Modeling Car Crash Management with KAOS

Antoine Cailliau, Christophe Damas, Bernard Lambeau, and Axel van Lamsweerde
Université catholique de Louvain (UCL)
Louvain-La-Neuve, Belgium
{antoine.cailliau, christophe.damas, bernard.lambeau, axel.vanlamsweerde}@uclouvain.be

***Abstract*—Getting the right software requirements under the right environment assumptions is a critical precondition for developing the right software. KAOS is a goal-driven, model-based approach for elaborating a complete, adequate, consistent, and well-structured set of measurable software requirements and environment assumptions. The modeling language and method cover the intentional, structural, functional, and behavioral facets of the target system. Declarative and operational sub-models are integrated. Semi-formal and formal techniques complement each other for model construction, analysis and evolution. They support early and incremental reasoning on partial models for a variety of purposes including goal satisfaction arguments, property checks, animations, the evaluation of alternative options, the analysis of risks, threats and conflicts, and traceability management.**

The paper illustrates the modeling language and method on a car crash management case study. The overall produced model integrates the goal, object, agent, operation and behavior sub-models of the system. The paper outlines some of the features supported by KAOS for incremental model elaboration, including goal identification and refinement, the structuring of domain concepts, risk analysis for increased requirements completeness, goal operationalization, the derivation of agent interfaces and the derivation of state machine behavior models.

***Index Terms*— Goal-oriented requirements engineering, multi-view system modeling, model construction, model analysis.**

I. Introduction

Requirements engineering (RE) is concerned with the elicitation, evaluation, specification, consolidation, and evolution of the objectives, functionalities, qualities, and constraints a software-based system should meet within some organizational or physical setting [5]. The RE process is intrinsically complex.

- A wide spectrum of concerns need to be addressed, ranging from high-level, strategic objectives to detailed, technical requirements.
- Two systems are involved: the system as it is before software development and the system-to-be. The latter includes software and environment components such as people, devices or pre-existing software.
- The involved stakeholders may have diverse, partial and conflicting concerns.
- Risks must be anticipated in order to achieve requirements completeness and system robustness.
- Numerous alternative options must be evaluated for selection of preferred ones.

KAOS is a rigorous method for requirements engineering aimed at addressing those challenges [5]. The method may be briefly characterized as follows.

- Model-based and multi-view: the elaborated system specification is organized at various levels of abstraction and according to multiple system facets.
- Goal-oriented: the software requirements and environment assumptions are derived so as to meet the system's functional and non-functional objectives.
- Constructive: systematic steps and heuristic rules at each step are available to provide guidance in elaborating high-quality requirements.
- Incremental: early reasoning on partial models is supported.
- Wide-spectrum: declarative and operational models are integrated.
- Formal when and where needed for more sophisticated analysis.
- Lightweight for use in practical situations, with formal details kept hidden wherever possible.

In KAOS, the multiple system facets are captured through complementary models integrated through inter-model consistency rules [5].

- The goal model interrelates all functional and non-functional objectives of the system considered (as-is or to-be).
- The object model defines and interrelates all concepts involved in the goal specifications.
- The agent model specifies the system components, their interfaces and their responsibilities with respect to goals.
- The operation model specifies the functional services derived from the goals assigned to specific agents.
- The behavior model captures agent behaviors for goal satisfaction in terms of interaction scenarios and parallel state machines.

The paper shows KAOS in action on the bCMS case study (barbados Car crash Management System) [3]. The proposed system is intended to coordinate the communication between a fire station coordinator (FSC) and a police station coordinator (PSC) to handle crises in a timely manner. Only a few modeling steps are discussed here for lack of space. Further modeling increments resulting in a larger portion of the bCMS model are available in [2].

Section II shows the elaboration of a first-sketch model for an ideal system; a few high-level goals are gradually refined into subgoals assignable to individual system agents. During such refinement, key concepts are identified and structured in an object model. Alternative system options are then discussed and evaluated in the light of soft goals and agent responsibilities. Section III de-idealizes the model presented in Section II through risk identification, assessment and resolution. Section IV shows how operational software specifications, behavior models and agent interfaces are obtained systematically from the models previously built. Section V concludes the paper by briefly discussing the strengths and limitations of the overall model obtained.

