Goal-directed Concept Acquisition in Requirements Elicitation

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

CIS-TR-91-08
March 13, 1991

*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. A conceptual meta-model in terms of which requirements models are acquired is first briefly presented. 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.

Department of Computer and Information Science, University of Oregon
Eugene, OR 97403

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 being involved:
- The acquisition of knowledge about the *composite* system involves concepts that usually are not found in the final formal specification, such as, e.g., goals to be achieved, agents involved and their responsibilities, etc. (In the following, we will use the term "composite system" [Fea87] to refer to the whole system made of the application to be automated and of the part of its environment which is of interest in formulating requirements).
- Acquisition processes are more relying on domain knowledge whereas formal specification processes are more relying on knowledge about the specification formalism being used.
- The basic operators being applied in such processes are rather different; requirements elicitation involves learning operators [Lam91] whereas formal specification involves data/operation decomposition and structuring, modularization, parameterization and instantiation, etc [Dub87].

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). Requirements elicitation 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. Among the major tasks of requirements elicitation are: (i) the acquisition of goals, (ii) the specification of those goals, and (iii) the integration of divergent goals [Rob90]. Once goals are acquired, they must be operationalized in a specification. Goal acquisition is thus a critical task for the Acquisition Assistant.

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

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 justify the presence of specification components;

- 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 [Rob89];

- They allow to trace back low-level descriptions, incomprehensible for clients, to global client goals.

This paper presents a strategy for concept acquisition which relies on the goals that must be met in a specification. The conceptual meta-model in terms of which requirements models are being acquired according to that strategy is first briefly discussed. 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. For example, the characteristics of the action meta-concept are propagated to the openDoors action concept: the openDoors action has a pre_ and a post_condition, a trigger_condition, etc; these characteristics are inherited from the meta-attributes of the action meta-concept. 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 (optional), the inheritance mode through specialization hierarchies, etc.

