Agent-Based Tactics for Goal-Oriented Requirements Elaboration

Emmanuel Letier and Axel van Lamsweerde

Département d'Ingénierie Informatique
Université catholique de Louvain
B-1348 Louvain-la-Neuve (Belgium)
{eletier,avl}@info.ucl.ac.be

Abstract

Goal orientation is an increasingly recognized paradigm for eliciting, structuring, analyzing and documenting system requirements. Goals are statements of intent ranging from high-level, strategic concerns to low-level, technical requirements on the software-to-be and assumptions on its environment. Achieving goals require the cooperation of agents such as software components, input/output devices and human agents. The assignment of responsibilities for goals to agents is a critical decision in the requirements engineering process as alternative agent assignments define alternative system proposals.

The paper describes a systematic technique to support the process of refining goals, identifying agents, and exploring alternative responsibility assignments. The underlying principles are to refine goals until they are assignable to single agents, and to assign a goal to an agent only if the agent can realize the goal.

There are various reasons why a goal may not be realizable by an agent, e.g., the goal may refer to variables that are not monitorable or controllable by the agent. The notion of goal realizability is first defined on formal grounds; it provides a basis for identifying a complete taxonomy of realizability problems. From this taxonomy we systematically derive a catalog of tactics for refining goals and identifying agents so as to resolve realizability problems. Each tactics corresponds to the application of a formal refinement pattern that relieves the specifier from verifying the correctness of refinements in temporal logic.

Our techniques have been used in two case studies of significant size; excerpts are shown to illustrate the main ideas.

Keywords
Requirements realizability, goal-oriented requirements engineering, specification refinement, reasoning about agent responsibilities.

1. INTRODUCTION

Requirements engineering (RE) is concerned with the identification of goals to be achieved by the envisioned system, the refinement of such goals and their operationalization into specifications of services and constraints, and the assignment of responsibilities for the resulting requirements to agents such as humans, devices and software [Lam00c].

Proceedings ICSE'2002,
24th International Conference on Sofware Engineering,
ACM Press, May 2002

Goal-oriented RE refers to the use of goals for requirements elicitation, elaboration, organization, specification, analysis, negotiation, documentation and evolution [Lam01].

Goals are objectives to be achieved by the system under consideration. The word "system" here refers to the software-to-be together with its environment [Fic92]. Goals are formulated in terms of optative statements [Zav97] which may refer to functional or non-functional properties and range from high-level concerns (such as "safe transportation" for a flight control system) to lower-level ones (such as "FlightPathAngle mode engaged until aircraft near desired altitude").

Goals play a prominent role in the RE process [Lam01]. They drive the elaboration of requirements to support them [Rub92, Dar93, Ant98, Kai00]. They provide a criterion for requirements completeness and pertinence [Yue87]. They induce rich specification structuring mechanisms such as goal AND-decomposition/composition for specification refinement/abstraction and OR-decomposition for reasoning about alternatives [Myl92, Dar93, Chu00]. Goals thereby provide a rationale for requirements and allow one to trace low-level details back to high-level concerns. The higher-level a goal is, the more stable it is likely to be; goals are thus essential elements for managing requirements evolution [Lam01]. Last but not least, goals have been recognized to be the roots at which conflicts can be detected and resolved [Rob89, Boe95, Lam98].

Agents are active system components (or "processors") which may have choice of behavior to ensure the goals they are assigned to [Fea87]. Achieving goals in general requires the cooperation of multiple agents. For example, the high-level goal of "safe transportation" might require the cooperation of the pilot, the autopilot software, the on-board TCAS software, the on-ground tracking system, etc. The essence of goal refinement is to decompose a goal into subgoals so that each subgoal requires the cooperation of fewer agents; the refinement process stops when goals are reached that can be assigned as responsibility of single agents [Dar93]. Terminal goals assigned to agents in the software-to-be become *requirements*; terminal goals assigned to agents in the environment become *assumptions* (or normative policies) --the latter cannot be enforced by the software-to-be [Lam00a]. In general, alternative responsibility assignments are to be explored; for example, the goal *"reverse thrust enabled when landing plane on runway"* [Lad95, Jac95] might be assigned to the Pilot or the Autopilot agent. Different goal

refinement and assignment alternatives yield alternative system proposals in which more or less features are automated.

Clearly, a goal should be assigned to an agent only if the latter has sufficient capabilities to ensure it; the agent should be able to "realize" the goal in some sense to be defined. For example, consider a simplified specification of the goal just introduced:

MovingOnRunway $\Rightarrow$ **o** ReverseThrustEnabled (G1)

(where "**o**" is the standard temporal logic operator for refering to the next state.) The Autopilot agent cannot realize this goal because it cannot monitor whether the plane is moving on the runway; in this case, the goal has a condition in its antecedent which is not *shared* by the agent and its environment [Jac95].

Surprisingly enough, such notion of goal realizability has not been studied so far in the literature. Goal realizability is important on a methodological standpoint as goal-oriented requirements elaboration might be driven by the process of refining high-level, unrealizable goals until realizable subgoals are reached that can be assigned to single agents; furthermore such a process might drive the identification of potential agents as well.

The purpose of the paper is to address this problem by defining precisely what it means for a goal to be realizable by an agent, and by proposing a comprehensive set of constructive tactics for refining unrealizable goals towards realizable subgoals and assignable agents. Each tactics corresponds to a refinement pattern; such patterns complement the ones discussed in [Dar96] in that they are agent-driven and guarantee strict progress towards realizable subgoals.

In the simple example above, one of our tactics would produce a refinement of the unrealizable goal G1 above into a functional subgoal

WheelsTurning $\Rightarrow$ **o** ReverseThrustEnabled (G2)

together with the following assumption on the environment

MovingOnRunway $\Leftrightarrow$ WheelsTurning (Ass)

(Obstacle analysis would reveal that this assumption is in fact too strong because of the possibility of, e.g., plane aquaplaning on a wet runway --but this is another story [Lam00a].)

The functional subgoal G2 is still not realizable by the Autopilot agent because this agent cannot directly monitor whether the wheels of the plane are turning. Applying one of our tactics again would produce a further refinement of goal G2 into an accuracy subgoal

WheelsTurning $\Leftrightarrow$ WheelsPulseOn (G3)

and a functional subgoal

WheelsPulseOn $\Rightarrow$ **o** ReverseThrustEnabled (G4)

The accuracy subgoal G3 is now realizable by a WheelsSensor agent whereas the functional subgoal G4 is realizable by the Autopilot agent.

The rest of the paper is organized as follows. Section 2 introduces our formal model of agents. Section 3 presents a semantic definition of goal realizability based on this model. Section 4 gives a necessary and sufficient set of conditions for realizability that provides a practical basis for checking whether a goal is realizable or not. This set of condition yields a taxonomy of unrealizability problems that can be shown to be complete; a comprehensive set of tactics for resolving unrealizability through refinement is derived systematically from this taxonomy in Section 5. The tactics provide systematic guidance for recursively refining goals and identifying agents; alternative tactics allow alternative goal refinements, agent interfaces and responsibility assignments to be explored. Section 6 provides a few snapshots from a real case study to illustrate the main ideas.

The paper provides a lightweight version of our technique in order to increase readability and stick to space limitations. A more formal treatment is available in [Let01].

