Model Checking Early Requirements Specifications in Tropos

Ariel Fuxman and ***John Mylopoulos*** (Univ. of Toronto)

Marco Pistore and ***Paolo Traverso*** (IRST, Italy)

Tropos Workshop — Trento, 15-16 November 2001

Formal Methods in the Software Development Process

GOAL: an effective integration and harmonization of formal methods in the software development process

MOTIVATION: the gap existing between Formal Methods and current Software Development Processes that limits the applicability and the effectiveness of formal methods in practice

CHALLENGE: take into account all the different aspects:

- techniques and tools
- methodology
- supporting platform
- soft-goals: practicality / cost-effectiveness / degree of automation ...

The Formal Tropos project

Formal Tropos: extend the Tropos approach with formal specification and V&V techniques

- Tropos aims at providing an uniform approach

  ⇒ one specification language for all the phases

  ⇒ smooth borders between the development phases

- Modeling concepts: actors and their dependencies

  ⇒ very general: appropriate for all large systems

  ⇒ compositional verification techniques are applicable

- Change of perspective: start from the early requirements

  ⇒ formal validation can be very convenient in this phase

In this talk: Application of Formal Methods for the verification of Early Requirements Specifications in Tropos

Motivations

- Early Requirements Specification is an important phase in software development
- Formal Methods provide very powerful specification and early-debugging techniques in the later phases of software development
- Formal Methods are difficult to apply in Early Requirements:
  - the typical approach of Formal Methods (validate an implementation against the requirements) does not apply;
  - Formal Methods require a detailed description of the behavior of the system;
  - the concepts of Formal Methods are not appropriate for Early Requirements.

⇒ **Our aim is to provide a framework for the effective application of Formal Methods in the Early Requirements phase.**

Formal Methods in Early Requirements

Formal Methods in Early Requirements cannot be used to prove the correctness of the specification.

However they can:

- show misunderstanding 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

The approach

The approach we are proposing builds on:

- i*, a framework for modeling social settings, 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

The approach

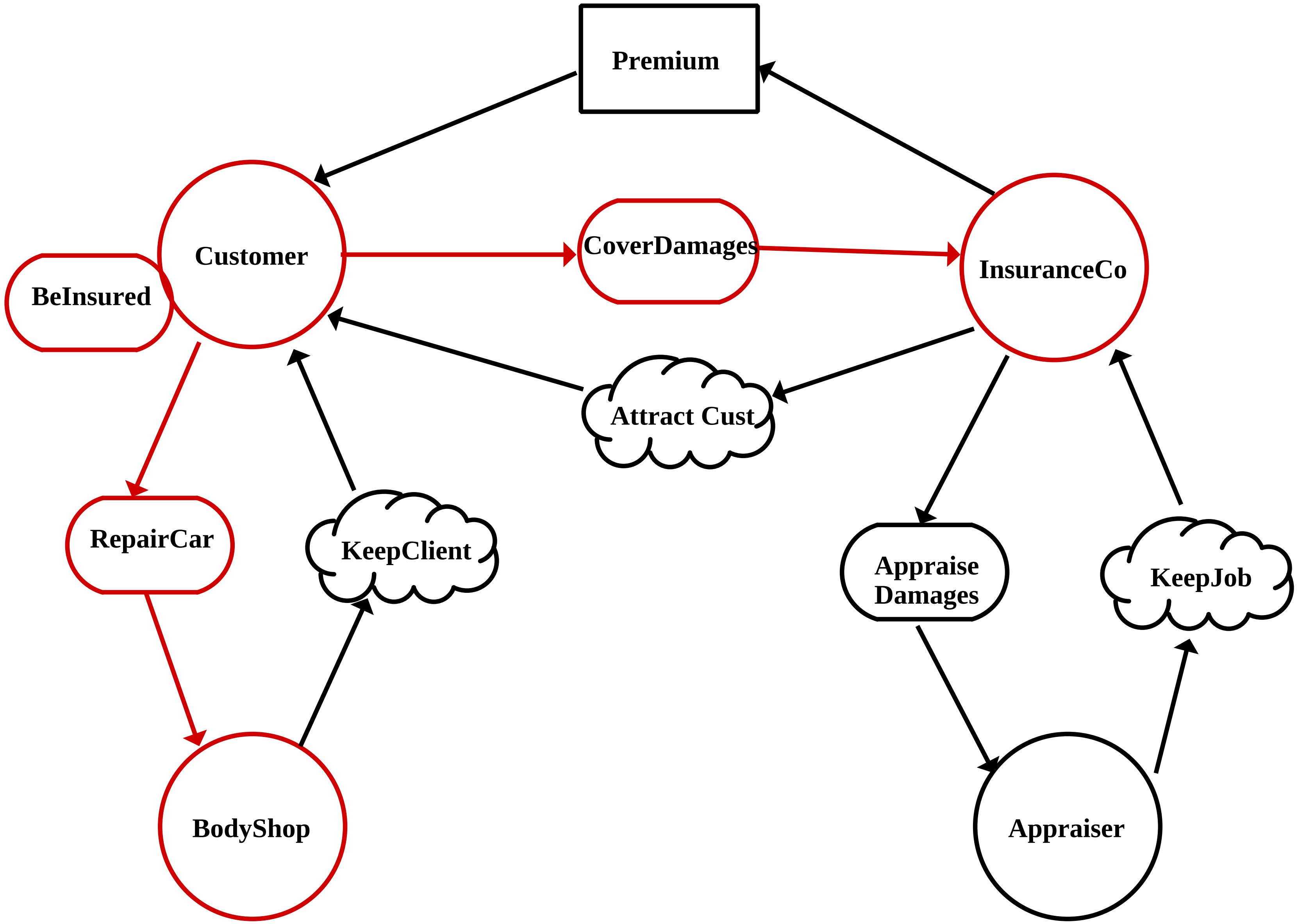
We have achieved the following results:

- definition of Formal Tropos, that integrates the primitive concepts of i* 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

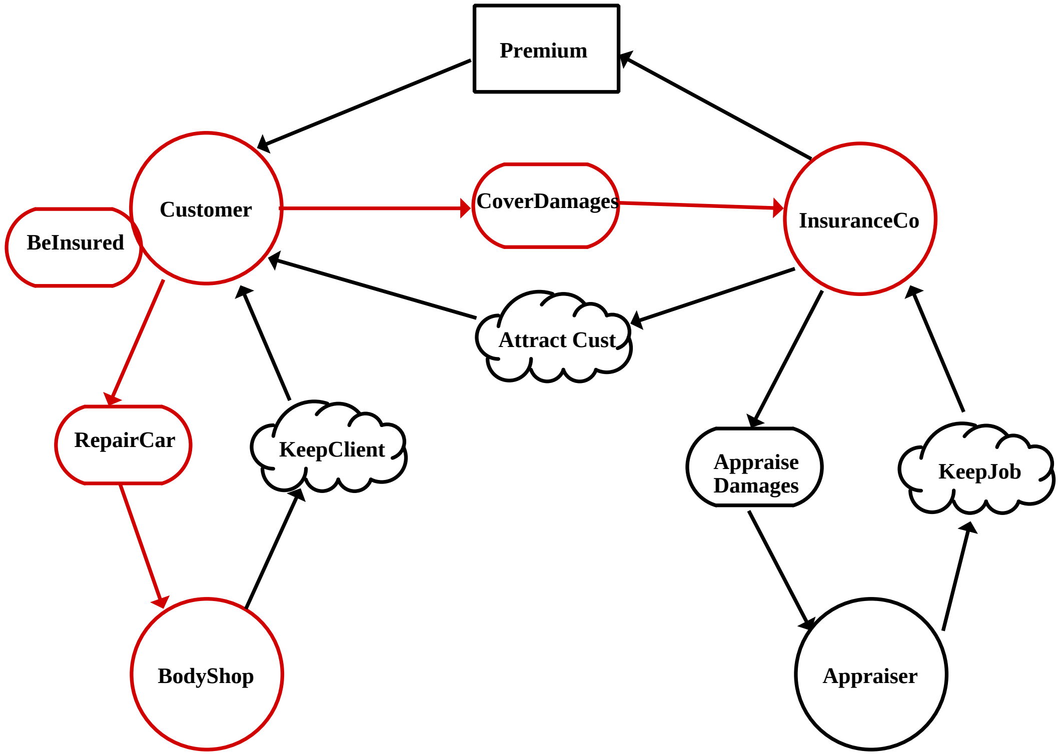
The original contributions:

- w.r.t. i*, a formal specification language, the possibility of applying formal methods techniques
- w.r.t. KAOS, a different ontology based on i*, different formal techniques (model checking rather than theorem proving)
- w.r.t. NuSMV, a new application domain, and a different specification language

Insurance Company case study in i*



The diagram does not show that...



- there are different instances of actors, goals, dependencies, and relations among these instances
- strategic dependencies have a temporal evolution (they arise, they are fulfilled...)

A textual notation for i*

Actor InsuranceCo

Actor BodyShop

Actor Customer
    Goal BeInsured

Dependency CoverDamages
    Type goal
    Depender Customer
    Dependee InsuranceCo

Dependency RepairCar
    Type goal
    Depender Customer
    Dependee BodyShop

Adding the “class” layer

Entity Car

  Attribute runsOK: **boolean**

Entity Damage

  Attribute constant car: Car

Actor InsuranceCo

Actor BodyShop

Actor Customer

  Goal BeInsured

Dependency CoverDamages

  (Type/Depender/Dependee)

  Attribute constant dam: Damage

Dependency RepairCar

  (Type/Depender/Dependee)

  Attribute constant dam: Damage

Modeling the temporal aspects

Formal Tropos places special emphasis in modeling the “strategic” aspects of the evolution of the dependencies.

The focus is on the two central moments in the life of dependencies and entities: **creation** and **fulfillment**.

Formal Tropos allows the designer:

- to specify different modalities for the fulfillment of the dependencies (e.g.: is it a maintain or an achieve goal?)
- to specify temporal constraints on the creation and fulfillment of dependencies and goals.

Goal modalities...

Actor Customer

Goal BeInsured

Mode maintain

Dependency CoverDamages

Type goal

Mode achieve

Dependee Customer

Dependee InsuranceCo

Dependency RepairCar

Type goal

Mode achieve

Depender Customer

Dependee BodyShop

... and behavioral properties

Dependency CoverDamages

Type goal

Mode achieve

Depender Customer

Dependee InsuranceCo

Attribute constant dam: Damage

Creation condition ¬dam.car.runsOK

Dependency RepairCar

Type goal

Mode achieve

Depender Customer

Dependee BodyShop

Attribute constant dam: Damage

Creation condition ¬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
- **invariant**s
- **fulfillment** properties

Creation and fulfillment properties may express:

- necessary **condition**s (for creation, fulfillment . . .)
- sufficient conditions, or **trigger**s
- necessary and sufficient conditions, or **definition**s

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:
  - $\Box\phi$ (always in the future), $\Diamond\phi$ (eventually) ...
  - $\blacksquare\phi$ (always in the past), $\blacklozenge\phi$ (sometimes in the past) ...

We aim to minimize the use of temporal operators. For instance,

- **maintain** hides a $\Box$.
- **achieve** hides a $\Diamond$.

Formal analysis

Formal Tropos allows for the following kinds of formal 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 stakeholder

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.

