Project DEPLOY
Grant Agreement 214158

"Industrial deployment of advanced system engineering methods for high productivity and dependability"

Goal-Oriented Requirements Engineering in Action in the Transportation Sector

Technical Report

28 May 2009

http://www.deploy-project.eu

Contributors:

Renaud De Landtsheer	CETIC
Christophe Ponsard	CETIC

Contents

1 Introduction

This report presents the goal-oriented approach to engineer high-quality requirements. It is centered on the KAOS approach, which is a good representative of goal-orientation [vL09]. It also includes some reference to non-functional requirements engineering approaches such as NFR, I* and Tropos [CNYM00, Yu95, BPG^{+}04]. Further information can be found in reference documents [vL09, CNYM00].

Starting from informally structured textual requirements, this report illustrates some of the benefits of using a goal-oriented approach in terms of guarantee of completeness, consistency, reasoning on the system boundaries, identifying the responsibilities and dealing with potential conflicts. This report is mainly on semi-formal notations but it also shortly considers the formalization process.

This report is consistently illustrated on the first simplified pilot problem developed by the transportation sector during the DEPLOY project and known as "mini-pilot" [F7 09]. The elaboration starts with an initial model extracted from the specification document and then proceeds according to the non-functional quality requirements included in this document to gradually make the requirements robust and compliant with these non-functional requirements. Finally, some formalization aspects are illustrated and discussed.

2 Structure of a typical goal-oriented requirements engineering language

We use here the KAOS language [vL09]. It is a modelling language composed of several sub-models related through concept sharing and *inter-model consistency rules*. **Figure 1** presents the key models and their inter relationships. In addition, UML-like class diagrams are also used to capture the structure of the domain. A, operation models is also available for the derived operational behaviour. This last model will not be considered here because it overlaps with Event-B. The aim is to focus only on the intentional part dealing with the requirements.

The *goal model* is the driving model. It declares the goals of the system-to-be. A goal defines an objective the system-to-be should meet, usually through the cooperation of multiple agents. For instance, a goal of the train control system is *Avoid[ATP mode on non-CBTC track]*. It states that "*The ATP mode should not be selected if the train is not on a CBTC track. This is because the ATP mode is fully automated and needs the presence of dedicated*

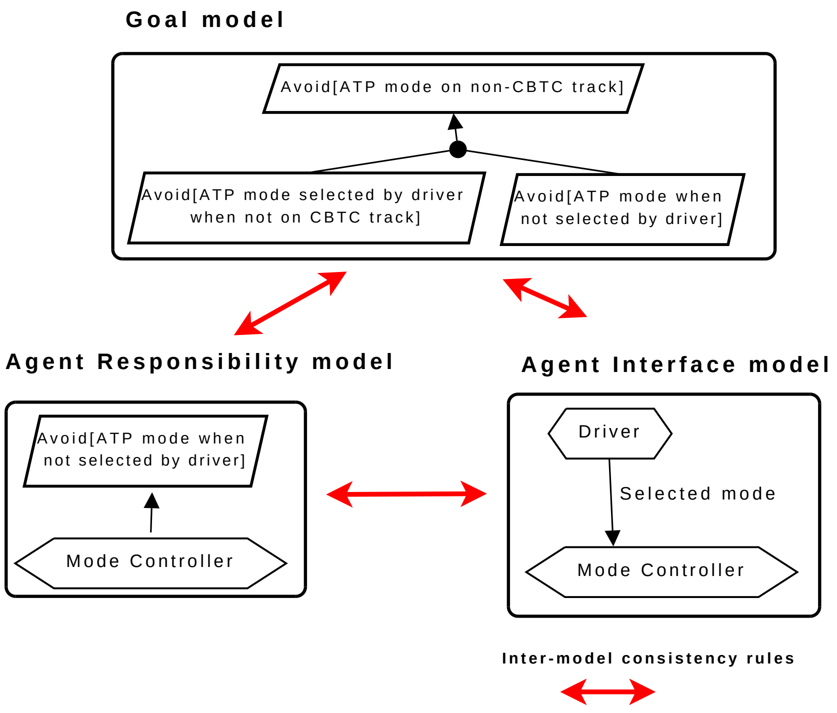
Goal model
Avoid[ATP mode on non-CBTC track]
Avoid[ATP mode selected by driver when not on CBTC track]
Avoid[ATP mode when not selected by driver]
Agent Responsibility model
Avoid[ATP mode when not selected by driver]
Mode Controller
Agent Interface model
Driver
Selected mode
Mode Controller
Inter-model consistency rules

Figure 1: Structure of the KAOS meta-model

device beside the track. This is called CBTC track. ". It is a safety goal.

Goal-refinement links relate a goal to a set of subgoals. A set of subgoals refines a parent goal if the satisfaction of all subgoals is sufficient for satisfying the parent goal [DvLF93, Dar95, DvL96]. As an example, the goal *Avoid[ATP mode on non-CBTC track]* is refined in **Figure 1** into the subgoals *Avoid[ATP selected when not selected by driver]* and *Avoid[ATP selected by driver when not on CBTC]*.

