Goal-driven approaches to requirements engineering

Stewart Green

stewart.green@uwe.ac.uk

Department of Computing,
University of the West of England,
Bristol

Abstract

A number of recent approaches to requirements engineering suggest that a new paradigm is emerging based upon the use of goals. To gain insight into these approaches, and thus into the emerging paradigm, five of them were surveyed, and four of the five were also applied to a common case-study: "the meeting scheduler problem". Knowledge gained from the surveys and case-studies enabled both the main contributions of each approach to goal-oriented requirements engineering, and the similarities and differences between the approaches to be identified. In addition the knowledge facilitated the sketching of an alternative approach based upon a synthesis of the "best" features of the approaches surveyed.

1 Introduction:

Within the discipline of software engineering, a number of recent approaches to requirements engineering suggest that a new paradigm is emerging based upon the use of goals. In this context goals are requirements which describe states to be either achieved, maintained or avoided by a system. In the new paradigm such goals provide a basis for deriving and evaluating designs. These features are exemplified in the approaches described below. They include an approach to the specification of composite systems developed by Martin Feather [Fea87] at ISI (Information Sciences Institute); a development of this work, the Critter approach [FH92]; the Knowledge Acquisition in AutOmated Specification (KAOS) approach [DvF93]; the Generic Modelling Approach to Requirements Capture (GMARC) [BJT+94]; and an approach which addresses non-functional requirements, developed by Lawrence Chung at Toronto University [Chu91].

In this paper I have attempted to elucidate the approaches by reviewing each, and applying all but one (the Critter approach [FH92] was excluded for reasons

given below) to a common example: the Meeting Scheduler System [vDM92] (See appendix A). I have also attempted to identify the main contribution of each approach, the similarities and differences between approaches, and any problems I feel an approach might have. Finally I suggest a synthesis which I hope may make them more useful to software practitioners.

2 The approaches:

2.1 The ISI approach:

As figure 1 shows, the ISI approach [Fea87] involves the creation of specifications of composite systems. These are systems which comprise multiple interacting, autonomous components, where each component might be, for example, a piece of software, or a human, or perhaps a machine. An elevator system is an example of a composite system; other examples include process control systems, operating systems and communication systems.

The input for the process is a "closed specification" of a composite system. A "closed specification" models both the desired behaviour of the composite system expressed in terms of global constraints, and an initial analysis of the composite system expressed in terms of putative agents, their actions, and the information they have access to.

The objective of the process is to decompose the global constraints into an equivalent set of sub-constraints where each sub-constraint may be satisfied by the actions of a single agent.

During the process an analyst first uses the initial analysis to identify agents, their actions, any information to which they alone have access, and their interfaces to local information owned by other agents.

The analyst next decomposes the global constraints into sub-constraints until each sub-constraint can be made the sole responsibility of one agent. Feather [Fea87] suggests that "decomposition of a constraint is achieved by choosing an implication of the constraint to make into a separate, explicit constraint, and thereafter simplifying the original constraint." He continues: "Judicious choice of the implication will give a constraint whose responsibility can be simplified to an individual agent."

When the process ends, the initial global constraints have been transformed into a set of equivalent sub-constraints each of which has been assigned as the responsibility of one agent. It is intended that each agent uses its actions, its access to local information, and its access to interface information to meet its constraints.

Agents are expressed in the specification language GIST [BGW82] augmented with the notion of agents. Each agent specification has a generative part, which generates a behaviour, i.e. a set of candidate histories (where each history is a sequences of actions), by using the state changing actions specified for the agent combined with control structures (sequentiality, conditionality, parallelism, etc); each agent also has a constraint portion denoting predicates

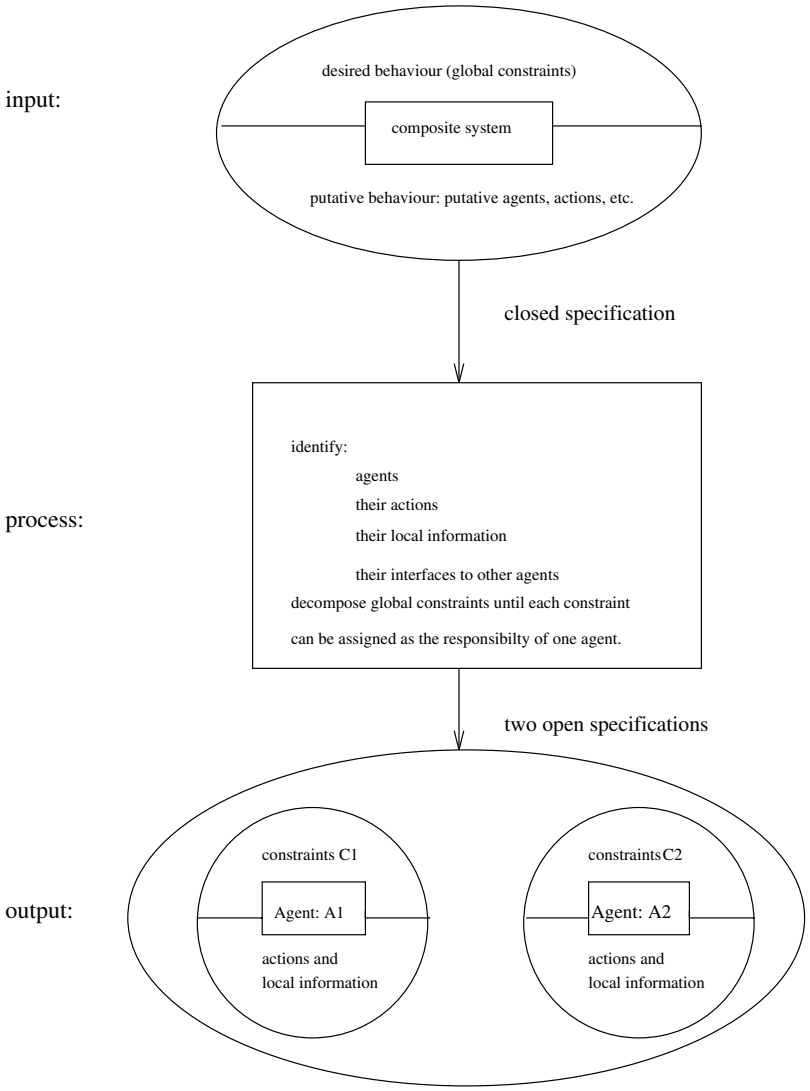

Figure 1: Overview of Feather's approach

which have to be satisfied by candidate histories. So a GIST agent specification denotes a behaviour which is the set of candidate histories satisfying the agent's constraints.

Each agent description may be viewed as an "open specification" of one of the components of the composite system: the behaviour of the component, in terms of its inputs, is defined by both the component's generative description (its actions) and its assigned constraints. The inputs to the component are completely defined by the generative descriptions of the other agents in the closed system and the constraints assigned to them.

2.1.1 Applying the approach to the Meeting Scheduler problem:

To illustrate the ISI approach further, I have attempted to apply it to the Meeting Scheduler problem referred to in the introduction. The results of this application are depicted in figure 2.

The approach was applied to the global constraint "Meeting scheduled rationally", which is the responsibility of all the agents in the meeting scheduler composite system: meeting scheduler, all intended participants, meeting convener, and so on.

Having identified global constraints, the approach encourages an analyst to select and remove an implication of the constraint, and then to simplify the remainder. One implication of the chosen constraint is that a meeting be requested and accepted. This is shown in figure 2 as (2a). After this was identified, the original constraint was then simplified to "Requested meeting scheduled rationally".

From figure 2 it can be seen that only two agents are responsible for the constraint "Meeting requested and accepted". However, when this is decomposed in turn via (3a) and (3b), the resulting constraints are each the responsibility of just one agent.

The remainder of the diagram was produced in a similar way.

2.1.2 Tool support:

There is currently no existing tool support for the ISI approach.

2.2 The Critter approach:

Fickas and Helm [FH92] have developed further the ideas behind the ISI approach to tackle the same problem, i.e. the specification of composite systems. This section describes the development.

The purpose of the Critter approach is to create open specifications for composite systems.

As figure 3 shows, the input to the Critter process is an initial (composite system) design state. This is equivalent to a closed specification in the ISI approach. The initial design state has a constraint part, where the desired

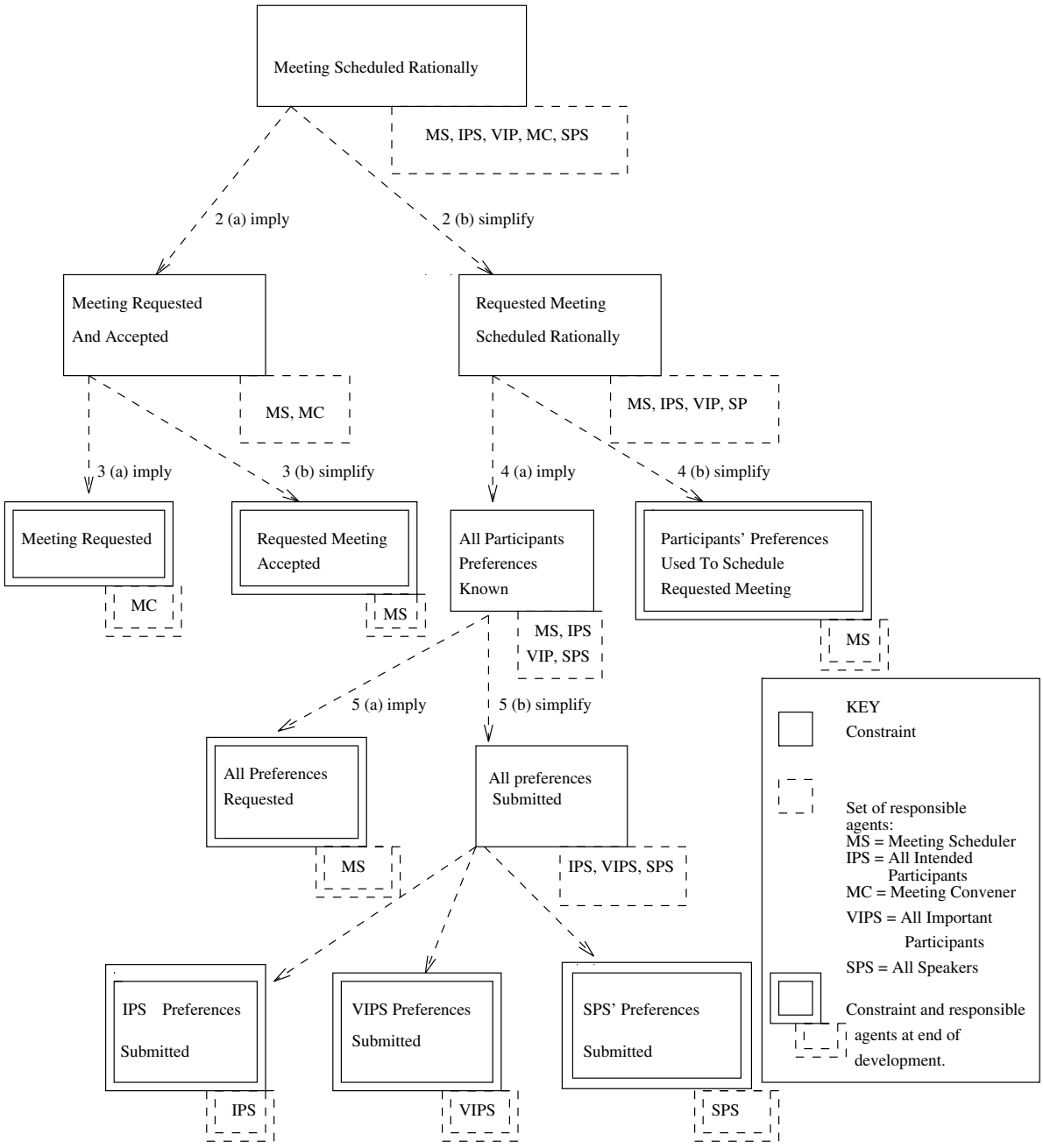

Figure 2: Application of the ISI approach to the MS system

behaviour of a composite system is modelled as a set of global constraints, expressed using a first order sorted temporal logic based on ERAE [DH89]. It also has a system part, where an initial description of a composite system's behaviour is expressed using a superset of Numerical Petri Nets [WH85] augmented with the concept of agents.

