Formal Tropos

Integrating Formal Methods and Software Engineering

Marco Roveri

joint work with M. Pistore, A. Fuxman, J. Mylopoulos, P. Traverso and L. Liu

`roveri@irst.itc.it`

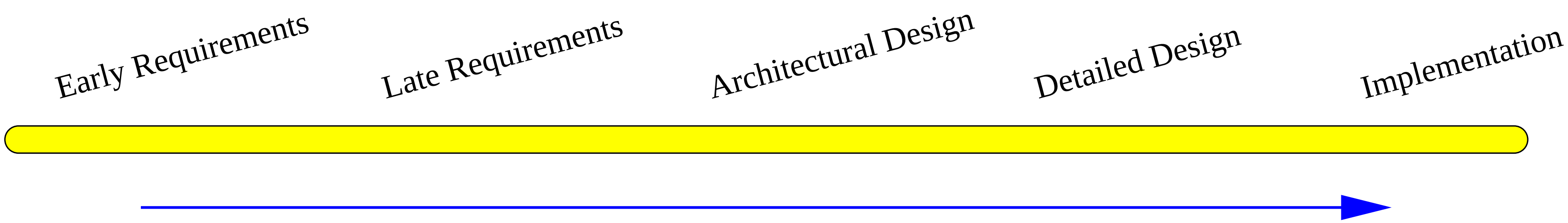
ITC-IRST, Via Sommarive 18, 38055 Povo, Trento, Italy

Outline

- Motivations
- The Formal Tropos Project
- Results so Far: Model Checking Early Requirements
- Conclusions and Future Works

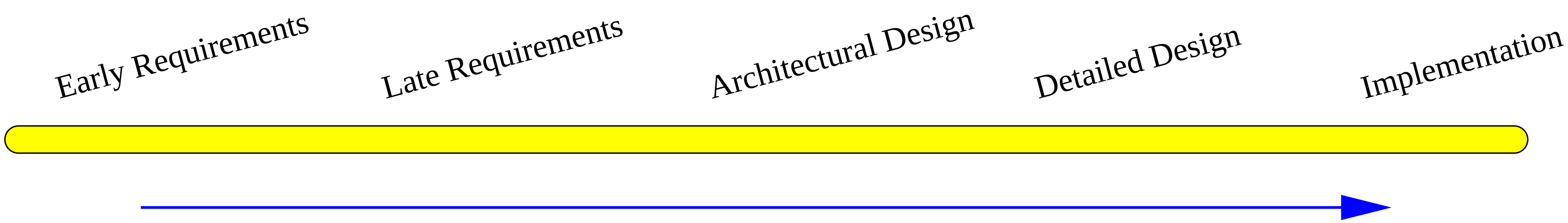
ITC irst

A Software Development Process



- Early Requirements amounts to the definition of the domain.

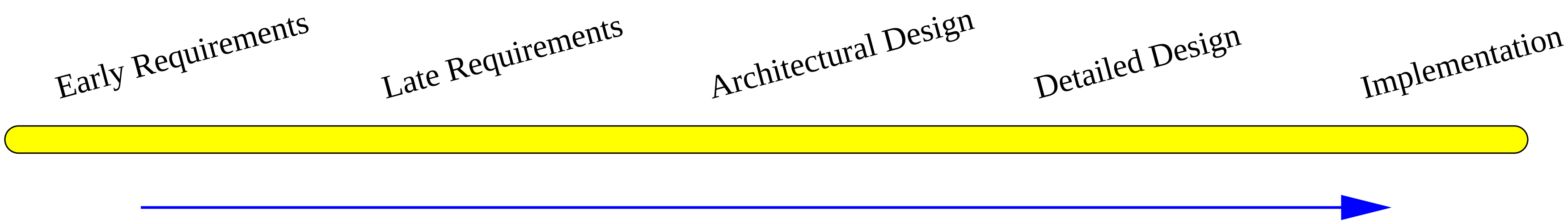
A Software Development Process



- Early Requirements amounts to the definition of the domain.
- Late Requirements amounts to explicitly introduce the *system* and in refining and completing the system definition.