2. MODELING AGENTS

The techniques in this paper are described in the context of the KAOS framework for goal-oriented requirements elaboration [Dar93, Lam98, Lam00c]. A KAOS application model is specified as a composition of four submodels: a *goal model* in which the goals to be achieved are described together with their refinement/conflict links; an *object model* in which the application objects involved are described together with their relationships; an *operation model* in which the services that operationalize the goals are described; and an *agent model* in which the agents are described together with their interfaces and responsibilities with respect to goals and operations. Each model has a separate semantics, and is related to the others through inter-model consistency rules [Dar93, Let01].

Our focus in this section is on the semantic foundation for the agent model. Our agent framework extends Feather's notion of agent [Fea87], Parnas' 4-variable model [Par95] and Jackson's principle of shared phenomena [Jac95].

An agent is characterized by the following items:

- an *interface* which declares two disjoint sets of state variables: a set of variables that the agent monitors and a set of variables that the agent controls;
- a *transition system* composed of an initial condition on the agent's controlled states and a "next state" total relation that maps each temporal sequence of states of monitored/controlled variables to a next state of controlled variables;
- a *responsibility relation* that maps the agent to the set of *goals* the agent is responsible for.

A *state variable* corresponds to an object attribute or relationship from the KAOS object model. A system *state* is defined as a mapping that assigns a value to each state variable [Man92].

Multiple agents run non-deterministically and concurrently; they interact through shared variables. In order to avoid interference between concurrent agents, each state variable can be controlled by at most one agent. (If a variable needs to be controlled by more than one agent, one can split the

variable into several variables, each controlled by a single agent.) A single variable can be monitored by several agents.

The agent interface component in our framework extends the *4*-variable model [Par95] to a *2N*-variable model; instead of one single software agent with one pair of input/output devices we consider *N* agents; some of them are software agents, others are sensors or actuators, others are humans, etc. In the KAOS language, agent interfaces can be represented by context diagrams in the spirit of [Jac95].

The agent transition system component provides the semantic domain for the KAOS operation model; the "next state" relation of an agent corresponds to applications of operations that are assigned to the agent and operationalize the goals the agent is responsible for. An *agent run* is an infinite sequence of system states generated by the transition system of the agent.

The agent responsibility component in our framework links agents to goals. A goal prescribes some set of possible histories of the system; a *history* is a temporal sequence of system states. A real-time linear temporal logic is therefore natural for specifying goals [Man92, Koy92, Dar93, Lam00b]. The following temporal operators will be used in this paper:

$\mathbf{o}$ (in the next state)	$\bullet$ (in the previous state)
$\Diamond$ (some time in the future)	$\Box$ (always in the future)
W (always in the future *unless*)	U (always in the future *until*)

$A \Rightarrow C$ (in every future state A implies C)

$@P$ (P holds in the current state and not in the previous state i.e., $\bullet \neg P \wedge P$)

$\Diamond_{\leq k u} P$ (P holds in some future state within *k* time units *u*)

$\Box_{\leq d} P$ (P holds in every future state up to some deadline *d*)

Assigning responsibility for a goal to an agent means that this agent must restrict its behavior so as to ensure the goal [Fea87, Dar93]. This is captured through the following *responsibility consistency rule* between the responsibility relation and the transition system of the agent:

if *Responsible (ag, G)* **then** *RUN(ag)* $\subseteq$ *HIST(G)*

where *RUN(ag)* denotes the set of all possible runs of agent *ag* and *HIST(G)* denotes the set of all possible histories prescribed by goal *G.*

To illustrate our agent framework briefly, consider the standard mine pump example in which water level has to be controlled [Kra83, Jos96]. There are three agents: PumpController, HighSensor and LowSensor. The HighSensor agent has WaterLevel as monitored variable and HighWaterSignal as controlled variable; the PumpController agent has HighWaterSignal and LowWaterSignal as monitored variables and PumpSwitch as controlled variable. The PumpController agent might be assigned the following goal:

HighWaterSignal = 'On' $\Rightarrow$ **o** PumpSwitch = 'On'

The responsibility consistency rule constrains the transition system of PumpController to be such that any run of this agent is among the possible histories prescribed by the goal assertion.

The responsibility assignment of a goal to an agent is further constrained by the monitoring and control capabilities of that agent. This is captured through the concept of realizability we discuss now.

3. GOAL REALIZABILITY

Our aim here is to provide a precise criterion for determining whether a goal is assignable to an agent.

We first provide a semantic definition of what it means for a goal to be realizable by an agent, that is, a definition in terms of agent runs and system histories. The next section will then provide an equivalent, more syntactic characterization to be used for checking realizability in practice. In the sequel, we will use the following notations:

Mon(ag):	set of monitored variables of agent *ag*
Ctrl(ag):	set of controlled variables of agent *ag*
State(V):	set of all possible states of variables in set *V*
Path(V):	set of all possible sequences of states of variables in set *V*

Definition *(Realizability).* A goal *G* is realizable by agent *ag* iff there exists a transition system

$$\Delta_{ag} = \langle \mathrm{Init}_{ag}, \mathrm{Next}_{ag} \rangle$$

with

- $\mathrm{Init}_{ag} \subseteq \mathrm{State}(Ctrl(ag))$
- $\mathrm{Next}_{ag} \subseteq \mathrm{Path}(Mon(ag) \cup Ctrl(ag)) \times \mathrm{State}(Ctrl(ag))$

such that

$\mathrm{RUN}(ag) = \mathrm{HIST}(G)$.

To illustrate this definition, we first come back to the goal Maintain[PumpSwitchOnWhenHighWaterDetected]:

HighWaterSignal = 'On' $\Rightarrow$ **o** PumpSwitch = 'On'

This goal is realizable by the PumpController agent; the latter monitors the HighWaterSignal variable and controls the PumpSwitch variable; a transition system whose set of runs is equal to the set of histories prescribed by the goal is given by the pair <*Init,Next*> such that

$s \in Init$ for all $s \in$ State(Ctrl(PumpController))

$(p, s) \in Next$ iff

if p_n(HighWaterSignal) = On **then** s(PumpSwitch) = On

(where p_n denotes the last state of path *p* and *s(v)* denotes the value of variable *v* in state *s*).

Consider now the goal Maintain[PumpMotorOnWhenHighWater]:

WaterLevel $\geq$ 'High' $\Rightarrow$ **o** PumpMotor = 'On'

In this case no transition system can be found for the PumpController agent since the WaterLevel state variable is not among the agent's monitored variables and the PumpMotor state variable is not among the agent's controlled variables; this goal is therefore not realizable by the PumpController agent.

With respect to the responsibility consistency rule in the previous section, we now require that the agent be able to achieve the goal *without being more restrictive than required by the goal*. To illustrate why we require equality

in the realizability condition above, let us consider the goal Maintain[PumpSwitchOnWhenHighWater], defined by:

$$\text{WaterLevel} \geq \text{'High'} \Rightarrow \mathbf{o}\ \text{PumpSwitch} = \text{'On'}$$

The PumpController agent controls the PumpSwitch variable, but does not monitor the WaterLevel variable. The agent could ensure this goal by always keeping the PumpSwitch variable set to 'On', regardless of the value of the WaterLevel variable. The following agent transition system could thus be identified:

$s \in \mathit{Init}$ for all $s \in$ State(Ctrl(PumpController))

$(p, s) \in \mathit{Next}$ iff s(PumpSwitch) = On

The agent runs generated by this transition system correspond to histories that satisfy the following assertion:

$\mathbf{o}\ \square$ PumpSwitch = 'On'

Such histories are stronger than those required by the goal. The equality requirement in our realizability condition thus prevents the PumpController agent from being responsible for the goal Maintain[PumpSwitchOnWhenHighWater] because this agent cannot ensure the goal without being more restrictive than required by the goal.