II. Declarative Model for an Ideal System

The main bCMS goals are first identified, specified and related to each other within goal model fragments (Section II.A). While doing so, a first sketch for the object model is derived from goal specifications (Section II.B). The goal model is enriched by refining goals until fine-grained subgoals are obtained that can be assigned to software or environment agents, respectively (Section II.C). Throughout this process, scenarios can be used for goal elicitation or for illustration of alternative goal refinements (Section II.D).

A. Elaborating Goal Model Fragments

A goal is a prescriptive statement of intent whose satisfaction requires the cooperation of agents forming the system [5]. Agents are active system components playing some role in goal satisfaction; they include people, devices such as sensors and actuators, legacy software or software-to-be components. The system thus comprises software-to-be agents together with environment agents. Functional goals prescribe services to be delivered by the system; non-functional goals constrain how such services should be delivered (e.g., quality constraints). Behavioral goals prescribe intended system behaviors; soft goals prescribe preferences among alternative behaviors. Goals cover a wide range of concerns –from strategic, high-level ones to technical, fine-grained ones. Requirements are goals under the sole responsibility of the software-to-be; expectations (sometimes called assumptions) are goals under the responsibility of some environment agent.

A goal model is an AND/OR graph showing how goals contribute to each other. An AND-refinement requires all subgoals to be satisfied for the parent goal to be satisfied. An OR-refinement captures alternative system options; the parent goal is satisfied provided one of the alternative subgoals is satisfied. Heuristic rules and refinement patterns are available for goal identification and refinement [5]. For example, parent goals may be obtained bottom-up by answering WHY questions about subgoals; subgoals may be obtained top-down by answering HOW questions about parent goals.

Our heuristic rule for eliciting goals from intentional or prescriptive keywords in natural language text yields the following goal stated in [3, p.3]:

Goal Achieve [CrisisResolved When Reported]
Def *Every crisis detected and reported both at the fire station and the police station shall be resolved in a timely manner.*

The Achieve prefix introduces a specific kind of behavioral goal; it prescribes behaviors where a target condition must sooner or later hold (here, CrisisResolved) whenever some current condition holds (here, CrisisReported). Goal specifications in the **Def** annotation should be made precise. This calls for further elicitation and disambiguation with stakeholders –e.g., what do "in a timely manner" or "resolved" exactly mean?

Refinement patterns provide effective modeling guidance by encoding common tactics for goal decomposition and subgoal specification [5]. The explanation in [3, p.7] on how a crisis should be resolved suggests reusing the milestone-driven pattern. This pattern refines a behavioral goal by identifying milestone conditions for reaching the goal's target condition. For the goal Achieve [CrisisResolvedWhenReported] it produces the following refinement into three subgoals (see Fig. 1):

Goal Achieve [CrisisRequirementsKnown When CrisisReported]
Def *For every crisis reported, the number of fire trucks and police vehicles needed for handling the crisis shall eventually be known by both coordinators.*

Goal Achieve [RoutePlanAgreementReached When CrisisRequirementsKnown]
Def *When the numbers of required fire trucks and police vehicles are known, the PSC and FSC shall eventually agree on a route plan to be deployed for resolving the crisis. The latter specifies the vehicles to be allocated to the crisis. For each of them, the route plan describes the route to be followed from its current position to the crisis location.*

Goal Achieve [CrisisResolvedWhenRoutePlanAgreementReached]
Def *When a route plan has been agreed by both coordinators, the crisis shall eventually be resolved.*

This refinement introduces two milestone conditions: CrisisRequirementsKnown and RoutePlanAgreement Reached.

B. Deriving Objects from Goal Specifications

A first portion of the object model may be derived from the goal specifications available so far.

The object model provides a structural view of how the system concepts are interrelated. A conceptual object is a system-specific concept whose instances are distinctly identifiable, can be enumerated in any system state, share similar features, and differ from each other in their individual behavior. An object is structurally modeled as an entity, association, event or agent depending on whether it is an autonomous, subordinate, instantaneous, or active object, respectively. The object model is represented by an operation-

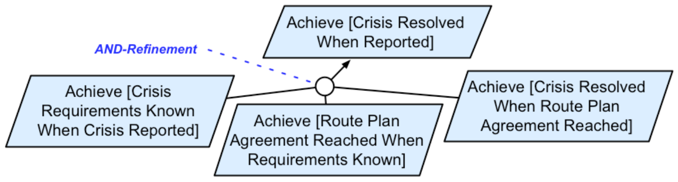
Achieve [Crisis Resolved When Reported]
AND-Refinement
Achieve [Crisis Requirements Known When Crisis Reported]
Achieve [Route Plan Agreement Reached When Requirements Known]
Achieve [Crisis Resolved When Route Plan Agreement Reached]