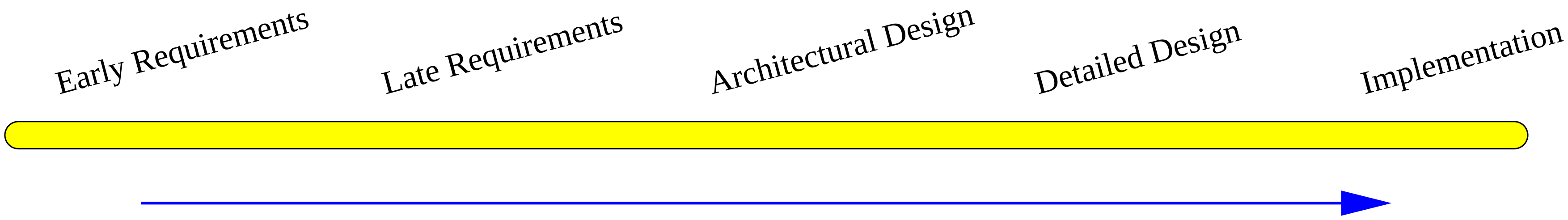
ITC irst

A Software Development Process



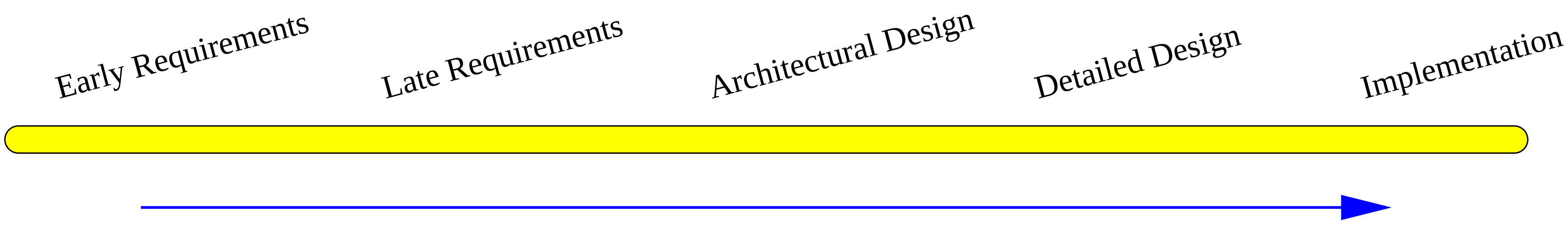
- Early Requirements amounts to the definition of the domain.
- Late Requirements amounts to explicitly introduce the *system* and in refining and completing the system definition.
- Architectural Design amounts to describe how system components work together.

A Software Development Process



- Early Requirements amounts to the definition of the domain.
- Late Requirements amounts to explicitly introduce the *system* and in refining and completing the system definition.
- Architectural Design amounts to describe how system components work together.
- Detailed Design amounts to refine the architectural components of the system.

A Software Development Process



- Early Requirements amounts to the definition of the domain.
- Late Requirements amounts to explicitly introduce the *system* and in refining and completing the system definition.
- Architectural Design amounts to describe how system components work together.
- Detailed Design amounts to refine the architectural components of the system.
- Implementation amounts to the effective coding.

Formal Methods and Software Development Process

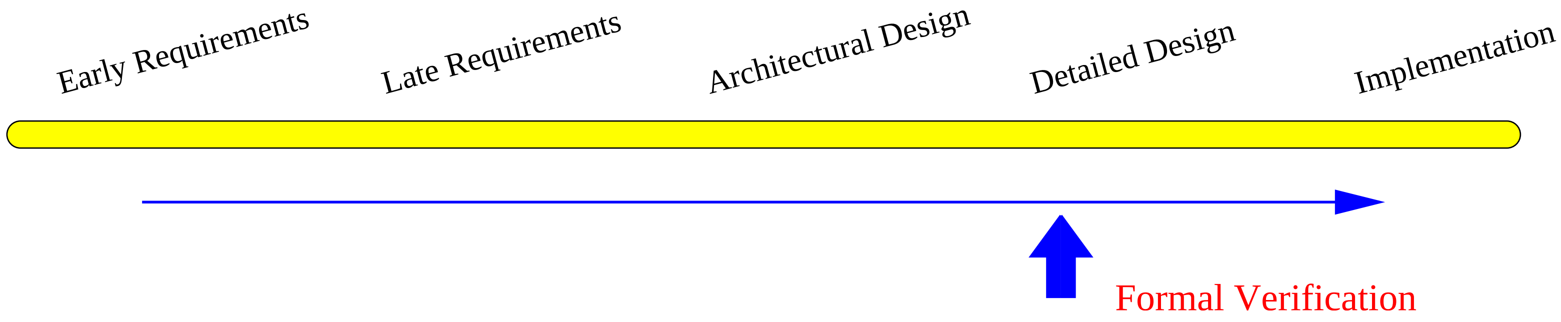
- Formal Methods provide a very powerful specification and early debugging techniques, but. . .

Formal Methods and Software Development Process

- Formal Methods provide a very powerful specification and early debugging techniques, but. . .
  - The gap between these two disciplines limits the applicability and effectiveness of Formal Methods in practice (e.g. language).