Note that realizability requires only the existence of an appropriate transition system, that is, the *possibility* for the agent to achieve the goal without being more restrictive than required by it; the *actual* transition system of the agent may be stronger than required by the goal. When several goals are assigned to the same agent, the actual transition system of that agent will generally be stronger than required by any single goal.

Our notion of realizability can be viewed as the counterpart at the goal level of the notion of realizable program specification [Aba89]; a specification there is said to be realizable if there exists a program that implements it. There are however two important differences: our notion of realizability explicitly refers to the variables monitored and controlled by the agent, and we require the existence of a transition system whose behaviors are *equal* to the set of histories admitted by the goal, whereas only inclusion is required in [Aba89].

4. IDENTIFYING UNREALIZABILITY PROBLEMS

Our aim now is to check whether a goal is realizable by some agent. If the goal is realizable, the agent may be candidate for responsibility assignment; if it is not, goal refinement should proceed further until realizable subgoals are reached. An additional concern is therefore to identify the cause of unrealizability so that appropriate elaboration tactics can be proposed to resolve the problem.

To show that a goal is not realizable by an agent, necessary and sufficient conditions for unrealizability can be used to avoid the brute force approach of showing that $\mathit{RUN(ag)} \neq \mathit{HIST(G)}$ for all possible agent transition systems [Let01]. This approach is still not practical though because such conditions are still defined at the semantic level in terms of sets of histories prescribed by the goal. Furthermore, it does not provide sufficient explanation about *why* the goal is not realizable by the agent.

An equivalent, more pragmatic characterization of realizability can be found in terms of syntactical conditions that can be checked more easily and provide such explanation.

To define these conditions, we partition the set of variables involved in the formulation of a goal into two subsets: the set *C* of state variables that are *intended to be constrained by the goal* and the set *M* of all other state variables. For an *Achieve* goal taking the form $\mathsf{P} \Rightarrow \Diamond \mathsf{T}$ for some current condition P and target condition T, for example, one will typically find the *C* and *M* variables in T and P, respectively.

A goal *G* can then be viewed as a relation

$$G_{M,C} \subseteq \mathit{Hist(M)} \times \mathit{Hist(C)}$$

where *Hist(M)* and *Hist(C)* denote the set of all possible histories of state variables in *M* and *C*, respectively. For example, the goal Maintain[PumpMotorOnWhenHighWater] is intended to constrain the PumpMotor variable and can be viewed as a relation between histories of the water level and histories of the pump motor. (This relation amounts at the goal level to a relation *REQ* between histories of monitored and controlled variables in the *4*-variable model [Par95].)

The following theorem defines conditions on a goal relation $G_{M,C}$ under which the goal *G* is not realizable by an agent.

Theorem *(Pragmatic conditions for unrealizability).*

A goal *G* is not realizable by an agent *ag* if and only if one of the following independent conditions holds at least.

(1) *Lack of monitorability*: the goal definition refers to some state variable in *M* but not in *Mon(ag)* ∪ *Ctrl(ag)*:

$$M \not\subseteq \mathit{Mon(ag)} \cup \mathit{Ctrl(ag)}$$

(2) *Lack of controllability:* the goal requires some state variable not in *Ctrl(ag)* to be controlled:

$$C \not\subseteq \mathit{Ctrl(ag)}$$

(3) *Reference to future:* the goal constrains the values of some variable in *C* in terms of future values of variables in *M*.

(4) *External unachievability:* some external condition *B* on unconstrained variables in *M* makes the goal impossible to achieve:

$$B \models \neg G \text{ with } B \not\models \textbf{false}$$

(5) *Unbounded achievement:* the goal does not constrain the finite behaviors of *ag*.

The full proof of this theorem is fairly long and technical; the interested reader may download it from [Let01]. We provide some intuitive arguments here to relate conditions (1)-(5) to the semantic definition of realizability in Section 3.

- *Lack of monitorability* and *lack of controllability* prevent a goal from being realizable by an agent because the definition of realizability requires the existence of a transition system for the agent that is defined in terms of variables monitored and controlled by it.
- *Reference to the future* prevents a goal from being real-

izable by an agent because the "next state" relation of any transition system for the agent constrains the next state of controlled variables in terms of the *previous* values of monitored and controlled variables.

- *External unachievability* means that there are histories on variables not controllable by the agent which falsify the goal for every possible history of the agent's controlled variables. Since any transition system for an agent can only constrain the values of its controlled variables, no transition system for the agent can prevent such falsification; hence the goal is not realizable by the agent.
- *Unbounded achievement* means that the agent could indefinitely postpone the satisfaction of the goal, here taking the form $P \Rightarrow \Diamond T$ or $P \Rightarrow Q \ U \ T$ for some current condition P, target condition T and other condition Q, without ever violating it along any finite run prefix in any transition system for the agent.
- Furthermore, the theorem states that the set of unrealizability conditions (1)-(5) is *complete*. When all of the conditions of the theorem are false, one can indeed construct a transition system for the agent from the goal assertion so that the set of runs generated by this transition system is equal to the set of histories defined by the goal. The intuition is roughly as follows (see [Let01] for technical details). Since all variables constrained by the goal are controlled by the agent and all other variables in the goal assertion are monitored or controlled by it, the goal relation $G_{M,C}$ defines a relation between histories of variables monitored and controlled by the agent, respectively. Since the goal formulation does not refer to future states of unconstrained variables and since there is no external condition on these variables that inhibit goal achievement, this relation between histories of monitored and controlled variables is total and may be mapped to a transition system for the agent. By construction, the set of finite sequences of states generated by this transition system equals the set of finite sequences of states that do not violate the goal in finite time. Since the goal is not a liveness property, this also means that the set of runs of the transition system equals the set of histories prescribed by the goal.

To illustrate the various unrealizability conditions above, let us first consider the following goal for an ambulance despatching system [LAS93].

Goal Achieve [AmbulanceMobilized]

InformalDef *For every urgent call reporting an incident, an available ambulance able to arrive at the incident scene within 11 minutes should be mobilized. The ambulance mobilization time should be less than 3 minutes .*

FormalDef $\forall$ c: UrgentCall, inc: Incident

@ Reporting (c, inc)

$\Rightarrow \Diamond_{\leq 3m} \exists$ amb: Ambulance

$\bullet$ Available(amb) $\wedge$ Mobilized (amb)

$\wedge$ amb.Destination = inc.Location

$\wedge \bullet$ TimeDist (amb.Location, inc.Location) $\leq$ 11

This goal is intended to constrain the state variables Mobilized(amb) and amb.Destination from values of the other state variables referenced in the goal. This goal is not realizable by the ambulance despatching software agent for the following reasons:

- lack of monitorability: the state variables inc.Location , amb.Location and Available(amb) are not among the agent's monitored or controlled variables, that is, the agent cannot monitor the actual incident location, ambulance location and ambulance availability;
- lack of controllability: the state variables Mobilized(amb) and amb.Destination, to be controlled, are not among the agent's controlled variables, that is, the agent cannot control the actual mobilization and destination of ambulances,
- external unachievability: the goal cannot be achieved under the agent's responsibility when there is no available ambulance near the incident scene.