The *agent responsibility model* declares responsibility assignments of goals to agents. Agents include software components that exist or are to be developed, external devices, and humans in the environment. Responsibility assignment provides a criterion for stopping the goal refinement process. A goal assigned as the responsibility of a single agent is not refined any further. The meaning of a responsibility assignment is that the agent responsible for a goal is the only one required to restrict its behaviour so as to ensure that goal [Let01].

The *agent interface model* declares which objects are monitored and controlled by each agent. In the agent interface model of **Figure 1**, the Driver agent controls the Selected mode, which is a switch of the mode selector agent.

There is an *inter-model consistency rule* between the agent interface and the corresponding responsibility assignment of a goal to an agent. Roughly speaking, a goal can be assigned as the responsibility of an agent only if the goal is stated in terms of objects that are monitorable and controllable by the agent and if the agent can behave so that it enforces the goal. This is known as the realizability meta-constraint.

3 Setting up the context

The context includes the variables of the system, and the agents of the system, together with their control and monitoring capabilities. Generally, one distinguishes between the variables of the system, structured into an object model, and the agent interface model. In this case, the running example has no notion of entities or relationships, rather, it has a finite set of variables. We therefore present here a simplified version of our goal-oriented approach where we use variables instead of an entity relationship attribute model [BC95].

Agents are active objects such as humans, devices, or software components that play some role towards goal satisfaction. Some agents define the software-to-be whereas others define the environment. Our train control system includes the Driver and the Mode controller agents.

Agents are declared by a name and a definition. The particular meta-features of agents are that they can monitor and control objects, and take responsibility for goals.

The Driver and Mode selector agents are defined as follows:

Agent Driver
Definition The driver is a human agent that monitors the track state and can select a driving mode according to some non-formulated requirements.

Agent Mode Selector
Definition This is an automated agent that inputs user wishes about the driving mode and select the most appropriate one. It also controls an emergency brake.

The system includes the following variables:

Variable DesiredTrainMode
Definition A switch that is controlled by the driver and part of the Mode Selector. It can have three possible values: ATP, ATPR, and Bypass.
Type: enumeration: {ATP, ATPR, Bypass}

Variable EmergencyBrake
  Definition To stop the train in emergency situations.
  Type: Boolean

Variable TrainMode
  Definition This is the mode that is applied to the speed regulator of the train. It can have three possible values: ATP, ATPR, and Bypass.
  Type: enumeration: {ATP, ATPR, Bypass}

Agent interfaces are declared through Monitoring and Control links between agents and concepts of the domain, here, variables.

The meaning of a *Monitoring* link between an agent and an object attribute is that the agent directly monitors ("reads") the value of the attribute.

The meaning of a *Control* link between an agent and an object attribute is that the agent directly controls ("writes") the value of the attribute. In other words, an agent controls an attribute if it is capable of controlling state transitions for that attribute. We also consider that attributes controlled by an agent are observable by that agent as well.

As an example, the Driver controls the position of the DesiredTrainMode switch of the mode selector, and the Mode selector monitors the position of this switch.

The control and monitoring capabilities of the agents are presented in **Figure 2**. The adopted syntax is the one of the KAOS approach from [Let01]. Agents are represented by flattened hexagons, variables are represented by rectangles. An arrow leading from a variable to an agent denotes that the agent monitors the variable (i.e.: sees the variable), and an arrow leading from an agent to a variable denotes that the agent controls the value of the variable (i.e.: chooses its value).

Notice that context diagrams presented here cover part of the Problem Frame concepts, with a more recent syntax [Jac01]. For examples, the phenomena of problem frames are renamed into variables here. Other parts will be covered in later sections.

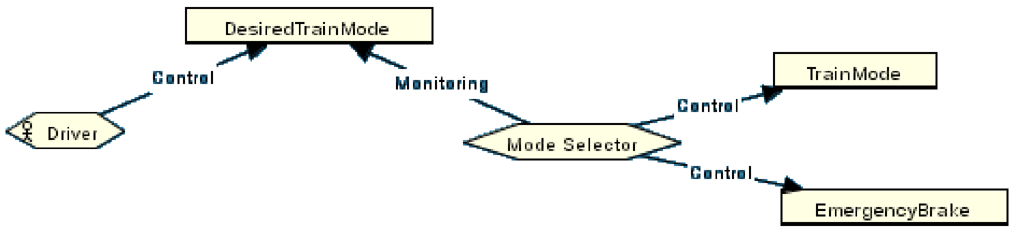

Figure 2: Context diagram

4 High-level goals and initial model

A *goal* is a prescriptive statement of intent about the considered system-to-be [vL01]. For example, *Maintain[TrainMode Equal To Driver mode]* is a functional goal of the train control system.

Each goal has a name and a definition. The *name* of the goal is used to identify the goal. The *definition* of the goal is the statement represented by the goal expressed in natural language. Goals can also be defined formally with the adjunction of a *formal def* field. The formal definition is generally expressed in first order temporal logics. This is not covered in this report, please refer to the reference document for further information [Let01, Lan07, vL09].

