

Goal-directed Concept Acquisition in Requirements Elicitation

Anne Dardenne*, Stephen Fickas**, Axel van Lamsweerde***

*University of Namur, B-5000 Namur (Belgium)

**University of Oregon, Eugene, OR 97403 (USA)

***University of Louvain, B-1348 Louvain-la-Neuve (Belgium)

Abstract

Requirements analysis includes an acquisition step where a global model for the specification of the system and its environment is elaborated. This model involves concepts that are usually not found in the final formal specification, such as goals to be achieved, agents and their responsibilities, etc. This paper presents an approach for model acquisition which is driven by such goals. We describe a conceptual meta-model in terms of which requirements models are acquired. Our acquisition strategy can be viewed as a systematic way to traversing this meta-model backwards from the goals. The goal-directed acquisition strategy and the use of the meta-model are illustrated with a case study, the specification of a simple elevator system.

1. Introduction

Requirements analysis is recognized as being the most critical step in the software lifecycle. Errors made during this first step may have disastrous effects on the subsequent development steps and on the quality of the resulting product. It is therefore useful to provide automated support for assisting analysts in conducting this step.

We view requirements analysis as being made of two coroutining substeps:

- *requirements elicitation*, where a preliminary architecture for the specification of the system and its environment is elaborated and expressed in an intermediate knowledge representation language;
- *formal specification*, where the global model elaborated during acquisition is refined and made further precise using constructs of a fully formal language suitable for formal proofs, prototype generation, etc.

This distinction arises from the nature of the tasks involved:

- The acquisition of knowledge about the *composite* system involves concepts that usually are not found in the final formal specification, such as goals to be achieved, agents involved and their responsibilities, etc. (the term "composite system"[6] refers to the application to be automated and its environment).

- Acquisition processes rely more on domain knowledge whereas formal specification processes rely more on knowledge about the specification formalism being used.

- The basic operators being applied in such processes are rather different; requirements elicitation involves learning operators [12] whereas formal specification involves data/operation decomposition and structuring, modularization, parameterization and instantiation, etc. [4].

We are involved in modelling and formalizing the processes for both substeps through two complementary projects: KAOS (for requirements elicitation) and ICARUS (for requirements formalization). Elicitation in KAOS is viewed as a cooperative learning task between clients and analysts. An Acquisition Assistant is being developed in that context to guide analysts in the elaboration of the preliminary requirements model.

As in learning-by-instruction systems, requirements elicitation should be structured in terms of a model for acquiring requirements models. A proposal for such a meta-model is sketched in Section 2.

The concept of goal is a central component of this meta-model. Goals are useful in several respects.

- They lead to the incorporation of specification components which should support them;
- They allow to justify and explain the presence of specification items which are not necessarily comprehensible to clients;
- They may be used to determine the respective roles of agents in the system; more precisely, they may provide the basis for defining which agents should best perform which actions (according to their responsibilities, ability, reliability, motivation,...);

- They provide the “roots” at which conflicts should be resolved and multiple viewpoints should be reconciled [10].

This paper presents a strategy for requirements acquisition which relies on the goals that must be met in a specification. Section 2 outlines the conceptual meta-model in terms of which requirements models are being acquired. The goal-directed acquisition strategy is described in Section 3, together with a scenario of acquisition for the specification of a simple elevator system. Some concluding remarks are made in Section 4.

2. A conceptual meta-model for requirements modelling

The requirements model built gradually during acquisition is maintained in a *requirements database*. The requirements database is organized according to components of a *conceptual meta-model*; this means that the requirements model built during acquisition is expressed as domain/task-specific *instances* of domain/task-independent components of this meta-model. An acquisition session consists of traversing the meta-model to acquire the corresponding instances by use of acquisition operators; the order for traversing the meta-model and the composition of acquisition operators is determined by the acquisition strategies selected among those made available (this paper presents one of those acquisition strategies).

