Instituto Tecnológico y de Estudios Superiores de Monterrey
Campus Guadalajara

Escuela de Graduados en Ingeniería y Arquitectura (EGIA)

Maestría en Ciencias de la Computación

Applying Model Checking to Detect Conflicts in Rule-Based Internet Plans

AUTOR: Alvaro Maza Maza

ASESOR: Dr. Gerardo Padilla Zarate

Guadalajara (Jal), 01 de Octubre de 2015

This thesis is dedicated to my parents and family.

Acknowledgements

I would like to express my sincere gratitude to my advisor Dr. Gerardo Padilla, for the continuous support and guidance during the thesis process. His valuable advice, patience and encouragement have been of great importance for this work.

Besides my advisor, I would like to thank the rest of the thesis committee: Dra. Gema Gudiño and Dr. Ivan Romero, for their insightful feedback, interest and tough questions.

EGIA, particularly the MCC program with all its members, played an invaluable role in my graduate education.

Last, but not least, I would like to thank my parents and family.

Abstract

This thesis presents a formal approach to verify rule-based Internet plans through Linear Temporal Logic (LTL) properties and to determine whether or not there is a conflict amongst the rules within a plan. Rule-based Internet plans are formalized as specifications using our proposed Promela model.

Current and future use-case scenarios obtained from the public domain, are used to validate and to explain our proposed Model. The entire set of identified scenarios was successfully modeled and verified using our Promela model and LTL formulas introduced as part of this thesis.

The results obtained, demonstrate that Model Checking is a valid and feasible tool to model and verify rule-based Internet plans.

Resumen

Esta tesis presenta un metodo formal para verificar planes de Internet basados en reglas, mediante Logica Temporal Lineal (LTL) y para determinar si hay conflicto entre las reglas dentro del plan. Los planes de Internet basados en reglas son formalizados y especificados utilizando nuestro modelo propuesto en Promela.

Escenarios actuales y futuros fueron tomados del dominio publico y utilizados para validar y explicar nuestro modelo propuesto. Todos los escenarios identificados fueron exitosamente modelados y verificados utilizando nuestro modelo en Promela y formulas LTLs introducidas como parte de nuestra tesis.

Los resultados obtenidos, demuestran que Model Checking es una herramienta válida y factible para modelar y verificar los planes de Internet basados en reglas.

