Using Linear Temporal Model Checking for Goal-oriented Policy Refinement Frameworks

Javier Rubio-Loyola[1], Joan Serrat[1], Marinos Charalambides[2], Paris Flegkas[2], George Pavlou[2], Alberto Lluch Lafuente[3]
[1]Universitat Politècnica de Catalunya, [2]University of Surrey, [3]Università di Pisa
[1]{jrloyola, serrat}@tsc.upc.edu, [2]{M.Charalambides, P.Flegkas, G.Pavlou}@eim.surrey.ac.uk
[3]lafuente@di.unipi.it

Abstract

Policy refinement is meant to derive lower-level policies from higher-level ones so that these more specific policies are better suited for use in different execution environments. Although it has been recognized as crucial, it has received relatively little attention. We present a policy refinement framework grounded in goal-elaboration methodologies and reactive systems analysis. Through Linear-Time Model Checking, we obtain system trace executions aimed at fulfilling lower-level goals refined with the KAOS goal-elaboration method. From system executions, we abstract managed entities, conditions and actions to encode the refined policies. We present our framework and provide a refinement scenario applied to the DiffServ QoS Management domain.

1. Introduction

A Policy-Based Network Management (PBNM) system should allow the description of high-level policies, enable their refinement into lower-level ones and map them to commands that ultimately configure the managed devices. Despite the enormous research done on languages for specifying policies and architectures for managing and deploying such policies into distributed environments for different application domains, policy refinement is a key area that still remains scarcely studied.

Goal-oriented Requirements Engineering has been proposed as a feasible alternative to formalize policy refinement [1]. Goal-elaboration assisted by domain-independent refinement patterns [2] has opened a new and promising research area for policy analysis.

The representation of individual or several managed objects is possible by defining finite state machines that describe the multiple states in which such managed objects can be [3], being possible to relate the behavior of an object or a set of objects to the value of one or more attributes that are used to characterize the states of the system. State transitions are directly related to changes of attributes, which policies configure and control. The general "*on-event* and *if-condition* then *action*" structure of policy rules makes it possible to consider policy-based systems as event/state-driven systems and use formal methods to analyze their behavior. Model checking [4] is a formal automated approach to exhaustively analyze whether event/state-based systems satisfy specific behavioral claims characterizing safety and reliability requirements. After modeling a system and its requirements in suitable formalisms, verification algorithms check whether the system satisfies its requirements by exhaustively testing all possible combinations. One of the keys in the success of model checking remains in its ability to find and report counterexamples as execution traces that show the processes, conditions, actions and states that make a requirement not to hold.

In this paper, we present a policy refinement approach based on Goal-oriented Requirements Engineering and Model Checking techniques. As initially proposed in [1], through goal-elaboration methods, we refine lower-level goals that logically entail high-level administrative guidelines. After this, making use of linear temporal logic formulae and model checking capabilities, we obtain execution traces aimed at fulfilling the refined lower-level goals. From system executions, relevant policy information is abstracted and eventually encoded into a set of refined policies. The novelty of the approach presented in this paper is the introduction of formal verification techniques in the context of goal-oriented policy refinement frameworks.

The main issue behind policy refinement is to abstract generic policy refinement patterns. Nevertheless, abstracting patterns applicable to all management domains is too difficult and probably impossible. We start our study towards this direction for a Differentiated Services (DiffServ) Quality of Service (QoS) Management domain and present a refinement scenario for this domain.

After this introduction, Section 2 provides the formalisms used in our approach. Section 3 reviews our policy refinement framework. Section 4 presents a refinement scenario and Section 5 discusses some important issues and future work as well. Section 6 presents the related work to conclude in Section 7.

2. Background

2.1. Pattern-driven goal elaboration

Goals capture, at different levels of abstraction, the various objectives a system should achieve [5]. They provide the rationale for requirements elaboration. Many goal classifications have been presented in the literature and different approaches for goal-oriented elaboration and reasoning techniques have been developed in the Requirements Engineering (RE) area [5]. A temporal classification of goals is based on the behavior prescribed by the goal. The following are identified:

- *Achieve* and *Cease* goals obey to system behaviors that require some target property to be eventually satisfied or denied respectively, in some future state.
- *Maintain* and *Avoid* goals restrict behaviors, in that they require some target property to be permanently satisfied or denied respectively, in every future state.

We will follow the above classification for reasoning analysis since it can be related to the obligation, refrain, authorization and negation concepts, widely used in policy-based management.

Several approaches have been proposed to formalize goal elaboration [5]. For the reasons expressed above, we propose to use support provided by goal refinement methodologies grounded in temporal logic. As initially proposed by [1], we use KAOS [2], a formal technique to elaborate goals inspired by the classic linear temporal logic of Manna and Pnueli [6]. In the remaining of this section we briefly outline KAOS.

By definition, a set of goal assertions G1, G2,…,Gn is a complete refinement of a goal assertion G iff the following conditions hold:

1. $G1 \wedge G2 \wedge \ldots \wedge Gn \models G$ (entailment)
2. $\forall\, i,j: j \neq i \rightarrow Gj \not\models Gi$ (minimality)
3. $G1 \wedge G2 \wedge \ldots \wedge Gn \not\models \text{false}$ (consistency)
4. $n > 1$ (nonequivalence)

The KAOS method contains two essential operators to relate goals: AND and OR refinement. The former relates a goal to a set of subgoals in which satisfying all subgoals in the refinement is a sufficient condition for satisfying the high-level goal. OR-refinement relates a goal to an alternative set, satisfying a refinement is a sufficient condition for satisfying the goal.