The objective of the Critter process is to decompose all the global constraints into a set of (not necessarily equivalent) sub-constraints where each sub-constraint is satisfied by the actions of one or more agents.

The Critter process interactively generates a design state space, i.e. an inverted tree of design states. Ideally this should contain at least one design state for which the behaviour produced by the system part (i.e. by the agents through their actions) satisfies all the constraints in the constraint part. Such a state is called a leaf design state.

A design state space is created as follows: Starting from an initial design state an analyst develops each global constraint in turn. The analyst may use three different tools (see below) to determine whether a design state's system description satisfies a constraint, i.e. to determine whether it is a leaf design state (with respect to the constraint). If the tools indicate that the selected constraint is violated by the system behaviour, the analyst may use his or her knowledge to select a Critter "operator" to apply to the design state to create a new design state where satisfaction of the constraint is more likely.

Each Critter operator embodies the framework of a composite system design strategy, e.g. the means of dividing responsibility among agents in a particular context. An operator consists of a matching pattern and a replacement pattern. An analyst applies the matching pattern to a design state by mapping each element of the pattern to an element of the design state. This generates a replacement pattern that introduces new concepts in a new design state. The new concepts might consist of a modified constraint, e.g. a weakened constraint, or new agents, or a new assignment of responsibilities to existing agents, and so on.

The analyst may be able to match several different operators to a design state and thus be able to produce several new design states. In this way the analyst may interactively create and explore the design state space generated from the initial design (see figure 3).

After an operator has been applied, the analyst uses the tools to check again whether the new system behaviour satisfies the constraint. This process of checking design states with the tools and decomposing them with the operators continues until a leaf design state (with respect to the constraint) is produced. At this point the analyst considers another constraint, and so on until all the constraints have been considered.

The output from the process is a design state space. Ideally it should contain at least one leaf design state. For leaf design states, the behaviour generated by their system agents is congruent with the final version of their global constraints.

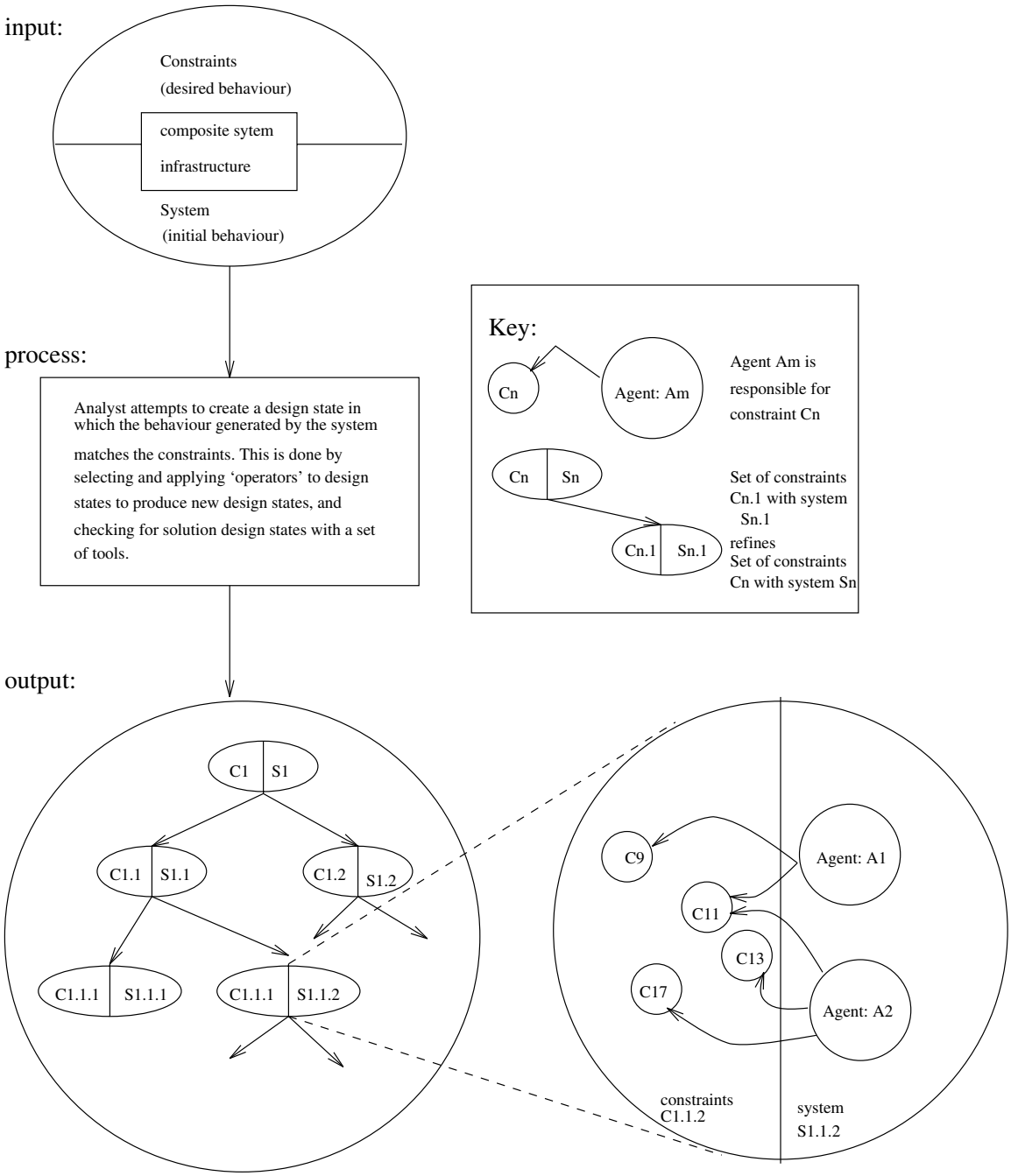

Figure 3: Overview of the Critter approach

2.2.1 Applying the approach to the Meeting Scheduler problem:

It was not possible to apply this approach to the meeting scheduler problem because general purpose operators are not available. (The operators which were developed [Hel93] were specific to a "railroad" example [FH92].)

2.2.2 Tool support:

The Critter approach is supported by three analysis tools. One tool uses a system description to attempt to generate a sequence of actions that leads to a violation of a constraint. Another allows the NPN system description to be "run" forward, thus simulating the behaviour of the composite system. The third creates a reachability graph and answers questions about it, such as, in the context of a meeting scheduling system, "is it possible for a requested meeting not to be scheduled eventually?" The latter tool is implemented on a Macintosh [Hel93].

In addition, operators were developed [Hel93] to a sufficient extent to enable Fickas et al. to tackle a "railroad example" [FH92].

2.3 KAOS:

Before either of the two approaches described above may be used for the design of a composite system, a description of its required behaviour and an initial analysis of its structure must be available. But how is such information obtained?

The aim of the KAOS approach [DvF93] is to generate such information by acquiring and formalising functional and non-functional requirements (i.e. goals) for a composite system.

Figure 7 shows the inputs to KAOS. (The meta model is actually an integral part of KAOS, but is included here as an input as it may be changed occasionally.) The meta model is like an ERA schema; here it characterises composite systems. As such it comprises meta concepts, e.g. goal, constraint, agent, action...; meta relationships, e.g. agent-performs-action, action-ensures-constraint, constraint-operationalises-goal...; meta attributes characterising both, e.g. pre-condition is a meta attribute of the action meta concept and strengthened pre-condition is a meta attribute of the ensures meta relationship; and meta constraints, e.g. constraints defining the cardinality of a meta relationship. Figure 4 shows a part of a KAOS meta model.

The order and manner in which meta objects are acquired is determined by the meta model traversal strategy (or acquisition strategy). The meta model could be traversed backwards, either from identified agents, or from client supplied scenarios; however, for the remainder of this section, only backwards traversal from a client's requirements (or goals) is considered. Knowledge about traversal strategies is included in meta knowledge used by a computer based acquisition assistant to guide an actual traversal. The assistant also uses domain knowledge of varying degrees of abstraction in order to perform analogical reasoning.

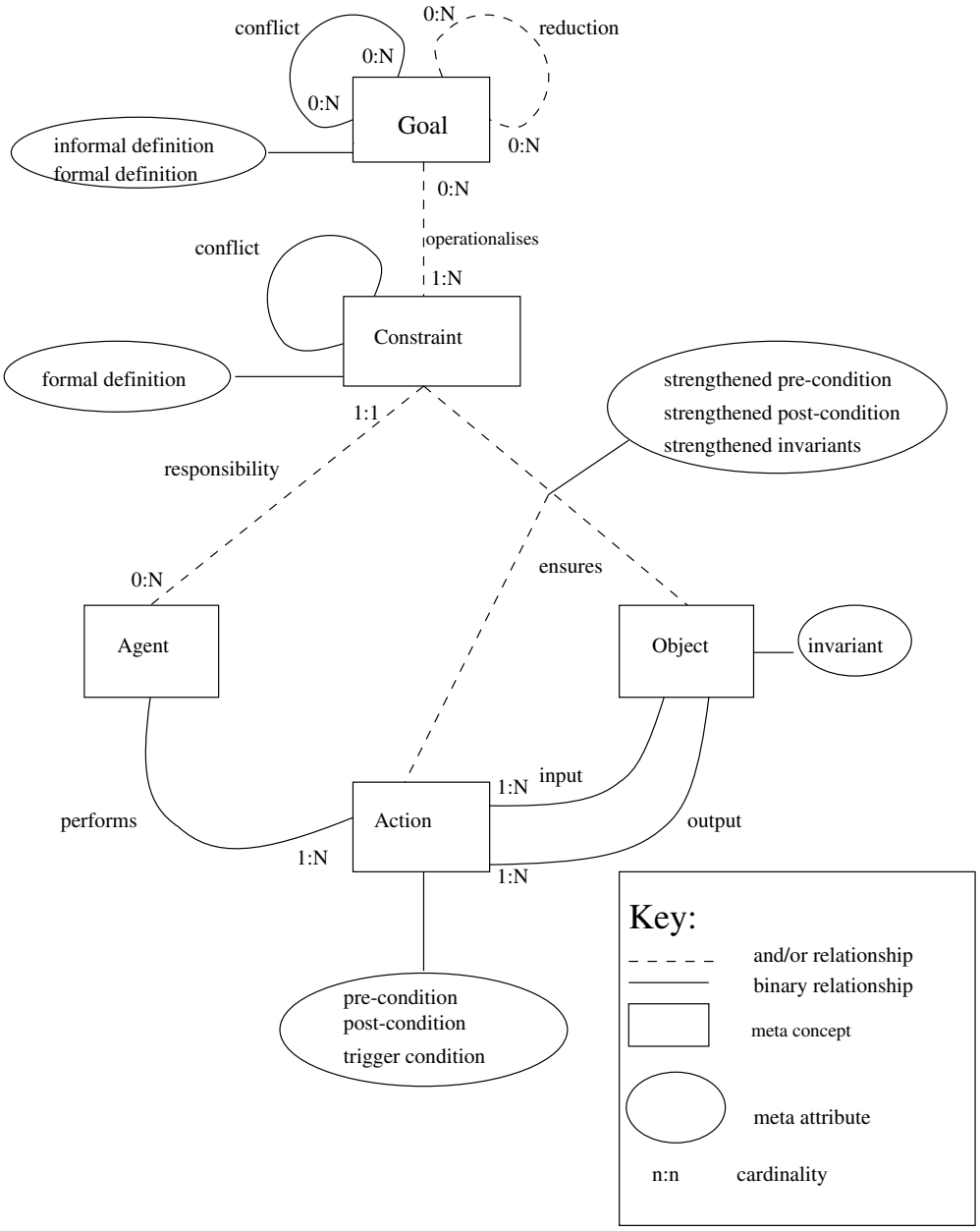

Figure 4: A portion of a KAOS meta model

The main objective of the KAOS approach is to elicit requirements for an application and to represent them in a formal structure called a requirements model.

