are AND/OR-decomposed
     - Resources needs and production are identified
  3. Contribution analysis
     - For each softgoal, possible incoming and outgoing contributions are identified
Actor model
Dependency model

- Allows to build a social (organisational) view
- The focus is on social dependencies
- Process
  1. Actors identification
     • The relevant actors are identified (scope of the model)
  2. Social dependencies are identified
     • Each actor is analyzed identifying incoming and outgoing dependencies (goal/task/resource dependency)
     • Actor models are used for the analysis
  3. Model refinement
     • More actors can be identified (go back to 1)
- The level of abstraction depends on the objective of the model
- No precise rules are given
Dependency model

[Diagram of dependencies between Paolo, Room, projector, high quality course, UNIMA, fees, lecture notes, learn about AOSE, attend AOSE lectures, provide a degree program, good degree program, UNIMA Student, UNIMA, project, high quality course connections]
Mixed model

- Is used to have a mixed view between actor models and their dependencies model
- Often used to build the dependency model or refine the actor model
- Has to be used very carefully
  - can become very complicated and difficult to read

Usual process

1. From an initial actor diagram, (new) dependencies are identified
   - E.g., Actor A has a goal G that is decomposed in two subgoals G1 and G2, goal G2 is delegated to actor B
2. The new dependencies introduce in the actor model(s) new goals/tasks
   - Actor B has now goal G2 that can be analyzed in its actor model
3. Go back to 1. or stop the process
Tropos modeling

- It is up to the analyst/designer to decide
  - the most appropriate level of abstraction (details)
  - When to stop
  - How (and whether) verify the completeness of models
  - Split the modelling problem in subproblems
- The questions to be asked are (strictly ordered)
  1. WHAT
  2. WHY
  3. HOW
  always try to find the WHY
The Tropos phases

- Early requirements
  - The socio and organizational setting is analyzed
  - We are not interested in describing the system-to-be, but just the most relevant actors and their relationships in the domain where the system will operate

- Late requirements
  - The system-to-be is introduced as a new actor of the social domain analyzed in the previous phase
  - The system-to-be is analyzed in terms of Tropos concepts

- Architectural design
  - The Actor system-to-be is designed
  - Subactors are introduced and delegated of goals/task
  - Agents are identified
  - Agent capabilities are identified

- Detailed design
  - Capabilities, protocols, and agent’s tasks/plan are specified in detail
The Tropos process

An iterative and transformational approach

Early requirements

Late requirements

Architectural design

Detailed design

Implementation

Strongly dependent on the (A)PL
Transformational approach

- We adopt a transformational approach:
  - for early and late requirements, and partially for the architectural design
- Starts with a limited list of *Tropos* conceptual elements
- iteratively and incrementally:
  - add details and revise dependency relationships
- Each step corresponds to the introduction/deletion of relationships/elements in the model.

Advantages
- provides systematic description of the process
- allows for process analysis
- provides guidelines to the engineer
- provides a sound basis for describing and evaluating requirement acquisition and design strategies
An Example  (Thanks people from IRST)

- The Conference Management System (CMS)
  - Pay attention, in the next lecture you have to do it !!!!

- Setting up and running a conference it is a multi-phase process involving several individuals and groups [e.g.Zam01]
- During the submission phase, the authors need to be informed that their paper has been received (a submission number assigned)
- Once the submission deadline has been passed, the program committee (PC) has to manage the review of the papers: contacting potential reviewers; asking them to review a number of papers
- Reviewers can decide to accept or not to review a paper and in case of acceptance they have to produce a review by a given deadline. Reviews are collected and used to decide about paper acceptance by the PC
- Authors will be notified by the PC about paper acceptance/rejection. In case of acceptance they are requested to produce a revised version of the paper (camera-ready).
- Finally, the publisher has to collect these final version and print the proceedings.
- Some organizational rules apply to the actors in the CMS: scenario: e.g. there must be at least three reviewers for each paper; a paper author does not review his own paper; ....
Early requirements

We analyze the environment (i.e. organizational setting) and model it in terms of relevant actors and their respective dependencies.