Fig. 1. Refining Achieve [CrisisResolved When Reported]

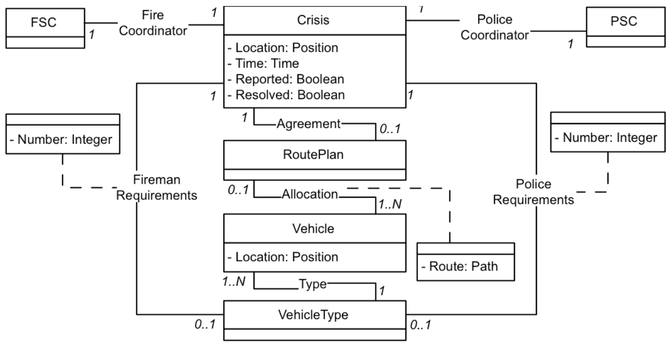

Fig. 2. First object model portion for the bCMS

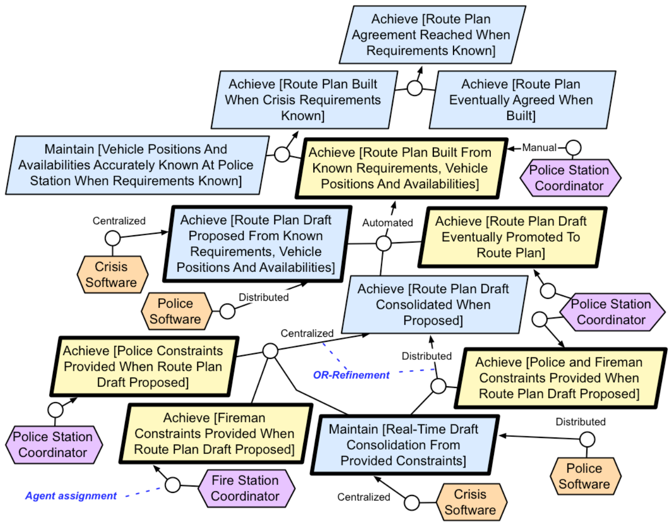
Achieve [Route Plan Agreement Reached When Requirements Known]
Achieve [Route Plan Built When Crisis Requirements Known]
Achieve [Route Plan Eventually Agreed When Built]
Maintain [Vehicle Positions And Availabilities Accurately Known At Police Station When Requirements Known]
Achieve [Route Plan Built From Known Requirements, Vehicle Positions And Availabilities]
Manual
Police Station Coordinator
Centralized
Crisis Software
Achieve [Route Plan Draft Proposed From Known Requirements, Vehicle Positions And Availabilities]
Automated
Achieve [Route Plan Draft Eventually Promoted To Route Plan]
Police Software
Distributed
Achieve [Route Plan Draft Consolidated When Proposed]
Police Station Coordinator
Centralized
Achieve [Police Constraints Provided When Route Plan Draft Proposed]
OR-Refinement
Distributed
Achieve [Police and Fireman Constraints Provided When Route Plan Draft Proposed]
Achieve [Fireman Constraints Provided When Route Plan Draft Proposed]
Maintain [Real-Time Draft Consolidation From Provided Constraints]
Distributed
Police Station Coordinator
Police Software
Agent assignment
Fire Station Coordinator
Centralized
Crisis Software

Fig. 3. Refining Achieve [RoutePlanAgreementReached When Requirements Known]

free, design-independent class diagram. It is gradually derived from goal specifications by use of heuristic rules [5], e.g., “identify associations from linking expressions in goal specs”.

Fig. 2 shows the object model obtained through such rules. Multiplicities may capture descriptive domain properties to be used when reasoning about the models –e.g., the multiplicities on Allocation express that a route plan requires one or more vehicles whereas a vehicle may be allocated to at most one route plan at a time. The early object model in Fig. 2 will be enriched with additional objects, attributes and associations as goal elicitation proceeds further.

C. Enriching the Goal Model

Goal refinement terminates when fine-grained subgoals can be assigned as requirements or expectations to software or environment agents, respectively. The process is guided by refinement heuristics, tactics, and patterns [5].