The initial basic source of information for this approach, using a goal-driven strategy, is a client's expression of functional and non-functional requirements for a composite system. These are input to the seven stage KAOS process, which is shown in figure 5.

During the KAOS process, instances of each of the meta objects discussed above are acquired as a requirements model is gradually created.

During stage one, the highest level goals of a system are identified from the requirements. Goals are deemed to be system objectives which cannot be met by the actions of just one agent. Such goals are reduced to an equivalent set of objectives each of which can be met by the actions of one agent. These objectives are called constraints and are formally defined during stage three. The reduction is achieved by decomposing goals into one or more sets of sub-goals, where each member of a set contributes to the satisfaction of a parent goal, and satisfying all the sub-goals in a set completely satisfies the parent goal [Nil71].

If it is possible, goals should be expressed formally using a first-order temporal logic (currently a formalism inspired by ERAE [Dub91] is used by the KAOS group).

As the goal decomposition proceeds, objects (agents, entities, relationships, and events) referred to in formal and informal goal descriptions are identified and abstracted for use in stage two.

During stage two, objects associated with goals are reviewed, and agents (objects which control state transitions) are identified along with their actions and any pre-conditions, post-conditions, and trigger-conditions for their actions. This knowledge of agents and actions is then used in stage one to identify when a goal may be reduced to a constraint, i.e. to identify leaf-goals. Thus stages one and two may be viewed as co-routining stages.

In stage three, each leaf-goal of a goal structure is converted into a constraint whose formal definition is expressed in terms of objects and actions available to some agent identified during stage two.

In stage four, new actions and objects described in the formal definitions of constraints during stage three are identified and acquired. In addition, new characteristics of already acquired concepts, such as new meta attributes or new pieces of invariants, pre-conditions, post-conditions, and trigger-conditions are also identified.

In general, actions, pre-conditions, post-conditions, and trigger-conditions identified in stage two are not necessarily sufficient to ensure that each constraint will be satisfied. So in stage five each constraint is examined and consequently, to ensure that each constraint will be met, some actions and objects may be modified, and new actions and objects may be introduced. Such modifications might involve strengthening an action's pre-condition, for example, or strengthening an object's invariant. This new information is held in a requirements model as meta attribute values of the ensures meta relationship. In addition,

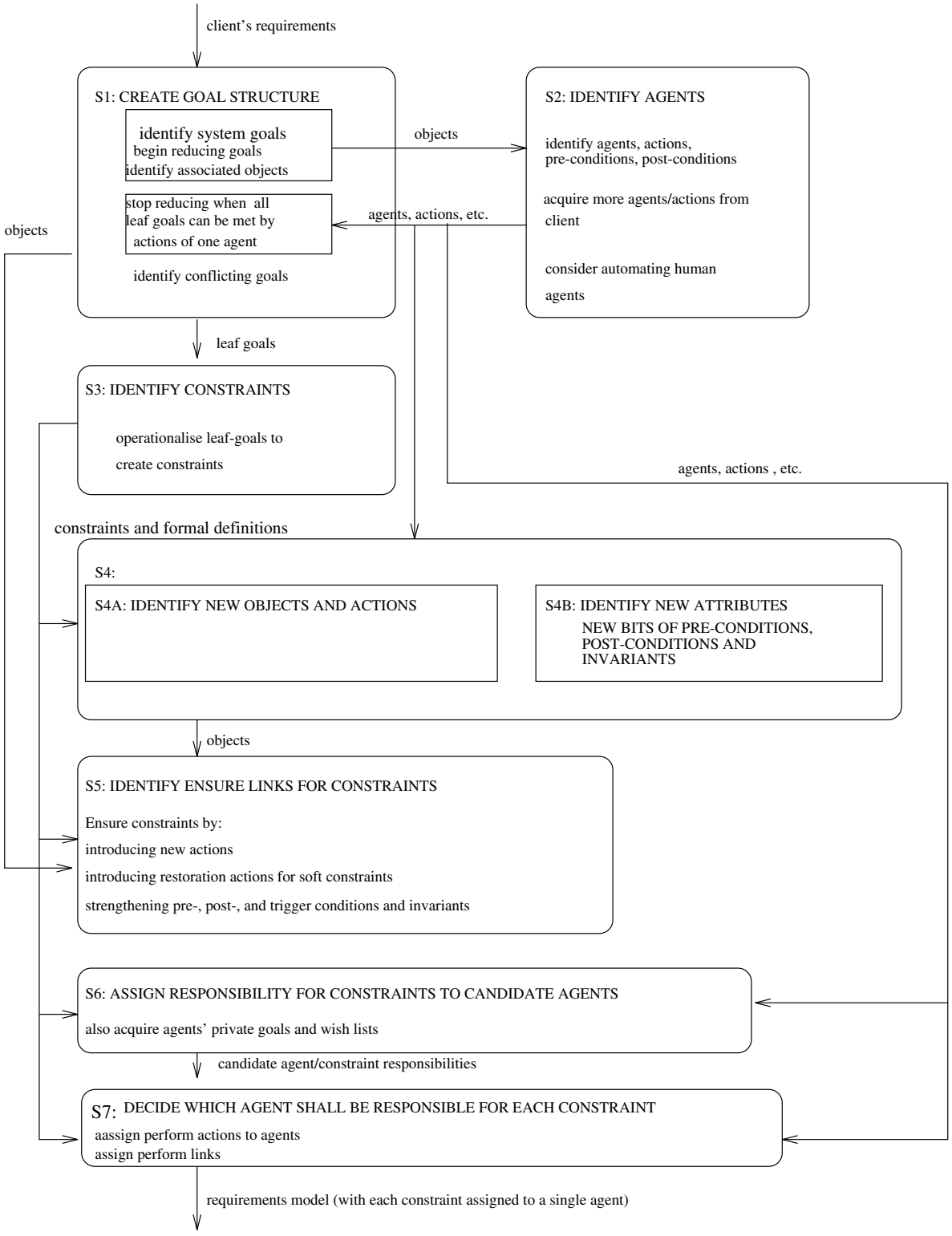

Figure 5: The KAOS process

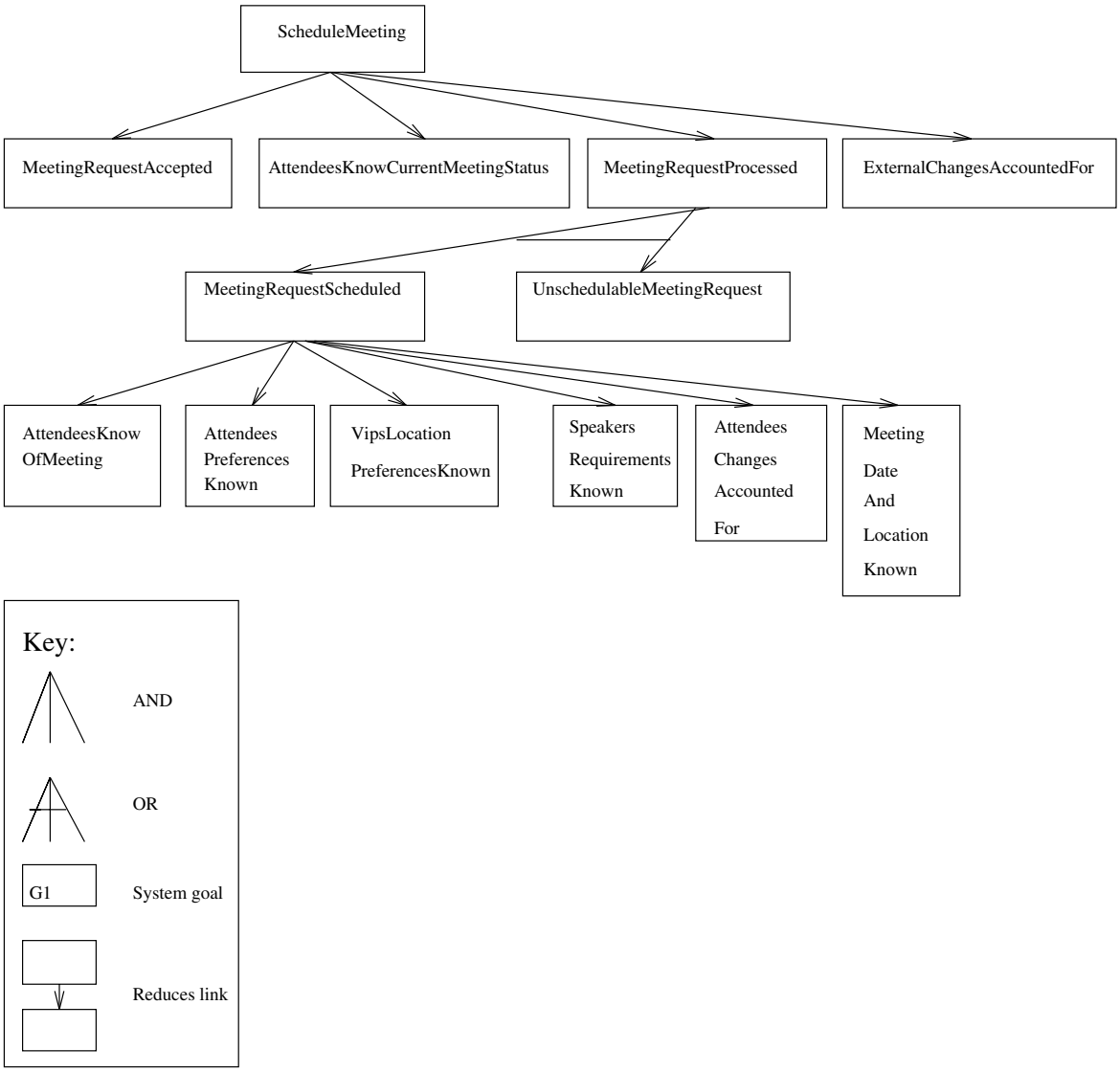

Figure 6: Partial decomposition of a meeting scheduler system goal

new actions are acquired which allow soft constraints (see below) which may have been violated to be restored.

In a complete requirements model each constraint is the responsibility of a single agent. During stage six, each constraint is reviewed in turn and agents which might be made responsible for it are linked to it by instances of the responsibility meta relationship.

Finally, in stage seven, the actions required to satisfy a constraint are assigned to the "best" agent from among the candidate agents for satisfying that constraint.

The rationale for the meta model and the acquisition process is as follows: The system goals correspond to a client's requirements; these are satisfied if all the leaf goals of the generated goal structure can be satisfied; these in turn can be satisfied if the constraints operationalising them can be met.