The KAOS meta-model (see Fig. 1) is made of meta-

concepts, meta-relations linking meta-concepts, meta-attributes characterizing meta-concepts and meta-relations, and meta-constraints on these various kinds of components (e.g., cardinality constraints on meta-relations); such constraints must be satisfied in the final state of the acquisition process. The requirements model acquired is made of instances of the meta-concepts, linked by instances of the meta-relations and characterized by instances of the meta-attributes. The “meta” prefix is used here to avoid confusions between the three following levels involved:

- the *meta level*, where domain-independent abstractions are defined;
- the *domain level*, where concepts specific to the application domain are defined as instances of meta-level abstractions;
- the *instance level*, where particular instances of domain-specific concepts are introduced (if necessary).

Each meta-concept is defined by a set of characteristics; a characteristic is either a meta-attribute or a meta-relation in which the meta-concept participates. This set of characteristics is inherited by any meta-concept instance. Each meta-relation is also defined by a set of characteristics; a characteristic here is either a meta-attribute or the ordered list of linked meta-concepts that make the meta-relation, together with their respective role and cardinality constraint. Finally, each meta-attribute is defined by features such as its naming, its intuitive meaning, its domain (i.e., the set of possible attribute values), the unit to which such values refer

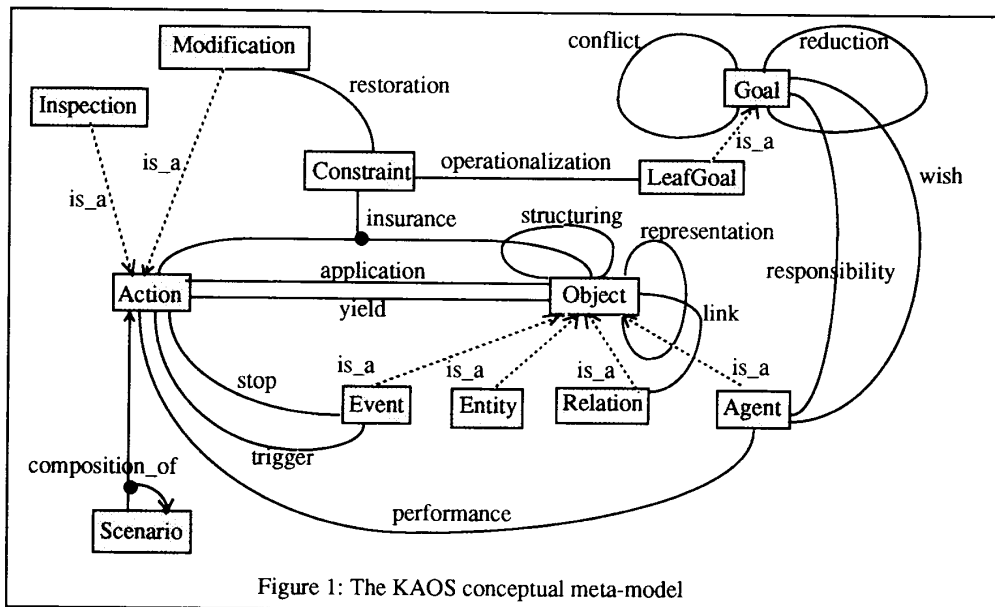


Figure 1: The KAOS conceptual meta-model

(optional), the inheritance mode through specialization hierarchies, etc.

The meta-model is just outlined here; a precise definition of its various components can be found in [11]; some of them will be made further precise as needed in Section 3. The current version covers the following *meta-concepts*: *object*, which can be specialized to *entity*, *relation*, *event*, and *agent*; *action*; *constraint*; *goal* which can be specialized to *leafGoal*; *scenario*. The *meta-relations* introduced so far include: actions *apply-to* or *yield* objects, actions/objects *ensure* constraints; events *trigger* or *stop* actions; agents *perform* actions; goals are *reduced* into goals; agents are *responsible-for* goals; constraints *operationalize* leafGoals; objects are *represented* by objects (any instance of this meta-relation links an external environment object to the internal object(s) that represent it in the automated application); etc. The *view* ternary meta-relation (not represented in Fig. 1) allows different views on a same object, action, or goal, to be associated with different agents. The *meta-attributes* introduced so far include *duration*, *pre_* and *post_conditions*, *trigger_* and *stop_conditions* for the action meta-concept; *reduction_mode* for the goal meta-concept (with “and”, “or” as possible values); *situation* for objects and actions (with possible values “external” or “internal” according as the corresponding object/action pertains to the external environment or to the sub-system to be automated); etc.