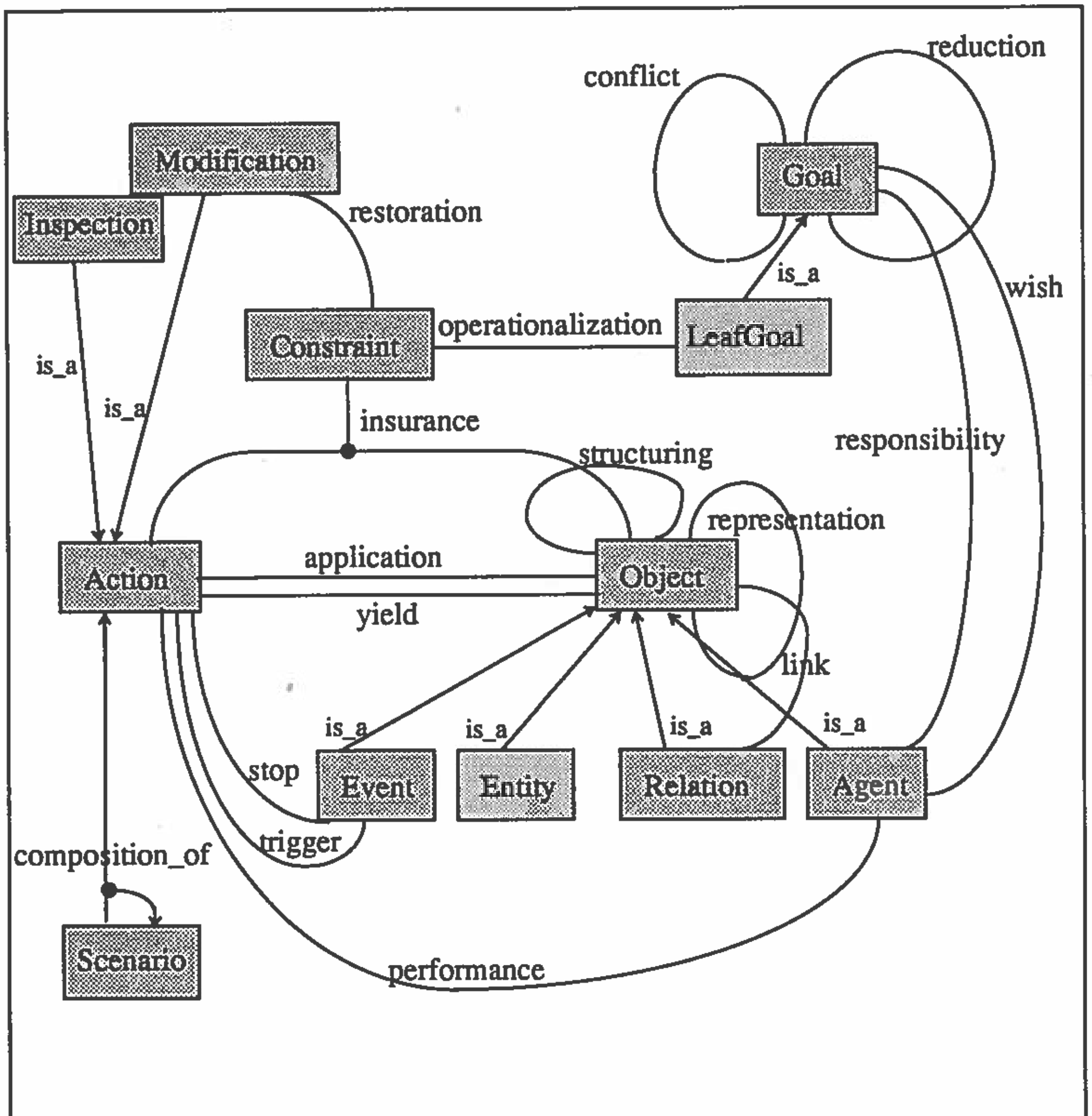


Figure 1: The KAOS Conceptual Meta-model

The meta-model is just outlined here; a precise definition of its various components can be found in [Lam90]; 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: ***application*** and ***yield*** between action and objects, ***insurance*** between constraint and actions or objects; ***trigger*** and ***stop*** between event and action; ***link*** between relation and objects; ***performance*** between agent and actions; ***reduction*** over goals; ***responsibility*** between agent and goals; ***operationalization*** between leafGoal and constraints [Mos83] (those constraints are ensured by restrictions defined on actions and objects under control of agents); ***composition-of*** between scenario and action; ***representation*** on objects (any instance of it 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; ***frequency*** for the event 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 subsystem to be automated); etc.

3. A Goal-directed Acquisition Strategy

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

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

3.1 Agents and Goals

3.1.1 Agents

In KAOS, an agent is an object which is a processor for one action at least (as any object, it can evolve from one state to the other).

3.1.2 Goals and Constraints

Our notions of goal and constraint can be defined as follows:

- A goal is an objective that has to be met by the composite system.

- A constraint is an operational assertion on object states which satisfaction contributes to the achievement of one or several goals.

In order to clearly distinguish goals from constraints, their specific features are more precisely described hereafter.

Goal

- A goal is in general not formalized; in general it cannot be described exclusively in terms of objects and actions of the composite system being considered: 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. An agent *is responsible for* a goal if it must guarantee, maybe jointly with others, that the goal will be achieved in the system (see [Fea87]). An agent can guarantee that a goal will be met by performing appropriate actions. Goals for which agents are responsible are called system goals. System goals are application-specific. System goals have to be met by the composite system.
An agent *wishes* a goal if it wants it to be met. Each agent may have its own goals that it wishes to be met. Goals wished by agents are called local goals. An example of local goal is the “gain time” goal wished by a passenger. Local goals are most often application-independent. Local goals do not have to be met by the composite 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 modeling

[Nil71]. 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 can be explicitly recorded. Two goals are linked via a conflict meta-relation instance if and only if they cannot be achieved both together.

Constraint

- A constraint can be formalized; it is *operational* in that it refers to actions and objects available to agents. Constraints are formalized in the (first-order) assertion language used to express pre_-conditions, post_conditions, trigger_conditions, etc.

An example of constraint is

$$\forall \text{ elevator: elevator.state='still'} \Rightarrow \exists \text{ floor} \mid \text{position(elevator,floor)}.$$

(This constraint says that when the elevator is not moving, it must be positioned at a floor).

Note that constraints are not directly linked to agents (the responsibility and wish relations apply to goals rather than constraints).

Link between Goal and Constraint

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 of them then contributes to the achievement of the goal. The operationalization meta-relation captures the fact that a constraint is some sort of abstract "implementation" of one or more leaf goals. Our model here somewhat refines the notion of operationalization in explanation-based learning ([Mos83], [Ell89]), 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 "go to requested floor" can be operationalized by being associated with the following constraint:

```
if performs(x,makeRequest) and
    applies-to(makeRequest, floor:f)
then triggers(ev,goToFloor) and
        applies-to(goToFloor,f).
```

This constraint is operational in the sense that it is exclusively described in terms of specification components (entity, action and event). More examples of goals, constraints and operationalization links can be found in Section 3.2.2.

3.2 Acquisition Strategy

For each step of our strategy, we explain what the step does, why it is necessary to do it at this 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 in fact to a directed way of acquiring instances of meta-concepts and meta-relations. Starting from the goal meta-concept, the specification is acquired incrementally and backwards 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 [Doe90]. We consider a single elevator system. The maximum capacity of the elevator is one passenger at a time. Only one passenger is considered at a time: elevator requests are treated in a fully sequential way (there is no interference between requests). This problem is a simplification of the elevator described in [Fea87].

3.2.1 Step 1: Identify Agents and Goals

This step consists of two co-routining substeps.

Step 1.1: 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 necessarily be discovered at this stage of acquisition. Local 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 (substep 1.2), this is the reason why they must be identified at this stage of acquisition. Coroutining with substep 1.2 occurs as new agents might be identified while reducing goals in substep 1.2.

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. In social systems, agent roles are more easily identified.