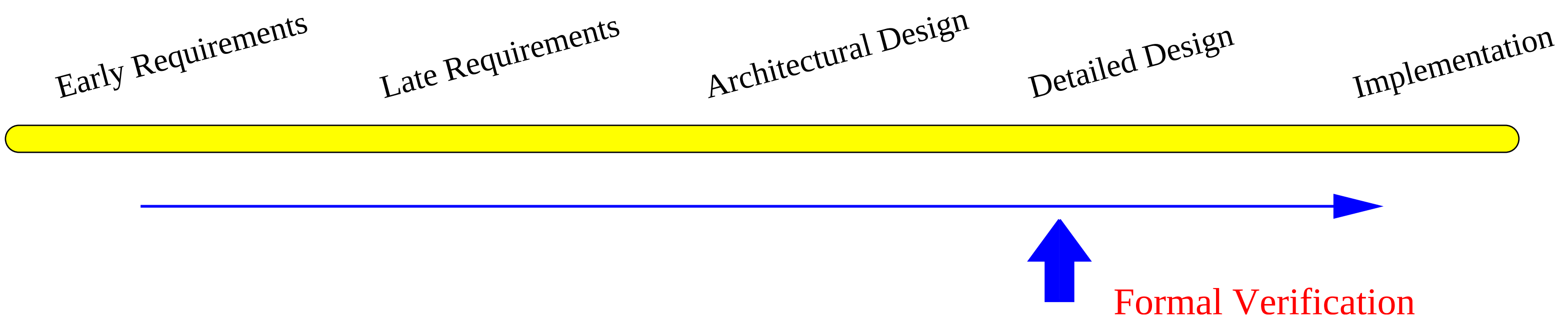
Formal Methods and Software Development Process

- Formal Methods provide a very powerful specification and early debugging techniques, but. . .
  - The gap between these two disciplines limits the applicability and effectiveness of Formal Methods in practice (e.g. language).
  - Formal Methods are usually applied to advanced stages in Software Development Processes.



Formal Methods and Software Development Process

- Formal Methods provide a very powerful specification and early debugging techniques, but. . .
  - The gap between these two disciplines limits the applicability and effectiveness of Formal Methods in practice (e.g. language).
  - Formal Methods are usually applied to advanced stages in Software Development Processes.



- System too big to be efficiently handled.
- Possible bugs discovered too late.

The Formal Tropos Project

Formal Tropos extends the Tropos approach with Formal Specification and Verification and Validation techniques.

The Formal Tropos Project

Formal Tropos extends the Tropos approach with Formal Specification and Verification and Validation techniques.

- Provides an uniform approach.
  - One specification language for all the phases.
  - Smooth borders between the phases.

The Formal Tropos Project

Formal Tropos extends the Tropos approach with Formal Specification and Verification and Validation techniques.

- Provides an uniform approach.
  - One specification language for all the phases.
  - Smooth borders between the phases.
- Modeling concepts: actors and their dependencies.
  - Very general: appropriate for large systems.
  - Compositional verification techniques are applicable.

The Formal Tropos Project

Formal Tropos extends the Tropos approach with Formal Specification and Verification and Validation techniques.

- Provides an uniform approach.
  - One specification language for all the phases.
  - Smooth borders between the phases.
- Modeling concepts: actors and their dependencies.
  - Very general: appropriate for large systems.
  - Compositional verification techniques are applicable.
- Change of perspective: start from the early requirements:
  - Formal verification can be very convenient in this phase.

The Formal Tropos Project

Formal Tropos extends the Tropos approach with Formal Specification and Verification and Validation techniques.

- Provides an uniform approach.
  - One specification language for all the phases.
  - Smooth borders between the phases.
- Modeling concepts: actors and their dependencies.
  - Very general: appropriate for large systems.
  - Compositional verification techniques are applicable.
- Change of perspective: start from the early requirements:
  - Formal verification can be very convenient in this phase.

GOAL: an effective integration and harmonization of Formal Methods in the Tropos Software Development Process.