3. A goal-directed acquisition strategy

Our aim is to support the elaboration of a requirements model that is guaranteed to satisfy the goals of the clients, and record how that satisfaction is realized.

Agents, goals and constraints, as they are defined in our meta-model, fulfill an important role in the acquisition strategy. They are therefore more precisely defined below.

3.1 Agents, goals and constraints

3.1.1 Agents: In KAOS, an agent is an object which is a processor for one action at least.

3.1.2 Goals: A goal is an objective that has to be met by the composite system. A goal is typically not formalized; in general, it cannot be described exclusively in terms of objects and actions of the composite system being considered. Hence, we say that the expression of a goal is *nonoperational*.

An example of goal for an elevator system is “safety

during transportation”. This goal refers to the concepts of “safety” and “transportation” which are not explicit components of the specification of an elevator system.

Goals are linked in two different ways to agents: by the responsibility and the wish meta-relations.

(i) An agent *is responsible for* a goal if it must guarantee, maybe jointly with others, that the goal will be achieved in the system. An agent can guarantee that a goal will be met by performing appropriate actions. Goals that *must* be met by the system are called *system* goals.

(ii) An agent *wishes* a goal if it wants it to be met. Each agent may have its own private goals that it wishes to be met. An example of a private goal is the “don’t waste time” goal wished by a passenger. *Private* goals *may* be met by the system.

A reduction meta-relation is defined to capture the goal-subgoal structure. This relation corresponds to the classical reduction operator in the problem reduction approach to problem solving [9]. The overall goal structure is an and/or graph. Alternative goal reductions can thus be captured; a goal node can have several parent nodes as it can occur in several reductions. A goal which is not reduced further is called a leafGoal.

Goals can also be conflicting; e.g., abstract goals such as “minimize waiting time for a resource” and “keep resources as long as desired” are conflicting as soon as multiple agents are considered. It is important that such conflicts be explicitly recorded. Two goals are linked via a conflict meta-relation instance if and only if they cannot be achieved simultaneously.

3.1.3 Constraints: A constraint is an operational assertion on object states; its satisfaction contributes to the achievement of one or more goals. A constraint can be formalized; it is *operational* in that it refers to actions and objects available to agents [8]. Constraints are formalized in the (first-order) assertion language used to express *pre_conditions*, *post_conditions*, *trigger_conditions*, etc. Constraints are *ensured by* restrictions defined on actions and objects under control of agents.

An example of a constraint is

$$\forall \text{ elevator: } \neg \text{elevator.state} = \text{'moving'} \\ \Rightarrow \exists \text{ floor: locatedAt}(\text{elevator, floor}).$$

3.1.4 Link between goals and constraints: The link between goals and constraints is captured in the *operationalization* meta-relation: a goal is operationalized by being translated into constraints; the satisfaction of each constraint then contributes to the achievement of the goal. The operationalization meta-relation captures the fact that a constraint is an abstract “implementation” of one or more leaf goals. Our model refines the notion of operationalization in explanation-based learning ([8],

[5]), in that we split the notion of operationalization of concepts through actions into the notions of (i) operationalization through constraints and (ii) constraint insurance through restrictions on actions and objects. This permits a finer grained explanation of how goals are supported by specification components such as actions and objects.

For an elevator system, the goal “safety during transportation” can be operationalized by being associated with the following constraint:

$\forall \text{ elevator: elevator.state} = \text{'moving'}$
 $\Rightarrow \text{door.state} = \text{'closed'}$.

This constraint is operational in the sense that it exclusively refers to actions/objects available to the elevator-controller agent. This constraint is ensured, among others, by strengthening the precondition of the goToFloor action with the assertion $\langle \text{door.state} = \text{'closed'} \rangle$.