The goal *Maintain[TrainMode Equal To Driver mode]* is fully defined as follows:

Goal Maintain[TrainMode Equal To Driver mode]
Definition The train mode should be the one selected by the driver. This is a functional goal.

The *pattern* of a goal is based on the temporal behaviour required by the goal. In the case of the KAOS language, the following four goal patterns are defined:

- **Achieve goals:** goals requiring that some property eventually holds
- **Cease goals:** goals requiring that some property eventually stops to hold
- **Maintain goals:** goals requiring that some property always holds
- **Avoid goals:** goals requiring that some property never holds

Goal patterns provide a lightweight way of declaring the temporal behaviour of a goal without writing formal goal definitions. The pattern of a goal is often used as the first part of the goal name, like in the goal here above.

Another goal of the system is defined as follows. This is a safety goal.

Goal Avoid[ATP Mode on non-CBTC track]
Definition The ATP mode should not be selected if the train is not on a CBTC track. This is because the ATP mode is fully automated and needs the presence of dedicated device beside the track. This is called CBTC track.

AND-refinement links relate a goal to a set of sub goals (called refinement); this means that satisfying all sub goals in the refinement is a sufficient condition in the domain for satisfying the goal. A goal can be AND-refined in several alternative ways. In this case, these are considered as alternative refinements and one of these alternatives must be chosen. For example, the goal *Avoid[ATP Mode on non-CBTC track]* is refined into the two following sub-goals:

Requirement Avoid[ATP selected when not selected by driver]
  Definition The Train mode should never be ATP if it is not the mode selected by the driver

Requirement Avoid[ATP selected by driver when not on CBTC]
  Definition The train mode should never be ATP if it is not on a CBTC track

A set of goals $\{G_1, \ldots, G_n\}$ AND-refines (or refines for short) a goal G in a domain theory *Dom* if the following three conditions hold [Dar95]:

- **completeness:** the satisfaction of the sub goals together with the satisfaction of domain properties in *Dom* is sufficient for satisfying the parent goal.
- **minimality:** if a sub goal is left out of the refinement, the remaining sub goals are not sufficient for satisfying the parent goal.
- **consistency:** the conjunction of the sub goals is logically consistent with the domain theory.

The formal definition of goals allows one to verify formally the completeness, minimality and consistency of goal refinements. The completeness and minimality conditions can be checked via model checking [PMR$^+$04].

There is also two other high-level goals that will drive our analysis. The first one is a non-functional goals, meaning that it cannot be expressed in term of a condition on the variables of the system.

Non-Functional Goal Driver Cannot be relied upon for safety goals
  Definition The driver cannot be relied upon for the enforcement of safety goal. Mistakes from him should not hinder the safety of the train; high-level security goals should be enforced even in the presence of human error.

Goal Emergency Brakes On When Error Made By Driver
  Definition Whenever the driver makes a mistake with respect to a safety requirements he is responsible for, the emergency brakes should be activated.

The whole initial picture is summarized in **Figure 3**. Goals are represented by parallelograms inclined on the right. Refinements are denoted by a bubble linking the parent goal with an arrow, and the child goals with regular lines. A line linking an agent to a goal denotes that the goal is a requirement under the responsibility of the agent.

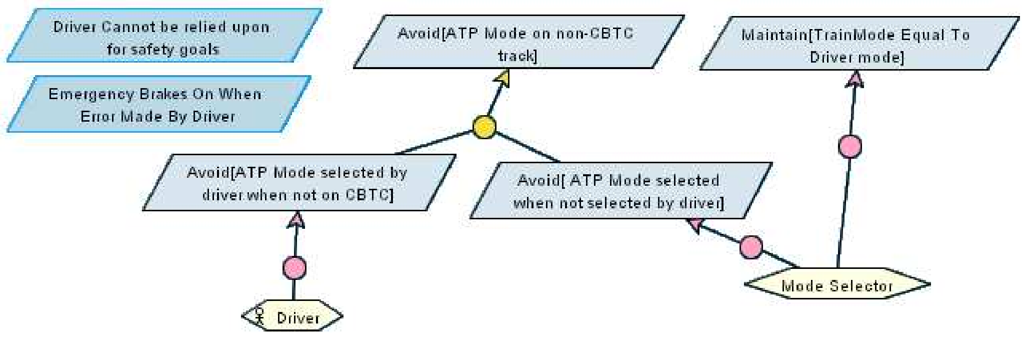
Driver Cannot be relied upon for safety goals
Emergency Brakes On When Error Made By Driver
Avoid[ATP Mode on non-CBTC track]
Maintain[TrainMode Equal To Driver mode]
Avoid[ATP Mode selected by driver when not on CBTC]
Avoid[ATP Mode selected when not selected by driver]
Mode Selector
Driver

Figure 3: Initial goal-oriented model of the train control system

5 Analyzing obstacles to safety requirements

First-sketch specifications of goals, requirements and assumptions tend to be too ideal; they are likely to be occasionally violated in the running system due to unexpected agent behaviour [vLL00]. The objective of obstacle analysis is to anticipate exceptional behaviours in order to derive more complete and realistic goals, requirements and assumptions.

