GOREMOCH: A Distributed Goal-oriented Policy Refinement Environment

Javier Rubio-Loyola, Joan Serrat
Universitat Politècnica de Catalunya
Barcelona, Catalonia. Spain
{jrloyola, serrat}@tsc.upc.edu

Marinos Charalambides, Paris Flegkas, George Pavlou
University of Surrey
Guildford, Surrey. United Kingdom
{M.Charalambides, P.Flegkas, G.Pavlou}@surrey.ac.uk

Abstract—**Goal-oriented requirements engineering methodologies have been suggested as an alternative to address the policy refinement paradigm. Moreover, practical approaches that capture the administrative and technical requirements to make policy refinement a systematic process are still missing although such integrated solutions are rather convenient to make policy-based management systems really useful. In this paper, we present GOREMOCH, a goal-oriented policy refinement environment grounded in goal-oriented requirements engineering methodologies, linear temporal logic and reactive systems analysis techniques. We describe the rationale of this integrated solution and the necessary mechanisms to achieve policy refinement in a systematic manner.**

Keywords: Policy refinement, Goal-oriented refinement, Goals

I. INTRODUCTION

Policy refinement is meant to derive lower level policies from higher level ones so these more specific policies are better suited for use in specific environments. Although policy refinement has been recognized as crucial, at the same time it has been a rather neglected area in policy-based management. Functional approaches are still missing in spite of the high relevance of policy refinement to policy-based management.

Goal-oriented Requirements Engineering techniques have been proposed as an alternative to address the policy refinement paradigm [1]. The generic policy refinement methodology presented in [1] is based on the KAOS goal-elaboration formal approach [2]. KAOS is used to elaborate goal graphs grounded in temporal logic [3]. High-level goals are decomposed into lower-level goals that logically entail the administrative guidelines prescribed by high-level goals. Goals are decomposed by using a set of pre-defined refinement patterns during the goal elaboration process [2].

In our previous work [4], we have proposed the use of linear temporal model checking to obtain system trace executions aimed at fulfilling lower level goals, elaborated with the KAOS goal elaboration method. From system executions, it is possible to abstract the managed entities, conditions and actions that need to be taken to fulfill lower-level goals, considering that these are temporally-related and that meaningful state transitions are policy-controlled.

Here, we move one step ahead towards the materialization of a functional approach to policy refinement following the above formalisms. We present GOREMOCH, a Distributed Goal-oriented Policy Refinement Framework. Our main contribution is the identification and description of the components, necessary to make policy refinement a systematic process. For this, we have identified a set of *functional issues* and additional *supporting activities* of our environment.

The *functional issues* addressed in GOREMOCH are those related to the practicability of the goal-oriented tasks (e.g. goal elaboration), the practicability to prescribe administrative decisions in accordance with the refined policies (e.g. selection of goals to fulfill with the refined policies), and the balancing of the computational complexity for large-scale refinements (i.e. distribution of management tasks). On the other hand, the *supporting activities* involve the practical support to document both, object distribution and system behavior within an integrated policy refinement environment.

The next section describes the refinement framework in which GOREMOCH relies. While Section 3 describes the GOREMOCH solution, Section 4 gives some flavor of how it is used to refine policies. Related work and conclusions are given in Sections 5 and 6 respectively.

II. POLICY REFINEMENT FRAMEWORK

GOREMOCH follows the goal-based approach to policy refinement initially proposed in [1], making use of linear temporal logic and model checking verification as policy analysis techniques [4]. The overall policy refinement framework on which GOREMOCH relies is shown in Figure 1. It demonstrates the steps to systematically deploy policies from high-level goals. We differentiate between operator directed processes and automated mechanisms. The formers are the *Goal Elaboration*, *Goal Selection* and *Request for Policy Refinement* steps. Regarding the automated mechanisms, namely the *Policy Refinement Mechanisms,* these are integrated by the *Requirements Formulation, Get System Behavior, Apply Translation Primitives* and *Encode/store Deployable Policies* steps. A brief description of these is given hereafter.