3.2 Acquisition strategy

For each step of the proposed strategy, we explain what the step does, why it is necessary to do it at a particular point in the strategy and how it is done. For each step, we also mention which components of the meta-model are involved: the strategy corresponds to a directed way of acquiring application-specific instances of meta-concepts and meta-relations. Starting from the goal meta-concept, the specification is acquired incrementally by considering at each step new instances of specific meta-concept(s) and/or meta-relation(s) (see Fig. 1 to follow the acquisition path through the meta-model).

The strategy will be illustrated by specifying a simple elevator system [2]. We consider one single elevator with a maximum capacity of one passenger.

Step 1 - Identify goals and agents

This step consists of two coroutining substeps.

Step 1.1: Build a goal structure for the particular application

WHAT: The system goals given by the client are progressively reduced in an overall goal-subgoal structure (an and/or graph). The leaf goals of the structure are primitive goals to which operational constraints will be associated in step 2. The elaboration of the goal structure consists of three substeps:

- (i) identify goals and associate them with the parent goal(s) they reduce (i.e. define instances of the goal meta-concept and of the reduction meta-relation);
- (ii) identify possible conflicts between system goals and private goals of agents (i.e. define instances of the conflict meta-relation);
- (iii) for each system goal, assign responsibility to agent(s) -i.e. define instances of the responsibility meta-

relation-(see also step 1.2 below).

These three substeps are not sequential, they are intertwined:

- the assignment of responsibilities determines the need for further reduction into subgoals (see the heuristics below);
- the state of the reduction process affects the assignment of responsibilities;
- the assignment of responsibilities may highlight the need for identifying a new agent and its associated private goals, which must be checked for conflict.

WHY: The reduction of system goals into primitive goals is necessary because global goals usually cannot be directly translated into constraints: only simple, primitive goals can be operationalized. Moreover, it is interesting to keep the whole goal structure as a history of the acquisition process because:

- it records the division of responsibility among agents;
- it ties specification components to informal text describing goals for the analyst's use;
- it will be used in case of negotiation required to solve conflicts [10];
- it can be used to replay some part of the acquisition process in other similar circumstances.

HOW: The identification of subgoals is a non-trivial task under control of the analyst. Goal structures may be refined with the help of the following heuristics.

1) ReduceGoal Heuristic: “A goal for which the responsibility is shared among several agents is reduced into subgoals for which the responsibility is shared among fewer agents”.

We believe that a goal under the responsibility of a smaller group of agents is easier to achieve than a goal under the responsibility of a larger group of agents, because less agents require less complex cooperation.

2) StopReduction Heuristic: “The reduction process ends when all the leaf goals of the structure are under the responsibility of one single agent or are considered as being simple enough to be operationalized by constraints”.

3) Assign Responsibility Heuristic:

“Given:

- $A = \{a_1, \dots, a_n\}$ the set of agents responsible for the parent goal(s) of G (this set includes all the agents if G has no parent);

- $g(a_i)$ the set of goals wished by agent a_i ($1 \leq i \leq n$)

Then:

The only agents which may be responsible for goal G should belong to the set $\{a \in A \mid \neg \exists g \in g(a) \text{ s.t. } \text{conflicts}(g, G)\}$,”

where the expression $\text{conflicts}(g, G)$ means that there is an instance of the conflict meta-relation between goal instances g and G .

The assignment of responsibility of goals to agents is

thus guided by the goals wished by each agent: if possible, none of the goals for which an agent is responsible should be in conflict with its private goals. It is therefore necessary to check for conflict between a system goal and all the agents' private goals before assigning responsibility for the system goal.

Other responsibility assignment heuristics can be used in addition to the main one above, e.g.,

- minimize multiple responsibilities (to avoid overloaded agents);
- minimize number of agents (to avoid coordination problems);
- avoid overcommunication between agents.

Such heuristics are further discussed in [7].

EXAMPLE: For our elevator system, the first system goal is given by the client and is assigned a joint responsibility of the passenger and the elevatorController (see top goal of the structure presented in Fig. 2). The three AND subgoals suggested by the analyst for that goal are "No delay to passenger", "Safety during transportation", "Satisfaction of transportation requests". In order to meet the first global goal, the three AND subgoals must be met. Responsibilities are assigned for each of these subgoals according to the AssignResponsibility Heuristic.