Table of Contents

Dedication I

Acknowledgements II

Abstract III

Resumen IV

List of Tables VIII

List of Figures IX

Listings XI

1. Introduction 1

1.1. Background 1

1.2. Problem Statement and Motivation 2

1.3. Aim and Objectives 3

1.3.1. Aim 3

1.3.2. Objectives 3

1.4. Scope, Limitations, and Delimitations 3

1.5. Justification 4

1.6. Hypothesis 6

1.7. Contribution of the Thesis 6

1.8. Organization of the Thesis 6

2. Literature Review **8**

2.1. Mobile Broadband services 8

2.1.1. Evolution 8

2.1.2. Examples of value-added services 10

2.2. Broadband communications technologies 11

2.3. Model Checking 12

2.3.1. Spin Model Checker and PROMELA 13

2.3.2. Linear Temporal Logic - LTL 14

2.4. Related Work 16

2.5. Summary of the chapter 17

3. Promela Model and LTL Properties **18**

3.1. Rule-based Plans Model 18

3.1.1. Overview 18

3.1.2. Process Definition 21

3.1.3. Example of rule-based plans 23

3.2. LTL Properties 24

3.2.1. Macros Definitions 24

3.2.2. Properties Definitions 25

3.2.3. Conflict verification through properties 27

3.3. Summary of the chapter 27

4. Case Study: MexCom Use Cases **28**

4.1. MexCom Overview 28

4.2. Core Rules 29

4.3. Roaming Bolt-on Rules 30

4.4. Family Rules 30

4.5. Parental-Control Rules 31

4.6. Limited Bolt-on Rules 32

4.7. Summary of the chapter 32

5. Case Study: Detecting Conflicts on MexCom Use Cases **33**

5.1. Core Rules . . . 33

5.1.1. Promela Model . . . 33

5.1.2. System Properties . . . 34

5.1.3. SPIN Verification . . . 36

5.2. Roaming Bolt-on Rules . . . 38

5.2.1. Promela Model . . . 38

5.2.2. System Properties . . . 39

5.2.3. SPIN Verification . . . 43

5.3. Family Rules . . . 46

5.3.1. Promela Model . . . 46

5.3.2. System Properties . . . 49

5.3.3. SPIN Verification . . . 50

5.4. Parental-Control Rules . . . 53

5.4.1. Promela Model . . . 53

5.4.2. System Properties . . . 55

5.4.3. SPIN Verification . . . 56

5.5. Limited Bolt-on Rules . . . 56

5.5.1. Promela Model . . . 56

5.5.2. System Properties . . . 58

5.5.3. SPIN Verification . . . 59

5.6. Result Analysis and Discussion . . . 60

6. Conclusion **63**

6.1. Summary of Contributions . . . 63

6.2. Future research . . . 64

References **68**

A. Promela Model Algorithms **68**

Vitae **71**

List of Tables

2.1. Key use-cases listed by *Elitecore Technologies* in [14]. 10

2.2. Semantics for the temporal operators 15

3.1. *Rule_ Type* Attribute definitions. 19

3.2. *Action_ Def* Attribute definitions. 19

3.3. *Rule_ Conditions* Attribute definitions. 20

3.4. *Time_ of_ day* Attribute definitions. 20

3.5. *Time Range* Attribute definitions. 21

3.6. Macros Definitions 24

3.7. Properties Definitions 25

3.8. Extended Properties Definitions 26

4.1. Core Limited Usage Rules. 29

4.2. Bolt-on Rules 29

4.3. Roaming Bolt-on Rules 30

4.4. Family Shared Rules 30

4.5. Family Shared Rules - Bolt-ons/Constraints 31

4.6. Rule Constraints 31

4.7. Limited Bolt-on Rules 32

5.1. Summary of results 61

List of Figures

2.1. Evolution of the Mobile Broadband services [2] 9
2.2. Model checking process 13

3.1. Typedef diagram 18

5.1. Core Rules - Linear Temporal Logic 35
5.2. Roaming Rules - Linear Temporal Logic 40
5.3. Roaming Rules - Linear Temporal Logic II 42

Listings

3.1. PriorityChecker high-level algorithm 22
3.2. Free Facebook Rule. 23
3.3. Free Navigation Rule. 23
3.4. Macros for each rule 24

5.1. Core-Rules-based Plan Model 34
5.2. Core-Rules LTL 36
5.3. Core-Rules Plan Verification 36
5.4. Increased priority of the Core Rule 37
5.5. Invalid Core-Rules Plan - Verification 37
5.6. Core-Rules-based Roaming Plan Model 38
5.7. Core-Rules-based Roaming Plan Model - II 41
5.8. Roaming-Rules LTL 42
5.9. Core-Rules Plan Verification 43
5.10. Increased priority of the Roaming Rule 44
5.11. Invalid Roaming-Rules Plan - Verification 45
5.12. Roaming-Rules LTL Invalidad 45
5.13. Invalid Roaming-Rules Plan - Verification II 46
5.14. Family Member#1 Model 47
5.15. Family Member#2 Model 47
5.16. Family Member#3 Model 48
5.17. Family Member#1 LTL 49
5.18. Family Member#2 LTL 49
5.19. Family Member#3 LTL 49
5.20. Family Member#1 Plan Verification 50
5.21. Family Member#2 Plan Verification 50
5.22. Family Member#3 Plan Verification 51

5.23. Invalidad Family Member#1 Model 52
5.24. Invalidad Family Member#1 Plan Verification 52
5.25. Parental-Control-Rules-based Plan Model 54
5.26. Parental-Control-Rules LTL 55
5.27. Parental-Control-Rules Plan Verification 56
5.28. Core-Rules-based Plan Model 56
5.29. Limited Bolt-on Rules LTL 58
5.30. Limited Bolt-on-Rules Plan Verification 59
5.31. Invalid Limited Bolt-on Rules LTL 59
5.32. Invalid Limited Bolt-on-Rules Plan Verification 60

A.1. PriorityChecker Process - Promela Code 68
A.2. OverQuota Process - Promela Code 70

CHAPTER 1

Introduction

1.1 Background

Internet is a collection of inter-connected private networks. These networks are operated by many Internet service providers (ISPs) or Network Operators in the world. How these networks interact with each other has strongly changed over the past years.

Just a decade ago, Internet users were just starting to use high-speed broadband access. By that time, there were only 569 million Internet users and we could download a song in around 10 minutes. Now, things changed and we are over 2.27 billion Internet users in the world and Real-time entertainment is, by far, the largest category of Internet traffic.

In order to sustain profitable business models, network operators are now looking at new ways of pricing models, beyond the standard expectation of volume-based plans [1].