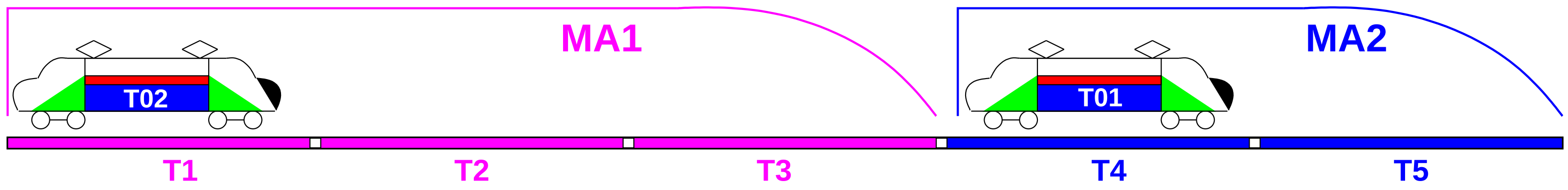
Applying FM to Early Requirements

- Early Requirement Specification is a crucial phase in the development process.
- Formal Methods are commonly used in advanced stage of the development process.
- Formal Methods are difficult to apply in Early Requirements:
  - The typical approach that amounts to validate an implementation against 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.

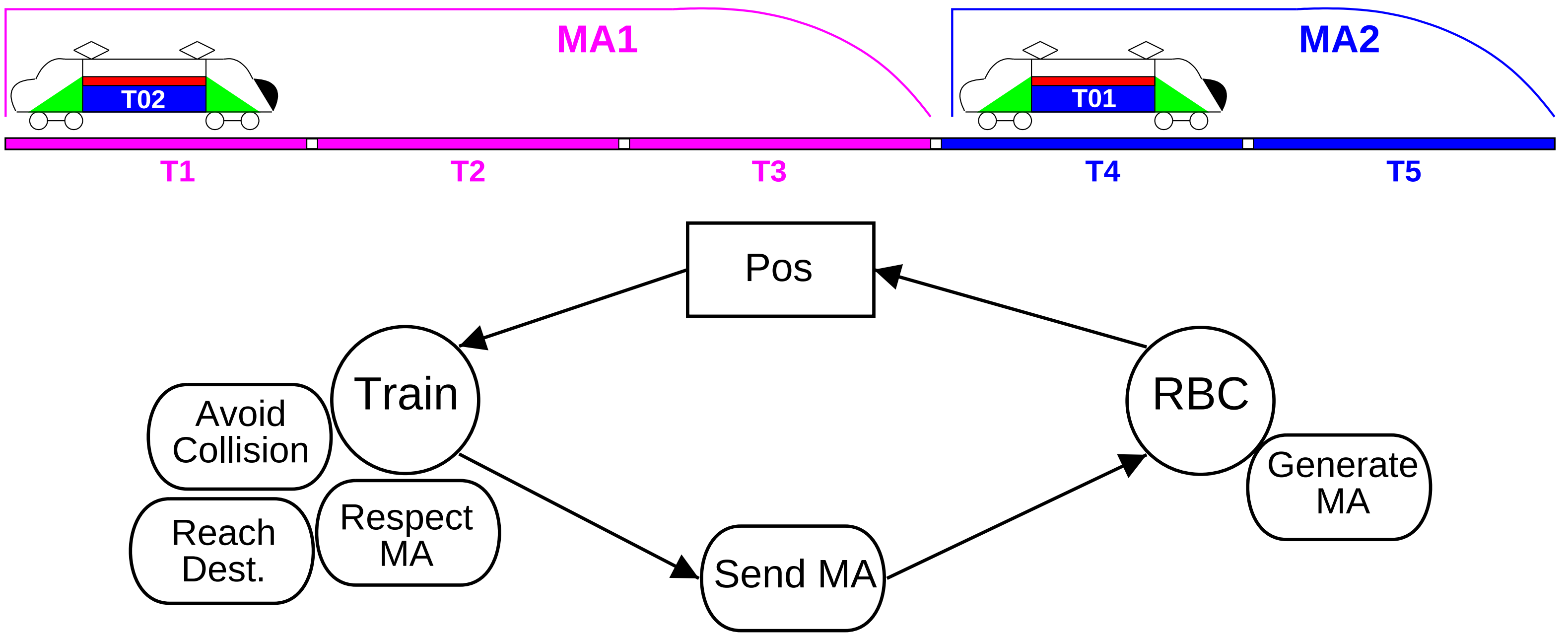
Applying FM to Early Requirements (II)

- Formal Methods in Early Requirements cannot be used to prove correctness of specifications.
- However they can . . .
  - show misunderstanding and omissions in the requirements that might not be evident in an informal setting.
  - assist the requirement elicitation by helping the interaction with stakeholders.
  - add expressive power to the requirement specification formalism.