This example combines three of the conditions above; each one would be sufficient for unrealizability.

Let us now illustrate the other conditions. Consider the following "utility" goal for the railroad crossing problem [Hei96]:

Goal Maintain[GateOpenWhenNoTrain]

InformalDef *When no train will be in the crossing during the next d time units, the gate should be opened.*

FormalDef $\forall$ cr: Crossing

$\square_{\leq d} \neg (\exists$ tr: Train) InCrossing (tr, cr)

$\Rightarrow$ cr.Gate = 'open'

This goal is intended to constrain the variable cr.Gate. It is not realizable by the GateController software agent because the value of the variable cr.Gate is constrained by future values of the variable InCrossing; moreover, this agent cannot monitor the variable InCrossing(tr,cr) and cannot control the variable cr.Gate (the GateController agent only controls the signal sent to the gate). Similarly, the goal

$\forall$ tr: Train

Moving (tr) $\Rightarrow$ tr.Doors = 'closed'

is unrealizable by the TrainController agent because the latter cannot monitor the variable Moving and control the variable Doors *within the same state*; no transition system allows controlled variables to be constrained by the current value of monitored variables. This particular case of reference to future is called *synchronization problem*.

Identifying a "reference to future" problem is not necessarily easy; a number of frequent temporal logic patterns of reference to the future are given in [Let01] to support the task.

The practical interest of the theorem above is that (a) it provides some guidance in the refinement process by explaining *why* the goal is not realizable by an agent, and (b) it gives a necessary and sufficient characterization of unrealizability from which a *complete taxonomy* of unrealizability problems can be built to define a comprehensive set of tactics for unrealizability resolution.

5. RESOLVING UNREALIZABILITY PROBLEMS

We now suggest a constructive technique for identifying agents and their capabilities, and for refining goals into subgoals until the latter are realizable by single agents. The general principle is to provide a catalog of *specification elaboration tactics* whose applications are driven by the need to resolve unrealizability problems. Specific tactics are provided for each unrealizability condition in the theorem given in the previous section; the entire space of unrealizability problems is therefore covered.

The tactics provide systematic guidance for recursively refining goals and for identifying new agents. They may produce new objects as well; new, enriched versions of the goal, agent and object models may thus be obtained thereby. Alternative goal refinements are explored through the application of alternative tactics.

In the same spirit as [Dar96], formal refinement patterns are associated with each tactics; the patterns are proved correct once for all; the specifier is thus relieved from the tedious task of verifying every pattern instantiation.

We first suggest what a single tactics may look like before outlining our catalog of tactics and highligthing a few tactics that will be used on a real example in the next section.

5.1 Describing agent-based refinement tactics

Each tactics is defined by the following items:

- a *motivation* that describes the process-level objective addressed by the tactics;
- a *precondition* that characterizes the conditions on the current specification model under which the tactics may be applied;
- some *heuristics* that suggests when the tactics should typically be applied;
- a *postcondition* that characterizes the state of the specification model after application of the tactics, in terms of effects on the goal model, the agent model, and the object model;
- *variants* and *specializations* of the tactics (if any);
- an example of using the tactics.

The following frequently used tactics illustrates this.

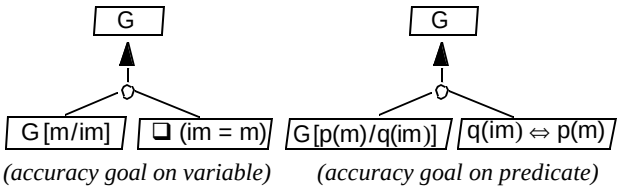
Tactics *Introduce Accuracy Goal*

Precondition: Agent *ag* cannot monitor some variable *m* appearing in goal *G* in order to realize that goal.

Heuristics: The tactics should be applied when an intermediate variable *im* can be identified as "image" of *m* so that *im* can be related to *m* through some accuracy goal or domain property.

Postcondition:

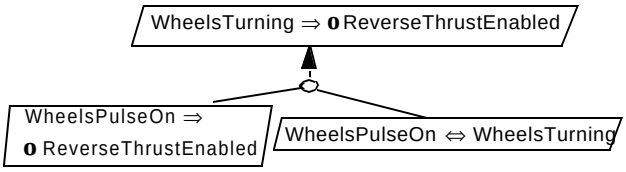
- *Object model:* a new attribute or relationship is introduced to capture the image *im* of attribute/relationship *m.*
- *Goal model:* the unrealizable goal *G* is refined into (i) a subgoal whose definition refers to variable *im* instead of *m,* and (ii) a companion accuracy goal that relates *im* to *m.* There are two alternative ways of applying the tactics according to the two following formal AND-refinement:



Variants: Introduce non-ideal accuracy goals involving tolerances and delays.

Specializations: *Introduce Tracking Object*, *Introduce Sensor Agent*

Example of use:



5.2 The catalog of agent-based refinement tactics

Each unrealizability condition from the theorem in Section 4 gives rise to a set of tactics for resolving the corresponding unrealizability problem. Because the taxonomy of unrealizability problems defined by this theorem is complete, the space of unrealizability problems is fully covered by our tactics. A tactics specialization hierarchy is associated with each specific unrealizability condition. The set of tactics found so far for each such condition is of course not complete; the tactics shown below were obtained by abstraction from a great number of examples in the literature on specification [Let01] and from several industrial case studies; the symmetry between lack of monitorability and lack of controllability was exploited as well.

To give an idea of what our current catalog looks like, Figures 1-5 outline portions of specialization hierarchies for the various unrealizability conditions. The reader may refer to [Let01] for more details and many examples of use.