In the train control system, obstacle analysis can be guided by the non-functional goal *Driver Cannot be relied upon for safety goals.*

These failures are modelled in requirements models through the concept of obstacle. *Obstacles* are a dual notion to goals; while goals capture desired conditions, obstacles capture undesirable (but nevertheless possible) ones. An obstacle obstructs some goal, that is, when the obstacle gets true the goal may not be achieved. The term "obstacle" denotes a goal-oriented abstraction, at the requirements engineering level, of various notions that have been studied extensively in specific areas, such as hazards that may obstruct safety goals [Lev95] or threats that may obstruct security goals [Amo94, vL04], or in later phases of the software lifecycle, such as faults that may prevent a program from achieving its specification [CL95, Gar99].

We focus here on the requirement *Avoid[ATP selected by driver when not on CBTC]* that is under the responsibility of the *driver*.

The obstacle is rather trivial in this case, as it is the negation of the requirement:

Obstacle ATP selected when not on CBTC
Definition The mode selected by the driver might be ATP and at the same time, the train might not be on a CBTC track

This obstacle is resolved through the introduction of the additional requirement:

Requirement ATPR mode selected when driver desires ATP on non CBTC
Definition When the mode desired by the driver is ATP and the train is not on a CBTC track, the train mode should be the default safe ATPR mode

On the other hand, the situation described by the obstacle is considered as an error made by the driver. According to the non-functional requirement *Emergency Brakes On When Error Made By Driver*, this situation must trigger the emergency brakes. We therefore introduce the following additional requirement:

Requirement Emergency Brakes On When ATP selected By Driver On Non CBTC Track
Definition The emergency brakes should be activated when the mode selected by the driver is ATP and the train is on a non-CBTC track. This situation is considered as an error of the driver.

This analysis together with the resolution of the obstacle are summarized in **Figure** 4. Obstacles are denoted by parallelograms inclined on the left. Obstruction of goals by obstacle is denoted by a red-marked arrow leading from the obstacle to the goal. The resolution of and obstacle by a goal is denoted by a green-marked arrow leading from the goal to the obstacle.

6 Detecting and resolving inconsistencies

This section introduces the concepts of inconsistencies between goals. A divergence between goals corresponds to situations where some particular combination of circumstances can be found that makes the goals logically inconsistent. Such a particular combination of circumstances is called a *boundary condition* [vLDL98, Lan07]. Inconsistencies can be automatically detected

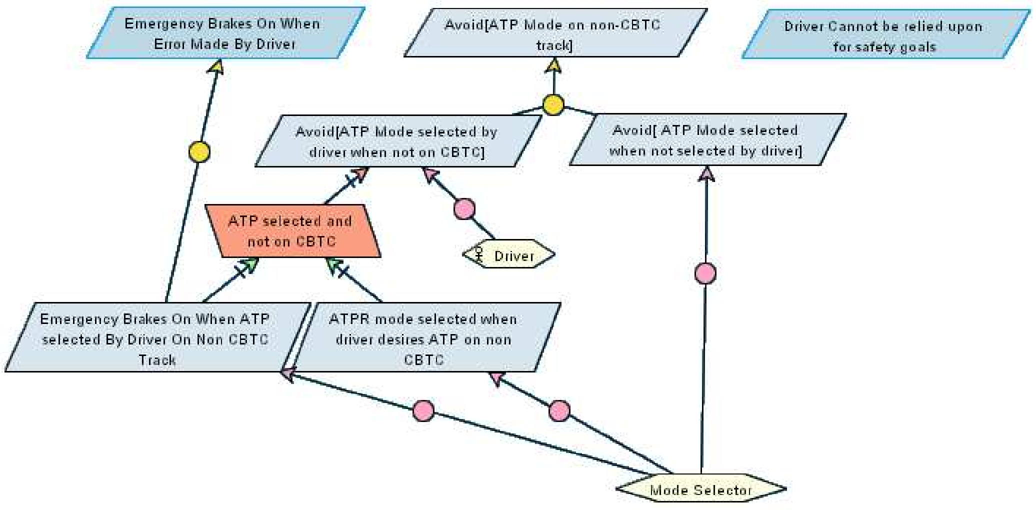
Emergency Brakes On When Error Made By Driver
Avoid[ATP Mode on non-CBTC track]
Driver Cannot be relied upon for safety goals
Avoid[ATP Mode selected by driver when not on CBTC]
Avoid[ATP Mode selected when not selected by driver]
ATP selected and not on CBTC
Driver
Emergency Brakes On When ATP selected By Driver On Non CBTC Track
ATPR mode selected when driver desires ATP on non CBTC
Mode Selector

Figure 4: Analysing unexpected conditions

by using dedicated formal methods [Lan07, vLDL98].

We need to check that there is no inconsistencies in the requirements model. Typically, functional requirements and safety requirements are often conflicting. In this case study, there is a conflict between the requirements *ATPR mode selected when driver desires ATP on non CBTC* and *Maintain[TrainMode Equal To Driver mode]*. The conflict arises when the driver selects ATP mode on a non-CBTC track. The conflict is defined as follows:

Conflict ATP selected on non-CBTC
Involved goal ATPR mode selected when driver desires ATP on non CBTC
Involved goal Maintain[TrainMode Equal To Driver mode]
Definition When the driver selects ATP on a non-CBTC track, the train mode should be ATPR. On the other hand, according to the goal Maintain[TrainMode Equal To Driver mode], they should always be equal.