EXAMPLE:
The agents identified in our elevator system are:

- *Passenger*.
A passenger agent wishes the following local, application-independent goals: "be where you want", "stay alive", "gain time". He might wish other local 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 local goal: "limit elevator maintenance".

A specification methodology such as Structured Common Sense [Fin87] also includes an explicit agent identification step; however, goals and their operationalizations are not explicitly handled there.

Step 1.2: 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 conflicts between newly defined system goals and the local goals of agents (i.e. define instances of the conflict meta-relation);
- (iii) for each goal, assign responsibility to agent(s) (i.e. define instances of the responsibility meta-relation).
These three substeps are not sequential, they are intertwined:

- the assignment of responsibilities determines the need for further reduction in subgoals;
- the state of the reduction process determines the assignment of responsibilities;
- the assignment of responsibilities may highlight the need for identifying a new agent and its associated local goals, which must be checked for conflict with system goals.

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

- it records division of responsibility;
- it ties specification components to informal text describing goals for the analyst's use;
- it can be used in case of negotiation required to solve conflicts, or to replay some part of the acquisition process.

Each goal identified is checked for conflict with each of the goals wished by the agents. Detection of possible conflicts between local and system goals is useful for the assignment of responsibility (see below the Assignment of Responsibility Heuristic).

HOW: Goal structures are refined with the help of heuristics.
- Goals are reduced with the help of the Reduction Heuristic.

Reduction 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 more simple to achieve than a goal under the responsibility of a larger group of agents, because more agents require more complex cooperation. This is the reason why responsibility is our criterion for identifying simple goals: using this criterion, even if an analyst should have a (vague) notion of concepts and characteristics to refine goals, he does not exclusively rely on this notion to find appropriate subgoals. Nevertheless, the identification of subgoals by the analyst remains a non-trivial task which would be impossible to automate.

- The End-of-Reduction Heuristic is applied to determine when to stop refining the goal structure. This heuristic relies on our belief that the less responsible agents, the simplest the operationalization of the goal by constraints.

End-of-Reduction 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.

- The assignment of *responsibility* of goals to agents is 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 wished goals. It is therefore necessary to check for conflict between a system goal and all the local goals before assigning responsibility for the system goal.

The assignment of responsibility for a goal to agent(s) is done according to the Assignment of Responsibility Heuristic described below.

<u>Assignment of Responsibility Heuristic</u>
(for assigning responsibility for goal G)

a) *G has no parent goal*

--- The agents which may be responsible for G are any of the system agents (there is no restriction on the assignment of responsibility for a root goal).

b) *G has at least one parent goal*

Given:

- $A=\{a_1, ..., a_n\}$ the set of agents responsible for the parent goal(s) of G;
- $g(a_i)$ the set of goals wished by agent a_i $(1 \le i \le n)$

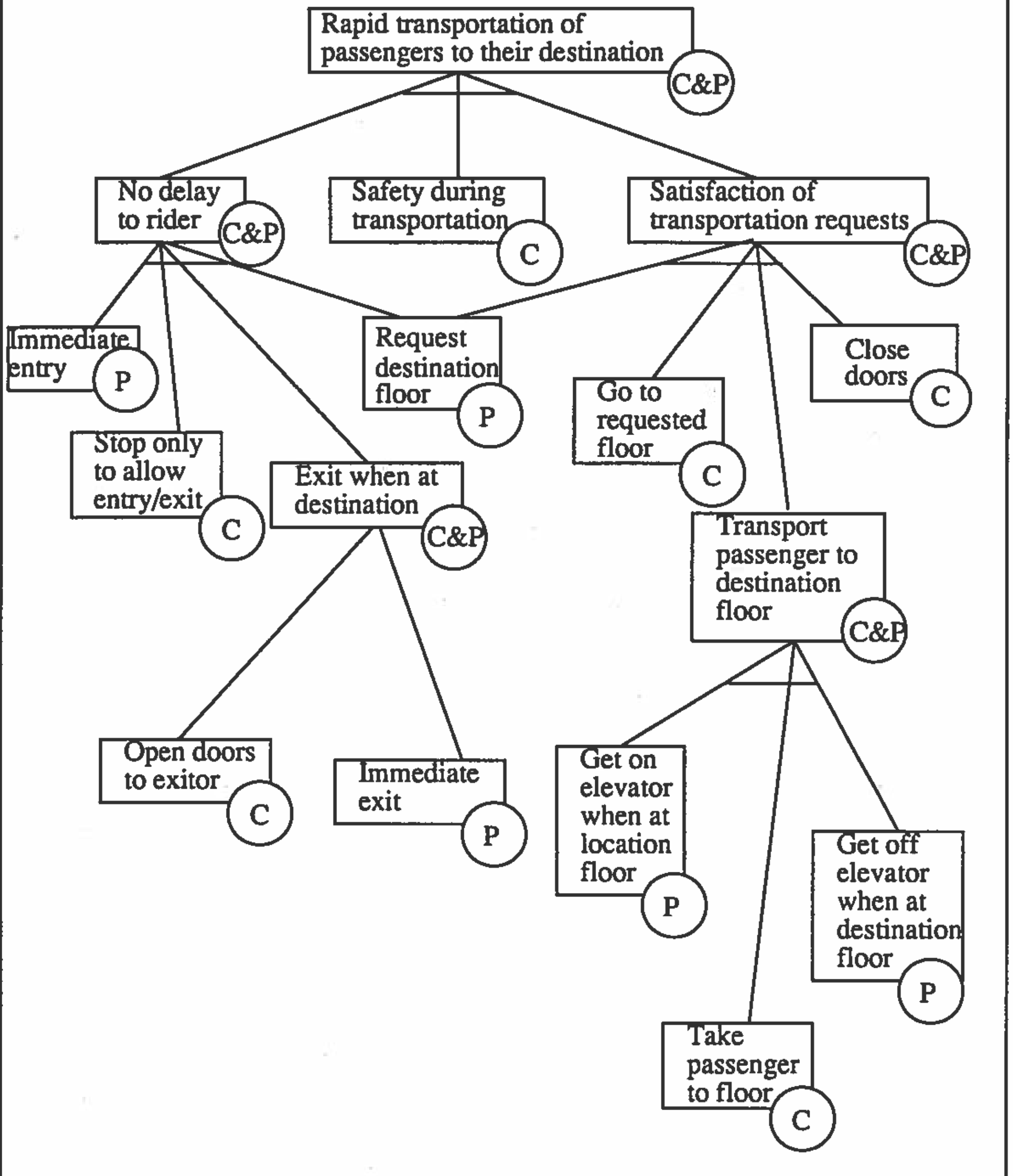
Then:

The only agents which may be responsible for G should belong to the set

$$\{a \in A \mid \exists\, g \in g(a) \text{ s.t. } \mathrm{conflict}(g,G)\}.$$

The expression conflict(g,G) means that there is an instance of the conflict meta-relation between the two goal instances g and G.

<u>EXAMPLE</u>:
For our elevator system, the first system goal is given by the client. This goal is identified as being "rapid transportation of passengers to their destination". The responsibility for this goal is assigned to both the elevatorController and the passenger. This goal is reduced into AND subgoals and the responsibilities are split among the different agents.
The three AND subgoals suggested by the analyst are "no delay to rider", "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 to each of these subgoals according to the Assignment of Responsibility Heuristic. None of the subgoals is in conflict with any of the local 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. As the goals "no delay to rider" and "satisfaction of transportation requests" have shared responsibilities, they should be further reduced.
Further in the reduction, a conflict is detected between the local "limit elevator maintenance" goal of the elevatorController and the system "request destination floor" goal. There is a conflict because any transportation request implies more work for the elevator and thus more maintenance. Making a request is thus in conflict with limiting maintenance. According to the Assignment of Responsibility Heuristic, the elevatorController may not be responsible for the "request destination floor" goal: the only agent which may be responsible for the "request destination floor" goal is passenger.
We apply the End-of-Reduction Heuristic, and we stop refining the goal structure when all leaf goals are under the responsibility of a single agent. The resulting structure is shown in Figure 2.


Rapid transportation of passengers to their destination
C&P
No delay to rider
C&P
Safety during transportation
C
Satisfaction of transportation requests
C&P
Immediate entry
P
Request destination floor
P
Go to requested floor
C
Close doors
C
Stop only to allow entry/exit
C
Exit when at destination
C&P
Transport passenger to destination floor
C&P
Open doors to exitor
C
Immediate exit
P
Get on elevator when at location floor
P
Get off elevator when at destination floor
P
Take passenger to floor
C

Key:

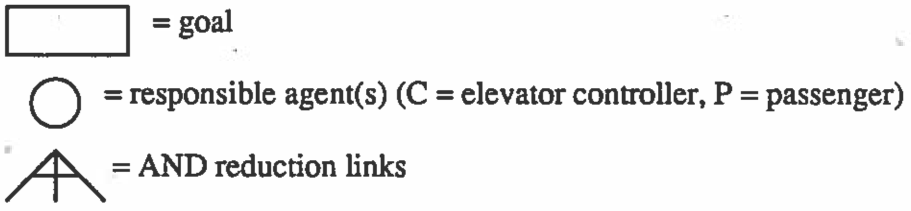

= goal
= responsible agent(s) (C = elevator controller, P = passenger)
= AND reduction links

Figure 2: Elevator's Goal Structure

It should be noted that local goals are not reduced. Once they are identified during step 1.1, they are not further reduced. The leaf goals of system goals structures do not include any refinement of local goals.

3.2.2 Step 2: Operationalize Goals

WHAT: Constraints are associated to the leaf goals of goal structures in order to operationalize these leaf goals (i.e. instances of the constraint meta-concept are defined, as well as instances of the operationalization meta-relation). Each constraint is an operational assertion on objects and actions.

WHY: Constraints are described to operationalize the leaf goals of the structure acquired in step 2. Such a operationalization of goals in necessary for traceability of specification components to goals. The identification of constraints is useful for the description of concepts required to ensure the constraints in the composite system (see the insurance meta-relation); complete concept characterizations are acquired during the following step.