Assume none of the subgoals is in conflict with any of the private goals of the agents. There is thus no restriction on the assignment of responsibilities: they may be assigned to any of the two agents responsible for the parent goal.

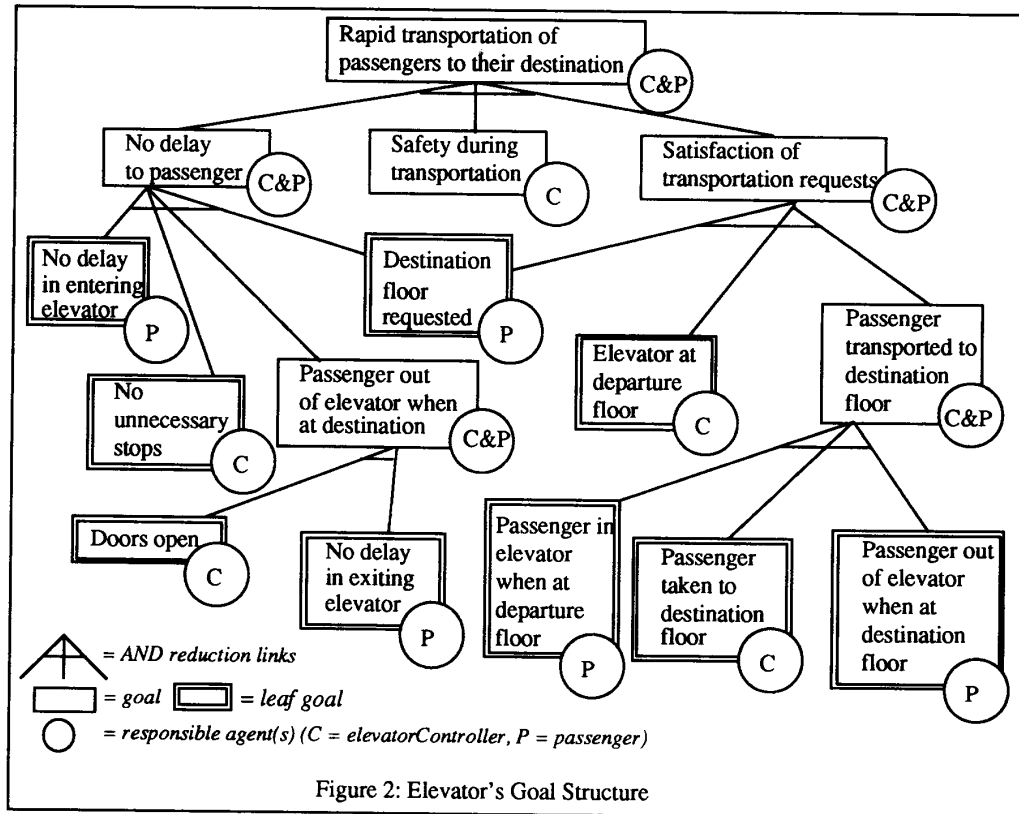
Further in the reduction process, we reach a state where all leaf goals are under the responsibility of a single agent (see leaf goals of Fig. 2). By applying the StopReduction heuristic, we terminate the reduction process.

Step 1.2: Identify agents

WHAT: Agents are identified (i.e., instances of the agent meta-concept are introduced), but not necessarily completely described; all their relevant characteristics might not be discovered at this stage of acquisition. Private goals wished by agents are identified (i.e., instances of the wish meta-relation are defined).

WHY: Agents are needed for the assignment of responsibility during the construction of goal hierarchies (step 1.1). Coroutinging with step 1.1 occurs as new agents are identified while reducing goals in step 1.1.

HOW: Agents are identified by interaction with the analyst. This might be a non-trivial step because in non-human systems, several alternative agents may appear as



candidate processors.

EXAMPLE: The agents identified in our elevator system are:

- Passenger: a passenger agent wishes the following private, application-independent goals: “be where you want”, “stay alive”, “don’t waste time”. He might wish other private goals, but we only consider here the goals that might be interesting in the context of an elevator system.