Particularly, mobile operators have realized that they need to offer a number of flexible and smartly designed data packages, aligned to the budgets of different customer segments and target groups. As an example, several mobile operators are now offering free access to certain applications such as Facebook or WhatsApp.

1.2 Problem Statement and Motivation

During the development of Broadband Plans, Network Operators specify several rules to make each plan. A rule generally defines an action to either allow or block certain applications. More complex rules, add additional constraints, such as a predefined bandwidth or a limited amount of bytes to be consumed over a period of time.

While a Broadband Plan is composed of one or more of these rules, they are usually designed individually and developed independently to offer new plan options to customers. When rules are added to a particular plan, an unexpected conflict may rise with the existing rules of the plan. For instance, when a new rule, such as *Unlimited Facebook*, which allows unlimited access to Facebook traffic, is added to a plan already including a rule to limit up to 1GB of monthly usage, a conflict happens in the case when Facebook traffic is generated and these two rules are configured. The reason is that these two rules can be applied to the Facebook traffic.

Such conflicts are best resolved as early as possible – ideally before the implementation of those rules. This work is motivated by the desire to prove if Model Checking can be used to detect such those conflicts.

Nowadays, Internet plans are evolving quickly. In fact, network operators are moving from unlimited or volume-based models to value-added data offerings [1, 2]. Consequently, it is reasonable to predict that their offer of plans will soon include a non-small number of rules, including conditions such as its current location, device, current time of the day, etc. Subscribers may even have the option to select the rules to make their own plan.

In order for us to use Model Checking to detect conflicts between rules within a plan, we first need to design a generic model to specify the different plans. Afterwards, we can use a formal verification language to exhaustively and automatically check whether the model meets the plan specification and to detect conflicts between the rules in the plan.

1.3 Aim and Objectives

1.3.1 Aim

Explore the use of formal methods to identify conflicts between rules, within an Internet Plan. A popular technique known as model-checking has been used in many other domains. Our aim is to apply model checking to verify the model of rule-based Internet Plans and identify conflicts.

1.3.2 Objectives

- Create an abstract model to specify the rule-based Internet Plans.
- Implement algorithms to complement the model-checking verification.
- Define real and future use-cases scenarios based on Internet Plans offered in the public domain.
- Exhaustively and automatically check whether the model meets the specification of the plan.
- Detect if there is any conflict between two or more rules within the rule-based Internet plan.

1.4 Scope, Limitations, and Delimitations

The model described in this Thesis was based on the fact that Internet plans, and Network Policies, have been commonly expressed in the form of rules and actions. Network specifications evolved from Network Policy languages, found in [3, 4, 5], to the well-known and currently widely-used 3GPP Architecture, found in [6, 7].

Network operators using 3GPP-based technologies, requires to specify their plans in the form of rules and actions. 3GPP defines a 5-tuple set of classifiers: source IP address,

destination IP address, source port number, destination port number, and protocol ID of the protocol above IP to identify a flow. However, many Network Operators are also extending them, to other classifications such as protocol/application, location and device.

A verification modeling language will be used to specify a number of current and future use-cases scenarios found in the public domain. Those use-cases will be verified against their expected properties expressed as Linear Temporal Logic (LTL) formulas.

The outcome of this Thesis is to prove that Model Checking is a feasible approach to specify and verify rule-based Internet plans; bearing in mind that plans with many rules would require much more CPU resources. Other methods or approaches to specify and verify such those plans are out-of-scope of this research.

1.5 Justification

World′s population is expected to exceed 7.6 billion early in 2020, up from the current 7.2 billion; while the number of internet-connected devices is expected to double in 5 years, from 25 billion, now in 2015, to 50 billion by 2020 [8].

With this proliferation of Internet-connected devices, essentially the number of Internet-connected devices in the world has grown faster than the number of people in the World, network operators are trying to differentiate themselves by offering new ways of pricing models, beyond the usual vectors of volume and bandwidth.

In most industries, customers have a choice in selecting the expected quality, which is usually tied to a well-known categorization. In the airlines industry, economy or business tickets can be bought. In the Hotel industry, the quality is dictated by the number of stars.

Particularly in the world of fast-food restaurants, we have seen an evolution in the last 5 years. At the very beginning, the meals were fixed and simple. Now the meal options

got more varied and, in some fast-food chains, it is even possible to choose all the *features* in your meal: which kind of bread, which kind of ham, which dressing, and so on [9].

On the other hand, in the world of Retail, we see examples of a new micro payment economy. Rather than long term commitments, people are making compulsives small purchases. As an example we can see people renting a house for weekly basis, or hourly renting a car, or even purchasing certain songs and not entire albums.