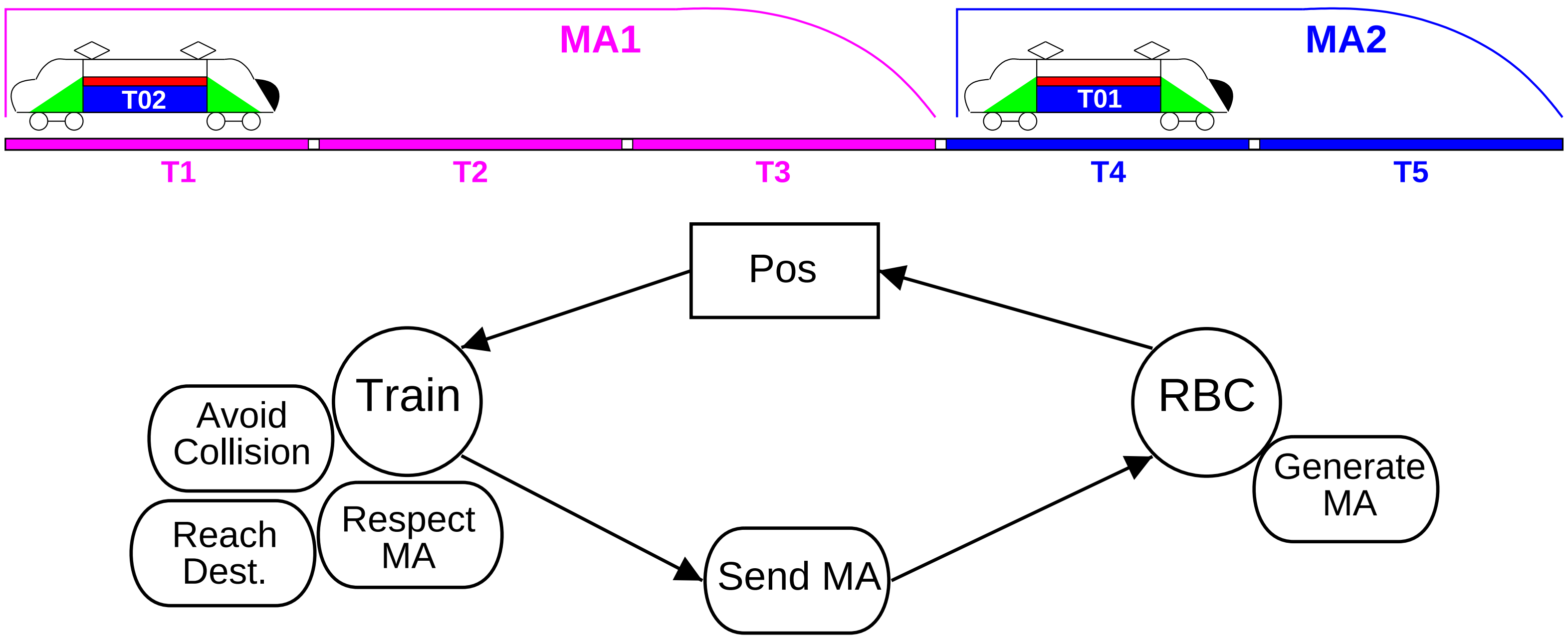
The RBC case study in Tropos



The RBC case study in Tropos

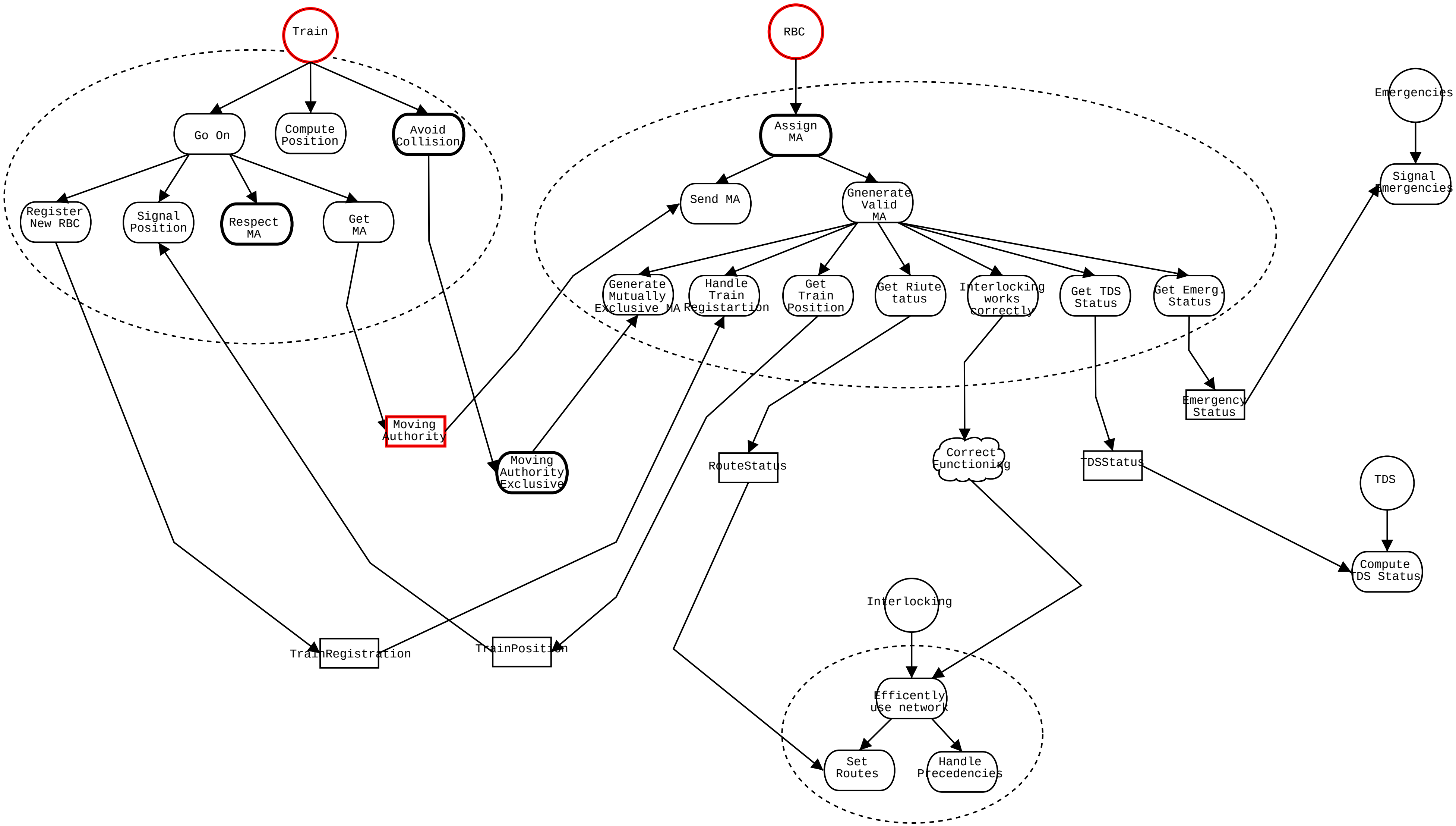


The RBC case study in Tropos



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

The RBC case study in Tropos (II)



The RBC in Formal Tropos

Actor Train

  Goal AvoidCollision

  Goal RespectMA

  Goal ReachDestination

Actor RBC

  Goal GenerateMA

Dependency Pos

  Type resource

  Depender RBC

  Dependee Train

Dependency SendMA

  Type Goal

  Depender Train

  Dependee RBC

The RBC in Formal Tropos (II)

Adding the “class” layer . . .

Entity Track
Entity MovingAuthority
  Attribute tracks : **set of** Track
Entity Position
  Attribute track : Track
Actor Train
  Goal AvoidCollision
  Goal RespectMovingAuthority
  Goal ReachDestination
  Attribute
    pos : Position
    ma : MovingAuthority
  . . . . . .

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.

The RBC in Formal Tropos (III)

Adding goal modalities . . .

Actor Train

Goal AvoidCollision

Mode maintain

. . .

Goal ReachDestination

Mode achieve

Dependency Pos

Type resource

Mode maintain

Depender RBC

Dependee Train

. . .

The RBC in Formal Tropos (IV)

Adding behavioral properties . . .

Actor Train

Goal AvoidCollision

Mode maintain

Fulfillment condition

exists rma : RespectMA (rma.**actor** = **self** $\wedge$ **Fulfilled**(rma))

. . .

Dependency SendMA

Type goal

Mode maintain

Depender Train **Dependee** RBC

Creation condition

exists gma : GenerateMA

(gma.**actor** = **dependee** $\wedge$ **Fulfilled**(gma))

. . .

Constraints Properties in Formal Tropos

- Constraint properties determine the possible evolutions of the objects in the specification.
- Constraint properties are specified with formulas given in a first-order linear-time temporal logic with past operators.
- 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.

Formal Analysis in Formal Tropos

Formal Tropos allows for the following kinds of formal analysis:

Formal Analysis in Formal Tropos

Formal Tropos allows for the following kinds of formal analysis:

- consistency check:
  - “the specification admits valid scenarios”.
  - “all the class will be created”.
  - . . .

Formal Analysis in Formal Tropos

Formal Tropos allows for the following kinds of formal analysis:

- consistency check:
  - “the specification admits valid scenarios”.
  - “all the class will be created”.
  - . . .
- possibility check: “there is *some* scenario for the system that respects certain **possibility** properties”.