In KAOS, a refinement pattern is a one-level AND-tree of abstract goal assertions such that the set of leaf assertions is a complete refinement of the root assertion. The KAOS method proposes the general principle of reusing domain-independent refinement patterns. These patterns have been included in a set of libraries [7] that have been previously proved to be correct. The libraries are grouped by the behavior prescription of the high-level goals, namely *Achieve, Cease, Maintain* and *Avoid*. Due to space limitations and to the nature of the scenario proposed in Section 4, we limit our study to *Achieve* refinement patterns; the interested reader might consult [7] for an extended description. Table 1 shows some AND-decomposition patterns for high-level *Achieve* parent goals.

Table 1.Some prepositional patterns for *achieve* goals

RP	Subgoals		
RP1	$P \wedge R \rightarrow \Diamond Q$	$P \rightarrow \Diamond R$	$P \rightarrow P\ \mathcal{W}\ Q$
RP2	$P \rightarrow \Diamond R$	$R \rightarrow R\ \mathcal{U}\ Q$	
RP3	$P \rightarrow \Diamond R$	$R \rightarrow \Diamond Q$	
RP4	$P \wedge P1 \rightarrow \Diamond Q1$	$P \wedge P2 \rightarrow \Diamond Q2$	$\Box(P1 \vee P2)$ $Q1 \vee Q2 \rightarrow Q$
RP5	$P \wedge \neg R \rightarrow \Diamond R$	$P \wedge R \rightarrow \Diamond Q$	$P \rightarrow \Box P$
RP6	$\neg R \rightarrow \Diamond R$	$P \wedge R \rightarrow \Diamond Q$	$P \rightarrow \Box P$

Table 1 presents different refinement patterns (RPs) that represent different possibilities to decompose the high-level goal into the respective subgoals. The *Achieve* goal is formally expressed as $P \rightarrow \Diamond Q$: If P then eventually Q in the future. We use the classical temporal operators: $\Diamond$ eventually in the future, $\Box$ always in the future, $\mathcal{U}$ always in the future until and $\mathcal{W}$ always in the future unless. RP3 for instance, defines a *milestone-driven* tactic where an intermediate state satisfying *R* must first be reached, from which a final state satisfying *Q* must be reached. RP4 proposes decomposition by cases. KAOS provides the necessary support to hierarchically structure goals as graphs by using different tactics in which the lower-level subgoals logically entail the higher-level goal.

2.2. Linear temporal model checking

Model Checking [4] is a formal and automated application of computational logic with high relevance in concurrent and distributed systems verification. As shown in Figure 1, it consists of three main processes: modeling system behavior, modeling the requirements specification of the system and verifying whether the system satisfies its specification.

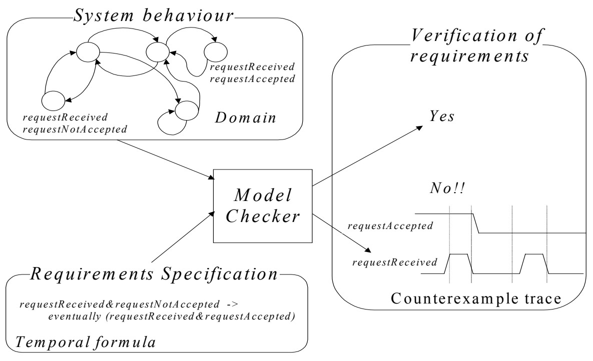


Figure 1. Model checking basic steps

Different formalisms have been proposed to model system behavior, each tailored for specific domains. Amongst the most common, labeled transition systems (LTSs) are typically used. An LTS is a set of states together with a transition set, modeling how a system changes its state. Additionally, a labeling function is used to relate states and transitions with observations.

The second process corresponds to the requirements specifications' modeling. At this stage, system observables like events, state of variables or the processes responsible of the transitions are the subject of interest. In fact, observations of a system are crucial to specify the requirements for the correctness of an event/state-based system. A fundamental dimension is time and how observables are time-related. This is precisely the aim of temporal logics. The use of a specific logic depends on the type of verification to be performed. Different model checkers have been developed for different temporal logics. Different temporal logics and model checkers can be found in [6] and [4] respectively.

The third process is the verification of the requirements specifications under "all" circumstances of system execution. It attempts to span the entire state space and verify every possible combination of inputs (events and conditions). If a requirement (i.e. a property) does not hold, model checkers can help to identify the input sequence that triggers the failure (i.e. conditions, events and states that made the property not to hold). This ability has made model checking so successful for reactive systems verification.

In our study, we need to have a mechanism that allows one to express the ordering of events in time where observations are extended with temporal connections such as “eventually in the future” or “always in the future”. We will then focus our work on Linear Temporal Logic (LTL) Model Checking verification. LTL formulae are interpreted over computations of a system with sequences of states representing executions of a system. If we let P be a set of observable predicates, and f and g be LTL formulae on P, then the representations shown on Table 2 are also LTL formulae. The predicates and boolean connectives expressed in the first row formulae have the usual meaning. For the second row formulae, operator ○ next state is used in addition to those described in Section 2.1, formula ○f would be satisfied if the next state of the computation satisfies f. The interested reader might consult [6] for extended description of LTL formulae.