In the ***Goal Elaboration*** process (Step 1 in Fig. 1), the operator expert documents goal-graph structures following the formal KAOS goal elaboration method [2]. The expert makes use of refinement patterns to decompose goals into sub-goals that logically entail the original goal. Since the refinement patterns have been proved to be correct and given that they take into account the temporal prescription of goals, goal-graph structures are considered logically consistent.

In the ***Goal Selection*** process (Step 1 in Fig. 1), the operator consultant chooses the goal strategies that better commit with his directives. He achieves this by consulting the goal-graph library in which goals are stored and classified.

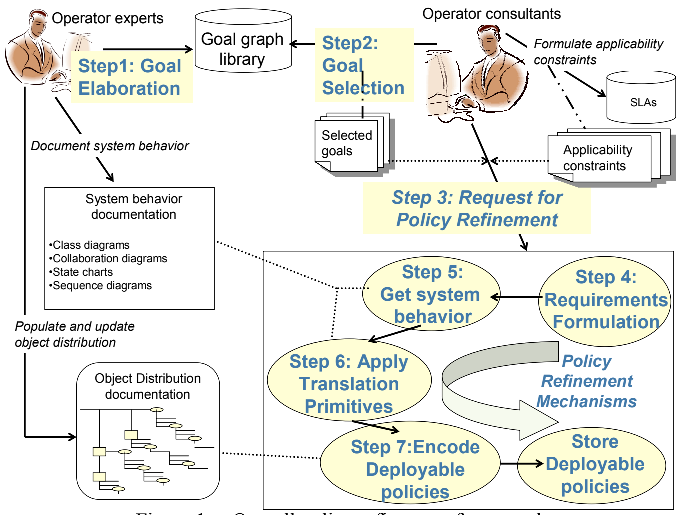

Operator experts
Goal graph library
Step1: Goal Elaboration
Step2: Goal Selection
Operator consultants
Formulate applicability constraints
SLAs
Selected goals
Applicability constraints
Document system behavior
System behavior documentation
•Class diagrams
•Collaboration diagrams
•State charts
•Sequence diagrams
Step 3: Request for Policy Refinement
Step 5: Get system behavior
Step 4: Requirements Formulation
Populate and update object distribution
Step 6: Apply Translation Primitives
Policy Refinement Mechanisms
Object Distribution documentation
Step 7:Encode Deployable policies
Store Deployable policies

Figure 1. Overall policy refinement framework

The policy refinement process starts with the submission of a ***Request for Policy Refinement*** (Step 3 in Fig. 1). It consists of the selection of goals that better suit the consultant needs and the applicability constraints (if necessary) for the to-be-refined policies (see Step 3 in Fig. 1).

The ***Policy Refinement Mechanisms*** are in charge of identifying the managed entities and the operations that they may take so that the system will meet the administrative requirements characterized as goals. In order to do so, the following processes are realized systematically:

The ***Requirements Formulation*** (Step 4 in Fig. 1) is aimed at producing the corresponding linear temporal logic (LTL) formulae that characterize goal fulfillment. This is achieved by abstracting the Temporal Relationships amongst the goal selection. In addition, this step also considers the refinement patterns with which the selected goals had been previously elaborated and the temporal prescription of its refinements.

The ***Get System Behavior*** (Step 5 in Fig. 1) aims to determine the necessary transitions a managed system may exhibit as to fulfill the goals. In our previous work [4], we demonstrated that by providing meaningful formulae characterizing goal fulfillment, model checking engines would automatically generate the system behavior, necessary to fulfill the corresponding goals. For this task we rely on the automated support provided by the SPIN model checker [5], specifically in its ability to find and report counterexamples. The two inputs of model checking are the specification of system behavior and that of requirements. Given that LTL requirements formulae can characterize goal fulfillment [4], we use SPIN to produce the necessary transitions that our managed system may exhibit as to fulfill the goals. Regarding system behavior specification, this is documented making use of standard UML notations such as state charts, class, collaborations and sequence diagrams. This information is managed properly during this step so that the model checking engine can take it as reference to find meaningful counterexamples.