Formal Analysis in Formal Tropos

Formal Tropos allows for the following kinds of formal analysis:

- consistency check:
  - "the specification admits valid scenarios".
  - "all the class will be created".
  - . . .
- possibility check: "there is *some* scenario for the system that respects certain **possibility** properties".
- assertion validation: "*all* scenarios for the system respect certain **assertion** properties".

Formal Analysis in Formal Tropos

Formal Tropos allows for the following kinds of formal analysis:

- consistency check:
  - “the specification admits valid scenarios”.
  - “all the class will be created”.
  - . . .
- possibility check: “there is *some* scenario for the system that respects certain **possibility** properties”.
- assertion validation: “*all* scenarios for the system respect certain **assertion** properties”.
- animation: the user interactively explores 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.

ITC irst

Possibility Check in Formal Tropos

A **possibility**:

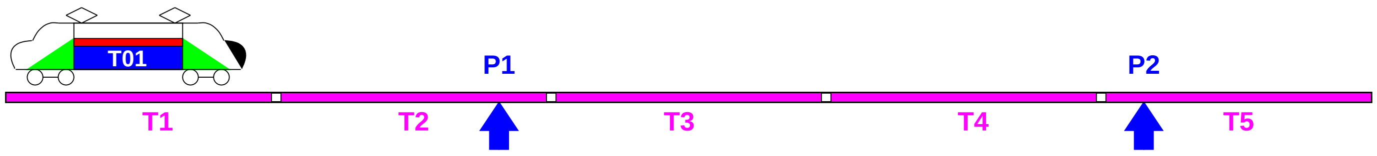
- describes *expected, valid* scenarios of the specification;
- is used to guarantee that the specification does not rule out any wanted execution of the system.

Possibility Check in Formal Tropos

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: “It is always possible for a train to achieve P1 and then P2”.



Global possibility

exists t : Train **F** ((t.pos = P1) $\wedge$ **F** (t.pos = P2))

Assertion Validation in Formal Tropos

An **assertion**:

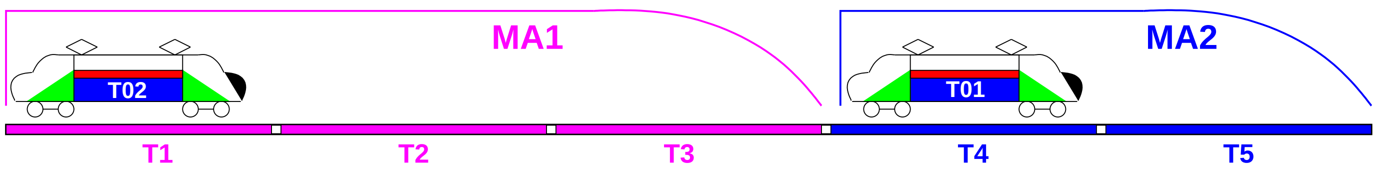
- describes *expected* conditions for all the valid scenarios;
- is used to guarantee that the specification does not allow for unwanted scenarios.

Assertion Validation in Formal Tropos

An **assertion**:

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

Example: “It is never the case that two different trains occupy the same position if they respect their moving authority.”



Global assertion

forall t1 : Train (**forall** t2 : Train

(t1 $\neq$ t2) $\wedge$ respectma(t1,t1.ma) $\wedge$ respectma(t2,t2.ma))

$\rightarrow$ (t1.pos $\neq$ t2.pos)))

The technical details

Our approach consists of the following 3 steps:

The technical details

Our approach consists of the following 3 steps:

1. The analyst writes a Formal Tropos specification.

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.

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. NuSMV performs the formal analysis on the Intermediate Language specification.

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. NuSMV performs the formal analysis on the Intermediate Language specification.

The Intermediate Language is:

- a small core language with a clean semantics.
- independent from Formal Tropos (the Intermediate Language may be applied to other requirements languages).
- independent from any particular analysis technique (model checking, LTL satisfiability, theorem proving).

So far we have...

So far we have. . .

- 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.

So far we have. . .

- 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.

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 (e.g. insurance company).
- heavy: it is difficult to write LTL specifications.

Ongoing Works

Ongoing Works

- Evaluation of the approach on a more complex case study to...
  - ... stress the scalability.
  - ... evaluate applicability and practicality (i.e. the RBC).

Ongoing Works

- Evaluation of the approach on a more complex case study to. . .
  - . . . stress the scalability.
  - . . . evaluate applicability and practicality (i.e. the RBC).
- Extension of the scope of the approach by. . .
  - . . . formalizing goal decomposition.