Table 2. Different LTL formulae

$\neg$ f	f || g	f && g	f $\rightarrow$ g	f $\leftrightarrow$ g
○f	□f	◊f	f *U* g	f *W* g

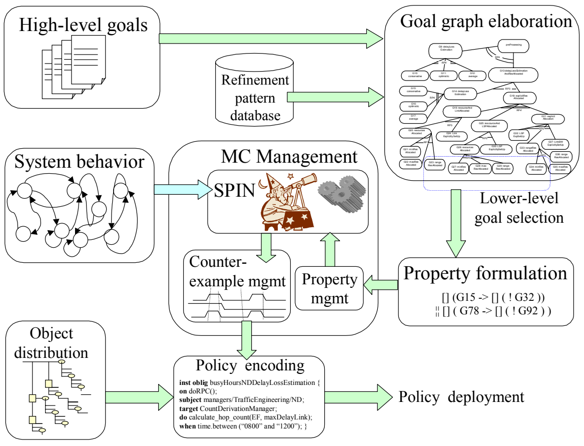
Broadly speaking, with LTL model checking we can design and verify typical temporal properties that express *absence, universality, existence, precedence* and *response* of observable predicates and any combination of these. A guide of how to design formulas using temporal descriptions can be found at [8].

If we were to verify a property of an event/state-based system in which P is globally absent (formally expressed as □!P), we could get a counterexample trace indicating the conditions, states, events and transitions that make such absence not to hold, namely the occurrence of P. From the counterexample, it is also possible to identify the managed entities that collaborate to make the absence not to hold. In other words, we can use LTL formulae to obtain the conditions, states, events and transitions and the managed entities’ collaboration that make P to hold. This is the basic idea behind the approach presented in this paper; counterexamples traces can be interpreted as plans that make the system satisfy determined properties using a fully automated formal method.

As a model checking tool we use SPIN [9], a LTL model checker that focuses on the verification of concurrent software systems. It uses Promela [9] as the modeling language. Promela specifications are basically state machines that communicate via message-passing or shared variables. Requirement specification can be done using some ad-hoc mechanism to express deadlock-freedom or validity of assertions, but, more generally, LTL is used. Not only does SPIN provide the possibility to verify properties but also to simulate the system and obtain/reproduce counterexamples. As we will see, these capabilities are relevant for our refinement analysis.

3. Policy Refinement Framework

Goal refinement must eventually result in the identification and specification of requirements whose responsibility must be assigned to agents [2]. Our approach to policy refinement is shown in Figure 2.



```
inst oblig busyHoursNDDelayLossEstimation {
on doRPC();
subject managers/TrafficEngineering/ND;
target CountDerivationManager;
do calculate_hop_count(EF, maxDelayLink);
when time.between ("0800" and "1200"); }
```

Figure 2: Overall process for policy refinement

The following steps may be followed to systematically deploy policies from high-level goals:

- *Goal graph elaboration*
- *Responsibility assignment to managed entities*
- *Operationalization* (both part of the *Counter-example Management*)
- *Policy encoding*

In the Goal graph elaboration step, AND/OR structures are built, defining goal hierarchies and their refinement links. High-level goals are decomposed using domain-independent refinement pattern libraries following the KAOS elaboration method. The desired outcome of this step is a set of lower-level goals that logically entail higher-level ones. From the many options of the structured goal graph, the administrator selects the lower level goals that better satisfy his requirements: lower-level goal selection sub step of Figure 2. While the KAOS elaboration method provides support for this first step, it does not provide support to relate system behavior to goal fulfillment finding. Nevertheless, in order to carry out the following steps, this gap needs to be filled.

As described in Section 2.2, the inputs of model checking are the specification of system behavior and that of requirements. For the former, we propose graphical representation, as depicted in Figure 2. As mentioned, the modeling language of SPIN is PROMELA [9] whose specifications are expressed basically as state machines that communicate via message passing or shared variables. The procedure to translate graphic state charts or other visual modeling languages into PROMELA is out of the scope of this paper, this issue has been widely studied in the literature [10]. Regarding the other input of Model checking, i.e. the specification of requirements, the Property formulation step shown in Figure 2 is aimed at designing property formulae that characterize lower-level goals. As in plan-based techniques [5], lower-level goals are identified by state predicates, hence the requirements must be characterized by such lower-level goals. The Property formulation step takes into account temporal information in which the lower-level goals have been constructed. As described in Section 2.2, a very important issue here is that the requirements are encoded into LTL formulae that basically describe the *absence* of the behavior prescribed by the low-level goal predicates. For example, if the low-level goals G11 and G31 must be fulfilled to satisfy a high-level goal G, and their temporal behavior is such that G11 must be fulfilled before G31, we might encode a temporal formula specifying that G31 is never fulfilled after G11. That way, the execution trace might indicate system behavior to achieve G13 after G11 and consequently G.

The property management process inside the MC Management module is aimed at coordinating the query to the model checker (SPIN). When the model checker is queried with LTL formulae, it generates counterexample traces that might be interpreted further for the following refinement steps. Due to the nature of the requirement specification (absence-based), the counterexample will display the execution trace that results as a consequence of the *presence* of the predicates and the desired temporal behavior.

For the *Managed entities responsibility assignment* and *Operationalization* steps, a systematic interpretation of the counterexample trace generated by SPIN is necessary. Both steps are part of the Counterexample management process shown in Figure 2. One of the outputs of SPIN is a sequence chart. For the responsibility step, we take this message sequence chart to select the managed entities responsible to achieve the administrative decisions. This information and the object distribution information are used in the final step of refinement as shown later.

Undoubtedly, the Operationalization process is crucial for policy refinement. It implicitly includes the responsibility assignment in the sense that the operationalization step goes into the details of the counterexample trace to identify the actions that the managed entities may take. For our refinement problem, the Operationalization step is particularly focused at finding the processes that imply decisions in order to identify the conditions, transitions and operations that are meaningful for policy encoding. For this step of the refinement we rely on the SPIN capabilities to generate detailed traces from which this valuable information is obtained.