The formal definition of each constraint defines a set of sequences of states (a behaviour in Feather's terms) in terms of properties of objects in the states, i.e in terms of patterns of states. In order to move from one state in a sequence to the next, actions must be performed on one or more of the objects in the input state. In general, a set of actions (with appropriate pre-conditions, post-conditions and trigger conditions) acting on objects (conforming to appropriate invariants) will be needed to meet a constraint.

A number of agents may be able to perform actions which will enable them to satisfy the constraint. One of these is selected as the agent which is responsible for meeting the constraint; this one has the appropriate actions assigned to it.

The approach is used to create a requirements model for a composite system. This model comprises, among other things, a set of agents, where each agent has a set of actions and a set of constraints for which it has been assigned responsibility. Different constraints may conflict with one another.

It should be clear that by either judiciously selecting agents and constraints so as to avoid conflicting constraints, or by resolving conflicting constraints, the starting point for the Critter approach might be obtained from a requirements model.

2.3.1 Applying the approach to the Meeting Scheduler problem:

I now illustrate the KAOS approach by applying it to the Meeting Scheduler problem. The initial basic source of information for this approach, using a goal-driven strategy, is a client's expression of functional and non-functional requirements for a composite system. For example, for the meeting scheduler system, such goals include the following: "to determine, for each meeting request, a meeting date and location so that most of the intended participants will effectively participate", and "privacy rules should be enforced; an ordinary participant should not be aware of constraints stated by other participants".

A client's requirements are input to the seven stage KAOS process, which is shown in figure 5.

Stage one:

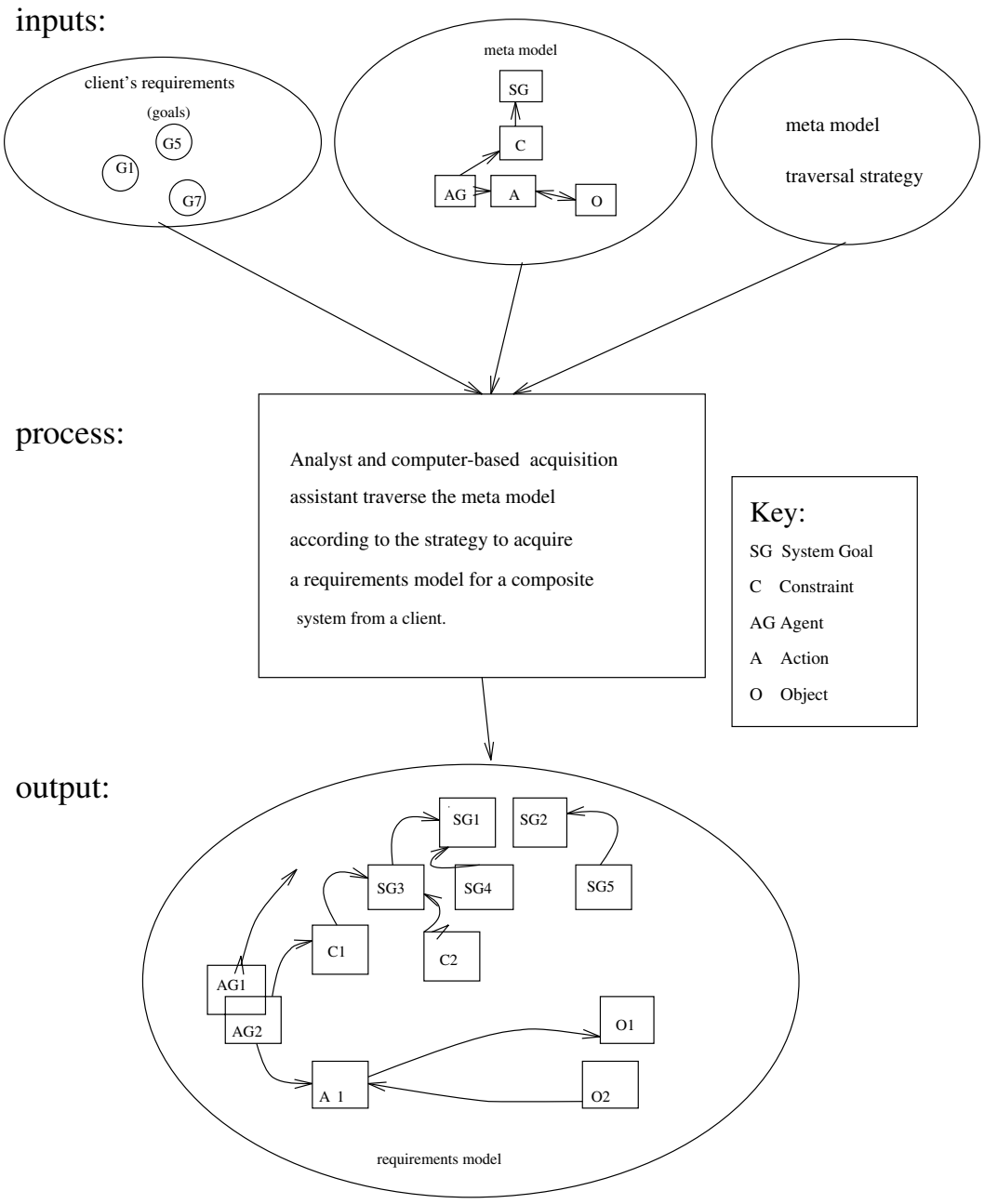

Figure 7: Overview of the KAOS approach

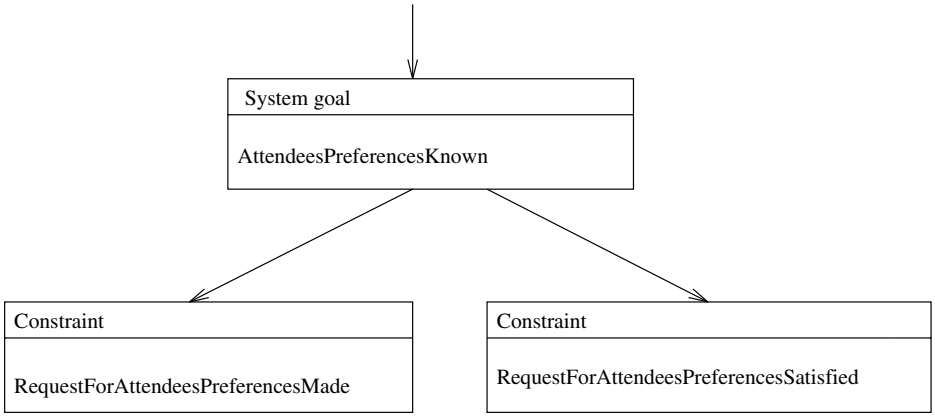

Figure 8: Operationalising goals into constraints

Figure 6 shows the results of partially decomposing one of the meeting scheduler system's functional goals.

Stage two:

From complete goal descriptions (not given) for the goals referred to in figure 6, it is possible to identify from the ScheduleMeetings goal the following objects: meeting request, date, location, meeting scheduler, attendee (i.e. meeting participant); and actions: participate. During stage two, these objects and actions are reviewed, and Attendee and MeetingScheduler are identified as agents, and Participate (at meetings) is identified as an action of the Attendee agent. Similarly, an AuthorisedUser agent, and RequestMeeting and AcceptMeeting actions are also identified during stage two.

Stage three:

Continuing the example, the AttendeesPreferencesKnown system goal is operationalised into two constraints: a hard constraint (hard constraints must not be violated), RequestForAttendeesPreferencesMade, and a soft constraint, RequestForAttendeesPreferencesSatisfied. This is shown in figure 8. The first constraint may be satisfied by a RequestAttendeesPreferences action available to the MeetingScheduler agent; the second may be satisfied by the SubmitPreferences action available to Attendee agents. It is assumed that both actions were acquired from the client during stage two.

Stage four:

The two constraints in the example contain no new objects or actions, nor do they contain anything which completes the description of objects and actions already identified.

Stage five:

RequestForAttendeesPreferencesSatisfied is a soft constraint which signifies that all the requests for attendee preferences will be satisfied within a predefined time period. If the constraint is violated, actions are required to help to restore it. One such action might be SendPreferenceRequestReminder; in the meeting

scheduler requirements model this action would be linked to the constraint by an instance of the restoration meta relationship.

Stage six:

In the example, the Attendee agent is linked by a responsibility meta relationship to the RequestForAttendeesPreferencesSatisfied constraint, and the SystemScheduler agent is similarly linked to the RequestForAttendeesPreferencesMade constraint. Had other agents been identified earlier which might also have been made responsible for either of these constraints, then they would also have been similarly linked.

Stage seven:

The RequestForAttendeesPreferencesMade is assigned to the MeetingScheduler agent by an instance of the performs meta relationship link.

Figure 9 shows a fragment of the requirements model acquired for the meeting scheduler system by the approach just described.

2.3.2 Tool support:

There are no tools currently available for supporting the KAOS approach. However there is a plan to instantiate a generic process tool to produce a tool to guide the KAOS acquisition process [Dar94].

2.4 Toronto approach:

I next turn to an approach developed developed at Toronto University [Chu91] which uses non-functional requirements to help to select both design components and implementation components for information systems. Clearly, the degree to which particular non-functional requirements are met in an implementation depends upon design and implementation decisions that have been made during the development, and in particular upon how each decision affects each non-functional requirement. To capitalise on this idea the Toronto approach encourages both analysis of non-functional requirements, and exploration of the impact of alternative design and implementation decisions on non-functional requirements.

Before proceeding to a detailed description of the Toronto approach and an example of its use, it is worth noting that it is just one part of a more comprehensive approach for developing information systems [CPM+91] This latter approach, an outcome of the DAIDA project [JMSV92], is represented in figure 10.

Initially, a customer's requirements for an application are re-expressed as a requirements specification with four main components: a system model, an environment model, an interaction model, and a set of non-functional requirements. (As is the case for some of the approaches described above—with their focus on composite systems—one can also see here a concern to model both a system and its environment.)

In developing a design specification, system model components are mapped to one or more corresponding design components. At the same time, non-