Fig. 3 shows the refinement of the goal Achieve [RoutePlan AgreementReached When RequirementsKnown] introduced in Section II.A. According to [3, p.7], a route plan must be built by the police coordinator and then agreed by the fire coordinator. Building a route plan requires the vehicle positions and availabilities to be known at the police station —a goal to be further refined in another diagram.

There are alternative options for meeting the goal Achieve [RoutePlanBuilt From Requirements&VehiclePositions&Availabilities], to be evaluated with respect to soft goals in the goal model.

1. *Manual System*. No software is used. As the goal assignment shows, the police coordinator is expected to build the route plan manually (e.g., through a physical map). This alternative corresponds to the system-as-is.
2. *Centralized System*. The police and fire station coordinators use a common centralized CrisisSoftware. The latter proposes a route plan that both coordinators may adapt by stating constraints on selected vehicles and routes. This option involves real-time collaboration between both coordinators and the software in order to build an accurate route plan with respect to actual vehicle positions and availabilities.
3. *Distributed System*. The police and fire station coordinators use their own software [3, p.3]. The police coordinator builds the route plan using the PoliceSoftware, in a way similar to the previous alternative. Here, however, the fire coordinator is not involved in route plan construction; she has to agree with the proposal.

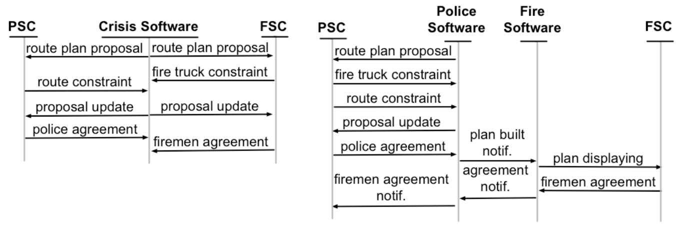

Fig. 4. Scenario for *Centralized* (left) and *Distributed* (right) *Systems*

D. Exploring Alternative Options with Scenarios

Scenarios capture typical examples of system behavior through sequences of interactions among its agents [5]. The MSC scenarios in Fig. 4 illustrate the centralized (left) and distributed (right) options for route plan construction.

The scenario illustrating the distributed option reveals the implicit assumption that firemen constraints, such as a fire truck being unavailable, are known by the PSC. Such hidden assumptions must be made explicit and carefully assessed, requiring further elicitation in case this option is selected.

E. Identifying Soft Goals for Option Selection

Alternative goal refinements or agent assignments capture different system designs. The selection of best alternatives is based on soft goals as the latter capture preferred behaviors.

Soft goals for the bCMS system can be identified from stakeholder objectives in [3, p.4-6], e.g., Minimize [TimeToGet ResourcesOnCrisisLocation], Maximize [TimeEstimatesAccuracy], Minimize [StressLevel], Minimize [SystemCost]. Such goals are more satisfied along some alternatives envisioned in Section II.C and less along others. As illustrated by the scenarios, favoring real-time collaboration among coordinators in the centralized option is likely to yield an agreement more quickly than in the distributed option. The centralized option would thus better meet the first soft goal –possibly at the expense of system cost.

Other sub-models may help assessing the satisfaction level of soft goals in a given alternative. For example, a responsibility diagram shows all responsibilities assigned to a given agent [5]. Comparing the responsibility diagrams of the police and fire coordinators in the alternative options helps assessing potential stress levels through load analysis; too many responsibilities may negatively impact on stress level.

III. Obstacle Analysis to Complete the Goal Model

Missing requirements are known to be among the major causes of software failures. Early goal models tend to be too ideal; the software and its environment are assumed to always behave as normally expected. Obstacle analysis is a goal-anchored form of risk analysis whereby exceptional conditions obstructing system goals are identified, assessed and resolved to produce