Finally, the *Policy encoding* step takes as input the information described above and the object distribution in order to encode the policies in a policy specification language. We have considered Ponder [11]. Following its deployment model, the refined policies can be compiled and deployed in the policy-based system.

4. Policy Refinement Application Scenario

The specialization of refinement patterns applicable to all management domains is very difficult. In practice, specialization patterns might be abstracted from specific policy applicability areas. We present a policy refinement scenario applied to DiffServ QoS management. We describe the steps of the refinement process applied to this domain and the feasibility of our approach.

4.1. Application domain

The application domain of our scenario relies on the framework developed in the context of the EU IST TEQUILA project [12]. TEQUILA provides a policy-based functional architecture for supporting QoS in IP DiffServ networks. The generic architecture of the TEQUILA framework is shown in Figure 3. It is decomposed into three major subsystems, namely; the Service Level Specification (SLS) management, the Traffic Engineering (TE) and Monitoring system. The former is responsible for agreeing QoS services (SLSs) with customers and handling respective requests while the TE subsystem is responsible for fulfilling the contracted SLSs by appropriately engineering the network. Due to space limitations, we will focus on the TE part of TEQUILA, particularly the Network Dimensioning (ND), in order for the reader to better understand the scenario proposed in Section 4.2. More details about TEQUILA can be found at [12] and [13].

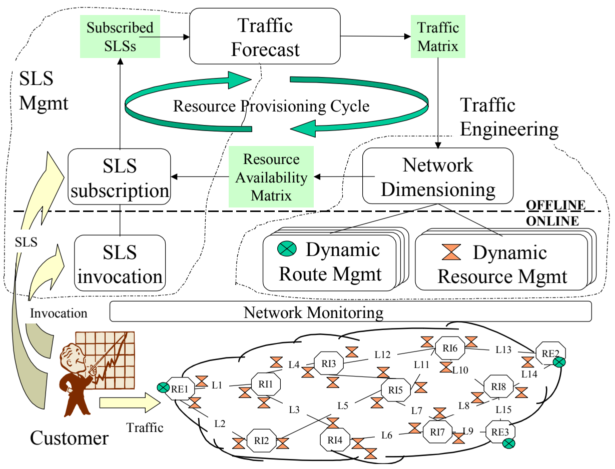


Figure 3: Generic architecture of TEQUILA

ND is responsible for mapping the traffic onto the physical network resources in order to accommodate the forecasted traffic demands. Configuration includes the definition of Label Switched Paths (LSPs) and anticipated loading for each Per Hop Behavior (PHB) on all interfaces. The output of ND is provided to DRtM and DRsM, and also to SLS Management in order to base the admission control decisions for future SLS subscriptions. ND is decomposed in the following subcomponents: *Traffic Matrix Manipulation*: this is responsible for retrieving the Traffic Matrix (TM) from Traffic Forecast and also provides functions for manipulating entries in the TM. *Network topology*: this holds the objects describing the physical network topology together with the physical capacity of the network links. *Explicit LSP and BW allocation*: this offers methods that can explicitly define the MPLS Labeled Switched Paths (LSPs) that Traffic Trunks (TTs, aggregates of traffic flows with the same origin-destination pair and same performance requirements) should follow (expLSP state in Fig. 4). This subcomponent also offers methods that can explicitly define the way bandwidth (BW) should be allocated to different traffic classes Ordered Aggregates (OAs, minResAlloc and masResAlloc states in Fig. 4). *Hop Count Derivation*: this provides functionality to handle the QoS requirements of the expected traffic in terms of delay and loss requirements by transforming them into maximum hop count constraints (see minDelayLoss, maxDelayLoss and avgDelayLoss states in Fig. 4). *Optimization Algorithm*: its objective is to find a set of paths for which the BW requirements of the TTs are met as well as the requirements for delay and loss by using the hop count constraint as an upper bound and at the same time optimize the use of network resources. ND allows setting upper bounds on the number of hops the calculated paths are permitted to have and on the number of alternative paths for every TT for load balancing purposes (hopBalancing and pathBalancing states in Fig. 4). It provides functions to customize the BW allocation by setting importance guidelines for a particular OA (cost function settings; min/max/expCostResAlloc states in Fig. 4). It also supports two optimization objectives: i) avoid overloading parts of the network while other parts are under loaded (minLinkLoad state in Fig. 4) or ii) provide overall low network load (maxNetLoad state in Fig. 4) . A compromise between these two options is also possible (netCompromised state in Fig. 4). *Spare/over-provisioned BW treatment*: this assigns residual physical capacity to the various classes (spareCap states in Fig. 4) or reduces the allocated capacity (OverCapacity states in Fig. 4) when link capacity cannot satisfy the predicted traffic requirements. Due to the complexity of the ND component, we consider the summarized behavior specification shown in Figure 4.

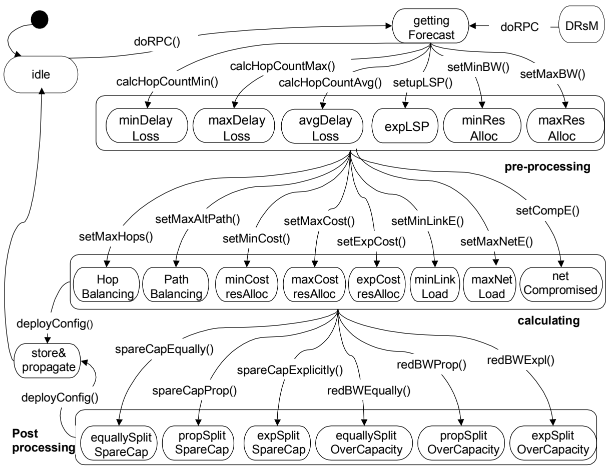


Figure 4: Simplified behavior of ND module

4.2. Scenario description

Consider the case where a set of subscribed SLSs includes one from the "AOL" client for which the administrator wants to ensure that specific dimensioning directives be enforced especially during specific periods of time, namely during busy hours (8:00 to 12:00hrs and 15:00 to 17:00hrs). Consider also that the AOL client in the network of Figure 3 has contracted an SLS with the following technical parameters:

- A pipe between routers RE1 and RE2 with EF PHB, 5ms delay, zero packet loss and assured throughput of 5Mbps.