The ***Apply Translation Primitives*** (Step 6 in Fig. 1) aims to abstract the policy fields from the counterexample reports produced in the *Get System Behavior* step. As counterexamples may include more than one policy enforceable transition, typical counterexamples may produce multiple sets of policy fields, each set corresponding to one policy enforceable transition. In order to abstract the policy fields from the counterexample reports, this step implements a set of translation primitives. The result of applying these primitives is a set of policy instances, one for each enforceable transition. Finally, the ***Encode Deployable Policies*** step takes as input this information and the object distribution to encode deployable policies in a policy specification language. In this paper we have considered the Ponder specification language [8].

III. GOREMOCH SOLUTION

The refinement process might become overwhelming and exhaustive for large-scale refinements. Additionally, different stakeholders might contribute to its update, management and usage. GOREMOCH has been proposed as a CORBA-based distributed component environment that logically entails the approach described above. Our current implementation integrates the following components: Objectiver package, Goal Manager, Behaviour Manager, Requirements Manager, Search Manager, Policy Encoder and Inventory. In Figure 2 we show the components that integrate GOREMOCH and the most relevant interactions of these components during a refinement process. A brief description of these components is given hereafter.

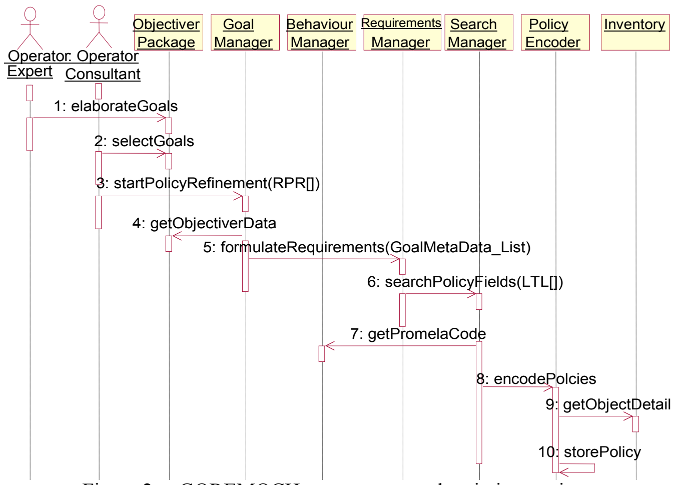


Figure 2. GOREMOCH components and main interactions

We begin our description with the **Objectiver package**. Objectiver [9] is a Requirements Engineering toolkit that supports the KAOS methodology. It is intended to help analysts identify, formalize requirements and write such requirements documents in a pure goal-oriented fashion. In our environment, Objectiver is the toolkit available for the operator expert to elaborate and document goals following the KAOS method. In this regard, it is the support for the *Goal Elaboration* step of the refinement process (`elaborateGoals` interaction in Fig. 2). For the consultant, it is the interface to select the goals that better fulfil his needs. In this sense, the Objectiver package is the support for the *Goal Selection* step of the policy refinement process (`selectGoals` interaction in Fig 2).

Goal Manager: The policy refinement process starts with the submission of a Request for Policy Refinement (RPR). Many RPRs may be submitted concurrently. A RPR consist of a selection of goals that better suit the consultant needs and the applicability constraints for the to-be-refined policies. The Goal Manager is the component in charge of processing the RPRs (`startPolicyRefinement` interaction in Fig 2). It is the logical component that bridges the gap between the *Goal Selection* step and the *Requirements Formulation* step. The Goal Manager exchanges information with the Objectiver package during the Goal Selection process. In this regard we could say that it shares the tasks with the Objectiver package for the *Goal Selection* step of the overall policy refinement process. The submissions of the Requests for Policy Refinement are carried out by the operator consultant. GOREMOCH provides an interface for RPR submissions. Since the goals selected by the operator may influence other goals stored in the Objectiver package database, one of the main tasks of the Goal Manager is to coordinate queries to Objectiver in order to get the relevant information about the goals selected by the consultant (`getObjectiverData` in Fig 2).