ITC irst

Ongoing Works

- Evaluation of the approach on a more complex case study to. . .
  - . . . stress the scalability.
  - . . . evaluate applicability and practicality (i.e. the RBC).
- Extension of the scope of the approach by. . .
  - . . . formalizing goal decomposition.
- Enhancement of the tool by . . .
  - . . . improving the interaction with the user.
  - . . . enhancing the animation techniques.
  - . . . developing specifically tailored verification algorithms.

Future Works

Future Works

- Extend the scope of the approach by allowing for the use of different specification languages (e.g. KAOS, UML, . . .).

Future Works

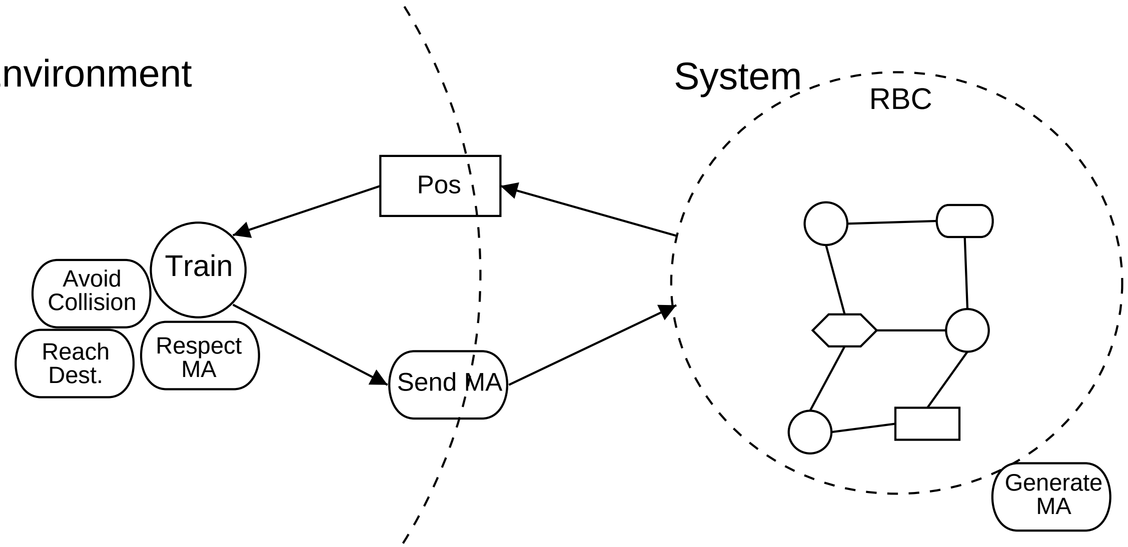
- Extend the scope of the approach by allowing for the use of different specification languages (e.g. KAOS, UML, . . .).
- Extend to later phases of the Tropos development process
  - same concepts / different interpretations / different V&V techniques.
  - automatic generation from the requirements.

Future Works

- Extend the scope of the approach by allowing for the use of different specification languages (e.g. KAOS, UML, . . .).
- Extend to later phases of the Tropos development process
  - same concepts / different interpretations / different V&V techniques.
  - automatic generation from the requirements.
- Integration of the informal and formal layer
  - graph transformations.

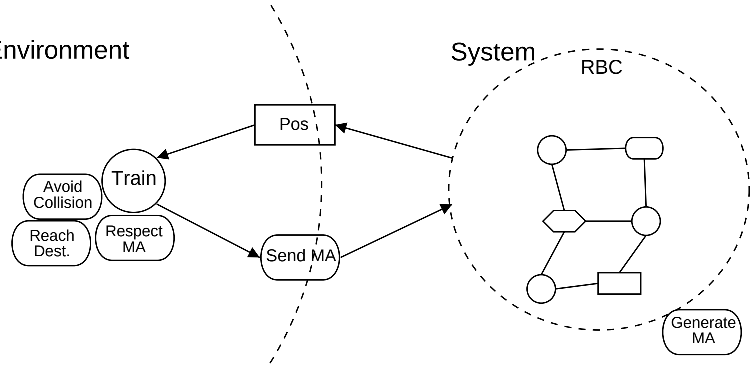
Formal Tropos and Late Requirements

- In Late Requirements the system component is explicitly added and it is refined by introducing new actors, goals and dependencies.



- In this phase the focus is in the refinement of the *system to be*.

Formal Tropos and Late Requirements



- The specification approach is similar to the approach used in early requirements, but here more details are added.
- Provided the system shows a “certain behavior at the interface” (*Assume*), then the environment works “correctly” (*Guarantee*).
- Verification/Validation can be performed using Assume/Guarantee reasoning.
- The process can be iterated within the system, thus allowing for a compositional approach.