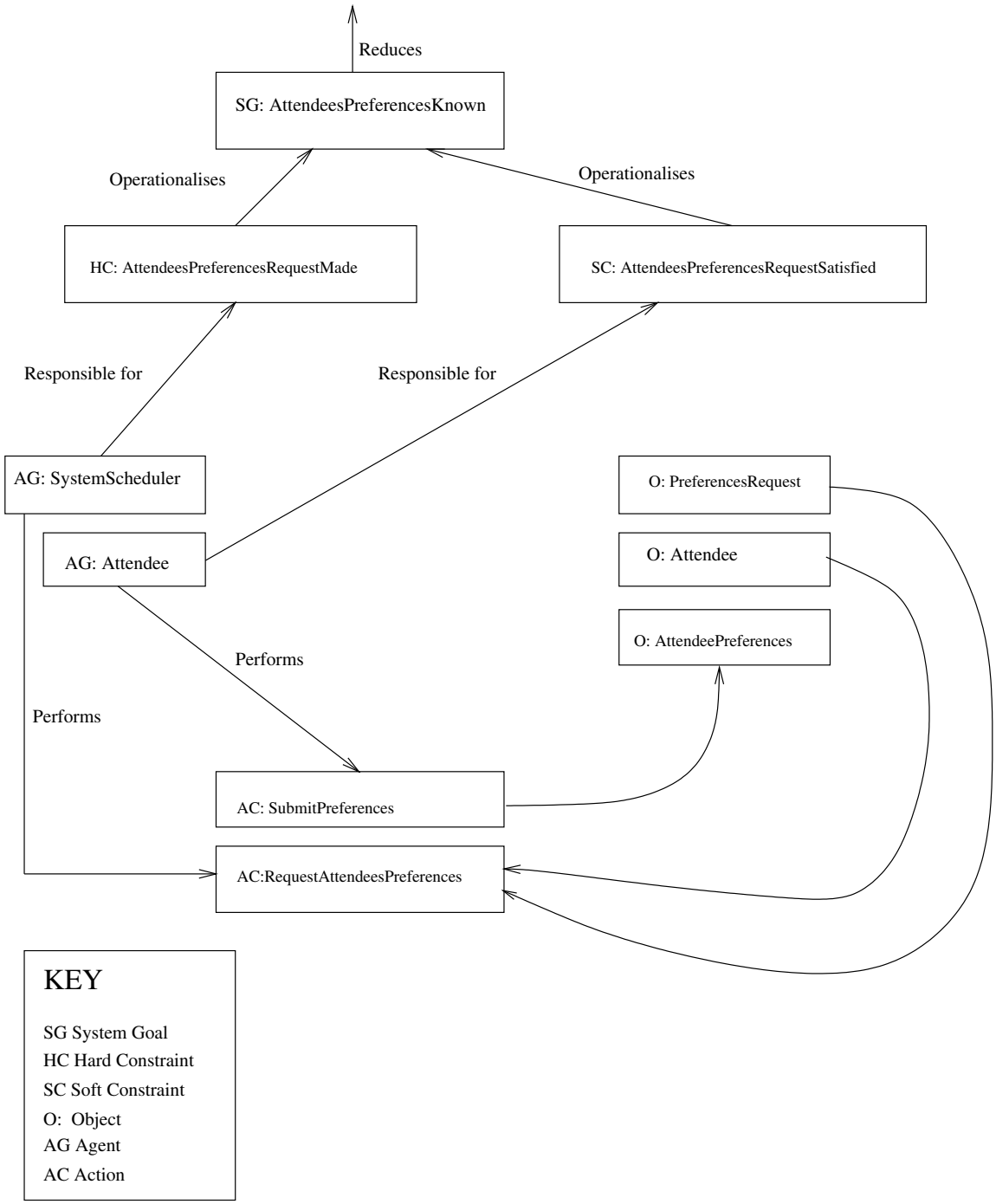

Figure 9: Fragment of a requirements model for a meeting scheduler system

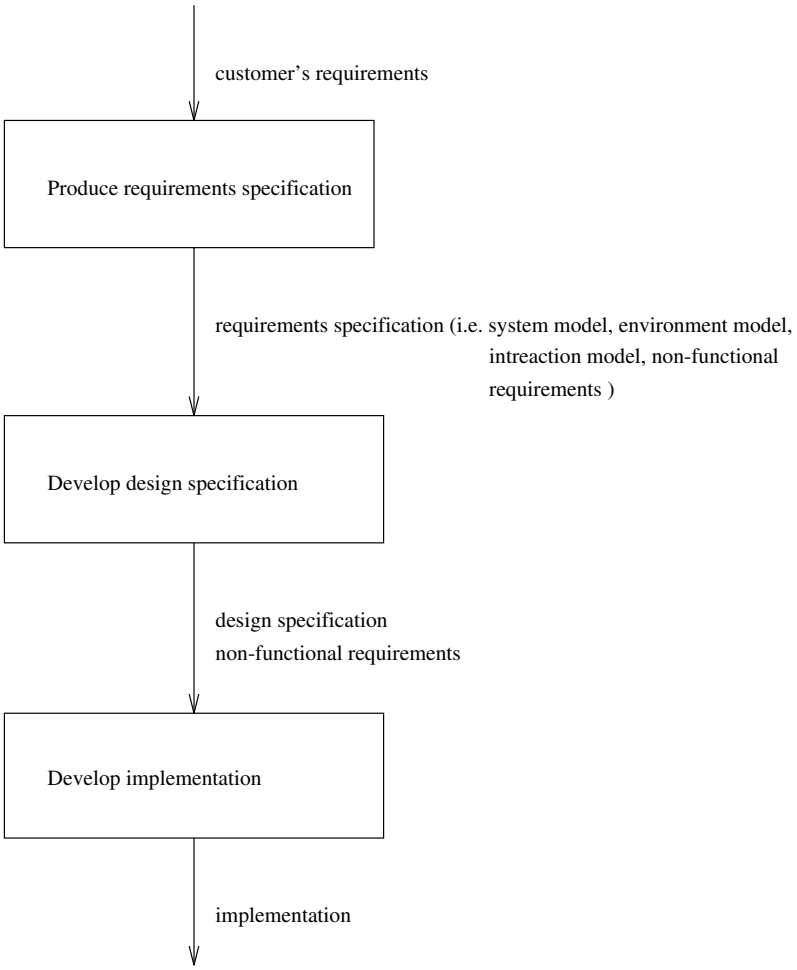

Figure 10: DAIDA system development process

functional requirements are decomposed to alternative (possibly conflicting) design decisions. These alternative designs may be evaluated for their effect on the non-functional requirements, and then the evaluations may be used to help to make decisions about which candidate design components will make up a design specification.

An implementation is produced in a similar manner [Nix94].

The purpose of the approach is to produce information systems whose global properties—performance, cost, accuracy of managed information, etc.—match customers' stated non-functional requirements more closely than is normal today.

The input to the Toronto approach is a set of non-functional requirements for a required information system.

The objective of the process is to generate knowledge which should facilitate the selection of those design components which will best satisfy the functional and non-functional requirements of a required application.

In outline the Toronto process may be described as follows. Each non-functional requirement is represented as a goal. Each such goal is decomposed to a point where sub-goals may be satisficed—satisfied within agreed limits—by one or more design decisions; these may themselves be refined further. The results of this process are represented in a number of goal structure diagrams (one for each non-functional requirement) similar to the one shown in figure 11. The diagrams should also record any conflict between design decisions which has been identified.

In more detail, the Toronto approach may be described with respect to five main components: goals, links, methods, correlation rules, and a labelling procedure.

Goals are used to represent non-functional requirements, design decisions, and arguments; the latter support both goals and links between goals.

Different kinds of link are used to denote the different ways in which goals may be related both to goals, and to links. In "and" links a parent goal is satisficed by satisficing all its child goals. Other types of link include "or" links, where a parent goal is satisficed by satisficing at least one child goal ; "sub" links, where a designer believes that there is strong evidence that satisficing a child goal will contribute to the satisficing of its parent goal; and other link types.

All the details of a particular decomposition of a parent goal into child goals may be separately worked out by a designer, or they may be achieved simultaneously through the application of an existing goal decomposition method to a parent goal. A goal decomposition method is like a rule or a template. Goal satisficing methods is another important class of methods. Satisficing goals are goals which represent design decisions. Satisficing methods may be applied to a parent goal to refine it into a satisficing goal and an appropriate link. Argument methods are a third class of method. These may be applied to goals or to links in order to generate argument goals; these support goals or links, either formally or informally.

The decomposition of one non-functional requirement may lead ultimately

to satisficing goals, i.e. design decisions, which conflict with satisficing goals derived in the goal structures of other non-functional requirements. Such conflict might be recognised either directly by the designer, or through the application of correlation rules, the fourth main component of the framework.

Finally, Chung has created a procedure, the labelling procedure, the fifth main component of his approach, which allows a particular goal structure to be evaluated and the extent of its contribution to the satisficing of a non-functional requirement to be determined. Using this procedure, alternative designs may be evaluated and the results may be used to choose design components for a design specification.

The output from the Toronto approach is a set of goal decomposition diagrams with one diagram for each non-functional requirement. The goal decomposition diagrams comprise goals, sub-goals, and design decisions. Conflict between design decisions in one diagram and goals in another is also shown.

2.4.1 Applying the approach to the Meeting Scheduler problem:

Before applying Chung's approach to a Meeting Scheduler non-functional requirement, I thought that it would be useful to outline a requirements specification for the Meeting Scheduler problem in the spirit of DAIDA. This is shown below:

Environment model:

1. Conveners have ideas for particular meetings.
2. Invited participants read meeting invitations and consider their response.

Interaction model:

1. Conveners request meetings
2. Scheduler acknowledges meeting requests
3. Scheduler invites intended participants
4. Scheduler prompts participants for replies
5. Participants respond to requests with date/time preferences

System model:

1. System manages interaction with conveners and participants
2. System schedules meetings rationally.

Non-functional requirements:

1. Minimize interaction overhead.
2. Minimize elapsed time between the time a meeting is requested and the time the meeting is scheduled.

3. Enforce privacy for participants.

4. Ensure system is usable by non-experts.

I chose to decompose and refine the second non-functional requirement in order to influence the selection of implementation components for a Meeting Scheduler system. A partial goal structure for this example is shown in figure 11.

The second non-functional requirement is represented by a non-functional requirement goal as follows:

```
MT[Schedule_requested_meeting(Meeting, Participants)]
```

Here MT stands for good minimisation of elapsed time.

I decomposed this non-functional requirement goal with an Implementation Components Method [Nix94] which I assumed to exist:

```
MT[Schedule_requested_meeting(Meeting, Participants)] <--and--

{MT[Input_meeting_request(Meeting, Participants)],
 MT[Schedule_meeting(Meeting, Participants)]}
```

I decomposed the

```
 MT[Schedule_meeting(Meeting, Participants)]
```

non-functional requirement goal in the same way:

```
MT[Schedule_meeting(Meeting, Participants)]} <--and--

{MT[Obtain_participants_preferences(Meeting,Participants)],
 MT[Compute_meeting_date_time(Meeting,Participants)]}
```

I decomposed the

```
MT[Obtain_participants_preferences(Meeting,Participants)]
```

non-functional requirement goal with an "and" link: (Although there might be a standard method for this type of decomposition in the context of meeting systems, here, I determined what child goals were required to satisfice the parent.)

```
MT[Obtain_participants_preferences(Meeting,Participants)] <--and--

{MT[Request_preferences(Meeting,Participants)],
 MT[Request_in_communication_channel],
 MT[Receive_valid_preferences(Meeting,Participants)]}
```