Questions to guide the analysis

- Which are the main actors?
- What are their goals?
- How can they achieve them?
- Does an actor depend on another one to achieve its goals?
Questions to guide the analysis

Taking the perspective of each single actor, analyze each goal:

- How can it be decomposed in sub-goals?
- Are there alternative ways to reach a goal? How to evaluate/choose an alternative?
- Are there means (e.g. plan /resources) to achieving them?
- Are there (soft) goals which may prevent the achievement of a goal?
- Are there (soft) goals which may contribute to the achievement of a goal?
Late requirements

We introduce the system actor and analyze its dependencies with actors in its environment identifying system’s functional and non-functional requirements.
Late requirements

The goals *decomposition*, *means-end* and *contribution* analysis are performed on the system’s goals.
Architectural design: 3 activities

1. Decomposing and refining the system actor diagram
   - definition of the overall architecture (patterns -- if necessary)
   - inclusion of new actors due to delegation of subgoals upon goal analysis of system's goals
   - inclusion of new actors according to the choice of a specific architectural style (design patterns)
   - inclusion of new actors contributing positively to the fulfillment of some Non Functional Requirements

2. Identifying actors’ capabilities

3. From *actors* to *agents*
Architectural design (step 1)
Focusing on System Sub-Actor dependencies
Architectural design (step 1 - with open balloon)
Architectural design (step 1)
Focusing on the the **Proceedings Manager** actor (agent)

<table>
<thead>
<tr>
<th>Agent Name</th>
<th>Capability</th>
<th>Means-end</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proceedings Manager</td>
<td><em>Cp1</em>  Deliver proceedings; send to publisher</td>
<td></td>
</tr>
<tr>
<td></td>
<td><em>Cp2</em>  Print proceedings; print to pdf</td>
<td></td>
</tr>
<tr>
<td></td>
<td><em>Cp3</em>  Build indexes; retrieve index from PC</td>
<td></td>
</tr>
<tr>
<td></td>
<td><em>Cp4</em>  Build indexes; automatic index gen</td>
<td></td>
</tr>
<tr>
<td></td>
<td><em>Cp5</em>  Align paper format; check style</td>
<td></td>
</tr>
<tr>
<td></td>
<td><em>Cp6</em>  Align paper format; recompile</td>
<td></td>
</tr>
</tbody>
</table>
Detailed design

- Specification of the agents’ micro level taking into account the implementation platform.
- We model:
  - capabilities
    - capability diagrams (currently UML activity diagrams)
  - plans
    - plan diagrams (currently UML activity diagrams)
  - agents interaction
    - agent interactions diagrams (currently (A)UML sequence diagrams)
- Automated transformations can be applied
  - we are not going to see this
Detailed design

Capability diagram

Agent interaction diagram
Implementation

- … follow step by step the detailed design specification …

- In JADE, agents behave carrying out specific tasks (actions) and communicating via asynchronous message passing, messages are expressed in ACL language.

- In Jade, [Pok05] agents try to achieve goals requested by the user by executing the connected Plans. A code (skeleton) executable on the Jadex platform (XML+Java) can be generated automatically from an agent's goal model knowledge level.

- In Jack, agents have explicit goals (desires) to achieve or events to handle, beliefs and capabilities. They are programmed with a set of plans that make them capable of achieving goals.
Implementation in Jadex

[Diagram showing agent interactions and network connections]

- GOAL = deliver_proceedings#1
- Value = RAchieveGoal(name=deliver_proceedings#1)
- Agent = ProceedingsManager@idacox:1099/JADE
- Seq = 16
- Causes = requestPlan_deliver_proceedings#1
Implementation in Jack
TAOM4E

- TAOM4E: Tool for Agent Oriented visual Modeling for the Eclipse platform
- Model-based approach
- Supports the Tropos methodology
- TAOM4E is based on the Eclipse Platform
  - offers a flexible solution to the problem of component integration.
- TAOM4E offers
  - a graphical interface for all the Tropos phases
  - Integration with
  - Automated jadex code generation
  - Access to reasoning tools (risk analysis, security analysis, goal analysis....)
TAOM4E’s GUI
Using Patterns

- Two type of patterns
  - Macro-level patterns
    - Bla Bla …
  - Micro-level patterns
    - Bla .. Bla ..
Organizational patterns

Structure-in-5

Joint-venture
Joint-Venture E-commerce Architecture
Multi-agent patterns

Matchmaker

Consumer
Requested Service
Provider
Locate Provider
Matchmaker
Advertise Service

Monitor
Subscriber
Forward Subscribed Change
Subject
Notify Change

Contract-Net
Client
Bid
Accept Proposal
Perform Contracted Work
Contractor

Mediator
Initiator
Route Requested Service
Mediator
Performer
Performs Service
Formal Tropos