Requirements Manager: This component is in charge of realizing the *Requirements Formulation* step of the overall policy refinement process. It receives the goal-related information from the Goal Manager (`formulateRequirements` interaction in Fig. 2). The Requirements Manager implements a finite-state verification property pattern design database [6]. The later allows producing LTL formulae from specific requirements in a systematic manner. By managing the information contained in this database in a proper manner, the Requirements Manager formulates LTL formulae that characterize goal fulfilment in our solution.

Search Manager: This component derives a set of policy fields (i.e. subjects, targets, events and actions) that would make the selected goals to be fulfilled. This component executes the *Get system behavior* and *Apply Translation Primitives* steps of the overall policy refinement process in order to derive the relevant policy fields. It receives the set of LTL formulae that characterize goal fulfillment from the Requirements Manager (`searchPolicyFields` in Fig 2). In order to execute its management tasks, the Search Manager interacts with the Behavior Manager (see `getPromelaCode` interaction in Fig 2). The first process carried out by the Search Manager is that of the *Get system behavior* of the overall policy refinement process. For this, this component integrates a SPIN toolkit for the counterexample acquisition. Once the meaningful counterexamples are produced, the Search Manager executes the *Apply Translation Primitives* step of the overall policy refinement process. During the later, the Search Manager applies a set of pre-defined Translation Primitives as to abstract the corresponding policy fields.

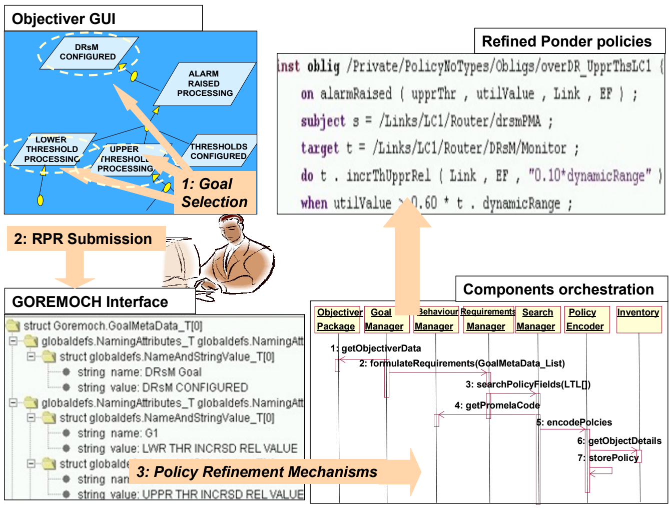
Behavior Manager: This component manages the documentation of the system behavior specification. It provides an interface to which other components get information related to the system behavior specification (e.g. `getPromelaCode` interaction in Fig. 2). In our environment, the administrator expert is enabled to document system bahaviour using standard representation such as UML models. These models may contain active classes with state machines, collaborations and interactions. In this sense, this component provides support for the *System behaviour documentation* supporting activity of the overall policy refinement process. Since the analysis technique of this approach relies on guided-search model checking, one of the main functions of this component is to coordinate the UML managed objects specifications and their translation into PROMELA (input language of SPIN). In order to achieve this, the Behavior Manager implements the libraries provided by HUGO [7]. In this sense, the *Behavior Manager* is a supporting component for the *Get System Behavior* step of the policy refinement process.