I decomposed the

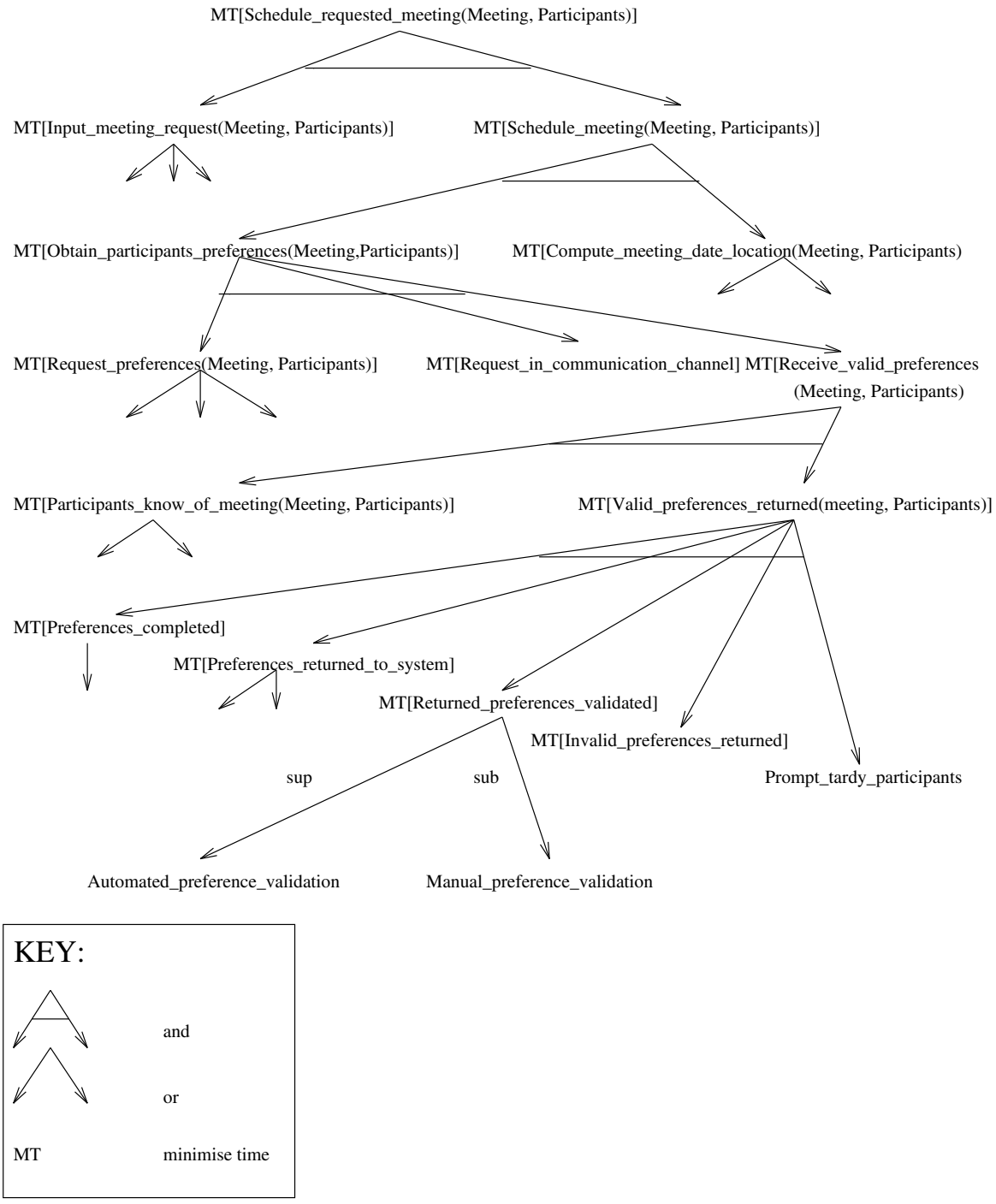

Figure 11: Refining the nfr "minimise elapsed time to schedule a requested meeting"

```
MT[Receive_valid_preferences]
```

non-functional requirement goal using my own knowledge:

```
MT[Receive_valid_preferences(Meeting,Participants)] <--and--
                    {MT[Participants_know_of_meeting(Meeting,Participants)],
                     MT[Valid_preferences_returned(Meeting,Participants)] }
```

The

```
MT[Valid_preferences_returned(Meeting,Participants)]
```

non-functional requirement goal was then decomposed:

```
MT[Valid_preferences_returned(Meeting,Participants)] <--and--

{MT[Preferences_completed],
 MT[Preferences_returned_to_system],
 MT[Returned_preferences_validated],
 MT[Invalid_preferences_returned_to_participants],
 Prompt_tardy_participants(Meeting,Participants)}
```

The last goal in this "and" set is a satisficing goal, i.e. an implementation decision, indicating that the system will prompt participants who have not responded with their preferences within a predefined period.

Finally the

```
MT[Returned_preferences_validated]
```

non-functional requirement goal was decomposed into two "or" satisficing goals:

```
MT[Returned_preferences_validated] <--or--

{Automated_preference_validation],
 Manual_preference_validation]}
```

These are both satisficing goals, i.e. implementation decisions; the first indicates that validation will be carried out automatically by a meeting scheduler system, the second that a meeting scheduler system will interact with a human who will validate participants' responses. Either satisfices the parent goal.

Conflict between satisficing goals can be seen if the reader considers the decomposition of the fourth non-functional requirement viz "Ensure the system is usable by non-experts". At some point in the decomposition it is likely that a satisficing goal will be posited which will conflict with the Manual_preference_validation satisficing goal already established, since a system in which validation has to be performed manually is relatively difficult to use. Such a conflict link might be detected by the non-functional requirement computer-based assistant.

2.4.2 Tool support:

A prototype tool has been developed which supports the management of the five main components of the approach viz goals, links, methods, correlation rules, and labels [Chu93].

2.5 GMARC:

In the GMARC (Generic Modelling Approach to Requirements Capture) approach [BJT+94], domain knowledge plays a central role.

The purpose of the GMARC approach is to make it easier than current practice allows for a non-technical client to produce formal specifications for a variety of applications within a domain of interest. Figure 12 below shows how GMARC fits into a software development process model, and also indicates its main inputs and outputs. It is intended that domain analysis be performed once for each new domain and that its result, a domain model, be re-used many times to create a variety of specifications for applications in the domain.

The GMARC approach has two main inputs: a client's requirements for a particular application and a domain model for the application area.

A domain model depicts the goals of a domain from high-level goals to low-level goals. It also shows which goals support each other, which undermine each other, which specialise each other, and the order in which goals must be considered for the inclusion of any associated formal specification fragments into a formal specification.

The BSG (Basic System Goal) is intended to model the most basic functionality goal of a domain. Formal specification fragments modelling the goal are stored with it.

Goals which specialise a BSG (directly or indirectly) are intended to model the variety of ways in which the functionality specified by the BSG can be augmented.

An example domain model is shown in figure 13.

The other main input to GMARC is a client's requirements.

The objective of the GMARC process is to allow a non-technical client to create a formal specification corresponding to an application need.

In order to create a specification, it is intended that a client and analyst would first examine the BSG, and then examine the other goals in a domain model. They do this in order to validate that the right domain model is being used, to see which domain goals will be supported and which undermined by their requirements, to familiarise themselves with the domain, to remind themselves of domain goals the may have forgotten, and to learn about new domain goals.

Following such an examination, it is intended that they would next navigate the domain model, selecting goals (corresponding to their requirements) which they wanted their intended application to meet. Formal specification fragments associated with selected goals are added to an emerging specification for the application.