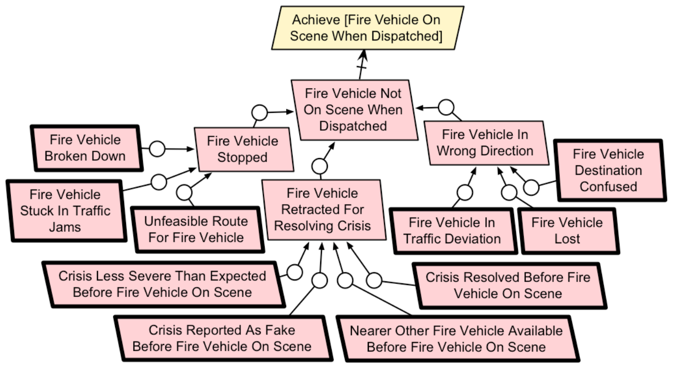
Achieve [Fire Vehicle On Scene When Dispatched]
Fire Vehicle Not On Scene When Dispatched
Fire Vehicle Broken Down
Fire Vehicle Stopped
Fire Vehicle In Wrong Direction
Fire Vehicle Destination Confused
Fire Vehicle Stuck In Traffic Jams
Unfeasible Route For Fire Vehicle
Fire Vehicle Retracted For Resolving Crisis
Fire Vehicle In Traffic Deviation
Fire Vehicle Lost
Crisis Less Severe Than Expected Before Fire Vehicle On Scene
Crisis Resolved Before Fire Vehicle On Scene
Crisis Reported As Fake Before Fire Vehicle On Scene
Nearer Other Fire Vehicle Available Before Fire Vehicle On Scene

Fig. 5. Obstacles for Achieve [FireVehicleOnScene When Dispatched]

more complete requirements [5]. An obstacle to a goal is a precondition for non-satisfaction of this goal. An obstacle tree is anchored on a leaf goal; it shows through AND/OR refinements how this goal can be violated. The root of the tree is the goal negation; the leaves are elementary obstruction conditions that are satisfiable by the environment.

A. Identifying Obstacles

Consider the following descendant of the goal Achieve [CrisisResolvedWhenRoutePlanAgreementReached] in Section II.A:

Goal Achieve [FireVehicleOnScene When Dispatched]
Def *Every fire vehicle dispatched to a crisis shall be on the crisis scene within the prescribed deadline.*

This leaf goal is assigned to the Fireman agent. Its root obstacle is obtained by negating it. The resulting obstacle is recursively refined into sub-obstacles until leaf obstacles are reached whose satisfiability, probability of occurrence, and resolution can easily be determined. Fig. 5 shows the resulting obstacle tree. For example, the fire vehicle may not be on the crisis scene in time because it was retracted or it took a wrong direction. The latter sub-obstacle might be caused by the driver confusing destinations, as shown in the refinement.

This example, focused on fire vehicle dispatching, shows that undesirable situations are not necessarily connected to communication problems among coordinators [3]. The restricted scope in [3] must thus be enlarged in order to anticipate problems that might severely affect the envisioned software. (We come back to this in the next section.)

The communication failures mentioned in [3] were all identified by obstacle generation using our regression calculus through domain properties [5]. Other problems missing in [3] were thereby discovered such as NetworkHardwareFailure, FloodingAttackCausingNetworkDelay, LargeTrafficPeak, Messages ModifiedByNetworkInfrastructure, etc.

The generated obstacles must be assessed in terms of probability of occurrence and criticality of consequences on the degree of satisfaction of high-level goals in the goal model [1].

B. Resolving Obstacles

The likely and critical obstacles must be resolved in order to produce a more robust goal model with new and/or deidealized goals, resulting in a more complete set of requirements. Various tactics are available for obstacle resolution, such as goal weakening, goal substitution, agent substitution, obstacle prevention or obstacle mitigation [5].

For example, the leaf obstacle FireVehicleDestinationConfused in Fig. 5 can be resolved in different ways (see Fig. 6). A first countermeasure might be to avoid sending fire vehicles in unfamiliar areas:

Goal Avoid [FireTruckDriver In UnfamiliarAreas]
Def *The planned fire truck routes shall avoid sending truck drivers in unfamiliar areas.*

This countermeasure decreases the probability of occurrence of the associated obstacle. The new goal has to be refined in turn towards assignable requirements and expectations. Note that this resolution has an impact on the software envisioned in [3]; the requirements discovered along the refinement of this new goal impose further constraints on route plan proposals in addition to those specified in [3]. They will appear more precisely in Section IV.A through postconditions on corresponding software operationalizations.

The alternative resolutions of FireVehicleDestinationConfused in Fig. 6 mitigate the goal obstruction by (a) introducing a GPS device to reduce the chance of confusion, or (b) dispatching another fire vehicle to the crisis together with driver guidance for reaching the crisis location.