HOW: Constraints are acquired by interaction with the analyst. We believe that the discovery of constraints and their associated operationalization links is a non-trivial task that cannot be fully automated with the Acquisition Assistant.
The analyst might already have in mind some concepts and associated characteristics when he describes the constraints. The support given by the Acquisition Assistant for the acquisition of constraints may consist of consistency checking based on the different types of concepts described in the meta-model: e.g., check that the same concept is not referred to as an action and in another constraint as a relation, etc.
On another hand, some constraints can be acquired through analogical learning when similar goals are being recognized [Dub90].

EXAMPLE:
Each of the leaf goals of the goal structure of our elevator system must be operationalized by being associated with constraints.
For example, the "safety during transportation" goal can be operationalized with the following constraint:

$\forall$ elevator: elevator.state='moving' $\Rightarrow$ elevator.doorState='closed' .

The satisfaction of this constraint requires inserting (i) the assertion <elevator.doorState='closed'> in the pre-condition for the goToFloor action, (ii) that same assertion in the pre-condition for the takeToFloor action, (iii) the introduction of the closeDoors action that allows states satisfying the right hand side of the implication to be reached, and (iv) the assertion <elevator.state='still'> in the pre-condition for the openDoors action.
The goal "immediate entry" can be operationalized with the constraint:

<u>if</u> performs(x,openDoors) <u>then</u> triggers(ev,getOnElevator).

The satisfaction of this constraint requires (i) the introduction of the openingDoors event, (ii) the definition of the openingDoors event as triggering the getOnElevator action, and (iii) the definition of the openDoors action as yielding an instance of the openingDoors event. As a result, the end of the execution of the openDoors action creates a new instance of the openingDoors event. The openingDoors event triggers the getOnElevator action. As soon as the doors of the elevator open, the passenger must immediately get on the elevator, resulting in the "immediate entry" goal being met.

More operationalization links between leaf goals and constraints for the elevator system are presented in Figure 3.

"safety during transportation"	$\forall$elevator: elevator.state='moving' $\Rightarrow$ elevator.doorState ='closed'
"immediate entry"	if performs(x,openDoors) then triggers(ev,getOnElevator)

"stop only to allow entry/exit"	if performs(x,stop)
	then position(elevator,f1) and
	destination(elevator,f2) and
	f1=f2

"open doors to exitor"	if performs(x,stop) then triggers(ev,openDoors)
"immediate exit"	if performs(x,openDoors) then triggers(ev,getOffElevator)

"go to requested floor"	if performs(x,makeRequest) and
	applies-to(makeRequest,floor:f)
	then triggers(ev,goToFloor) and
	applies-to(goToFloor,f)

"get on elevator when at location floor"	if position(elevator,f1) and location(passenger,f2) and request(passenger,f3) and f1=f2=f3 then performs(x,getOnElevator)
"get off elevator when at destination floor"	if location(passenger,elevator) and position(elevator,f1) and request(passenger,f2) and f1=f2 then performs(x,getOffElevator)
"close doors"	if performs(x,getOffElevator) then triggers(ev,closeDoors)

Key: "leaf goal" $\xrightarrow{\text{operationalization}}$ constraint

Figure 3: Elevator's Goals and Constraints

3.2.3 Step 3: Acquire Concept Characterizations

WHAT: Complete concept characterizations are acquired for all the concepts of the specification considered. All the characteristics of the concepts (i.e. attributes and relations) must be fully described. At this stage of the acquisition process, instances of the event, entity, relation, and action

meta-concepts are defined, and constraints are related to these concepts (i.e. instances of the insurance meta-relation are defined).
Agents are not considered here because their characterizations at this stage of the acquisition process should already be almost complete. The only missing information should be which actions they perform. This will be defined in the next acquisition step.

WHY: Complete characterizations must be acquired so as to ensure the constraints resulting from step 3.

HOW: Complete concept characterizations are acquired in two substeps:
- (i) acquisition from constraints;
- (ii) acquisition from background knowledge.

(i) Constraints are operational assertions on characteristics of objects and actions; the referenced characteristics must be explicitly defined. As constraints are described in a structured constraint language, they can be used to drive the acquisition of concepts and characteristics they are refering to. E.g., the constraint

∀ elevator: elevator.state='moving' ⇒ elevator.doorState='closed'

implies the existence of the concept "elevator" and the characteristics "state" and "doorState" for this concept.
Thus, goals and constraints are essential because they provide partial concept characterizations to begin with; the concept acquisition process thus has not to start from scratch.

(ii) The partial characterizations acquired from constraints are then refined and completed interactively by learning operators working on domain background knowledge (e.g., describing typical concepts in the domain of resource management systems), and domain-independent background knowledge (viz. knowledge about the meta-model and acquisition heuristics). Currently, our domain background knowledge includes characterizations of various concepts related to resource management systems: these concepts identify some basic characteristics of resource management systems together with multiple specializations of them [Dar90b]. Hierarchies of predefined concepts (from previously acquired specifications) already exist and are stored in the background knowledge. They are used during the acquisition of new concepts.