This conflict is resolved by weakening the functional goal *Maintain[TrainMode Equal To Driver mode]* into the following one:

Goal Maintain[Train Mode Equal To Driver Mode Except If ATP selected and non-CBTC]
Definition The train mode should be equal to the one selected by the driver except if this selected mode is ATP and the train is not on a CBTC track.

The goal *Maintain[TrainMode Equal To Driver mode]* was not involved in any refinement. Should it be the case, we would need to transfer all the refinements from this goal to its refined version, and possibly, propagate the weakening along the goal graph.

This analysis is summarized in **Figure 5**. Divergences and conflicts are graphically represented using the same notation as obstacle, except that they relate to several goals with a red-marked arrow.

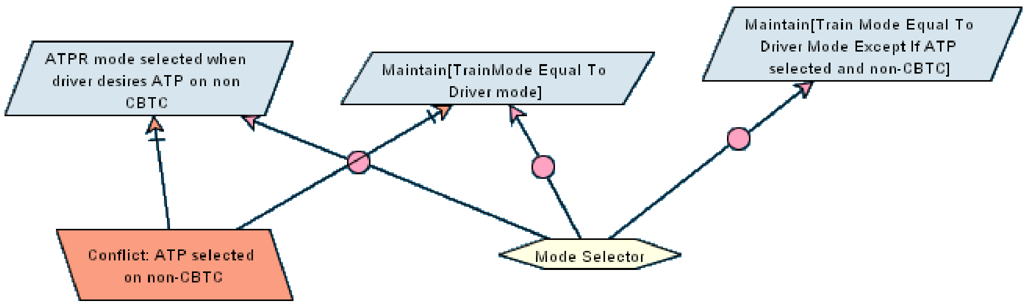
ATPR mode selected when driver desires ATP on non CBTC
Maintain[TrainMode Equal To Driver mode]
Maintain[Train Mode Equal To Driver Mode Except If ATP selected and non-CBTC]
Conflict: ATP selected on non-CBTC
Mode Selector

Figure 5: Analysing and resolving conflicts in the train controller model

7 Final requirements model

The requirements under responsibility of the *Mode selector* are summarized below. We focus on these requirements because they will be implemented, as the *Mode selector* is a software agent. Requirements under the responsibility of the Driver will need to be incorporated into some user procedure, and possibly into some training.

Requirement Avoid[ATP selected when not selected by driver]
Definition The Train mode should never be ATP if it is not the mode selected by the driver

Requirement ATPR mode selected when driver desires ATP on non CBTC
Definition When the mode desired by the driver is ATP and the train is not on a CBTC track, the train mode should be the default safe ATPR mode

Requirement Emergency Brakes On When ATP selected By Driver On Non CBTC Track
Definition The emergency brakes should be activated when the mode selected by the driver is ATP and the train is on a non-CBTC track. This situation is considered as an error of the driver.

Requirement Maintain[Train Mode Equal To Driver Mode Except If ATP selected and non-CBTC]

Definition The train mode should be equal to the one selected by the driver except if this selected mode is ATP and the train is not on a CBTC track.

The final model displaying only the requirements is shown on **Figure 6**.

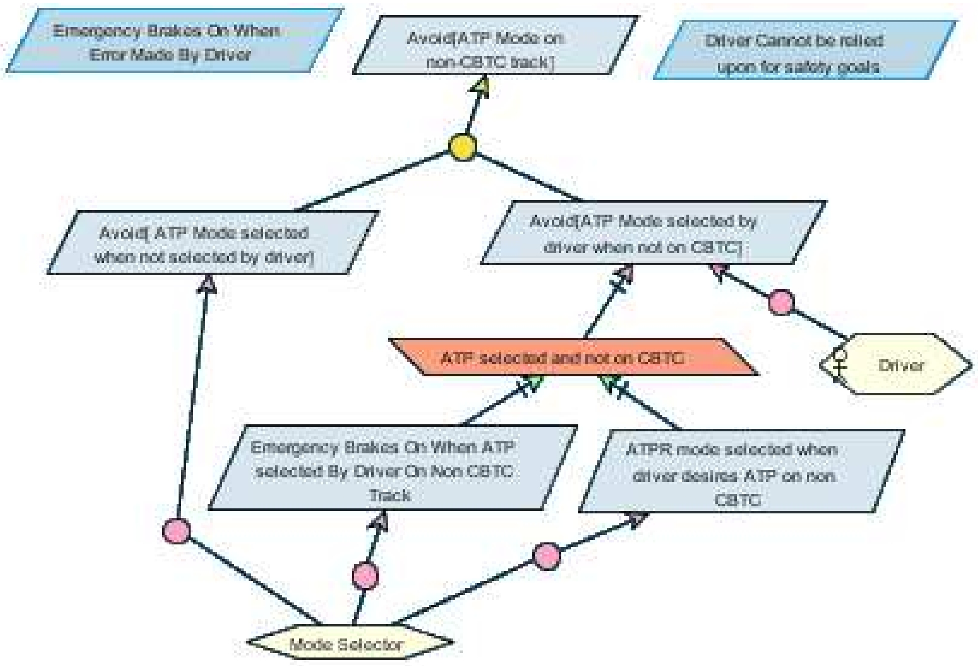
Emergency Brakes On When Error Made By Driver
Avoid[ATP Mode on non-CBTC track]
Driver Cannot be relied upon for safety goals
Avoid[ATP Mode selected when not selected by driver]
Avoid[ATP Mode selected by driver when not on CBTC]
ATP selected and not on CBTC
Driver
Emergency Brakes On When ATP selected By Driver On Non CBTC Track
ATPR mode selected when driver desires ATP on non CBTC
Mode Selector