Leaf goals along those resolutions might be obstructed by further obstacles, leading to a new cycle of obstacle analysis.

An anti-goal model with the goals of potential attackers should be built for security analysis using similar techniques [5] –notably, for generating attacks on the compromiser's goals and resolving the corresponding threats. Potential conflicts among goals in the goal model should similarly be detected and resolved [5].

IV. Deriving the Agent Model

An agent model captures the agents forming the target system, their capabilities, their responsibilities on goals and operations, and their interfaces with each other. The boundary between the software-to-be and its environment is thereby specified. The capabilities of an agent are defined in terms of state variables the agent can monitor and control. State variables correspond to object attributes and associations from the object model. Capabilities and agent interfaces are specified in a context diagram; the latter can be systematically derived from leaf goal specifications and responsibility assignments [5]. An agent model provides a basis for responsibility assignment heuristics, load analysis, vulnerability analysis, and software architectural design [5]. It may also show how agents and responsibility assignments are refined into finer-grained ones.

Fig. 7 shows a context diagram fragment derived from the specifications of the goals assigned to the corresponding agents. The labels there mean that the source agent controls the corresponding attribute/association whereas the target agent monitors it. For example, to satisfy its assigned goal of proposing a route plan draft to the coordinators, the CrisisSoftware needs to monitor information about vehicle

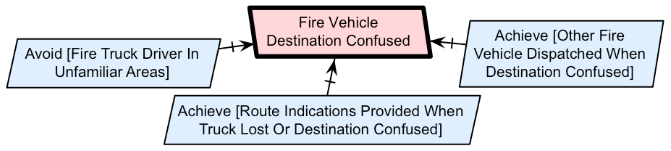
Fire Vehicle Destination Confused
Avoid [Fire Truck Driver In Unfamiliar Areas]
Achieve [Other Fire Vehicle Dispatched When Destination Confused]
Achieve [Route Indications Provided When Truck Lost Or Destination Confused]

Fig. 6. Resolutions of obstacle FireVehicleDestinationConfused

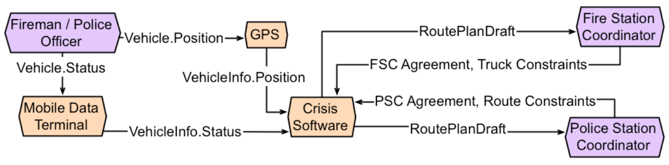

Fig. 7. Context diagram for bCMS agents

positions and availabilities. This information is sent by the GPS and the mobile data terminal inside each vehicle. The real, physical position and availability status of vehicles is controlled by fireman/police officer agents; these state variables are not directly monitorable by the software.

The precise relationship between real and assumed information about objects and their attributes/associations (such as vehicles here) must be prescribed by accuracy goals [5]. The latter are often overlooked in requirements documents in spite of being generally critical.

V. Deriving Operational Models

All leaf goals from the goal model must be operationalized into specifications of software operations or environment agent tasks. An operation model focuses on the operations to be performed by the software-to-be. The specifications of these operations are systematically derived from goal specifications using operationalization techniques [5]. Operation applications define events that trigger state transitions on state variables. A behavior model captures the required behaviors of an agent on the state variables it controls so as to meet its assigned goals.

This section considers the CentralizedSoftware alternative discussed in Section II.C and illustrates the building of an operation model fragment (Section V.A) and a behavior model fragment (Section V.B).

A. Deriving Operations from Goal Specifications

Let us consider the software operation ProposeRoutePlanDraft proposing a route plan draft to the coordinators, as illustrated by the first scenario interaction in Fig. 4 (left). This operation operationalizes leaf goals from the goal model such as Route PlanDraftProposedPromptly, DraftMeetsRequirements, DraftMeets Constraints and RoutePlanDraftFeasible.

The specification of this operation has three parts.

- The signature declares the input/output state variables. ProposeRoutePlanDraft takes as input a crisis info together with vehicle information (their current known availability status and position). It outputs a route plan draft.
- The domain pre- and postconditions capture what the operation intrinsically means in the domain regardless of any prescription for goal satisfaction. Here, the domain conditions capture a transition from a state where there is no route plan draft for the crisis to a state where such draft has been proposed to the coordinators.
- The required pre-, trigger- and postconditions must ensure that the goals underlying the operation are satisfied. A required pre- (resp. trigger-) condition for some goal captures a permission (resp. an obligation); under this condition the operation may (resp. must) be applied to ensure that goal. For example, the trigger condition hereafter prescribes that the operation must be performed as soon as the crisis requirements have been encoded. A required postcondition for some goal prescribes an additional effect that the operation must have to ensure that goal –in addition to the effect captured by the operation's domain postcondition.