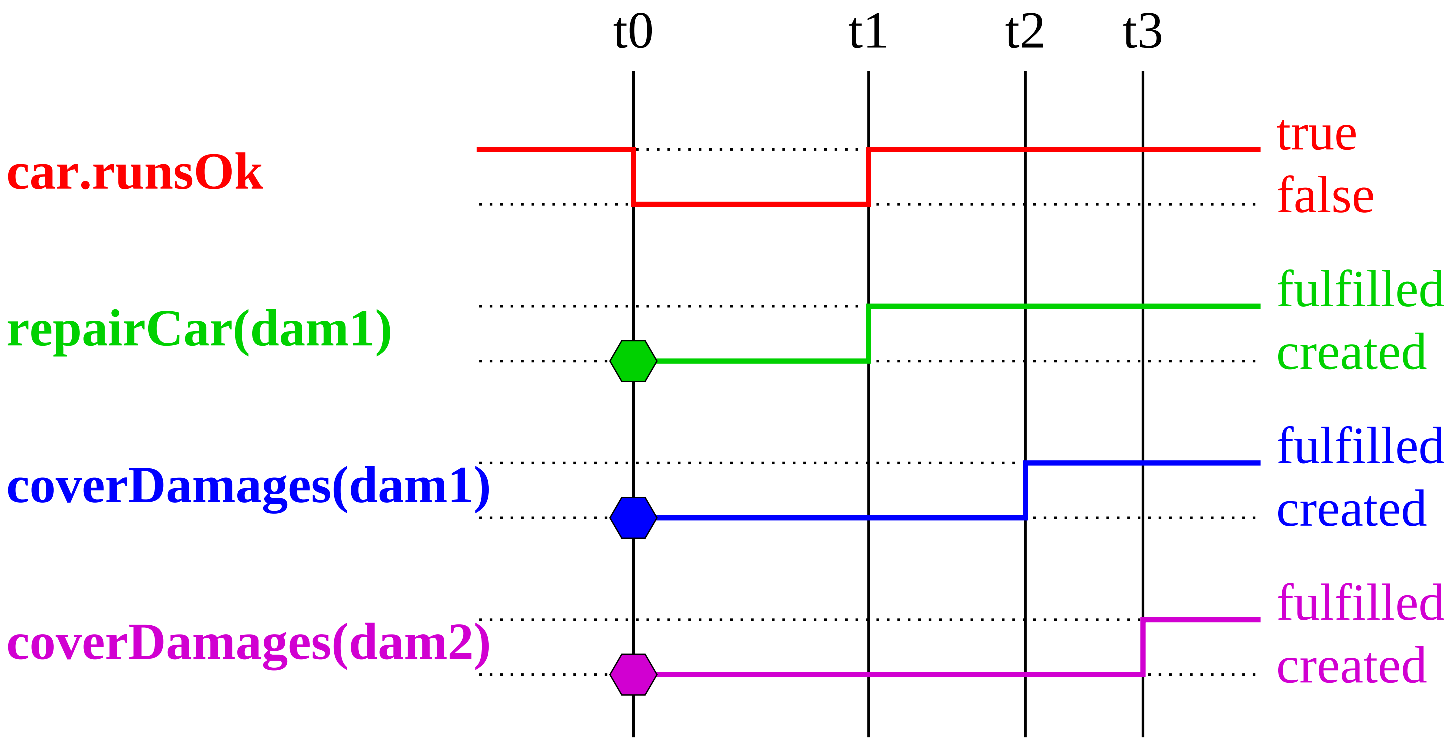
Assertions are specified in Formal Tropos with the same syntax as constraints, but they have a different semantics.

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

Dependency CoverDamages
    Fulfillment assertion condition
        $\text{dam.car.runsOK} \rightarrow \exists \text{rep} : \text{RepairCar}\ (\text{rep.dam} = \text{dam} \wedge \text{Fulfilled}(\text{rep}))$

A counterexample...

Outcome: The tool returns a counterexample scenario:



Possible fix: add to **Dependency** CoverDamages the following constraint:

Dependency CoverDamages

  Fulfillment condition $\exists \text{rep} : \text{RepairCar}(\text{rep.dam} = \text{dam} \wedge \text{Fulfilled}(\text{rep}))$

Possibility check

A **possibility**:

- describes *expected*, *valid* scenarios of the specification;
- is used to guarantee that the specification does not rule out any wanted execution of the system.

Example: “when cars are so damaged that they cannot be repaired, the insurance company is still responsible for covering damages”

Dependency CoverDamages
Creation possibility condition
$\Diamond \mathrm{Fulfilled(cov)} \wedge \Box \neg \mathrm{cov.dam.car.runsOK}$

Possibility check

Outcome: The possibility check fails. There is *no* suitable *example* scenario.