We consider that pipe defines the boundaries of the QoS to be enforced as a “one-to-one” ingress-egress SLS model. During busy hours, the administrator wants to ensure that traffic entering the domain from the node RE1 and exits from the node RE2 belonging to such Ordered Aggregate (OA), follows the route RE1-RI1-RI3-RI6-RE2. Additionally, due to the strict requirements of delay and packet loss, the administrator wants to be extremely conservative for the hop-count estimation for traffic of the same QoS-class as that of the “AOL” client. During busy hours, the administrator wants to avoid under-loaded parts of the network when other parts are overloaded and wants to make sure that any spare capacity is equally split amongst the PHBs.

The administrator will need to encode these administrative goals into policies, considering the managed entities that would enforce them, the actions, conditions and constraints for their execution. Considering that not only might the administrator design policies for the ND module but for other components in the TEQUILA framework, this task might become extremely difficult.

4.3. Proposed solution

4.3.1 Step 1: Goal graph elaboration

Since the requirement of the administrator in the above scenario is to ensure a given behavior, we use the *Achieve* goal refinement patterns described in Section 2.1. The final goal of the administrator is that appropriate administrative directives are stored and propagated to the underlying components. This high-level goal can be formulated using the following Linear Temporal specification:

G1 busyHoursDimensioning:

$$TM\_busyHours \rightarrow \Diamond\ configStored\&Propagated$$

G1 is interpreted as: "when retrieving the traffic matrix, appropriate ND directives for busy hours should eventually be configured, stored and propagated to the underlying components". Applying the case-driven refinement pattern RP4 of Table 1, we derive the subgoals G2 and G3, formally expressed as follows:

G2 preProcessing&Calculation:

$$TM\_busyHours \wedge preProcessed\&Calculated \rightarrow \Diamond\ configStored\&Propagated$$

G3 preProcessingAndPostProcessing:

TM_busyHours /\ pre&PostProcessed →
◊ configStored&Propagated

For each of these sub-goals (G2 and G3), we apply a milestone-driven refinement tactic (RP3 of Table 1). For G3, this has to be extended since RP3 is only applicable for a single milestone refinement pattern. We propose to extend RP3 and RP4 to make them suitable for a multiple milestone-driven and case-driven fashion respectively. RP3 and RP4 are modified as depicted in Table 2.

Table 2. Some extended refinement patterns

RP	Subgoals
RP3’	P → ◊ R R → ◊ S S → ◊ Q
RP4’	P /\ P1 → Q1, P /\ P2 → Q2, P /\ P3 → Q3, □(P1\/P2\/P3), Q1\/Q2\/Q3 → Q

Applying RP3 and RP3' to G2 and G3 respectively, we elaborate the goal graph shown in Figure 5.

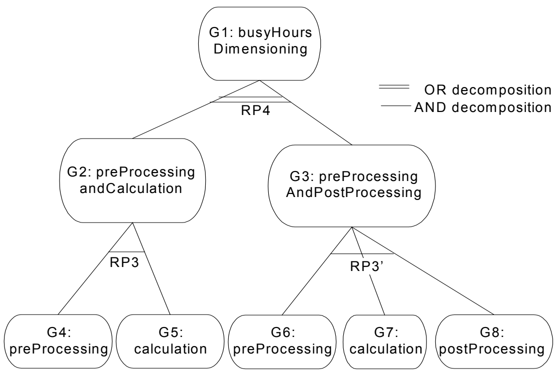


Figure 5: Initial high-level goal elaboration

Lower-level goals *preProcessing* (G4), *calculation* (G5), *preProcessing* (G6), *calculation* (G7) and *postProcessing* (G8) are represented as follows:

G4: TM_busyHours /\ preProcessed&CalculatedReq/\ preProcessedReq → ◊ calculated

G5: calculatedReq → ◊ configStored&Propagated

G6: TM_busyHours /\ pre&PostProcessedReq /\ preProcessedReq → ◊ calculated

G7: calculatedReq → ◊ postProcessing

G8: postProcessingReq→ ◊configStored&Propagated

From this initial goal-decomposition, it is not feasible for the administrator to select the sub-goals that best satisfy the high-level goals, so further decomposition is needed for these lower-lever goals (G4 to G8).

Applying similar refinement guidelines, we elaborate the goal graph for the *preProcessing* subgoal (G4 and G6) as shown on Figure 6. In order to compose this goal graph, RP3 and RP4 have been used. We have used the extended RP4’ shown in Table 2 to refine G9, G14 and G18 into their respective subgoals. Taking *delayLossEstimation* (G9) as a high-level goal, subgoals *conservative* (G10), *optimistic* (G11) and *average* (G12) are refined. They are formally represented as follows:

G9: preProcessReq/\delayLossEstReq→ ◊ calculated

G10: preProcessedReq/\delayLossEstReq/\ conservativeReq → ◊ calculated

G11: preProcessedReq/\delayLossEstReq/\ optimisticReq → ◊ calculated

G12: preProcessedReq/\delayLossEstReq/\ averageReq → ◊ calculated

Similar temporal representations can be obtained for the remaining subgoals. Regarding the *calculation* and *postProcessing* sub goals shown in Figure 5, they are refined similarly and their goal elaboration hierarchies are shown in Figures 7 and 8 respectively.