Policy Encoder and Inventory Manager: The former coordinates the *Encode Deployable Policies* process of the overall policy refinement process. It receives the set of policy fields produced by the Search Manager and the administrative constraints provided by the consultant (see `encodePolicies` interaction in Fig. 2). It follows the syntax of the Ponder specification language [8] and includes in its internal architecture, a Ponder Policy Editor and compiler (slightly modified to automate the compilation process). Special attention might be put to the subject and target objects for the deployable policies. On one hand, we consider that the administrative guidelines may influence a set of managed objects (e.g. routers, sub networks, etc) to which the consultant refers during the RPR submissions. On the other hand, we consider that the actual managed objects are classified following a domain-dependent logical representation in which a well-defined structure of this information may allow automating the policy refinement process. For this purpose, we have included a support component; the **Inventory Manager**. This component provides support for the administrator to document object distribution. In addition, the Inventory Manager enables other components to access to this information. For instance, the Policy Encoder asks for meaningful information (`getObjectDetails` interaction in Fig. 2) as to encode and store deployable policies. In this sense, the Inventory Manager component provides support for the *Object Distribution documentation* supporting activity of the policy refinement process. The introduction of the Inventory Component has enabled to automate the policy refinement process considering that its internal database is populated for a specific policy applicability domain.

IV. REFINING POLICIES WITH GOREMOCH

In this section we give a brief description of how the operator consultant refines policies with GOREMOCH. In general terms, GOREMOCH has been designed and implemented as a distributed system where several components instances may be running for large-scale refinements. It provides support for all the *functional issues* and *supporting activities* mentioned along the paper. Due to space limitations, we only give a general overview of the refinement procedure. We consider that the operator expert has previously elaborated the goal graphs with Objectiver, documented system behaviour making use of UML standards and populated the Inventory database with the appropriate information according to the applicability domain.

Under these circumstances, the refinement process starts when the operator consultant selects the goals that better satisfy his requirements (Step 1 in Fig. 3). For this purpose he browses through the Objectiver GUI attached to GOREMOCH. The Objectiver databases store all the available goal strategies elaborated for the managed system. After the goals have been selected, the operator consultant submits the Request for Policy Refinement (RPR), including the arguments for the goals if necessary (Step 2 in Fig.3). In order to submit this and subsequent RPRs (concurrently if desired), the operator consultant uses the GOREMOCH interface available for this purpose. After RPR submission, GOREMOCH executes the automated Policy Refinement Mechanisms mentioned in Section 2 resulting in an orchestration of its components (Step 3 in Fig.3). Having the components executed their management tasks and internal algorithms, the outcome of the refinement process is a set of deployable policies.



```
inst oblig /Private/PolicyNoTypes/Obligs/overDR_UpprThsLC1 {
    on alarmRaised ( upprThr , utilValue , Link , EF ) ;
    subject s = /Links/LC1/Router/drsmPMA ;
    target t = /Links/LC1/Router/DRsM/Monitor ;
    do t . incrThUpprRel ( Link , EF , "0.10*dynamicRange" )
    when utilValue > 0.60 * t . dynamicRange ;
```

```
struct Goremoch.GoalMetaData_T[0]
⊟ globaldefs.NamingAttributes_T globaldefs.NamingAtt
   ⊟ struct globaldefs.NameAndStringValue_T[0]
        ● string name: DRsM Goal
        ● string value: DRsM CONFIGURED
⊟ globaldefs.NamingAttributes_T globaldefs.NamingAtt
   ⊟ struct globaldefs.NameAndStringValue_T[0]
        ● string name: G1
        ● string value: LWR THR INCRSD REL VALUE
   ⊟ struct globalde
        ● string nan
        ● string value: UPPR THR INCRSD REL VALUE
```