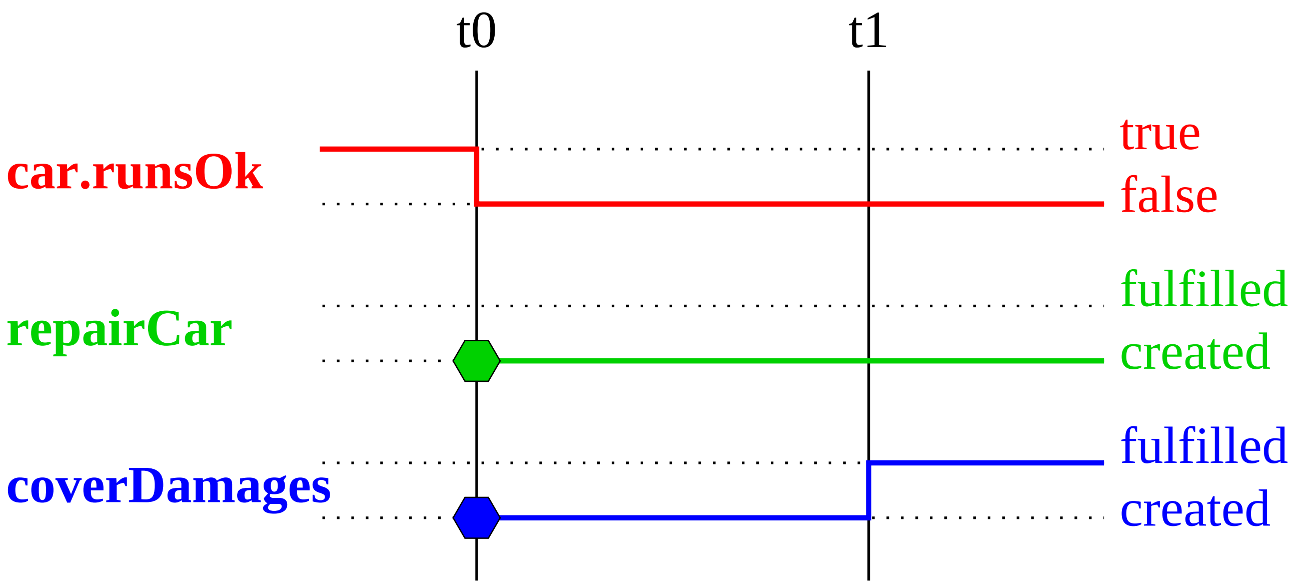
Possible fix: modify the constraint for CoverDamages as follows:

Dependency CoverDamages

Fulfillment condition

$\exists \mathrm{rep} : \mathrm{RepairCar}(\mathrm{rep.dam} = \mathrm{dam} \wedge \mathrm{Fulfilled}(\mathrm{rep})) \vee \Box \neg \mathrm{dam.car.runsOK}$

Outcome: the possibility is satisfied by the following trace:



The technical details

Our approach consists of the following 3 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 specificities of Formal Tropos (the Intermediate Language may be applied to other requirements languages)
- independent from any particular analysis technique (model checking, LTL satisfiability, theorem proving)
- (more details in the paper...)

Conclusions

We have defined:

- Formal Tropos, a formal language for specifying early requirements
- a methodology to extend the requirements with assertions on expected behaviors of the system
- a prototype tool (based on NuSMV) to support the proposed approach

Outcomes: the approach is

- feasible: we obtained feedback from the formal analysis even when dealing with just a few instances
- useful: we were able to identify ambiguities and problems in the informal requirements
- heavy: it is difficult to write LTL specifications

Future work

- Evaluate the approach
  - more complex case studies
  - stress the scalability
  - evaluate applicability and practicality
- Extend the scope of the approach
  - goal decomposition
  - KAOS-like languages
  - “agent-oriented” flavor
- Enhance the tool
  - better interaction with the user
  - improve the animation techniques
  - develop specifically tailored verification algorithms

Future work (long term)

- Extend to later phases
  - same concepts / different interpretations / different V&V techniques
  - automatic generation form the requirements
- Integration of the informal and formal layer
  - graph transformations
- Real applications
  - information systems (eg. security problems)
  - safety critical systems