Figure 6: Final set of requirements for the train control system

8 Why goal-orientation for requirements engineering?

This section borrows from [vL01]. The goal-oriented modelling paradigm is particularly well-suited for requirements engineering because it can provably reach the purpose of requirements engineering, namely:

- Achieve requirements completeness. This is a major concern of requirements engineering. Completeness is about ensuring that all the necessary requirements are identified. Several definitions of completeness have been proposed [Yue87, Lan07]. Goals provide a precise criterion for sufficient completeness of a requirements specification; the specification is complete with respect to a set of goals if all the goals can be proved to be achieved from the specification and the properties known about the domain considered [Yue87, Lan07].

- Detect, represent and resolve conflicts between requirements [NKF94, vLDL98, Lan07].
- Support the identification of unexpected conditions, and drive the elaboration of robust requirements that take into account those unexpected conditions [vLL00].
- Avoid irrelevant requirements. Goals provide a precise criterion for requirements pertinence; a requirement is pertinent with respect to a set of goals in the domain considered if its specification is used in the proof of one goal at least [Yue87, vL01].
- Explain requirements to stakeholders. Goals provide the rationale for requirements, in a way similar to design goals in design processes [Lee91]. A requirement appears because of some underlying goal which provides a base for it [DFvL91, SS97]. More explicitly, a goal refinement tree provides traceability links from high-level strategic objectives to low-level technical requirements. In particular, for business application systems, goals may be used to relate the software-to-be to organizational and business contexts [Yu95].
- Provide a natural mechanism for structuring complex requirements documents for increased readability.
- Provide support for the exploration of alternative during the requirements elaboration process. Alternative goal refinements allow alternative system proposals to be explored [vL00].
- Goals drive the identification of requirements to support them; they have been shown to be among the basic driving forces, together with scenarios, for a systematic requirements elaboration process [DFvL91, RG92, DvLF93, AP98, EDP98, Kai00, vL00].

9 Formal support for requirements elaboration

Formal methods can be usefully deployed at requirements time, to assist the requirements engineer in various tasks. Through formal methods, it is possible to more deeply check requirements completeness, consistency, and minimality [PMR+04, vLL00, vLDL98, Lan07]. A formalized specification can also be usefully animated, so that users of the future system can perform hands on validation of the system to be through animation sessions [VvLMP04, HABJ05]. Specifications can also be inferred through automated user interrogation, thanks to machine learning techniques [DLD05].

Generally (except for machine learning techniques), the first step towards formal support is to formalize the goals and requirements. They are generally formalized in first order temporal logics. The concepts referenced in these formalization are defined in a model of the context of the system to be. Generally, this is an entity relationship attribute model, that is represented through UML-like graphical notations [Let01, Obj].

The formal definitions of goals is attached to the concepts of goals in a dedicated field *formal def.* For instance, the goal *Avoid[ATP selected when not selected by driver]* is formally defined as follows:

Requirement Avoid[ATP selected when not selected by driver]
Definition The Train mode should never be ATP if it is not the mode selected by the driver
Formal Def $\text{TrainMode} = \text{ATP} \Rightarrow \text{DriverSelectedTrainMode} = \text{ATP}$

The formalization captures the informal definition; $P \Rightarrow Q$ is a shortcut for $\Box(P \implies Q)$, and $\Box$ is the *always* operator from temporal logics.

One can also specify real-time properties in temporal logics, using real-time temporal operators. For instance, the requirement *Emergency Brakes On When ATP selected By Driver On Non CBTC Track* prescribes some reaction. We can suggest a time bound for this reaction, say 300ms. The formalized requirements is as follows:

Requirement Emergency Brakes On When ATP selected By Driver On Non CBTC Track
Definition The emergency brakes should be activated when the mode selected by the driver is ATP and the train is on a non-CBTC track. This situation is considered as an error of the driver.
Formal Def $\text{DriverSelectedTrainMode} = \text{ATP} \land \neg\text{OnCBTC}$
$\Rightarrow \Diamond_{\leq 300ms}\text{EmergencyBrakesOn}$

Built on these definition one can check refinements:

- testing the completeness of refinements can be done through temporal model-checking techniques [PMR$^+$04]
- testing the consistency of requirements models requires realizability testing engines [vLDL98, Lan07]

- potential obstacles can also be inferred automatically through regression procedures [vLL00]
- through a proper compilation process that generates state machines out of declarative requirements, it is also possible to animate requirements models [VvLMP04]

Formal methods enable one to:

- Reach a higher level of assurance in the correctness of the requirements document. This is especially valuable for system-wide reasoning, such as the detection of inconsistencies, that the human mind can difficultly handle
- Automate the verification of the model, thus discharging the requirements engineer from heavy verification tasks
- support extra functionalities such as animation and inductive learning of requirements [DLD05, VvLMP04, HABJ05]

10 From formal requirements to Event-B specifications

To support the Event-B formalisation process, Event-B models have to be derived from declarative requirements. We report here some ideas and on-going work on this topic.