Operation ProposeRoutePlanDraft
Input *c: CrisisInfo, v1..vn: VehicleInfo*
Output *rp: RoutePlanDraft*
DomPre *No route plan draft exists for resolving c*
DomPost *rp is proposed for resolving c*
ReqTrig for Achieve [RoutePlanDraftProposedPromptly]:
The number of fire trucks and police vehicles needed for handling c has been encoded by coordinators.
ReqPre for Maintain [DraftMeetsRequirements]:
There are sufficient vehicles available so as to meet c's requirements while meeting all stated constraints.
ReqPost for Maintain [DraftMeetsRequirements]:
rp allocates at least as many fire truck and police vehicles as stated in the fire and police requirements for crisis c, respectively.
ReqPost for Maintain [DraftMeetsConstraints]:
rp meets all constraints stated by fire and police coordinators
ReqPost for Maintain [RoutePlanDraftFeasible]:
Every vehicle in rp is currently available; the time to reach the crisis location from its current location is below X minutes.

The operation model may provide a basis for eliciting further requirements in a bottom-up fashion. Some missing goals underlying required pre-, trigger- and postconditions might be discovered through WHY questions about these. For example, consider the additional trigger condition on ProposeRoutePlanDraft stating that a new constraint is being given by a coordinator; this new obligation should prompt a WHY question to elicit the underlying goal.

B. Deriving Agent Behavior Models

State machine diagrams are used for prescribing the behaviors of an agent in terms of admissible sequences of state transitions for the state variables the agent controls. Depending on the kind of analysis to be performed, such diagrams can be UML statecharts or LTS diagrams. They are systematically derivable from goal operationalizations or from MSC scenarios [5].

Fig. 8 shows a partial statechart model derived for the crisis software agent. The diagram refers to the building and agreement of a route plan draft. The agreement process is modeled by a composite state made of three parallel state machines for the corresponding controlled state variables. In view of the guards, this composite state can be left only when both coordinators have agreed on the route plan draft, see Fig. 8 (left). The states, events and guards are derivable from domain pre- and postconditions and from required pre-, trigger- and postconditions from the operation model [5].

The goal, object, agent, operation and behavior models are integrated through inter-model consistency rules [5]. For example, every event appearing on a state transition in Fig. 8 should correspond to a scenario event in Fig. 4; the event route plan proposal must be controlled by the software in the agent model, and correspond to an application of the operation ProposeRoutePlanDraft specified in Section IV.A.

VI. Discussion

The paper illustrated the KAOS method for model-based RE on a number of modeling steps for a car crash management system. The full model integrates the intentional, structural, operational, and behavioral dimensions of RE. Along each

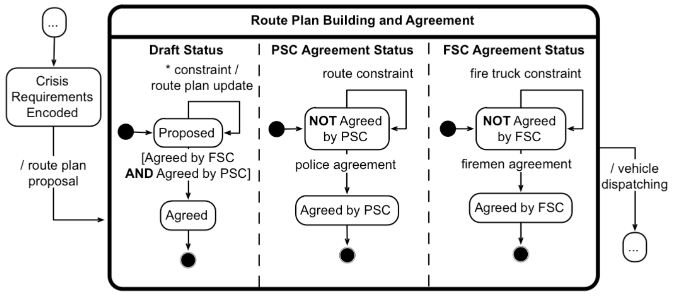

Fig. 8. Partial behavior model for the crisis software

dimension, dedicated elicitation or derivation techniques were used for model construction; consistency rules were used for integrating those multiple dimensions [5]. Goals were refined until they can be assigned to specific agents. Key structural concepts were systematically derived from goal specifications. The refinement process enabled reasoning on alternative options, illustrated by scenarios, and selected using soft goals from the goal model. A more complete and robust version of the goal model was obtained through obstacle analysis. From there, an agent model was derived together with goal-based specifications of software operations and agent behaviors.