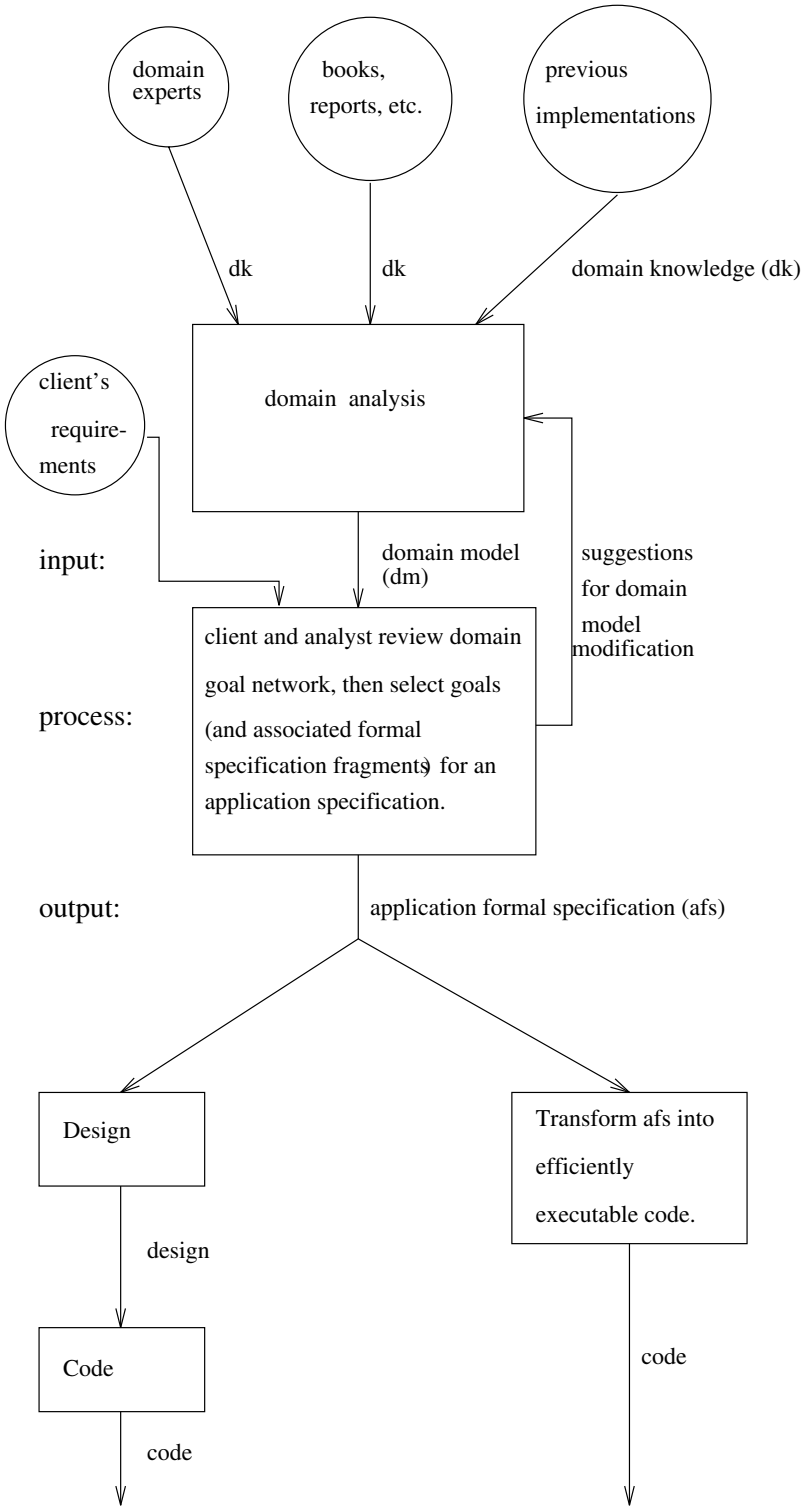

Figure 12: The role of GMARC in the software development process

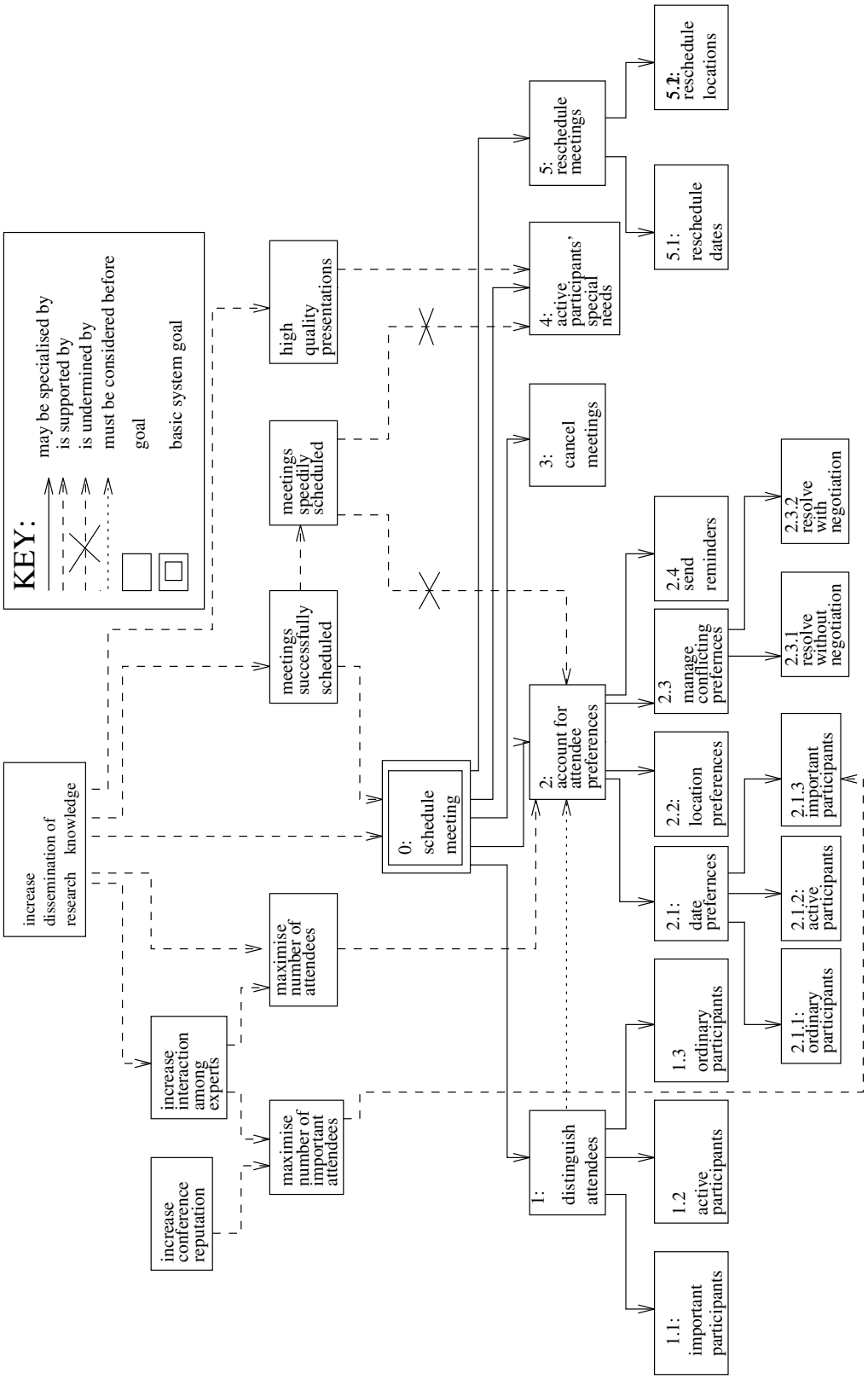

Figure 13: GMARC domain model for a conference scheduler

In general, it may be the case that selection of certain goals will inhibit selection of some remaining goals, and force the consideration of others.

Finally, if a client has some goals in mind for an application which are not covered by the corresponding domain model, it is intended that the client would be able to modify the application specification accordingly. It is expected that in such circumstances consideration would be given to augmenting the domain model to account for the missing goals.

The output of the GMARC process is a formal specification for a client's application, and possibly some suggestions for modifying a domain model.

2.5.1 Applying the approach to the Meeting Scheduler problem:

Figure 13 below depicts a partial GMARC domain model for a particular kind of meeting scheduler, an academic-conference scheduler. This model depicts high-level goals such as "disseminate research knowledge", and low-level goals such as "consider the equipment needs of active participants".

The Basic System Goal (BSG) is to determine dates and locations for requested conferences, prevent conferences being scheduled at the same location at the same time, announce conferences to intended participants, and call for conference papers. Formal specification fragments modelling the goal are stored with it.

Here, one way of augmenting the BSG might be to take the preferences of intended participants for dates and/or locations of meetings into account. Consequently the domain model contains a specialising goal which represents this. This goal is associated with fragments of formal specification which are intended to realise the goal by modelling the acquisition of participants' preferences and the generation of an appropriate conference date and location based upon them. In addition, the goal is linked negatively to the goal "meeting speedily scheduled" (since taking participants' preferences into account will increase the time taken to arrive at a suitable date and location), and positively to the goal "maximise number of meeting participants" (since taking participants' preferences into account is likely to result in a date and location which is convenient for more participants).

The other main input to GMARC is a client's requirements: here, it would be a client's requirements for a particular conference scheduling application.

In order to create the specification the client and analyst would first examine the BSG, and then examine the other goals in the domain model. Following this examination, they would navigate the domain model, selecting goals (corresponding to their requirements) which they want their intended application to meet. For example, this client might select goals from the conference scheduler domain model to create an application specification which allows conferences to be rescheduled and cancelled, but which doesn't take into account participants' date and location preferences. Another client might make a selection of goals which, in addition to those of the former client, includes a goal related to eliciting the preferences of the intended participants, and a goal related to issuing reminders to participants to send in their preferences.

Finally, if this client has some goals in mind for the application which are not covered by the domain model, he or she would be able to modify the application specification to take account of them.

2.5.2 Tool support:

A tool has been developed to support the GMARC approach [BJT+92]. The tool is built on top of the Xerox Parc NoteCards hypertext product.

3 Critique of reviewed approaches:

In the preceding section a number of goal-driven approaches to requirements engineering are presented. In this section I first present the main contribution of each approach. This is followed by a discussion of the similarities and differences between the approaches. The section ends with a review of some of the problems encountered with each approach.

3.1 Contribution:

The ISI approach [Fea87] first proposed and began to explore the idea that specifications for individual agents in multi-agent systems might be derived by decomposing the global goals of such systems.

Developing this idea, the Critter approach [FH92] introduced a new strategy for creating formal specifications for composite systems: it identifies behaviours that violate constraints, and then applies design operators to overcome these deficiencies.

Critter also refined some aspects of the ISI approach, e.g. by introducing formal expression of goals and system behaviour; extended some aspects, e.g. by introducing formal operators, which weaken goals and introduce new agents, etc.; and modified others, e.g by permitting both multiple agents to satisfy a goal, and global goals to be altered in the decomposition process. However, perhaps its most important contribution was to propose and explore the idea that a variety of configurations of agents and constraints might be properly derived from the same global goals.

The KAOS approach [DvF93] extended the two previous approaches by taking the starting point for goal decomposition not as the global goals of a composite system, but as the higher-level goals of the domain—organisation, super-system, etc.—in which the composite system is embedded. In addition, specifications (or requirements models, in KAOS terminology) are created by systematically instantiating an elaborate meta-model (ERA schema) which characterises composite systems.

At Toronto University, Lawrence Chung has created a goal-decomposition approach [Chu91] which can be applied to different types of non-functional requirements in order both to suggest new formal specification components and to measure the contribution of alternative candidate formal specification components towards meeting stated goals.

Finally, the main contribution of the GMARC approach [BJT+94] has been to introduce the idea of reusing existing goal decompositions for creating formal specifications for new applications in related areas.

3.2 Similarities and differences:

It should be clear that each of the five approaches may be characterised as goal-driven. However, while the GMARC approach requires users to select goals from existing goal-decompositions, each of the other four approaches may involve analysts in decomposing goals, either directly or indirectly: in the KAOS, ISI, and Toronto approaches it is goals that are decomposed, while in the Critter approach it is design states, which include goals, that are decomposed.

The KAOS, ISI, and Critter approaches are explicitly directed towards creating specifications for composite systems. Because of this the notion of agents plays an important role in each approach. The Toronto approach, however, is explicitly directed towards facilitating the creation of information systems, and has no notion of agents. The GMARC approach also has no notion of agents.

The Toronto approach is the only one to explicitly treat only non-functional requirements. Both KAOS and GMARC address functional and non-functional requirements. (These two are also the only ones to explicitly address higher level domain goals as well as application specific goals, and the only ones to be concerned with the elicitation of requirements.) Critter and the ISI approach, while noting the importance of non-functional requirements, both address just the functional requirements of composite systems.

Within the goal "decomposition" group two other important properties are shared by the KAOS, Critter, and Toronto approaches but not by the ISI approach:

First, while the decomposition structure of all four approaches is formal, the representations of goals and and behaviour is also formal for the former group; but in the ISI approach, goals are expressed informally in English. This feature has allowed the former group to develop knowledge-based tools to support their decomposition processes. It has also allowed other kinds of tools to be developed, e.g. the analysis tools in Critter. Within GMARC goals are described informally in English, while the behaviour to achieve them is described formally in Z [Spi92]. The GMARC formal decomposition structure has allowed a prototype tool supporting its use to be developed.

Second, the ISI, KAOS, and Critter approaches each support the notion of alternative decompositions to behaviour which can satisfy the same basic goals. This feature is also shared by the GMARC approach but is not present in the ISI approach.

Finally, there is very little obvious overlap between the kinds of relationships between goals adopted by each approach:

The KAOS, Toronto, and GMARC approaches all include the "goal conflicts with goal" relationship type. And the KAOS and Toronto approaches both include "and reduction" and "or reduction" relationships between goals too.

The GMARC "goal supports goal" relationship type seems similar to the "sup" and"sub" relationship types of the Toronto approach.

All the other relationships between goals are specific to an approach. For example the "goal x is an implication of goal y" relationship type is specific to the ISI approach.

3.3 Problems:

Although I appreciate that the ISI approach [Fea87] is concerned with presenting an original idea rather than a fully elaborated approach, nevertheless, it is felt that any approach which were to be developed from the idea would have to solve a number of problems. First, decomposing constraints (by selecting suitable implications and then simplifying) is difficult. To make this a more tractable task, heuristics are required which could help an analyst both to recognise implications in a constraint and to judge between implications with regard to their suitability for decomposition. Second, I feel that when constraints are decomposed, it must be possible to demonstrate that behaviour is correctly preserved, i.e. that the sub-constraints are equivalent to the original constraints.

Critter is a more detailed approach with a set of supporting tools. However, I have identified one possible problem with the process. After a first constraint has been developed and an associated system description perhaps modified, the second and subsequent constraints are developed. However, in general, as each additional constraint is developed and the system description modified, it may be necessary to ensure that some constraints developed earlier are still satisfied by the current system description. These additional checks are likely to make the process time-consuming to carry out. In addition, in some cases it seems possible that the process of checking constraints, modifying system descriptions, rechecking constraints, and so on, might not terminate.

Moving on to KAOS, the major problem perceived with this approach is that a number of its steps are difficult to perform. For example, it was found to be difficult to generate a goal structure (step one) for the Message Scheduling System. This is because it is difficult to identify sub-goals of a goal, and to know when all the required sub-goals have been identified.

Step five is also difficult to perform. The acquisition of actions and objects for a given system is completed at step four. However, in general there is no guarantee that these will meet the constraints. To alleviate this, in step five each constraint must be matched against the available actions to see whether the state transitions defined by the actions match the constraint. The match may reveal that a pre-condition, post-condition, or invariant associated with an action must be strengthened in order to ensure that a constraint is met. To help an analyst perform such matching, some inference rules for deriving strengthened pre-conditions, post-conditions, and invariants have been identified. However, it is felt that, in general, it will prove difficult for an analyst manually to select a correct rule and then to apply it accurately.

Turning now to GMARC, domain models have only been created for some simple domains, e.g. rail ticket system. It may prove more difficult to construct

domain models for more complex domains. In particular it may prove more difficult or even impossible to organise the specialising sub-goals so that they are independent of one another. In addition, in the work to date on GMARC, the question of which specification language is used to express goals has not been considered an issue. In the examples to date Z has been used. However, it is now seen that when such a language is used the set of behaviours, i.e. sequences of allowable states, is only implicit in the specification. It is now clear that it would be preferable to use a temporal logic to express some goals (as sequences of possible state patterns) at one level, and language like Z to express lower level goals through describing operations which allowed the state pattern sequence to be achieved.

4 Synthesis:

Previous sections describe five different goal-oriented approaches to requirements engineering and illustrate them with the meeting scheduler problem. In this section I have attempted to synthesise an approach from the five considered by selecting what I feel are the best features of each approach and organising them to provide a new approach. This synthesised approach is shown in outline below:

Either

1. Select specification components from domain model

or

1. Obtain goals
2. Decompose goals to specification components
3. Select "best" specification component alternatives
4. Create or modify domain model

This approach is now described in greater detail.