- Formal Tropos extends the Tropos approach with formal specification
  - Formal Methods for the verification of Tropos early requirements
- Formal methods are difficult to apply in early requirements
  - Formal Methods require a detailed description of the behaviour of the system
- Formal methods in early requirements cannot be used to prove the correctness of the specification
- However they can:
  - Show misunderstandings and omissions in the requirements specification that might not be evident in an informal setting
  - Assist the elicitation of the requirements by helping the interactions with the stakeholders
  - Add expressive power to the requirements specification formalism
Formal Tropos Approach

Formal Tropos builds on:
- **Tropos** for modeling social setting, based on the notions of actors, goals, dependencies, …
- **KAOS**, a goal-oriented requirements framework that provides a rich temporal specification language
- **NuSMV**, a (symbolic) model checker initially developed for the verification of hardware systems

Formal Tropos:
- definition of Formal Tropos language, that integrates the primitive concepts of Tropos with a temporal specification language inspired by KAOS
- extension of existing model checking verification techniques in order to allow for the mechanized analysis of Formal Tropos specifications
- implementation of a prototype tool, called T-Tool, that supports the given approach, and that uses NuSMV as verification engine
Insurance Company example in Tropos
Formal Tropos Specification

Entity Car
   Attribute runsOK: boolean

Entity Damage
   Attribute constant car: Car

Actor InsuranceCo
Actor BodyShop
Actor Customer

Goal BeInsured
Mode maintain

Dependency CoverDamages
   Type goal
   Mode achieve
   Depender Customer
   Dependee InsuranceCo
   Attribute constant dam: Damage
   Creation Condition
   \neg dam.car.runsOK

Dependency RepairCar
   Type goal
   Mode achieve
   Depender Customer
   Dependee Bodyshop
   Attribute constant dam: Damage
   Creation Condition
   \neg dam.car.runsOK
   Fulfillment condition
   dam.car.runsOK
Constraint properties

- Constraint properties determine the possible evolutions of the objects in the specification
- Three kinds of properties:
  - creation properties
  - invariants
  - fulfillment properties
- Creation and fulfillment properties may express:
  - necessary conditions (for creation, fulfillment, ...)
  - sufficient conditions, or triggers
  - necessary and sufficient conditions, or definitions
Temporal formulas

- Properties are specified with formulas given in a first-order linear-time temporal logic
  - Special predicates “JustCreated(obj), “Fulfilled(dep)” identify particular moments in the life of the objects
  - Past and future temporal operators can be used in the formulas:
    - □φ (always in the future), ◇φ (eventually), …
    - ◻φ (always in the past), ◤φ (sometimes in the past), …
- We aim to minimize the use of temporal operators. For instance,
  - maintain hides a
  - achieve hides a ◇
Formal Analysis

Formal Tropos allows for the following kinds of analysis:

- **Consistency check**: “the specification admits valid scenarios”
- **Assertion validation**: “all scenarios for the system respect certain assertion properties”
- **Possibility check**: “there is some scenario for the system that respects certain possibility properties”
- **Animation** allows the user to interactively explore valid scenarios for the system
  - Gives immediate feedback on the effects of the constraints
  - Makes it possible to catch trivial errors
  - is an effective way of communicating with the stakeholders
Assertion validation

An assertion

- describes expected conditions for all the valid scenarios
- is used to guarantee that the specification does not allow for unwanted scenarios.

**Example:** “the requirements should guarantee that the insurance company does not cover damage for which there is no proof (e.g., an invoice) that the car was repaired”

**Dependency** CoverDamages

**Fulfillment assertion condition**

dam.car.runsOK →

\[ \exists rep: \text{RepairCar}(\text{rep.dam=}\text{dam} \land \text{Fulfilled(rep)}) \]
The technical details

Three steps:
1. The analyst writes a Formal Tropos specification
2. T-Tool automatically translates the specification into an Intermediate Language
3. (An enhanced version of) NuSMV performs the formal analysis on the Intermediate Language specification

The intermediate language is:
- Small core language with a clean semantics
- Independent from the specifications of Formal Tropos (the Intermediate Language may be applied to other requirements languages)
- Independent from any particular analysis technique