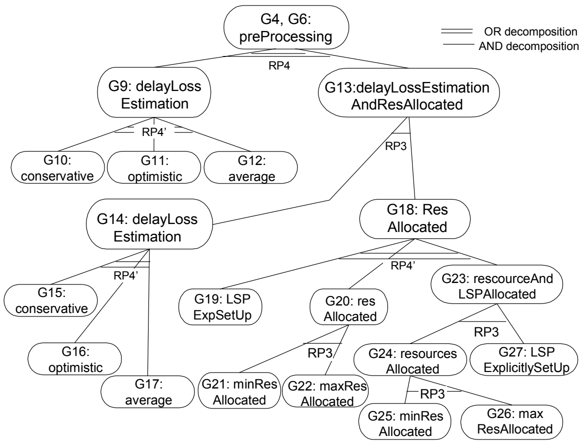


Figure 6: Pre processing goal graph

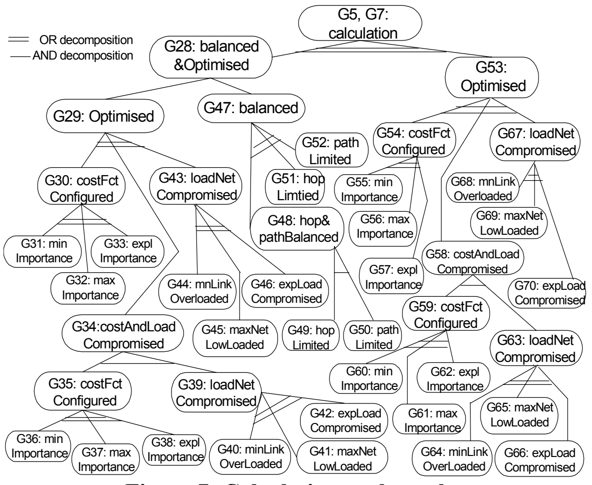


Figure 7: Calculation goal graph

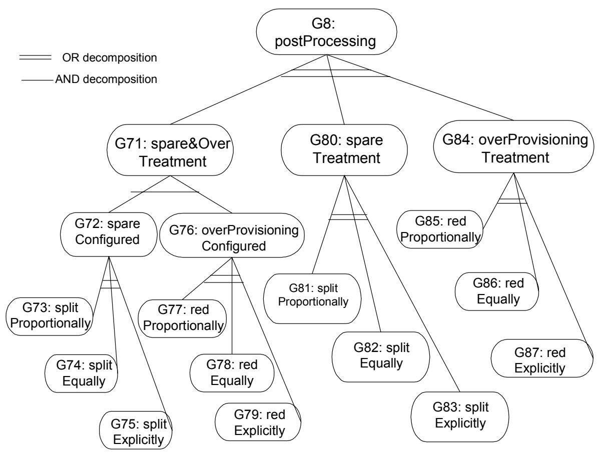


Figure 8: Post processing goal graph

Analyzing the description of the administrative high-level goal, we consider that the administrator's requirements are fulfilled by satisfying the subgoals G15, G19, G68 and G82.

4.3.2 Step 2: Responsibility assignment and operationalization of goals

Due to the distribution of modules in the TEQUILA architecture, as in any distributed system, fulfilling a high-level goal may require the cooperation of a combination of multiple components. The fist step towards Responsibility assignment in our framework is to design a temporal property that characterizes *absence* of the system behavior that could make the lower-level goals be fulfilled. For this purpose, we design the following LTL property:

$$\square(G15 \rightarrow \square(\neg G19)) \parallel \square(G68 \rightarrow \square(\neg G82)) \quad \ldots. \text{(P1)}$$

The interpretation of P1 is: "there is no system behavior in which either, state G19 is true after G15 or G82 is not fulfilled after G68". By querying SPIN with P1, the counterexample generated would give the opposite: "system behavior in which both G19 is fulfilled after G15 and G82 after G68 respectively". For a detailed description of how to make up similar specification patterns the reader may consult [8]. One of the outputs of the counterexample generation process is a message sequence chart from which the managed entities in charge of enforcing the administrative guidelines are easily identified. The abstract message chart of our scenario is shown in Figure 9. From this chart, we identify the following components responsible to enforce the administrative guidelines: hop Count Option module, explicit Route/BW Selection, optimisation and spare/overprovisioning treatment modules.

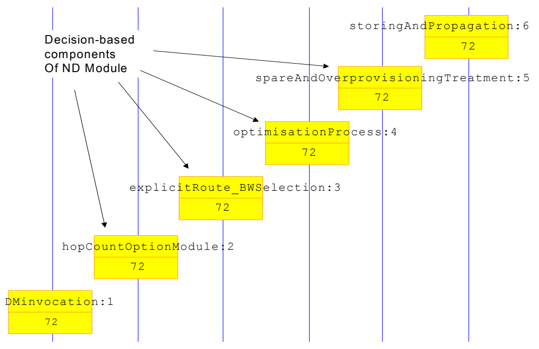
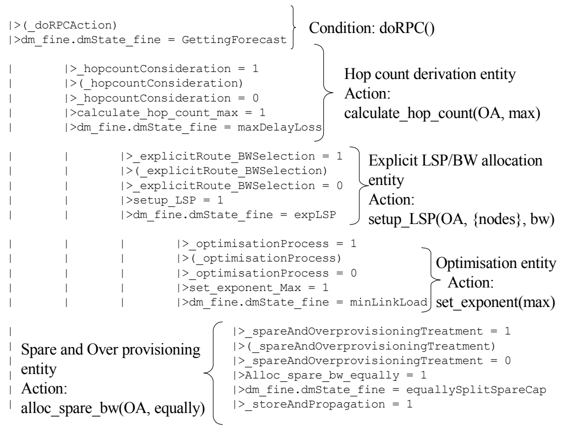