In the world of broadband data, network operators are also rethinking the notion of the long term contracts and fixed tiered plans. Network Operators are basically moving from unlimited or volume-based models to value-added data offerings to basically offer price transparency and access to certain applications for specified amounts of time, at affordable rates [2]. For example, Vox Telecom, a DSL provider in South Africa, partnered with a local retailer around the launch of the game "Call of Duty: Black Ops II" to provide 40GB of "gamer optimized" ADSL Bandwidth to customers who redeemed a voucher included with the purchase of the game [10]. Similarly, China Mobile currently offers "Lite Data Service Plans" at reduced prices, where heavy-data-usage applications, such as Peer-To-Peer applications, are blocked, in order to rationalize the use of the data plans [11].

Given that Network Operators are moving to value-added Internet plans, the spectrum of possible combinations of rules to form plans is huge. Consequently, Network Operators will require tools to validate the rules within a plan, and most importantly, an automated system that verifies the rule-based plans against its specification and that finds any possible conflict among the rules.

In that sense, Model Checking can be used to exhaustively and automatically establish whether a given rule-based Internet Plan, satisfies a predefined set of properties (formal specification) and determine whether or not there is a conflict within the rules.

1.6 Hypothesis

This thesis will intend to prove the hypotheses listed below:

- **H1:** Rule-based Internet plans can be abstracted and specified in a verification modeling language.
- **H2:** Model Checking can be used to verify whether the model meets the specification of the plan and to detect conflicts between the rules within the plans.

1.7 Contribution of the Thesis

The main contributions of this Thesis are:

- Provide a framework to specify rule-based Internet plans via a verification modeling language. This model will let specify the rules-based Internet Plans unambiguously.
- Provide real and future use-cases scenarios based on Internet Plans offered in the public domain. These scenarios will let us verify our proposed model with their properties.
- Provide a set of LTL formulas to exhaustively and automatically verify rule-based Internet plans. These LTL formulas will let us verify the identified scenarios and any other possible plan specified with our proposed model.
- Prove that Model Checking is a powerful approach to detect conflicts between the rules within Internet plans.

1.8 Organization of the Thesis

The Thesis is organized as follows:

1. In Chapter 1, the introduction to the thesis is provided which also includes our motivation and defines its scope.

2. In Chapter 2, we review the State of the Art.

3. In Chapter 3, we introduce our model to specify the Rule-based Internet Plans, and the LTL to be used to verify them.

4. In Chapter 4, we introduce MexCom, a not real Mexican Mobile Operator company offering use-cases we found in the Public Domain.

5. In Chapter 5, we validate the use-cases described in the previous chapter with the model studied in chapter 3. The results are analyzed and discussed at the end of the chapter.

6. We conclude in Chapter 6 by discussing further improvements and future work.

In the next chapter, the evolution of Internet Plans is reviewed and Model Checking is explained and discussed as a widely-used approach in many other domains.

CHAPTER 2

Literature Review

2.1 Mobile Broadband services

2.1.1 Evolution

There seems to be a consensus on how mobile broadband services have evolved and, more importantly, in which direction they go. In 2012, Kimbler and Taylor characterized the evolution of strategies for mobile broadband data services in three phases as follows [2]:

- Phase One: Unlimited data plans.
- Phase Two: Volume-based charging.
- Phase Three: Value-added mobile broadband services.

Ezziane was not wrong when in 2005 predicted that the traditional flat rate will no longer be valid. Instead, accounting and billing for emerging 3G services will be content-based and usage-based [12].

In 2013, Sen et al, introduced the term Smart Data Pricing (SPD), described as a broad set of ideas and principles that go beyond the traditional at-rate or byte-counting models. Such SDP models can include any of the following mechanisms: Time-based, location-based, application-based, quota-aware content distribution [1].