- ElevatorController: an elevatorController agent wishes the private goal: “limit elevator maintenance”.

Step 2 - Operationalize goals

WHAT: Constraints are introduced to operationalize the leaf goals of goal structures (i.e., instances of the constraint meta-concept are defined, as well as instances of the operationalization meta-relation). They are formal assertions that refer to objects and actions available to agents.

WHY: An operationalization of goals by constraints is necessary for traceability of specification components back to goals. Making constraints explicit is useful for the specification of objects and actions that ensure such constraints in the composite system (see step 3 below, where complete object/action characterizations are acquired).

HOW: The elaboration of constraints and their associated operationalization links is a non-trivial task under control of the analyst. Domain knowledge can be used; some constraints can be acquired through analogical learning when similar goals from other applications or from other domains are being recognized [3].

EXAMPLE: The “safety during transportation” goal can be operationalized with the following constraint:

$\forall \text{ elevator: elevator.state} = \text{'moving'}$
 $\Rightarrow \text{door.state} = \text{'closed'}$.

The insurance of this constraint requires (i) inserting the assertion $\langle \text{door.state} = \text{'closed'} \rangle$ in the pre-condition of the `goToFloor` action and the `takeToFloor` action, (ii) the introduction of a `closeDoors` action that allows states satisfying the right hand side of the implication to be reached, and (iii) the assertion $\langle \text{elevator.state} = \text{'still'} \rangle$ in the pre-condition for the `openDoors` action.

This constraint is operational in the sense that it exclusively refers to objects/actions available to the elevatorController agent.

Step 3 - Acquire object/action characterizations

WHAT: Complete characterizations in terms of attributes and links are acquired for all the objects and actions involved in the specification. Instances of the event, entity, relation, and action meta-concepts must be defined, and linked to the constraints they contribute to en-

sure.

WHY: Complete characterizations must be acquired to ensure the constraints resulting from step 2.

HOW: Concept characterizations are acquired in two substeps: (1) acquisition from constraints, and (2) acquisition from background knowledge.

(1) Constraints are specified by formal assertions that refer to objects and actions. These references are used to drive the acquisition of (i) the type of the corresponding objects (entity, relation, event, etc.), (ii) their attributes and links to other objects, and (iii) the actions responsible for changing their attribute values and links.

For example, the constraint

$\forall \text{ elevator: elevator.state} = \text{'moving'}$
 $\Rightarrow \text{door.state} = \text{'closed'}$

implies the existence of two objects, viz. “elevator” and “door”; they both have an attribute named “state” with domain including ‘moving’ and ‘closed’, respectively; they might be classified as instances of the “entity” meta-concept and linked together by a “part_of” link; the actions “goToFloor”, “stop” and “closeDoors”, “openDoors” would be obtained as actions responsible for changing the value of the “state” attribute of “elevator” and “door”, respectively.

(2) The partial characterizations acquired from constraints are refined and completed interactively by applying concept acquisition operators working on domain-independent knowledge (viz. knowledge about the meta-model and acquisition heuristics), and domain knowledge. Currently, our domain knowledge includes characterizations of typical concepts involved in resource management systems. These concepts are expressed in terms of the KAOS meta-model and structured according to various levels of specification of resource management systems.

The main concept acquisition operators we have identified so far include:

- *instance-to-class acquisition*: acquire a concept from descriptions of instances of it. The resulting concept characterization includes the characteristics of the instances, usually generalized according to inductive or explanation-based learning techniques [12].

- *class-to-class acquisition*: acquire a concept from more specific ones that are defined, or from approximations of it. The resulting concept characterization includes the common characteristics of the considered specific concepts, usually generalized;

- *instance classification*: find in the domain knowledge a possible parent concept for an instance given by the analyst (*instance-of* link between the input given and the target concept). It is interesting to find concepts that are similar to the one being acquired (i.e., that are almost parents) because these similar concepts can be used to

suggest lacking characteristics for the new concept.

- *concept classification*: find in the domain knowledge a possible parent concept for the concept being acquired (*Is-a* link between the new concept and the parent concept). If a real parent concept cannot be found, but other available concepts are almost parent concepts, the characteristics of these concepts can be suggested to complete the characterization of the new concept.