The concept acquisition operators presented in [Dar90c] can be applied to acquire missing knowledge about concept characterizations. The main concept acquisition operators we have identified so far include:

- instance-to-class acquisition: acquisition from instances of the considered concept. The resulting concept characterization includes the characteristics of the instances, usually generalized according to inductive or explanation-based learning techniques [Lam91].

- class-to-class acquisition: acquisition from more specific already defined concepts, or from approximations of concept characterizations. The resulting concept characterization includes the common characteristics of the considered specific concepts, usually generalized;

- instance classification: search in the background knowledge for a possible parent concept of an instance given by the analyst (instance-of link between the instance and the concept). As the instance is intended to define a new concept, it should not be possible to find among the already defined concepts a concept that could be its parent. In fact, it is interesting to find concepts that are similar to the one being acquired (that are almost parents) because these similar concepts can be used to suggest lacking characteristics for the new concept.

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

- analogical acquisition: a similar source concept is recognized in the background knowledge, and its characteristics are transferred to the new concept under acquisition, possibly with some adaptation.

A complete description of these acquisition operators together with examples can be found in [Dar90c], [Dub90].

The acquired concepts are described with the help of our requirements language. The model of our requirements language is the meta-model concisely described in Section 2.

EXAMPLE:
For our elevator system, at this stage of the acquisition process, we must still acquire complete characterizations for instances of the following meta-concepts: event, action, entity, relation. We will consider each type of concept in turn, indicating for each type what information from goal operationalization can be usefully used to simplify the acquisition task.

Event Characterization Acquisition
The characterizations of five events can be automatically inferred from constraints operationalizing goals: stopping, openingDoors, requesting, gettingOff and gettingOn. The characterizations of events are complete with the information contained in the constraints: we know when they occur and which actions they trigger.

Action Characterization Acquisition

Each action may be characterized by the following attributes: pre- and postconditions, trigger and stop conditions, duration.
All the constraints identified in step 3 describe pre- and postconditions of actions. Some actions can thus be already partially characterized. Nevertheless, all actions do not have their pre- and postconditions already described. The missing pre- and postconditions must thus be acquired by interaction with the analyst. The trigger and stop conditions could be derived from the events associated to the actions. The duration must be explicitly given by the analyst.

Actions can fulfill roles in several meta-relations. These roles must be defined for each action in order to have complete action characterizations.
The instances of the trigger meta-relation between actions and events have already been described during event characterization acquisition. Instances of the performance meta-relation are acquired during the next acquisition step.

Actions and events are the concepts that benefit the most from the information of the operationalization of goals: they are almost completely characterized by the information contained in the operationalizing constraints.

Entity Characterization Acquisition

Partial characterizations of entities can be acquired from the operationalizing constraints. The two attributes of elevator (doorState and state) are already given by the constraints, but must still be completely described. Their informal description and domain must be defined. The informal descriptions are given by the analyst. The domains can be inferred from the values appearing in the constraints: 'closed' and 'open' for doorState, and 'still' and 'moving' for state. The client is asked if additional values should be included in the domains. To complete the elevator characterization, the Acquisition Assistant tries to classify this concept. As we believe that most of the systems can be described as resource management systems, there should always be at least one entity that is a resource. The Acquisition Assistant suggests to classify elevator as a specialization of resource. This reminds the client of defining an attribute similar to the state attribute of a resource describing its availability. This lacking attribute is defined as the load attribute taking the value 'empty' or 'loaded'.

Relation Characterization Acquisition

Partial characterizations of relations can be acquired from the operationalizing constraints. If we consider the request relation, it is defined between passenger and another unidentified concept. In order for the characterization to be complete, the (i) unidentified related concept, (ii) cardinalities and (iii) other possible attribute have to be determined.
(i) To discover the unidentified related concept, the Acquisition Assistant suggests first each of the application-specific entities and agents. Entities and agents are considered before relations and events (the two other types of concepts that might also be related) because usually the application-specific relations are defined between entities and agents. In our case, the unidentified related concept is the floor entity.
(ii) The cardinalities are acquired by depicting to the client various situations where different cardinality values are considered. He decides on which are relevant, and the Acquisition Assistant infers the corresponding cardinalities.
For the request relation, one role is fulfilled by passenger, and the other role is fulfilled by floor. A passenger may not have more than one request at a time (cardinality 0-1 for the role fulfilled by passenger). When the passenger arrives at his requested floor, his associated request relation is deleted (postcondition of getOffElevator). There may be zero, one or several requests for the same floor (cardinality 0-N for the role fulfilled by floor).
(iii) According to the constraints and action characterizations, the request relation has an attribute describing its state. This attribute can take the values 'active' or 'pending'. A request is pending when the passenger has made his request and the request has not yet been considered. A request is active when it is being served by the elevator.

The agents, entities and relations of the elevator system are presented in Figure 4.