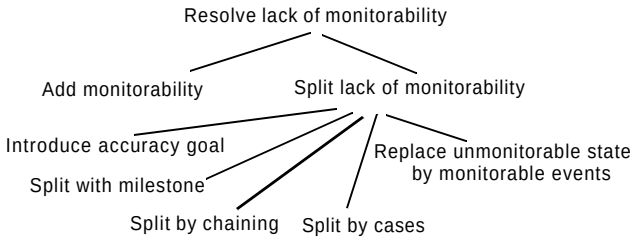


Figure 1 - *Tactics for resolving lack of monitorability*

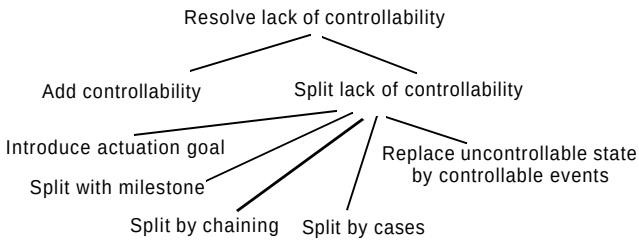


Figure 2 - *Tactics for resolving lack of controllability*

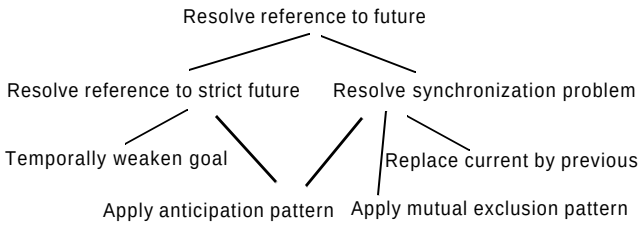


Figure 3 - *Tactics for resolving reference to future*

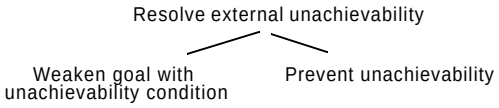


Figure 4 - *Tactics for resolving external unachievability*

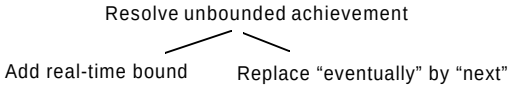


Figure 5 - *Tactics for resolving unbounded achievement*

Further specialized tactics could be explored by identifying specialized ways of resolving unrealizability problems for specific goal categories (such as satisfaction goals, information goals, security goals, usability goals, and so forth [Dar93]). The basic idea for such specialized tactics would be similar in spirit to the idea of using problem frames [Jac01].

We now highlight a few tactics that will be used on a real example in the next section. For example, the *Introduce Tracking Object* specialization of the tactics *Introduce Accuracy Goal* introduced before is defined as follows.

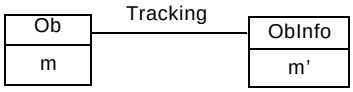
Tactics *Introduce Tracking Object*

Specializes *Introduce Accuracy Goal*

Heuristics: This tactics should be considered when lack of monitorability for an object *Ob* can be resolved by maintaining an internal image of the object.

Postcondition:

- *Object model:* If the unmonitorable variable *m* is an attribute of some object *Ob*, an intermediate variable *m'* is modelled as an attribute of a new object *ObInfo* representing an internal image of the object *Ob*. A *Tracking* relationship is also introduced to relate the objects *Ob* and *ObInfo*:



- *Goal model:* The unrealizable goal *G* is refined into the subgoals

  Maintain[ObjectTracked],
  Maintain[AccurateObjectInfo],
  G [Object/ObjectInfo]

  The first subgoal requires that every object Ob is related to an object ObInfo by a one-to-one *Tracking* relationship. The second subgoal is an accuracy goal relating an object and its image. The third subgoal is obtained roughly by replacing references to the actual object by references to the object's image.

Example of use: Consider the patient monitoring problem [Ste74, Jac95] and the goal Achieve[AlarmForCriticalPulseRate], defined as follows:

$\forall$ p: Patient
p.PulseRate $\notin$ p.SafePulse
$\Rightarrow \Diamond$ ($\exists$ al: Alarm) al.Raised $\wedge$ al.Location = p.BedNr

To resolve the lack of monitorability of the patient's pulse rate, safe pulse range and bed number by the *PatientMonitoring* software agent, the tactics is instantiated as follows:

Ob: Patient m: PulseRate, SafePulse, BedNr
ObInfo: PatientInfo m': PulseRate, SafePulse, BedNr

The following three subgoals are thereby produced:

Goal Maintain [PatientTracked]
FormalDef ($\forall$ p: Patient)($\exists!$ pi: PatientInfo) Tracking (pi, p)
$\wedge \forall$ p: Patient, pi: PatientInfo
Tracking (pi, p) $\Rightarrow \Box$ Tracking (pi, p)

Goal Maintain [AccuratePatientInfo]
FormalDef $\forall$ p: Patient, pi: PatientInfo
Tracking (pi, p) $\Rightarrow$ pi.PulseRate = p.PulseRate
$\wedge$ pi.SafePulse = p.SafePulse
$\wedge$ pi.BedNr = p.BedNr

Goal Achieve [AlarmForCriticalPulseRateInfo]
FormalDef $\forall$ pi: PatientInfo
pi.PulseRate $\notin$ pi.SafePulse
$\Rightarrow \Diamond$ ($\exists$ al: Alarm) al.Raised $\wedge$ al.Loc = pi.BedNr

The tactics *Split Lack of Controllability With Milestone* appears among the tactics for resolving lack of controllability in Figure 2. It is defined as follows.

Tactics *Split Lack of Controllability With Milestone*

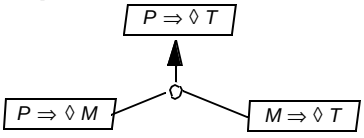
Motivation: Resolve lack of controllability

Precondition: The unrealizable goal is an *Achieve* goal of the form $P \Rightarrow \Diamond T$ and the agent cannot control a variable appearing in the target condition *T*.

Heuristics: The tactics is worth being considered when an intermediate milestone *M* can be identified for reaching the target *T* from *P*.

Postcondition:

- *Object model*: the object model is enriched with the new objects, attributes and relationships appearing in the definition of the milestone predicate *M*.
- *Goal model*: the *Achieve* goal is refined according to the following milestone-driven refinement pattern (or one of its variants) [Dar95, Dar96]:



Example of use: Consider the patient monitoring problem and the goal *Achieve[NurseInterventionForCriticalPulseRate]*, defined as follows:

$\forall$ p: Patient
p.PulseRate $\notin$ p.SafePulse
$\Rightarrow \Diamond$ ($\exists$ n: Nurse) Intervention (n, p)

To resolve the lack of controllability of the predicate Intervention(n,p) by the *PatientMonitoring* software agent, the tactics is instantiated as follows:

M: ($\exists$ al: Alarm) al.Raised $\wedge$ al.Loc = p.BedNr

A new *Alarm* entity is thereby introduced; the goal is refined into the following two subgoals:

Goal Achieve [AlarmForCriticalPulseRate]
FormalDef $\forall$ p: Patient:
p.PulseRate $\notin$ p.SafePulse
$\Rightarrow \Diamond$ ($\exists$ al: Alarm): al.Raised $\wedge$ al.Loc = p.BedNr

and

Goal Achieve [NurseInterventionForAlarm]
FormalDef ∀ al: Alarm, p: Patient:
al.Raised ∧ al.Loc = p.BedNr
⇒ ◊ (∃ n: Nurse) Intervention (n, p)

The tactics *Prevent Unachievability* appears among the tactics for resolving external unachievability in Figure 4. It is defined as follows.

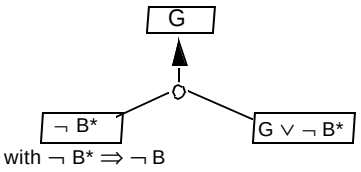
Tactics *Prevent Unachievability*

Motivation: Resolve external unachievability.

Precondition: The goal *G* is unsatisfiable when the unachievability condition *B* holds.

Heuristics: The tactics is worth being applied when *G* is a safety goal and the unachievability condition *B* cannot be tolerated. (Otherwise one can consider the alternative tactics *Weaken Goal With Unachievability Condition*.)