Decision-based components Of ND Module
storingAndPropagation:6
72
spareAndOverprovisioningTreatment:5
72
optimisationProcess:4
72
explicitRoute_BWSelection:3
72
hopCountOptionModule:2
72
DMinvocation:1
72

Figure 9: SPIN abstract message chart

For the operationalization step, we go into the detailed counterexample trace generated by SPIN. This implicitly includes the responsibility assignment step in the sense that operationalization goes into the details of the trace to identify the actions that the managed entities may take to achieve the administrative guidelines. In Figure 10 we show the counterexample trace for our ND scenario after a basic filtering process. The first bracket corresponds to the condition that enables the system behavior of our counterexample. The subsequent brackets indicate which actions the managed entities may take to fulfill the high-level guidelines. The first three lines of each of these brackets indicate that the managed entity needs to be invoked and the invocation needs to be attended (values from 1 to 0). The fourth line indicates what action the managed entity needs to take. Finally, the 5th line sets the attributes of the managed entity that determine the desired state goal.



```
|>(_doRPCAction)
|>dm_fine.dmState_fine = GettingForecast

|       |>_hopcountConsideration = 1
|       |>(_hopcountConsideration)
|       |>_hopcountConsideration = 0
|       |>calculate_hop_count_max = 1
|       |>dm_fine.dmState_fine = maxDelayLoss

|       |       |>_explicitRoute_BWSelection = 1
|       |       |>(_explicitRoute_BWSelection)
|       |       |>_explicitRoute_BWSelection = 0
|       |       |>setup_LSP = 1
|       |       |>dm_fine.dmState_fine = expLSP

|       |       |       |>_optimisationProcess = 1
|       |       |       |>(_optimisationProcess)
|       |       |       |>_optimisationProcess = 0
|       |       |       |>set_exponent_Max = 1
|       |       |       |>dm_fine.dmState_fine = minLinkLoad

|                                 |>_spareAndOverprovisioningTreatment = 1
|                                 |>(_spareAndOverprovisioningTreatment)
|                                 |>_spareAndOverprovisioningTreatment = 0
|                                 |>Alloc_spare_bw_equally = 1
|                                 |>dm_fine.dmState_fine = equallySplitSpareCap
|                                 |>_storeAndPropagation = 1
```

Condition: doRPC()

Hop count derivation entity
Action:
calculate_hop_count(OA, max)

Explicit LSP/BW allocation entity
Action:
setup_LSP(OA, {nodes}, bw)

Optimisation entity
Action:
set_exponent(max)

Spare and Over provisioning entity
Action:
alloc_spare_bw(OA, equally)

Figure 10: SPIN Detailed trace

4.3.3 Step 3: Policy encoding

Once conditions, managed entities and actions have been identified, we need to encode the policies whose enforcement might reproduce the counterexample traces obtained in the previous step. In Ponder [11], Obligation policies are event-triggered and define the activities (actions) that **subjects** must perform on **targets**. The event triggering our policies is the condition identified in the last section (*doRPC*). The subject is the automated manager component in charge of enforcing the ND policies (*ND_PMA*). The targets are the managed entities and the actions are identified in the last section (*hop Count Derivation Manager, Explicit Allocation Manager, Optimisation Manager* and *Spare/Over Provisioning BW Manager*). It is evident that the applicability of these policies is restricted by administrative decisions (only during busy hours), hence these constraints are directly mapped to the obligation policy constraints. The resulting set of the refined policies for our scenario is shown in Figure 11.

```
inst oblig busyHoursNDDelayLossEstimation {
on doRPC();
subject ND_PMA;
target managers/TE/ND/hopCountDerivationManager;
do calculate_hop_count(EF, maxDelayLink);
when time.between ("08:00", "12:00") and time.between ("15:00",
"17:00"); }

inst oblig busyHoursNDAllocation {
on doRPC();
subject ND_PMA;
target managers/TE/ND/ExplicitAllocationManager;
do setup_LSP(EF, {RE1, RI1, RI3, RI6, RE2}, 5000);
when time.between ("08:00", "12:00") and time.between ("15:00",
"17:00"); }

inst oblig busyHoursNDOptimisation {
on doRPC();
subject ND_PMA;
target managers/TE/ND/OptimisationManager;
do set_exponent(maxValue);
when time.between ("08:00", "12:00") and time.between ("15:00",
"17:00"); }

inst oblig busyHoursNDOverProvisioning {
on doRPC();
subject ND_PMA;
target managers/TE/ND/SpareOverProvisioningBWManager;
do alloc_spare_bw(EF, equally);
when time.between ("08:00", "12:00") and time.between ("15:00",
"17:00"); }
```

Figure 11: Set of policies resulting from the refinement process

5. Discussion and future work

Two issues about counterexample generation deserve discussion. The first is when no counterexamples are found. Two options might cause this problem additional to exhaustion of the time and space available to the model checker: 1; Wrong goal refinement patterns applied to elaborate the goal-graph. 2; The behavior of the system mismatches the temporal goal elaboration. To overcome these problems, an additional activity of Requirements Engineering is needed, namely *Alternative selection of goals* [5]. An additional alternative would be to extend the system specification. A potential compromise between these two options needs to be studied further.