- *analogical acquisition*: recognize similar "source" concepts in the domain knowledge, and transfer their characteristics to the new concept under acquisition, possibly with some adaptation.

A more complete description of these acquisition operators together with examples can be found in [1], [3].

The acquired concepts are described with the help of our requirements modelling language. This language closely reflects the meta-model outlined in Section 2. The organization of some concepts from our elevator system is summarized in Fig. 3.

Step 4 - Assign agents to actions

WHAT: Instances of the performance meta-relation must be defined to produce assignments of agents to actions.

WHY: Agents must perform actions in such a way that the goals they are responsible for are met. A goal is met if the constraints operationalizing it are ensured. Agents must thus perform actions in a way that ensures those constraints.

HOW: The AssignPerformance Heuristic is used to define which agent should best perform which action. As a result of applying this heuristic, a "performing" agent is suggested to the analyst. The analyst can accept or reject the suggestion. The heuristic is based on the indirect link existing between actions and agents through the insurance meta-relation (linking actions and constraints), the operationalization meta-relation (linking constraints and leafGoals), and the responsibility meta-relation (linking agents and goals)-see Fig. 1. The heuristic is the following: "Action Act should be performed by one of the agents responsible for the goals operationalized by the constraints ensured by Act."

The performance meta-relation has three attributes: ability, reliability and motivation. The value of the motivation attribute should always be high. High motivation is ensured by the combination of the AssignResponsibility Heuristic (no agent will be responsible for a goal in conflict with its private goals) and the AssignPerformance Heuristic (agents will perform actions that allow them to satisfy their responsibilities). Values for the ability and reliability attributes of performance links are expected to be defined by the analyst. They allow an additional heuristic to be applied in case the AssignPerfor-

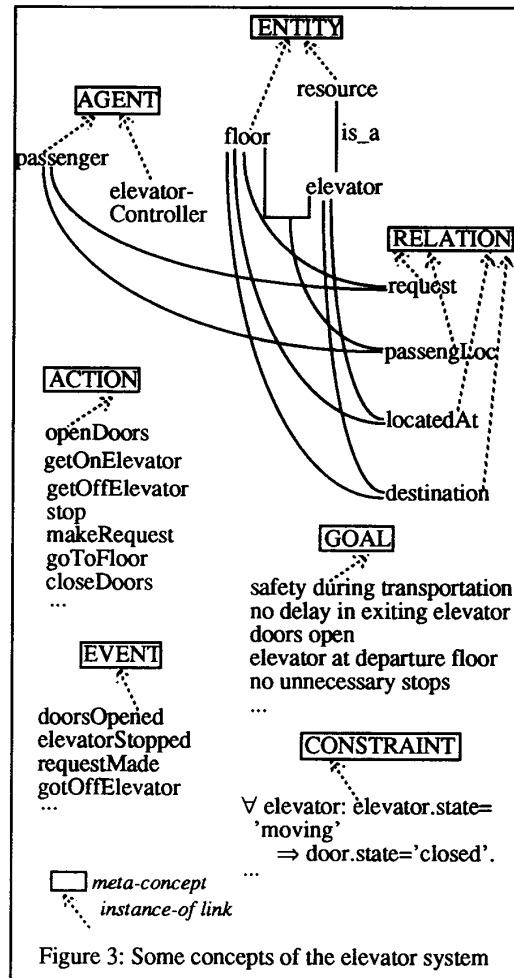


Figure 3: Some concepts of the elevator system

mance heuristic yields several candidate agents. The SelectAgent heuristic is stated as follows: "If there are several candidate agents to perform an action, assign performance links so that the values of the ability and reliability attributes are maximized."

EXAMPLE: Let us come back again to the goal "Safety during transportation" which was operationalized into

$\forall \text{ elevator: elevator.state} = \text{'moving'} \Rightarrow \text{door.state} = \text{'closed'}$

which itself was ensured, among others, by the introduction of the closeDoors action. Two possible candidates for performing this action could be the elevatorController agent and the passenger agent. Since the former has been assigned the responsibility for the safety goal, the application of the AssignPerformance heuristic will result in selecting it to perform the closeDoors action.