Intuitively, the Event-B context and state declaration are extracted from the context model such as the one displayed on **Figure 2**. Invariants of the Event-B model are identified and formalised from requirements with the proper temporal pattern (i.e "safety" requirements characterised with a single outern "always" temporal operator). This approach was already discussed in [pon06] in the context of B.

Deriving Event-B event declarations is more complex and there is on-going work on this topic such as based on patterns approach [BAA+08, AAB+09]. It requires some extension of the Event-B to fully take care of progress properties (captured by the "eventual" temporal symbol). An alternative, more generative, approach has also been studied by the FP6 GridTrust project with which DEPLOY is collaborating [FP609]. A restriction is however the use of past LTL only. So far is has been applied to the POLPA policy language [AAM+08], but Event-B could be considered as target language. Finally there is also some on-going work based goal refinement patterns [MGL08].

Some non-functional requirements can also be captured in Event-B, as they can constraint the interface of the system. Those must be studied on a case-by-case basis.

References

[AAB+09] Benjamin Aziz, Alvaro Arenas, Juan Bicarregui, Christophe Ponsard, and Philippe Massonet. From goal-oriented requirements to event-b specifications. In *proceedings of the First NASA Formal Methods Symposium (NFM 2009)*, April 6-8 2009.

[AAM+08] Benjamin Aziz, Alvaro Arenas, Fabio Martinelli, Ilaria Matteucci, and Paolo Mori. Controlling usage in business process workflows through fine-grained security policies. In *TrustBus '08: Proceedings of the 5th international conference on Trust, Privacy and Security in Digital Business*, pages 100–117, Berlin, Heidelberg, 2008. Springer-Verlag.

[Amo94] E.J. Amoroso. *Fundamentals of Computer Security.* Prentice Hall, 1994.

[AP98] A.I. Anton and C. Potts. The use of goals to surface requirements for evolving systems. In *Proc. ICSE-98: 20th Intrnational Conference on Software Enginering*, Kyoto, April 1998.

[BAA+08] Juan Bicarregui, Alvaro Arenas, Benjamin Aziz, Philippe Massonet, and Christophe Ponsard. Towards modelling obligations in event-b. In *ABZ*, pages 181–194, 2008.

[BC95] Robert H. Bourdeau and Betty H. C. Cheng. A formal semantics for object model diagrams. *IEEE Trans. Softw. Eng.*, 21(10):799–821, 1995.

[BPG+04] Paolo Bresciani, Anna Perini, Paolo Giorgini, Fausto Giunchiglia, and John Mylopoulos. Tropos: An agent-oriented software development methodology. *Autonomous Agents and Multi-Agent Systems*, 8(3):203–236, 2004.

[CL95] F. Cristian and M.R. Lyu. *Software Fault Tolerance*, chapter Exception Handling. Wiley, 1995.

[CNYM00] L. Chung, B. A. Nixon, E. Yu, and J. Mylopoulos. *Non-Functional Requirements in Software Engineering.* Kluwer Academic Publishers, Boston, 2000.

[Dar95] R. Darimont. *Process Support for Requirements Elaboration.* PhD thesis, Université catholique de Louvain, Dépt. Ingénierie Informatique, Louvain-la-Neuve, Belgium, 1995.

[DFvL91] A. Dardenne, S. Fickas, and A. van Lamsweerde. Goal-directed concept acquisition in requirements elicitation. In *Proc. IWSSD-6 - 6th Intl. Workshop on Software Specification and Design*, pages 14–21, Como, 1991.

[DLD05] Christophe Damas, Bernard Lambeau, and Pierre Dupont. Generating annotated behavior models from end-user scenarios. *IEEE Trans. Softw. Eng.*, 31(12):1056–1073, 2005. Member-Axel van Lamsweerde.

[DvL96] R. Darimont and A. van Lamsweerde. Formal refinement patterns for goal-driven requirements elaboration. In *Proc. FSE4 - Fourth ACM SIGSOFT Symp. on the Foundations of Software Engineering*, pages 179–190, San Francisco, October 1996.

[DvLF93] A. Dardenne, A. van Lamsweerde, and S. Fickas. Goal-directed requirements acquisition. *Science of Computer Programming*, 20:3 – 50, 1993.

[EDP98] E. Yu E. Dubois and M. Petit. From early to late formal requirements: A process-control case study. In *Proc. IWSSD'98 - 9th International Workshop on Software Specification and Design*, pages 34 – 42. IEEE CS Press, April 1998.