Postcondition: the unachievable goal is refined according to the following refinement pattern:



Example of use: Consider an ambulance dispatching system and the goal *Achieve[AmbulanceMobilizedInSector]*, defined by

∀ inc: Incident, s: Sector
inc.Reported ∧ InSector (inc, s)
⇒ $\Diamond_{\leq 3m}$ ∃ amb: Ambulance
Mobilization (amb, inc) ∧ • (amb.Available ∧ InSector(amb,s))

The unachievability condition *B* for this goal is given by

◊ ∃ inc: Incident, s: Sector
inc.Reported ∧ InSector (inc, s)
∧ $\Box_{\leq 3m}$ ¬ (∃ amb: Ambulance) amb.Available ∧ InSector(amb,s)

The unachievability condition is prevented through a subgoal requiring that in every sector there is always an ambulance available:

□ (∃ amb: Ambulance) amb.Available ∧ InSector(amb, s)

The companion subgoal produced by this tactics is then

∀ inc: Incident, s: Sector
inc.Reported ∧ InSector(inc, s)
⇒ $\Diamond_{\leq 3m}$ ∃ amb: Ambulance
Mobilization (amb, inc) ∧ • (amb.Available ∧ InSector(amb,s))
∨ ¬ (∃ amb: Ambulance) amb.Available ∧ InSector (amb, s)

The goal obtained requires an available ambulance to be mobilized from the sector in which the incident occurred except if there is no ambulance available in that sector.

Formal refinement patterns capture the general idea of the associated tactics. In practice, the formal definitions generated by a strict application of these patterns may have to be adapted for better adequacy with the situation at hand.

6. Example: Agent-Based Refinement for the London Ambulance System

We now show how some of the agent-based tactics above may be combined to build a portion of the goal graph for the LAS system [LAS93]. The detailed formal derivations are skipped here for lack of space; the interested reader may download them from [Let01] where a significant portion of the LAS goal graph is built formally.

We come back to the goal Achieve[AmbulanceMobilized] which was seen in Section 4 to be unrealizable by the Computer-Aided Despatching software agent (CAD) due to lack of monitorability, lack of controllability and external unachievability. Figure 6 shows a goal refinement graph produced by resolving each of these unrealizability problems.

The lack of monitorability of incident location can be resolved using the tactics *Introduce Tracking Object* defined in

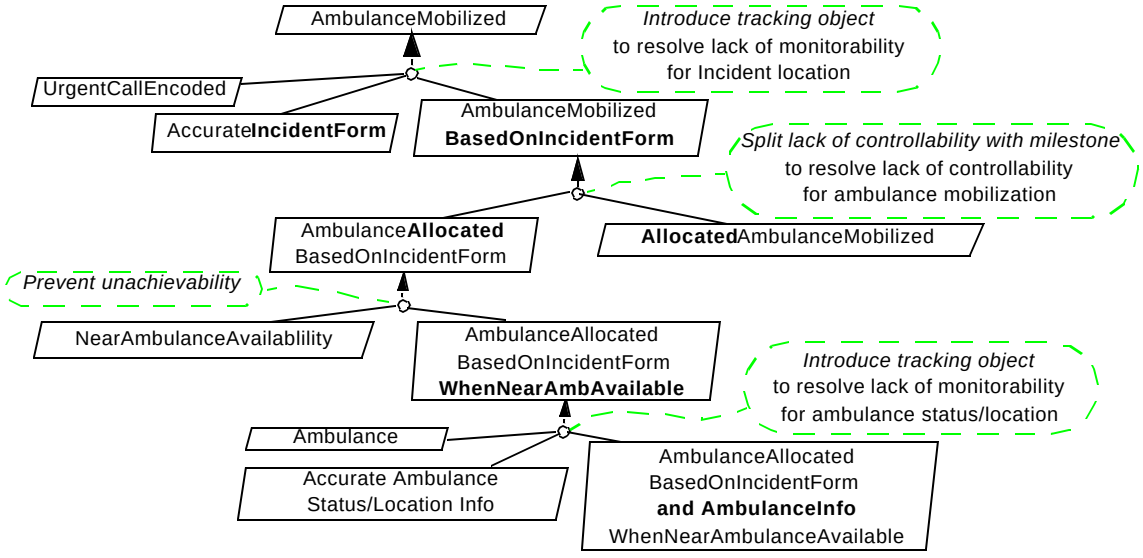


Figure 6 - *Applying agent-based tactics for the London Ambulance Service system*

the previous section. A new object IncidentForm is thereby introduced in the object model to track details about incidents; we obtain the following three subgoals:

Goal Achieve [UrgentCallEncoded]

InformalDef *For every urgent call reporting an incident, there shall be an incident form recording details about the reported incident. The time needed to handle the call and fill in the incident form should take no more than f time units.*

FormalDef $\forall$ c: UrgentCall, inc: Incident
Reporting (c, inc)
$\Rightarrow \Diamond_{\leq f} \exists$ icf: IncidentForm
icf.Encoded $\wedge$ Encoding (icf, c)

Goal Maintain [AccurateIncidentForm]

InformalDef *The incident form should record the accurate location of the incident and the time at which the call was taken. (Further details about the incident such as the number of injured persons and the kind of emergency services needed are ignored in this simplified spec.)*

FormalDef $\forall$ inc: Incident, c: UrgentCall, icf: IncidentForm
Reporting (c, inc) $\wedge$ Encoding (icf, c)
$\Rightarrow$ icf.Location = inc.Location $\wedge$ icf.CallTime = c.Time

Goal Achieve [AmbulanceMobilizedBasedOnIncidentForm]

InformalDef *For every incident form, an ambulance able to arrive at the incident scene within 11 minutes should be mobilized to the corresponding location. An ambulance should be mobilized less that 3 minutes after the reception of the call.*

FormalDef $\forall$ c: UrgentCall, icf: IncidentForm
@ icf.Encoded
$\Rightarrow \Diamond_{\leq \text{ icf.CallTime+3m}} \exists$ amb: Ambulance
amb.Mobilized $\wedge$ amb.Destination = icf.Location
$\wedge \bullet$ amb.Available
$\wedge \bullet$ TimeDist (amb.Location, icf.Location) $\leq$ 11

The agent model is enriched in parallel by introducing a ControlAssistant agent and assigning the responsibility for the goals Achieve[UrgentCallEncoded] and Maintain [AccurateIncidentForm] to that agent.

The new goal Achieve[AmbulanceMobilizedBasedOnIncidentForm] is not realizable by the CAD agent due to lack of controllability of ambulance mobilization (actually controlled by AmbulanceStaff agents). This can be resolved using the tactics *Split Lack of Controllability With Milestone* introduced in Section 5.2 with the following milestone:

M: $\exists$ al: AllocationOrder, amb: Ambulance
al.Issued $\wedge$ Concerning (al, amb)
$\wedge$ al.DestinationLoc = icf.Location
$\wedge \bullet$ amb.Available
$\wedge \bullet$ TimeDist (amb.Location, icf.Location) $\leq$ 11'

The following two subgoals are obtained using this tactics:

Goal Achieve [AmbulanceAllocatedBasedOnIncidentForm]

InformalDef *For every incident form, an available ambulance able to arrive at the incident scene within 11 minutes should be allocated to the corresponding location. The ambulance allocation time should take no more than g time units.*

FormalDef $\forall$ icf: IncidentForm
@ icf.Encoded
$\Rightarrow \Diamond_{\leq g} \exists$ al: AllocationOrder, amb: Ambulance
al.Issued $\wedge$ Concerning (al, amb)
$\wedge$ al.DestinationLoc = icf.Location
$\wedge \bullet$ amb.Available
$\wedge \bullet$ TimeDist (amb.Location, icf.Location) $\leq$ 11

and

Goal Achieve[AllocatedAmbulanceMobilized]

InformalDef *When an ambulance is allocated to an incident location, it should eventually be mobilized to that location. This should take no more than h time units.*

FormalDef $\forall$ al: AllocationOrder, amb: Ambulance, loc: Location
al.Issued $\wedge$ Concerning (al, amb) $\wedge$ al.DestinationLoc = loc
$\wedge \bullet$ amb.Available
$\Rightarrow \Diamond_{\leq h}$ amb.Mobilized $\wedge$ amb.Destination = loc

For simplicity and brevity, we omit further subgoals and domain properties needed to establish the correctness of this refinement.