4. Conclusion

We have presented an approach for requirements acquisition which is driven by the objectives that have to be met in the composite system under consideration. Goals and their operationalization through specification components are handled explicitly. A meta-model for representing preliminary requirements models has been suggested. Our acquisition strategy can be viewed as a directed way to traverse the meta-mode backwards from the goals. The acquisition strategy and the use of the meta-model have been illustrated with fragments of a specification of a simple elevator system.

The acquisition strategy presented in this paper can be refined further; we plan to formalize it through a body of rules to be evaluated by the Acquisition Assistant. The most critical part of the acquisition strategy is the acquisition of goals and their operationalization through constraints; this should receive special attention and be further investigated to evaluate the opportunities for automated support. For example, the entire tracking of specification components to goals should be visualized; appropriate checks during acquisition should be defined and mechanized as well. Last but not least, goal conflict resolution strategies should be carefully investigated.

We are currently applying our proposals to a real-world software project, that of a multi-agent, distributed meeting scheduler being developed at the University of Oregon.

Acknowledgments

Special thanks to John Anderson, Brian Durney, Rob Helm and Bill Robinson for critiquing successive versions of the acquisition strategy and the meta-model.

We also thank Martin Feather for helpful comments on an earlier version of this paper.

The KAOS Project is supported by the Belgian Ministry of Scientific Affairs in the context of the Belgian Incentive Program for Fundamental Research in Artificial Intelligence under grant RFO/AI/13. The research reported here was done while the first author was visiting the University of Oregon on an additional grant (AI/13/3) from that same Ministry.

References

- [1] Dardenne A., van Lamsweerde A., *Towards Concept Acquisition in Requirements Elicitation*, Internal report No. 12, KAOS Project, Institut d'Informatique, Facultés Universitaires de Namur, Belgium, 1990.
- [2] Derry E., Feather M., Fickas S., Helm R., *Deriving Interface Requirements through Composite System Design*, TR 90-12, Computer Science Dept., University of Oregon, Eugene, OR 97403, May 1990.
- [3] Dubisy F., van Lamsweerde A., *Requirements Acquisition by Analogy*, Internal report No. 13, KAOS Project, Institut d'Informatique, Facultés Universitaires de Namur, Belgium, 1990.
- [4] Dubois E., van Lamsweerde A., "Making Specification Processes Explicit", *Proceedings 4th Intl. Workshop on Software Specification and Design* (Monterey, Ca), IEEE Cat. No THO181-8, April 1987, pp. 169-177.
- [5] Ellman T., "Explanation-Based Learning: A Survey of Programs and Perspectives", *ACM Computing Surveys*, Vol. 21, N°2, June 1989, pp. 163-222.
- [6] Feather M., "Language Support for the Specification and Development of Composite Systems", *ACM Transactions on Programming Languages and Systems*, Vol. 9, No. 2, April 1987, pp. 198-234.
- [7] Feather M., Fickas S., Helm R., "When Things Go Wrong: Predicting Failure in Multi-Agent Systems", *Niagara Workshop on Intelligent Information Systems*, April 91 (also available as TR-91-15, Computer Science Dept., University of Oregon, Eugene, OR 97403).
- [8] Mostow J., "A Problem Solver for Making Advice Operational", *Proceedings AAAI-83*, Morgan Kaufman, 1983, pp. 279-283.
- [9] Nilsson Nils J., *Problem-Solving Methods in AI*, McGraw Hill, 1971
- [10] Robinson W., "Integrating Multiple Specifications Using Domain Goals", *Proceedings 5th International Workshop on Software Specification and Design*, May 19-20, 1989, pp. 219-226.
- [11] van Lamsweerde A., Dardenne A. and Dubisy F., *KAOS Knowledge Representations as Initial Support for Formal Specification Processes*, RP 24/91, Facultés Universitaires de Namur, May 1991.
- [12] van Lamsweerde A., "Learning Machine Learning", in *Introducing a Logic Based Approach to Artificial Intelligence*, A. Thayse (Ed.), Vol. 3, Wiley, 1991.