Despite the linear presentation here, a KAOS model elaboration is intended to support a seamless transition from high-level concerns to operational requirements and vice versa; the process turns out to be both top-down (e.g., through HOW questions or refinement patterns) and bottom-up (e.g., through WHY questions about goals, scenarios, multiplicities in the object model, or required pre-, trigger- and postconditions in the operation model).

The modeling language, method, and derivation/analysis techniques associated with each step are intended to increase the quality of the requirements document derived from the model —in terms of requirements completeness, consistency, adequacy, precision, measurability, pertinence, and structuring. These quality attributes define the problem space a requirements modeling approach should address; fit criteria for them should provide a problem-oriented basis for comparing different modeling approaches for RE. Surprisingly, the solution-oriented comparison criteria in [4] ignores them.

The paper was more focused on the model elaboration process, thereby sacrifying to: the presentation of a more comprehensive model; the formalization of critical aspects; and the illustration of formal analysis techniques on critical issues. A more extensive model can be found in [2]. The high-level goal introduced in Section II.A is entirely refined within the scope of the three alternative options discussed in Section II.C. Other high-level goals are identified and refined. The model is therefore richer than here, with more soft goals, conceptual objects, obstacles and resolutions, responsibilities, agent interfaces, operations, and behavioral specifications.

The scope of our bCMS modeling is significantly wider than the one suggested in [3]. The latter puts a strong emphasis on the communication between the fire and police station coordinators –with premature decisions such as owning a T1 link or making use of the https protocol. Such decisions pertain to the solution space rather than the problem space addressed by the RE process. Many other aspects of the system are claimed to be out of scope, such as the internal communication among police personnel or with vehicles. Internal communication issues are definitely part of the problem space. For instance, are vehicle positions manually sent by radio or through an automated positioning system? This may strongly impact on the bCMS software to be developed.

In the context of an academic modeling exercise, the absence of real stakeholders makes it difficult to make realistic choices among possible options. The lack of domain descriptions in [3] made us explore system options within the wider context of resolving a car crash crisis rather than simply communicating among coordinators. As already suggested with the obstacle analysis in Section III, such wider-scope modeling appears highly relevant to the requirements-related quality attributes mentioned before. It enables reasoning about environment behaviors that will definitely impact on the quality of the software-to-be. Moreover, RE modeling approaches should be compared within realistic settings; RE is inherently an in-the-large process where the context of the proposed application has to be investigated. Zooming now on the communication among coordinators, our extended modeling in [2] suggests that the distributed design decided in [3] is not necessarily the best option.

The following improvements should be considered for making our bCMS model more complete.

- More security, accuracy and usability goals should be identified and refined in the goal model.
- Obstacle analysis should be extended to more leaf goals. More anti-goals for the communication compromiser should be identified and refined to increase the level of security in the system. Conflicts in the goal model should be identified and resolved as well [5] (e.g., NoFeasibleRoutePlanMeetingAllCrisis Requirements).
- Critical goals should be formalized in the KAOS real-time temporal logic to support formal analysis when and where needed, e.g., to prove the correctness of refinements and operationalizations, to detect conflicts formally, and to generate obstacles automatically [5].
- Our single-system model should be extended into a product line model using the OR-refinement and OR-assignment constructs to specify variation points [5].

References

[1] A. Cailliau, A. van Lamsweerde. *Assessing requirements-related risks through probabilistic goals and obstacles*, Req. Eng. Journal Vol. 18 No. 2, June 2013, pp 129-146.

[2] A. Cailliau et al., *Modeling Car Crash Management with KAOS*, UCL, May 2013, kaos.info.ucl.ac.be/bcms.html and www.cs. colostate.edu/remodd/v1/sites/default/files/Models-KAOS.pdf

[3] A. Capozucca, B. H.C. Cheng, G. Georg, N. Guelfi, P. Istoan, G. Mussbacher, *Requirements Definition Document for a Software Product Line of Car Crash Management Systems*, May 2012, http://cserg0.site.uottawa.ca/cma2013re/CaseStudy.pdf.

[4] Geri Georg et al., *Modeling Approach Comparison Criteria for the CMA@RE Workshop at RE'2013*, http://cserg0.site. uottawa.ca/cma2013re/ComparisonCriteria.pdf

[5] A. van Lamsweerde, Requirements Engineering: From System Goals to UML Models to Software Specifications. Wiley, 2009.