The second issue is when more than one counterexample is found. This might imply that the administrator could choose between different options, corresponding to different policy encodings, possibly for different conditions. A comparative notion of multiple counterexamples will also be part of our future work.

The core of our work will be directed to provide tool support to automate the processes presented in this paper. We have envisaged the use of Objectiver [14] for goal-elaboration/management and ArgoUML and Hugo/RT [10] for behavior and PROMELA translation processes respectively. Additionally, for large-scale systems, work by Edelkamp and Lluch [15] will be included to provide support with guided search techniques additional to those provided by SPIN.

Additionally, we will study other temporal formulae and goal elaboration patterns aimed at formalizing inconsistencies between goals and between policies. Additionally, the verification of other temporal behavior prescription like universality, existence, precedence and response would be used to explore system behavior aimed at verifying such properties to find inconsistencies or potential conflicting behaviors.

6. Related work

POWER [16] is one of the few policy refinement approaches hitherto implemented. Our approach differs to POWER in the sense that the latter is an environment in which the user is guided to choose policies from pre-designed policy templates designed by an expert, tailored for specific use. Instead, we use space exploration to find system behavior that satisfies pre-refined high-level goals in a goal-oriented framework. We consider reactive system analysis through temporal verification and model checking.

More recently, work by Bandara et al [17] 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. Our approach and the EC-based approach differ in the way system behavior is analyzed and in how policy information is abstracted. While Event Calculus and abduction is used in the former 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. The main advantage of the model checking-based framework over the EC-based framework is that the former can be used in situations where it is necessary to account for an explicit temporal execution of the goals when performing refinement. This has not been addressed in the EC-based approach [17]. At the time of this publication, there is no evidence of performance evaluation of any refinement approach. Future work will also address comparative evaluations between the EC-based approach and our framework.

7. Conclusions

We have presented an approach to policy refinement based on Requirements Engineering and model checking techniques. It allows find system executions aimed at fulfilling low-level goals that logically entail high-level administrative guidelines. From system executions, policy information is abstracted and eventually encoded into a set of refined policies specified in Ponder [11]. We have described the foundations, the reasoning of our approach and the refinement process through a scenario applied to the DiffServ QoS Management domain.

The main contribution of our work is the introduction of formal verification techniques in the context of policy refinement. This approach is novel and opens a new front of study for policy analysis. We hope that our proposal may contribute to solve the policy refinement problem, so many times recognized as crucial but at the same time so much dismissed.

Several outstanding issues have been identified. Amongst the most important, we can mention the lack or the existence of more than one solution (i.e. none or more than one counterexample), the partial or total automation of the involved processes and finally, the scalability of framework directly related to “state explosion” problem. All these issues have been assessed and will be the focus of our work.

8. References

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

[2] R. Darimont and A. van Lamsweerde, "Formal Refinement Patterns for Goal-Driven Requirements Elaboration," 4th ACM Symposium on the Foundations of Software Engineering (FSE4) No. 179-190, 1996.

[3] C.J. Strassner. *Policy-based Network Management, Solutions for the Next Generation*. Elsevier, Morgan Kaufmann Publishers 2004. ISBN: 1-55860-859-1

[4] E.M. Clarke, O. Grumberg, and D.A. Peled. *Model Checking*. The MIT Press, 1999.

[5] A. van Lamsweerde “Goal-Oriented Requirements Engineering: A Guided Tour”. 5th IEEE International Symposium on Requirements Engineering, Toronto, August, 2001, pp. 249-263.

[6] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification*. Springer, 1992

[7] R. Darimont, *Process Support for Requirements Elaboration*, PhD Thesis, Université Catholique de Louvain, Louvain-la-Neuve, Belgium, 1995.

[8] M. B. Dwyer, G. S. Avrunin and J. C. Corbett. “Property Specification Patterns for Finite-State Verification” Workshop on Formal Methods in Software Practice 1998.

[9] G. Holzmann. *The SPIN Model Checker: Primer and Reference Manua*". A. Wesley. ISBN 0-321-22862-6. 2004

[10] M. Balser, S. Bäumler, A. Knapp, W. Reif, A. Thums. “Interactive Verification of UML State Machines”. Proc. 6th Int. Conf. Formal Engineering Methods (ICFEM'04)

[11] N. Damianou, T. Tonouchi, N. Dulay, E. Lupu, and M. Sloman. "Tools for Doamin-based Policy Management of Distributed Systems", (NOMS2002), Friorence, Italy, 2002

[12] P. Flegkas, P. Trimintzios, and G. Pavlou, "A Policy-based Quality of Service Management Architecture for IP DiffServ Networks," IEEE Network Magazine, 2002.

[13] P.Trimintzios etal. "A Management and Control Architecture for Providing IP Differentiated Services in MPLS-based Networks". IEEE Communications s Magazine, 2001.

[14] E. Delor, R. Darimont, A. Rifaut. “Software Quality Starts with the Modelling of Goal-Oriented Requirements”. (ICSSEA 2003), December 2-4, 2003

[15] S. Edelkamp, A. Lluch , S. Leue. “Directed explicit-state model checking in the validation of communication protocols”, Software Tools for Technology Transfer, 2003.

[16] M. Casassa, et al “POWER prototype: towards integrated policy-based management”. NOMS 2000

[17] A. Bandara et al. "Policy refinement for DiffServ Quality of Service Management". To appear in International Symposium on Integrated Network Management (IM 2005).