The new subgoal Achieve[AmbulanceAllocatedBasedOnIncidentForm] is still not realizable by the CAD agent due to the external unachievability problem propagated from its parent goals. The unachievability condition for this goal is given by the assertion

B: $\Diamond \exists$ icf: IncidentForm
@ icf.Encoded
$\wedge \Box_{\leq g} \neg \exists$ amb: Ambulance
amb.Available
$\wedge$ TimeDist (amb.Location, icf.Location) $\leq$ 11

Since this goal is the SafetyGoal category, the tactics *Prevent Unachievability* defined in Section 5.2 is used to resolve the unrealizability problem. The first subgoal is obtained by strengthening the negation of the unachievability condition:

Goal Maintain [NearAmbulanceAvailability]

InformalDef *For every location, there should always be an available ambulance able to arrive at that location within 11 minutes.*

FormalDef $\forall$ loc: Location
$\Box \exists$ amb: Ambulance
amb.Available $\wedge$ TimeDist (amb.Location, loc) $\leq$ 11

The second, companion subgoal is then:

Goal Achieve [AmbulanceAllocatedBasedOnIncidentForm
WhenNearAmbulanceAvailable]

InformalDef *For every incident form, an available ambulance able to arrive at the incident scene within 11 minutes should be allocated to the corresponding location except if there is no such ambulance available. The ambulance allocation time should take no more than g time units.*

FormalDef $\forall$ icf: IncidentForm
@ icf.Encoded
$\Rightarrow \Diamond_{\leq g} \exists$ al: AllocationOrder, amb: Ambulance
al.Issued $\wedge$ Concerning (al, amb)
$\wedge$ al.DestinationLoc = icf.Location
$\wedge \bullet$ amb.Available
$\wedge \bullet$ TimeDist (amb.Location, icf.Location) $\leq$ 11
$\vee \neg \exists$ amb: Ambulance
amb.Available $\wedge$ TimeDist (amb.Location, if.Location) $\leq$ 11

The latter subgoal is still not realizable by the CAD agent due to lack of monitorability of the actual location and availability of ambulances. Our tactics *Introduce Tracking Object* is therefore used again to produce the following three subgoals:

Goal Maintain[AmbulanceTracked]

InformalDef *Every ambulance is tracked by exactly one AmbulanceInfo object.*

FormalDef

$(\forall$ amb: Ambulance$)(\exists!$ ai: AmbulanceInfo) Tracking (ai, amb)
$\wedge \forall$ amb: Ambulance, $\forall$ ai: AmbulanceInfo
Tracking (ai, amb) $\Rightarrow \Box$ Tracking (ai, amb)

Goal Maintain [AccurateAmbulanceStatus&LocationInfo]

InformalDef *Information about ambulance availability and location should be accurate.*

FormalDef $\forall$ amb: Ambulance, ai: AmbulanceInfo
Tracking (ai, amb)
$\Rightarrow$ ai.Available $\leftrightarrow$ amb.Available
$\wedge$ ai.Location = amb.Location

Goal Achieve [AmbulanceAllocatedBasedOnIncidentForm andAmbulanceInfoWhenNearAmbulanceAvailable]

InformalDef *For every incident form, based on ambulance information (status and location), an available ambulance able to arrive at the incident scene within 11 minutes should be allocated to the corresponding location except if there is no such ambulance available. The ambulance allocation time should take no more than g ime units.*

FormalDef $\forall$ icf: IncidentForm
@ if.Encoded
$\Rightarrow \Diamond_{\leq g} \exists$ al: AllocationOrder, ai: AmbulanceInfo
al.Issued $\wedge$ Concerning (al, amb)
$\wedge$ al.DestinationLoc = icf.Location
$\wedge \bullet$ ai.Available
$\wedge \bullet$ TimeDist (ai.Location, icf.Location) $\leq$ 11
$\vee \neg \exists$ ai: AmbulanceInfo
ai.Available $\wedge$ TimeDist (ai.Location, icf.Location) $\leq$ 11

This last subgoal is now realizable by the CAD software agent. The accuracy subgoal needs to be refined further, and leads to alternative responsibility assignments for the AmbulanceStaff, the RadioOperator, the MobileDataTerminal, the AmbulanceTracking and the CAD agents.

Obstacle analysis may then be applied to the model and specification to produce a more robust model/specification, see [Lam00a] for an application of such analysis to this case study.

7. CONCLUSION

This paper has reported on ongoing efforts to provide more constructive guidance in the requirements engineering process. The techniques proposed here allow alternative goal refinements and agent assignments to be explored in a systematic way. The techniques are grounded on a simple formal model of agent responsibility, monitoring and control. A pragmatic counterpart of a semantic characterization of goal unrealizability was seen to play a central role in deriving a set of specification elaboration tactics that cover the entire space of unrealizability problems and guarantee strict progress towards realizable goals and assignable agents. Several of the tactics pay significant attention to accuracy goals. The role of these non-functional goals is often neglected in the literature on formal specification; they are known to be responsible for many serious accidents credited to poor requirements specifications.

In our experience, the systematic identification and resolution of unrealizability problems provides useful, practical guidance for elaborating goal graphs and associated responsibility assignments. We observed that by applying these tactics systematically we have been able to build fairly large goal graphs significantly faster than before.

Labelling goal refinements with the tactics used to produce them makes complex goal graphs easier to understand. Each refinement step is motivated by the resolution of an unrealizability problem, and the tactics applied to produce the refinement documents how this problem was solved.

Although grounded on a formal framework, the tactics can also be used by someone familiar with their repeated use in *shortcut mode*, to produce refinements systematically without necessarily getting into detailed formalizations (as we did in the previous section). In any case, much creative thinking and domain knowledge is still required to instantiate the tactics and combine them in a useful way. The tactics provide a way to organize such creative thinking; they do not automatically generate all alternative system proposals by themselves, of course.

A tool for doing all clerical work of retrieving tactics that match the current specification state, providing interactive help in their instantiation and application, and recording alternative refinements and assignments for later use would obviously be of great help. Frequently, however, the strict application of formal refinement patterns produces first-sketch formal definitions that need to be adapted manually to fit the details of the particular application.