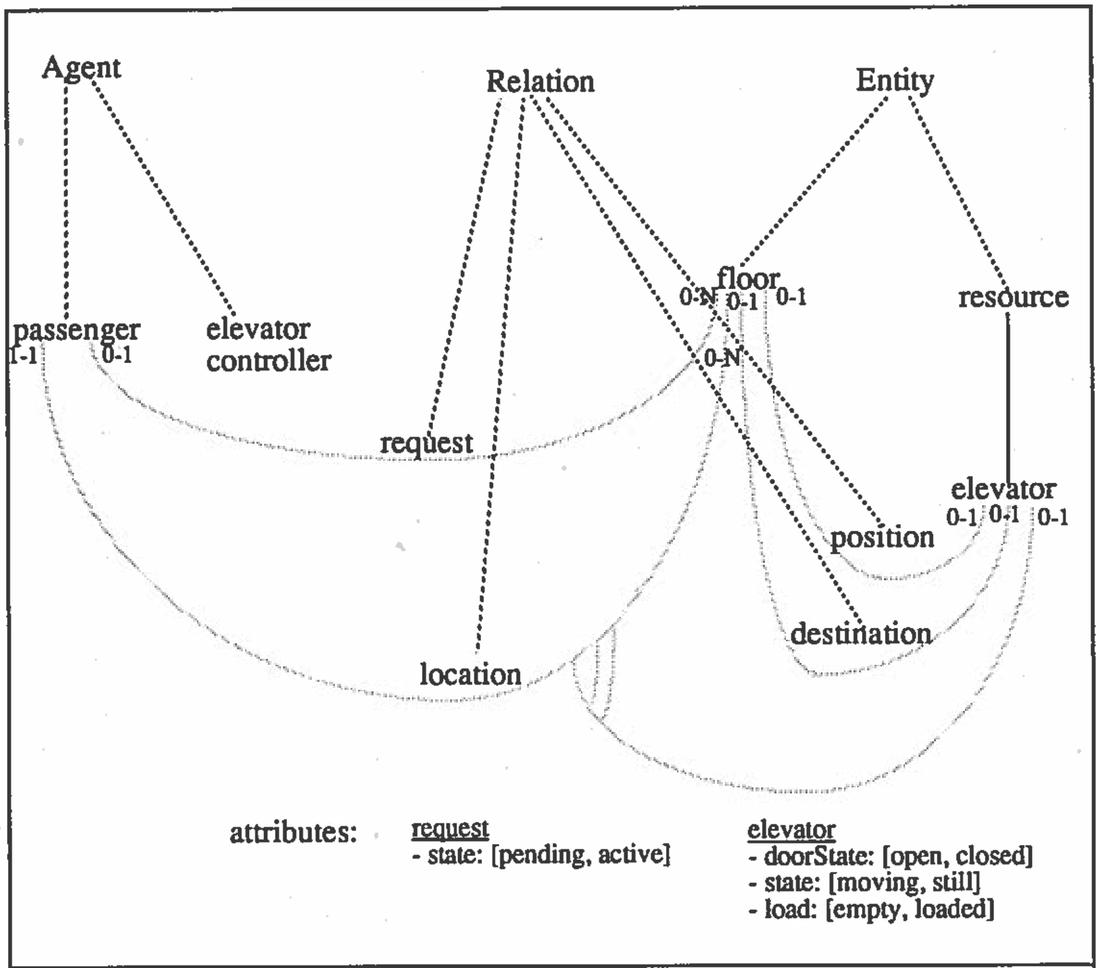
3.2.4 Step 4: Assign Agents to Actions

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

WHY: Agents must perform actions so that constraints that operationalize those goals which they are responsible for are ensured.

HOW: A 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 (between actions and constraints), the operationalization meta-relation (between constraints and leafGoals), and the responsibility meta-relation (between goals and agents)-see Fig. 1. The heuristic used to assign performance is described hereafter.

We don't take into account for the assignment of performance constraints that describe synchronization between several actions, because those constraints do not only refer to a single action: they refer to at least two actions. Therefore, it is not possible to deduce specific information about *one*



Key:

--------- = instance-of link
——— = is-a link
.............= application-specific relation
Blabla = meta-concept
blabla = concept

Figure 4: Elevator's Agents, Entities and Relations

particular action from such constraints. As we need specific information to assign performance, we reject synchronization constraints.
When the heuristic is not applicable for an action (n=0), the Acquisition Assistant suggests to choose between the agents responsible for actions similar to the action considered.

Let us illustrate the use of this heuristic with the help of our elevator example (see Fig. 5).

The action stop is associated with two operationalizing constraints:

```
- c1: if performs(x,stop)
      then position(elevator,f1) and
           destination(elevator,f2) and
           f1=f2.
```

<u>Assignment of Performance Heuristic</u>

(for the assignment of performance of action Ac)

<u>Given</u>:
- c1, ..., cn the constraints ensured by the action Ac, which do not describe synchronization between several actions;
- g1, ..., gm the goals operationalized by c1, ..., cn.

<u>Then</u>:
The action Ac should be performed by one of the agent(s) responsible for g1, ..., gm.

- c2: <u>if</u> performs(x,stop)
  <u>then</u> triggers(ev,openDoors).