4.1 Select specification components from domain model

The previous section on GMARC [BJT+94] described a goal-oriented domain model containing high and low-level domain goals, as well as specification components denoting behaviour required to satisfy low-level goals. If such a model existed for a particular domain, e.g. for the domain of meeting scheduling, it would seem sensible for a client who wanted to build a particular meeting scheduler to use the domain model in the manner described previously. Thus they might review the high and low-level goals in the domain model, and select those goals and associated specification components that matched their goals for the particular application.

4.2 Obtain goals

On the other hand, if such a model is not available, then the alternative path, shown above, might be followed. This path begins with "obtaining goals". Here an attempt is made to capture the functional and non-functional goals of a required application. In addition, the rationale for each goal should also be captured. By rationale I mean the reason given by the goal-owner for the goal. In general I would expect the reason for a goal to be given in terms of the way it contributes to or inhibits the achievement of other domain goals.

Clearly there will often be conflicting goals stated by different goal-owners, and even by the same goal owners. In such cases, the conflict should be resolved before moving on to the next step.

In general, all the information obtained at this stage might be considered when creating or modifying a domain model during the last stage.

4.3 Decompose goals to specification components

At the beginning of this stage a requirements engineer should classify the goals obtained during the previous stage as either functional or non-functional goals. Then they should process the two sets of goals separately (either sequentially or in parallel).

To process the functional goals, the requirements engineer must first identify the highest level, or global, goals of the required application when it is viewed as a composite system. The goal decomposition approach of, for example, KAOS [DvF93] is now applied: the high-level goals are decomposed until sub-goals are identified whose satisfaction may be made the responsibility of a single agent; such agents are identified during the decomposition along with actions and objects associated with satisfying goals (as we saw in the KAOS section).

At the end of this stage, a set of leaf-goals equivalent to the set of global goals will have been identified. And one or more sets of agent, actions and objects will be associated with each leaf-goal. It is understood that a such an association indicates that an agent may satisfy the leaf-goal by acting on the objects.

During this stage the non-functional requirements are also treated as goals. Each is decomposed using the "methods" of the Toronto approach [Chu91] until all the leaf-goals may be satisfied by by one or more design decisions. As a proposed modification to the Toronto approach I feel that it would be desirable if the design decisions could be stated in terms agents, actions and objects as they are in KAOS.

Such design decisions may correspond to functionality that has also been derived through decomposing the functional requirements, or they may identify the need for new functions whose role is solely to help satisfy particular non-functional requirements.

At this point, the various decomposition hierarchies should be inspected either manually, or automatically with a tool based perhaps on the Toronto approach's correlation rules, and potential conflict between design decisions should

be recorded.

Before proceeding to the next stage, the impact of functional requirement design decisions on non-functional requirements should be assessed and recorded, and the impact of non-functional requirement design decisions on functional requirements should be assessed and recorded.

4.4 Select "best" specification component alternatives

When this stage is reached functional and non-functional goal decomposition diagrams are available which explicitly show various ways in which all the goals (of both types) for an application may be met by candidate design decisions. And which also show the impact (either positive or negative) that these candidate design decisions have on all goals. These diagrams should now be inspected and the optimum design decision set identified. This is expected to be the one that meets all the functional requirements and as many non-functional requirements as possible. It seems likely that this task might be automated.

4.5 Create or modify domain model

In the final stage, the knowledge about an application domain in terms of goals, constraints, agents, and actions acquired during the earlier stages should be used to create or modify a GMARC style domain model. Such a domain model would then be either available for re-use if just newly created, or possibly better suited for re-use, if just modified, when a new application in the domain was required.

5 Summary and future work:

In this paper five goal-oriented approaches to requirements engineering have been characterised and four of them have been applied to a common example: "the meeting scheduler problem". In addition, the main contribution of each approach, and the similarities and differences among the approaches have been identified.

The outline of a new approach has also been sketched. This is based upon the "best" features of the five presented approaches. It is intended that this new approach be further elaborated in future work.

References

[BGW82] R. Balzer, N. Goldman, and D. Wile. Operational specifications as the basis for rapid prototyping. *Software Engineering Notes*, 7(5), 1982.

[BJT+92] D. Bolton, S. Jones, D. Till, D. Furber, and S. Green. A framework for knowledge-based support of requirements elicitation. Technical

Report TCU/CS/1992/18, City University, 1992. GMARC Project Report R42.

[BJT+94] D. Bolton, S. Jones, D. Till, D. Furber, and S. Green. Using domain knowledge in requirements capture and formal specification construction. In M. Jirotka and J. Goguen, editors, *Requirements Engineering: Social and Technical Issues*. Academic Press, 1994.

[Chu91] L. Chung. Representation and utilisation of non-functional requirements for information system design. In *Proceedings of CAiSE*, pages 3–50, 1991.

[Chu93] L. Chung. *Using Non-Functional Requirements: A process Oriented Approach*. PhD thesis, Department of Computer Science, University of Toronto, 1993.

[CPM+91] L. Chung, K. Panagiotis, M. Manolis, M. Mertikas, J. Mylopoulos, and Y. Vassiliou. From information system requirements to designs: A mapping framework. *Information Systems*, 16(4):429–461, 1991.

[Dar94] R. Darimont. Tool support. Private communication to the author, 1994.

[DH89] E. Dubois and J. Hagelstein. A logic of action for goal-oriented elaboration of requirements. In *Proceedings 5th International Workshop on Software Specification and Design*, volume 14, pages 160–168, 1989.

[Dub91] E. Dubois. A formal language for the requirements engineering of computer systems. In A. Thayse, editor, *Introducing a Logic Based Approach to AI*, volume 3, pages 357–433. Wiley, 1991.

[DvF93] A. Dardenne, A. vanLamsweerde, and S. Fickas. Goal-directed requirements acquisition. *Science of Computer Programming*, 20:3–50, 1993.

[Fea87] M. S. Feather. Language support for the specification and development of composite systems. *ACM Transactions on Programming Languages and Systems*, 9(2):198–234, April 1987.

[FH92] S. Fickas and R. Helm. Knowledge representation and reasoning in the design of composite systems. *IEEE Transactions on Software Engineering*, 18(6):470–482, April 1992.

[Hel93] R. Helm. Critter operators and tools. Private communication to the author, 1993.

[JMSV92] M. Jarke, J. Mylopoulos, J. W. Schmidt, and Y. Vassiliou:92. Daida: An environment for evolving information systems. *ACM Transactions on Information Systems*, 10(1), 1992.

[Nil71] N. J. Nilsson. *Problem Solving Methods in Artificial Intelligence.* McGraw-Hill Computer Science, 1971.

[Nix94] B. Nixon. Representing and using performance requirements during the development of information systems. In *Proceedings 4th International Conference on Extending Database Technology*, 1994.

[Spi92] J. M. Spivey. *The Z Notation: A Reference Manual.* Prentice Hall International Series in Computer Science, 2nd edition, 1992.

[vDM92] A. vanLamsweerde, R. Darimont, and P. Massonet. The meeting scheduler problem: Preliminary definition. Technical report, Université Catholique de Louvain, Belgium, 1992.

[WH85] M. C. Wilbur-Ham. Numerical petri nets—a guide: Report 7791. Technical report, Telecom Australia Research Laboratories, 1985.

A The Meeting Scheduler Problem: Preliminary Definition

A.1 Problem Statement

The purpose of a *meeting scheduler* is to support the organization of meetings - that is, to determine, for each meeting request, a meeting *date* and *location* so that most of the intended participants will effectively participate. The meeting date and location should thus be as convenient as possible to all participants. Information about the meeting should also be made available as early as possible to all potential participants. The intended system should considerably reduce the amount of overhead usually incurred in organizing meetings where potential attendees are distributed over many different places.

Meetings are typically arranged in the following way. A *meeting initiator* asks all potential meeting attendees for the following information based on their personal agenda:

- a set of dates on which they cannot attend the meeting (hereafter referred as *exclusion set*);
- a set of dates on which they would prefer the meeting to take place (hereafter referred as *preference set*).

A *meeting date* is defined by a pair (calendar date, time period). The exclusion and preference sets are contained in some time interval prescribed by the meeting initiator (hereafter referred as *date range*).

The initiator also asks *active* participants to provide any special equipment requirements on the meeting location (e.g., overhead-projector, workstation, network connection, telephones, etc.); he/she may also ask *important* participants to state preferences about the meeting location.

The proposed meeting date should belong to the stated date range and to none of the exclusion sets; furthermore it should ideally belong to as many preference sets as possible. A *date conflict* occurs when no such date can be found. A conflict is strong when no date can be found within the date range and outside all exclusion sets; it is weak when dates can be found within the date range and outside all exclusion sets, but no date can be found at the intersection of all preference sets. Conflicts can be resolved in several ways:

- the initiator extends the date range;
- some participants remove some dates from their exclusion set;
- some participants withdraw from the meeting;
- some participants add some new dates to their preference set.

A meeting room must be available at the selected meeting date. It should meet the equipment requirements; furthermore it should ideally belong to one

of the locations preferred by as many important participants as possible. A new round of negotiation may be required when no such room can be found.

The meeting initiator can be one of the participants or some representative (e.g., a secretary).

The system should assist users in the following activities.

- Plan meetings under the constraints expressed by participants (see above).
- Replan a meeting dynamically to support as much flexibility as possible. On one hand, participants should be allowed to modify their exlusion set, preference set and/or preferred location *before* a meeting date/location is proposed. On the other hand, it should be possible to take external constraints into account *after* a date and location have been proposed - e.g., due to the need to accommodate a more important meeting. The original meeting date or location may then need to be changed; sometimes the meeting may even be cancelled.
- Support conflict resolution according to resolution policies stated by the client.
- Manage all interactions among participants required during the organization of the meeting - to communicate requests, to get replies even from participants not reacting promptly, to support the negotiation and conflict resolution processes, to make participants aware of what's going on during the planning process, to keep participants informed about schedules and their changes, to make them confident about the reliability of the communications, etc.

The meeting scheduler must in general handle several meeting requests *in parallel*. Meeting requests can be competing by overlapping in time or space. Concurrency must thus be managed.

The following aspects should also be taken into account.

- The system should accomodate decentralized requests; any authorized user should be able to request a meeting independently of his whereabouts.
- Physical constraints may not be broken - e.g., a person may only attend one meeting at a time, a meeting room may not be allocated to more than one meeting at the same time, etc.
- The system should provide an appropriate level of performance, for example:
  - the elapsed time between the submission of a meeting request and the determination of the corresponding meeting date/location should be as small as possible;
  - the elapsed time between the determination of a meeting date/location and the communication of this information to all participants concerned should be as small as possible;

  - a minimal delay should exist between the determination of a meeting date and the actual date of the meeting.

- Privacy rules should be enforced; an ordinary participant should not be aware of constraints stated by other participants.
- The system should be usable by non-experts.
- The system should be customizable to professional as well as private meetings. These two modes of use are characterized by different restrictions on the time periods that may be allocated (e.g., meetings during office hours, private activities during leisure time).
- The system should be flexible enough to accommodate evolving data - e.g., the sets of concerned participants may be varying, the address at which a participant can be reached may be varying, etc.
- The system should be easily extendable to accommodate the following typical variations:
  - introduction of explicit status and priorities among participants;
  - introduction of explicit priorities among dates in preference sets;
  - introduction of explicit dependencies between meeting date and meeting location;
  - participation through delegation - a participant may ask another person to represent him/her at the meeting;
  - partial attendance - a participant can only attend part of the meeting;
  - variations in date formats, address formats, interface language, etc.
  - partial reuse in other contexts - e.g., to help establish course schedules.