[F7 09] F7 DEPLOY consortium. Join Deliverable 1 - Report on Knowledge Transfer, March 2009.

[FP609] FP6 GridTrust Consortium. Framework for Reasoning about Trust ans Security in Grids at Requirement and Application Levels. public deliverable D4.1, 2009.

[Gar99] F.C. Gartner. Fundamentals of fault-tolerant distributed computing in asynchronous environment. *ACM Computing Surveys*, 31(1):1–26, March 1999.

[HABJ05] Constance Heitmeyer, Myla Archer, Ramesh Bharadwaj, and Ralph Jeffords. Tools for constructing requirements specifications: The scr toolset at the age of ten. *International Journal of Computer Systems Science and Engineering*, 20:19 – 35, January 2005.

[Jac01] Michael Jackson. *Problem Frames*. Addison Wesley, 2001.

[Kai00] H. Kaindl. A design process based on a model combining scenarios with goals and functions. *IEEE Trans. on Systems, Man and Cybernetic*, 30(5):537 – 551, September 2000.

[Lan07] Renaud De Landtsheer. *Elaborating Complete and Consistent Requirements for Security-Critical Systems*. PhD thesis, Université Catholique de Louvain, Dépt. Ingénierie Informatique, Louvain-la-Neuve, Belgium, June 2007.

[Lee91] J. Lee. Extending the potts and bruns model for recording design rationale. In *Proc. ICSE-13 - 13th Intl. Conf. on Software Engineering*, pages 114 – 125. IEEE-ACM, 1991.

[Let01] E. Letier. *Reasoning about Agents in Goal-Oriented Requirements Engineering*. PhD thesis, Université Catholique de Louvain, Dépt. Ingénierie Informatique, Louvain-la-Neuve, Belgium, May 2001.

[Lev95] N.G. Leveson. *Safeware - System Safety and Computers*. Addison-Wesley, 1995.

[MGL08] Abderrahman Matoussi, Frédéric Gervais, and Régine Laleau. A first attempt to express kaos refinement patterns with event b. In *ABZ*, page 338, 2008.

[NKF94] B. Nuseibeh, J. Kramer, and A. Finkelstein. A framework for expressing the relationships between multiple views in requirements specifications. *IEEE Transactions on Software Engineering*, 20(10):760 – 773, October 1994.

[Obj] Object Management Group. Unified Modeling Language. http://www.uml.org.

[PMR+04] Ch. Ponsard, P. Massonet, A. Rifaut, J.F. Molderez, A. van Lamsweerde, and H. Tran Van. Early verification and validation of mission-critical systems. In *Proceedings of FMICS'04,* *9th International Workshop on Formal Methods for Industrial Critical Systems*, Linz (Austria), 2004.

[pon06] From requirements models to formal specifications in b. In *Proc. International Workshop on regulations Modelling and their Validation and Verification (REMO2V)*, June 2006.

[RG92] K.S. Rubin and A. Goldberg. Object behavior analysis. *Communications of the ACM*, 35(9):48 – 62, September 1992.

[SS97] I. Sommerville and P. Sawyer. *Requirements Engineering: A Good Practice Guide.* Wiley, 1997.

[vL00] A. van Lamsweerde. Requirements engineering in the year 00: A research perspective. In *Invited Paper for ICSE'2000 - 22nd International Conference on Software Engineering,ACM Press*, Limerick, 2000.

[vL01] Axel van Lamsweerde. Goal-oriented requirements engineering: A guided tour. In *RE'01 - 5th IEEE International Symposium on Requirements Engineering*, pages 249–263, Toronto, August 2001.

[vL04] Axel van Lamsweerde. Elaborating security requirements by construction of intentional anti-models. In *Proceedings of the 26th International Conference on Software Engineering*, pages 148–157. IEEE Computer Society, 2004.

[vL09] Axel van Lamsweerde. *Requirements Engineering From System Goals to UML Models to Software Specifications.* Wiley, January 2009. ISBN: 978-0-470-01270-3.

[vLDL98] A. van Lamsweerde, R. Darimont, and E. Letier. Managing conflicts in goal-driven requirements engineering. *IEEE Transactions on Software Engineering, Special Issue on Managing Inconsistency in Software Development*, pages 908 – 926, November 1998.

[vLL00] A. van Lamsweerde and E. Letier. Handling obstacles in goal-oriented requirements engineering. *IEEE Transactions on Software Engineering, Special Issue on Exception Handling*, 26(10):978 – 1005, October 2000.

[VvLMP04] H. Tran Van, A. van Lamsweerde, P. Massonet, and Ch. Ponsard. Goal-oriented requirements animation. In *Proceedings of RE'04, 12th IEEE Joint International Requirements Engineering Conference*, pages 218–228, Kyoto, 2004.

[Yu95] Eric Yu. *Modelling Strategic Relationships for Process Reengineering*. PhD thesis, Dept. of Computer Science, University of Toronto, 1995.

[Yue87] K. Yue. What does it mean to say that a specification is complete? In *Proc. IWSSD-4, Fourth International Workshop on Software Specification and Design*, 1987.