For a goal raising several unrealizability problems, it is not a priori clear in what order those problems should be resolved. During the actual elaboration of the goal graph for complex systems such as the LAS and BART systems [Let01], we frequently switched the order in which the agent-based tactics were applied. This did not result in different requirements at the end of the requirements elaboration process but had an impact on the definition of intermediate goals and on the presentation of the goal graph. The order of application of tactics that was finally chosen was mostly driven by our objective of making the goal graph and the formal definition of goals as easy to understand as possible.

The techniques presented in this paper provide support for generating alternative refinements and assignments through the use of alternative tactics; they provide no support for evaluating alternatives and selecting some preferable one among them. This is an issue we start working on; in particular, we are investigating the applicability in our context of qualitative frameworks in the spirit of [Chu00].

Acknowledgement

The work reported herein was partially supported by the "Communauté Française de Belgique" (FRISCO project, Actions de Recherche Concertées Nr. 95/00-187 - Direction générale de la Recherche) and the Fond National de la Recherche Scientifique (FNRS). Thanks to the ICSE reviewers for constructive suggestions on improving the clarity of some parts of the paper.

REFERENCES

[Aba89] M. Abadi, L. Lamport and P. Wolper, "Realizable and Unrealizable Specifications of Reactive Systems", *Proc 16th ICALP*, 1989, LNCS 372, pp. 1-17.

[Ant98] A.I. Anton and C. Potts, "The Use of Goals to Surface Requirements for Evolving Systems", *Proc. ICSE-98: 20th Intrnational Conference on Software Enginering*, Kyoto, April 1998.

[Chu00] L. Chung, B. Nixon, E. Yu and J. Mylopoulos, *Non-functional requirements in software engineering*. Kluwer Academic, Boston, 2000.

[Boe95] B. W. Boehm, P. Bose, E. Horowitz, and Ming June Lee, "Software Requirements Negotiation and Renegotiation Aids: A Theory-W Based Spiral Approach", *Proc. ICSE-17 - 17th Intl. Conf. on Software Engineering*, Seattle, 1995, pp. 243-253.

[Dar93] A. Dardenne, A. van Lamsweerde and S. Fickas, "Goal-Directed Requirements Acquisition", *Science of Computer Programming*, Vol. 20, 1993, 3-50.

[Dar96] R. Darimont and A. van Lamsweerde, "Formal Refinement Patterns for Goal-Driven Requirements Elaboration", *Proc. FSE'4 - Fourth ACM SIGSOFT Symp. on the Foundations of Software Engineering*, San Francisco, October 1996, 179-190.

[Fea87] M. Feather, "Language Support for the Specification and Development of Composite Systems", *ACM Trans. on Programming Languages and Systems* 9(2), Apr. 87, 198-234.

[Fic92] S. Fickas and R. Helm, "Knowledge Representation and Reasoning in the Design of Composite Systems", *IEEE Trans. on Software Engineering*, June 1992, 470-482.

[Hei96] C. Heitmeyer and D. Mandrioli (eds.), *Formal Methods for Real-Time Computing*. Wiley, 1996.

[Jac95] M. Jackson, *Software Requirements & Specifications - A Lexicon of Practice, Principles and Pejudices*. ACM Press, Addison-Wesley, 1995.

[Jac01] M. Jackson, *Problem Frames - Analyzing and Structuring Software Development Problems*. Addison-Wesley, 2001.

[Jos96] M. Joseph, *Real-Time Systems: Specification, Verification and Analysis*. Prentice Hall, 1996.

[Kai00] H. Kaindl, "A Design Process Based on a Model Combining Scenarios with Goals and Functions", *IEEE Trans. on Systems, Man and Cybernetic*, Vol. 30 No. 5, September 2000, 537-551.

[Koy92] R. Koymans, *Specifying message passing and time-critical systems with temporal logic*, LNCS 651, Springer-Verlag, 1992.

[Kra83] J. Kramer, J. Magee, M. Sloman et al, CONIC: an Integrated Approach to Distributed Computer Control Systems. *IEE Proceedings*, Part E 130, 1, January 1983, pp. 1-10.

[Lad95] P. Ladkin, in *The Risks Digest*, P.G. Neuman (ed.), ACM 15.13, October 1995.

[Lam98] A. van Lamsweerde, R. Darimont and E. Letier, "Managing Conflicts in Goal-Driven Requirements Engineering", *IEEE Trans. on Sofware. Engineering*, Special Issue on Inconsistency Management in Software Development, Vol. 24 No. 11, November 1998, pp. 908-926.

[Lam00a] A. van Lamsweerde and E. Letier, "Handling Obstacles in Goal-Oriented Requirements Engineering", *IEEE Transactions on Software Engineering*, Special Issue on Exception Handling, Vol. 26 No. 10, October 2000, pp. 978-1005.

[Lam00b] A. van Lamsweerde, "Formal Specification: a Roadmap". In *The Future of Software Engineering*, A. Finkelstein (ed.), ACM Press, 2000, pp. 147-160.

[Lam00c] A. van Lamsweerde, "Requirements Engineering in the Year 00: A Research Perspective". Invited Keynote Paper, *Proc. ICSE'2000: 22nd International Conference on Software Engineering*, ACM Press, 2000, pp. 5-19.

[Lam01] A. van Lamsweerde, "Goal-Oriented Requirements Engineering: A Guided Tour". Invited minitutorial, *Proc. RE'01 - International Joint Conference on Requirements Engineering*, Toronto, IEEE, August 2001, pp.249-263.

[LAS93] Report of the Inquiry Into the London Ambulance Service, February 1993. The Communications Directorate, South West Thames Regional Authority, ISBN 0-905133-70-6.

[Let01] E. Letier, *Reasoning about Agents in Goal-Oriented Requirements Engineering*. Ph. D. Thesis, University of Louvain, May 2001; *http://www.info.ucl.ac.be/people/eletier/thesis.html.*

[Man92] Z. Manna and A. Pnueli, *The Temporal Logic of Reactive and Concurrent Systems*, Springer-Verlag, 1992.

[Myl92] J. Mylopoulos, L. Chung and B. Nixon, "Representing and Using Nonfunctional Requirements: A Process-Oriented Approach", *IEEE Trans. on Sofware. Engineering*, Vol. 18 No. 6, June 1992, pp. 483-497.

[Par95] D.L. Parnas and J. Madey, "Functional Documents for Computer Systems", *Science of Computer Programming*, Vol. 25, 1995, 41-61.

[Rob89] Robinson, W.N., "Integrating Multiple Specifications Using Domain Goals", *Proc. IWSSD-5 - 5th Intl. Workshop on Software Specification and Design*, IEEE, 1989, 219-225.

[Rub92] K.S. Rubin and A. Goldberg, "Object Behavior Analysis", Communications of the ACM Vol. 35 No. 9, September 1992, 48-62.

[Ste74] W.P. Stevens, G.J. Myers and L.L. Constantine, 'Structured Design', *IBM Systems Journal*, Vol. 13, No. 2, 1974, pp. 115-139.

[Yue87] K. Yue, "What Does It Mean to Say that a Specification is Complete?", *Proc. IWSSD-4, Fourth International Workshop on Software Specification and Design*, IEEE, 1987.

[Zav97] P. Zave and M. Jackson, "Four Dark Corners of Requirements Engineering", *ACM Transactions on Software Engineering and Methodology*, 1997, 1-30.