If we apply the Assignment of Performance Heuristic, we only consider the first constraint because c2 describes a synchronization between the stop and the openDoors actions (see Fig. 5). The c1 constraint operationalizes the goal "stop only to allow entry/exit". This goal is under the responsibility of the elevatorController agent. This agent should thus perform the stop action.

The interest of such a heuristic is not only to automatically suggest performing agents so that the analyst simply has to acknowledge, but it is also a way to check the consistency of the decisions taken by analysts. When an analyst spontaneously assigns a performing agent to an action, the Acquisition Assistant applies the heuristic. If the agent suggested by the heuristic is different form the one chosen by the analyst, the analyst is notified of the difference.

```
action stop
  applies_to elevator with arg e
  yields stopping with res st,
          elevator with res new_el
  performed_by elevatorController
  pre_condition: <position(e,f1) and destination(e,f2) and f1=f2>
  post_condition: <e.state='still'>
  ...                                    ↖
end stop
                                          synchronization
action openDoors                         ↙
  applies_to elevator with arg e
  yields openingDoors with res st,
          elevator with res new_el
  performed_by elevaotrController
 triggered_by stopping
  pre_condition: <e.state='still'>
  post_condition: <e.doorSTate='open'>
  ...
end openDoors
```

<u>Fig. 5: Synchronization between two actions of the elevator system</u>

The performance meta-relation has three attributes: ability, reliability and motivation. We currently consider that the analyst must give values for the ability and reliability attributes. The value of the motivation attribute should always be high. High motivation is ensured by the combination of the Assignment of Responsibility Heuristic (no agent will be responsible for a goal in conflict with its local goals) and the Assignment of Performance Heuristic (no agent will perform an action ensuring a constraint operationalizing a goal for which the agent is not responsible).

4. Conclusion

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

The acquisition strategy presented in this paper should 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.

Acknowledgments

Special thanks to John Anderson, Brian Durney, Rob Helm and Bill Robinson for critiquing the successive versions of the acquisition strategy and the meta-model according to their view of goals and agents in composite systems.

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

The KAOS Project is supported by the Belgian Ministery 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 Ministery.

References

[Dar90a] Dardenne A., Delcourt B., Dubisy F., van Lamsweerde A., *A Conceptual Meta-model for Representing Product-level Knowledge in Requirements Acquisition*, Internal report No. 8, KAOS Project, Institut d'Informatique, Facultés Universitaires de Namur, Belgium, 1990.

[Dar90b] Dardenne A., Dubisy F., *A Description of Resource Management Background Knowledge*, Internal report No. 10, KAOS Project, Institut d'Informatique, Facultés Universitaires de Namur, Belgium, 1990.

[Dar90c] 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.

[Doe90] Doerry 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.

[Dub90] Dubisy F., van Lamsweerde A., *Requirements Acquisition by Analogy*, Internal report No. 13, KAOS Project, Institut d'Informatique, Facultés Universitaires de Namur, Belgium, 1990.

[Dub87] Dubois E., van Lamsweerde A., "Making Specification Processes Explicit", *Proceedings 4th Intl. Workshop on Software Specification and Design* (Monterey, Ca), IEEE Cat. Nr THO181-8, April 1987, pp. 169-177.

[Ell89] Ellman T., "Explanation-Based Learning: A Survey of Programs and Perspectives", *ACM Computing Surveys*, Vol. 21, N°2, June 1989, pp. 163-222.

[Fea87] 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.

[Fin87] Finkelstein A., Potts C., "Building Formal Specifications Using 'Structured Common Sense' ", *Proceedings 4th Intl. Workshop on Software Specification and Design* (Monterey, Ca), IEEE Cat. Nr THO181-8, April 1987, pp. 108-113.

[Lam90] A. van Lamsweerde, A. Dardenne and F. Dubisy, *KAOS Knowledge Representations as Initial Support for Formal Specification Processes*, Report N°15, KAOS Project, Facultes Universitaires de Namur, December 1990.

[Lam91] van Lamsweerde A., "Learning Machine Learning", in *Introducing a Logic Based Approach to Artificial Intelligence*, A. Thayse (Ed.), Vol. 3, Wiley, 1991.

[Lub89] Lubars M., *Representing Design Dependencies in the Issue-Based Information System Style*, MCC Technical Report N. STP-426-89, 3500 West Balcones Center Drive, Austin, TX 78759, 1989.

[Mos83] Mostow J., "A Problem Solver for Making Advice Operational", *Proceedings AAAI-83*, Morgan Kaufman, 1983, pp. 279-283.

[Nil71] Nilsson Nils J., *Problem-Solving Methods in AI*, Mc Graw Hill, 1971

[Rob89] Robinson W., "Integrating Multiple Specifications Using Domain Goals", *Proceedings 5th International Workshop on Software Specification and Design*, May 19-20, 1989, pp. 219-226.

[Rob90] Robinson W., Fickas S., *Negotiation Freedoms for Requirements Engineering*, CIS-TR-90-04, University of Oregon, Eugene, OR 97403, April 1990.