Figure 3. Generic overview of policy refinement with GOREMOCH

V. RELATED WORK

Practical approaches to policy refinement are scarce. POWER [10] is a tool for refining policies from policy templates designed by an expert so that consultants could create business policies from such templates. Policies are expressed as Prolog programs and an inference engine interprets them to guide the user through the refinement process. Our approach differs to POWER in the sense that ours is a goal-oriented policy refinement approach in which the consultant selects the goals that better satisfy his needs, having the possibility to formulate any combination of goals as required, with no pre-definition of policy template choices. More recently, work by Bandara et al [1] has contributed significantly to the policy refinement area. They propose an approach for transforming both policy and system behavior specifications into a formal notation based on Event Calculus (EC). The authors use goal elaboration and abductive reasoning to derive strategies that would achieve high-level goals. From these strategies, policies are encoded. While Event Calculus and abduction is used in the EC-based approach to infer the sequences of actions that achieve particular goals, our approach goes through state exploration to obtain system behavior that fulfils lower-lever goals elaborated through temporal refinement patterns. We encode policies using the information abstracted from the execution trace while in the EC-based approach these are encoded using the generated strategies.

Besides the above, explicit approaches for policy refinement are scarce. Recent efforts mostly cope with application specific policy translations. In [11] the authors address refinement introducing a set of decomposition rules and pre-defined resource type hierarchies for access control environments. In [12] the authors propose a refinement mechanism based on multi-layer system modeling. Here, low-level policy generation for security systems is achieved by analyzing system object's relationships between layers.

VI. CONCLUSIONS

We have presented a functional environment for goal-oriented policy refinement grounded in Requirements Engineering methodologies, Linear Temporal Logic and Reactive systems analysis techniques. We have demonstrated in general terms, the necessary mechanisms for an integrated solution that provides support to capture the administrative and functional requirements to achieve systematic policy refinement in a goal-oriented fashion.

We have demonstrated that systematic policy refinement is feasible once we have identified and defined a set of distributed components which interact accordingly to carry out the policy refinement process. As far as it is reflected in the literature, policy refinement is at its initial stage. In this sense, we hope that GOREMOCH may contribute to solve the policy refinement problem by providing a practical approach that captures the administrative and technical requirements to automate the refinement process, so many times recognized as crucial but at the same time so much dismissed.

REFERENCES

[1] A.K. Bandara, E.C. Lupu, J. Moffett, A. Russo; "A goal-based approach to policy refinement" IEEE Intl. Workshop on Policies for Distributed Systems and Networks, 2004

[2] R. Darimont and A. van Lamsweerde, "Formal refinement patterns for goal-driven requirements elaboration," Symp. on Found. of Soft. Eng. 1996

[3] Z. Manna, A. Pnueli. The Temporal Logic of Reactive and Concurrent Systems: Specification. Springr-Verlag, 1992

[4] J. Rubio-Loyola, J. Serrat, M.Charalambides, P. Flegkas, G. Pavlou, A. Lluch. "Using linear temporal model checking for goal-oriented policy refinement frameworks" IEEE International Workshop on Policies for Distributed Systems and Networks Stockholm, Sweden June 6-8, 2005

[5] G. Holzmann. The SPIN Model Checker: Primer and Reference Manual. A. Wesley. ISBN 0-321-22862-6. 2004

[6] M. Dwyer, G. Avrunin, and J. Corbett. "A system of specification patterns". http://patterns.projects.cis.ksu.edu/

[7] M. Balser, S. Bäumler, A. Knapp, W. Reif, and A. Thums. "Interactive verification of UML state machines". *Intl. Conference on Formal Engineering Methods 2004*

[8] N. Damianou, T. Tonouchi, N. Dulay, E. Lupu, and M. Sloman. "Tools for doamin-based policy management of distributed systems", NOMS, Friorence, Italy, 2002

[9] E. Delor, R. Darimont, A. Rifaut. "Software quality starts with the modelling of goal-oriented requirements". Intl. Conference of Software & Systems Engineering, 2003

[10] M. Casassa, A. Baldwin, C. Goh. "POWER prototype: towards integrated policy-based management" NOMS 2000

[11] S. Linying et al. "Automated Decomposition of Access Control Policies". IEEE POLICY, Stockholm, Sweden 2005.

[12] J.P. de Albuquerque et al. "Policy Modeling and Refinement for Network Security Systems". IEEE POLICY, Stockholm, Sweden 2005.