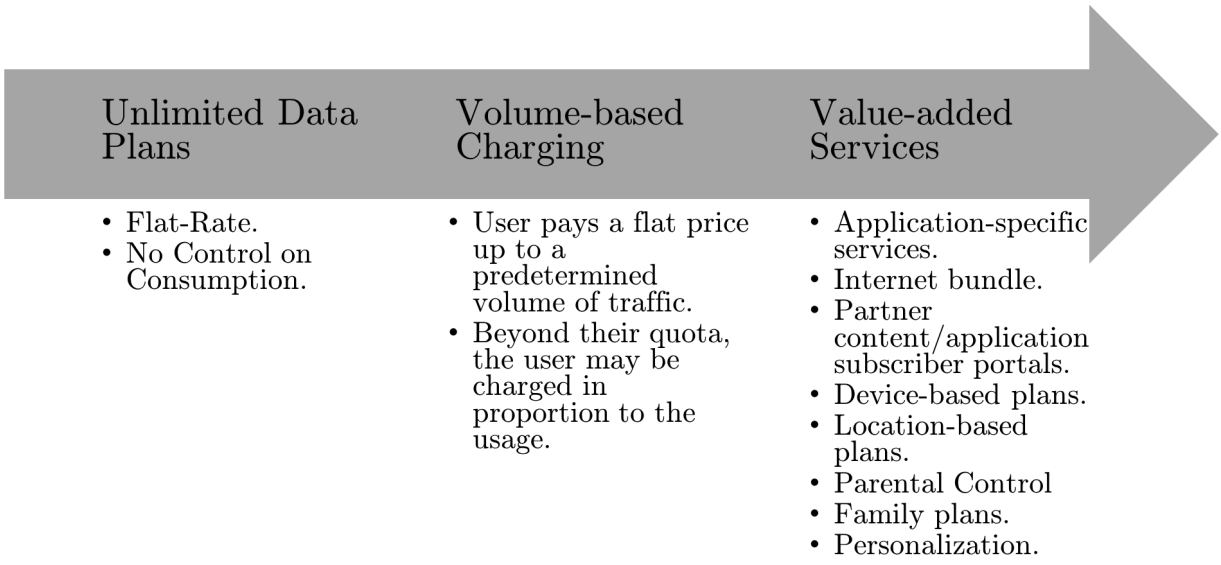

Figure 2.1: Evolution of the Mobile Broadband services [2]

Later in 2013, Hart and Brown defined similar services to be enabled on an LTE network: Fair usage, Time of Day, Parental Control, Shared Data Plans, Turbo Boost, Bundling of Popular Services (e.g. Twitter, Facebook, etc.) and HD Voice and Video [13].

Similarly to the authors mentioned above, telecommunication companies have also published their research regarding this topic. For example in 2012, Elitecore Technologies listed a number of emerging use cases, such as: turbo boost, family plans, application-based & device-based plans, service personalization, etc [14]. While Sandvine, which offers solutions to communication service providers (CSPs), talks about: Application-based service tiers and bolt-ons, roaming packages, shared data plans, sponsored connectivity and third-party bundles [15]. Nowadays, Alcatel Lucent and Allot Communications, other telecommunication companies, are also offering solutions about application-based charging, turbo boost and Parental Control [16, 17].

It is clear that network operators have already started to create value added data services and specialized packages that can be sold on top of or instead of the traditional tiered volume-based data packages. This approach is allowing operators to create incremental data revenues and effectively monetize their data traffic in an smarter way [2].

2.1.2 Examples of value-added services

The table 2.1 below, provides an overview of some of the most common volume and value added data services available today. The real advantage of value-based data plans is that they are designed to meet the wants and needs of the end-users and customers [14].

Pricing Policy	**Description**	**Example of Global Telco's**
Access Network	Policy based on the access network. e.g. 2G, 3G, 4G, WiFi, WiMax.	AT&T, Verizon, T-Mobile (USA), Movistar (Colombia), MTS(Russia), Aircel, Celcom, Singtel.
Tiered Plans	Different Quota-based plans with FUP and overage.	AT&T, Verizon, T-Mobile (USA), Celcom, Movistar (Colombia), Singtel, Aircel, Vodafone (India).
Device-based plans	Plans based on the accessed device. e.g. iPad, iPhone, Smart TV, Laptop	AT&T, Verizon, T-Mobile (USA),Celcom, Movistar (Colombia), MTS(Russia), Aircel, Singtel.
Parner-based plans	Partner-based plans enable operators offer content at subsidized rate.	AT&T, Verizon, T-Mobile (USA), Maxis, Celcom, SingTel, Aircel, Vodafone (India).
Parner-based plans	Partner-based plans enable operators offer content at subsidized rate.	AT&T, Verizon, T-Mobile (USA), Maxis, Celcom, SingTel, Aircel, Vodafone (India).
Time-based plans	Time of day, hourly plans, Seasonal plans.	AT&T, Movistar (Colombia), MTS (Russia), Maxis, Celcom.
Service Personalization	Subscribers can self select from webselfcare.	AT&T, Verizon, T-Mobile (USA), Movistar (Colombia), MTS (Russia), Maxis, SingTel, Aircel.
Parental Control	Quota management or Application control for child account.	AT&T, Verizon, Verizon, Maxis.
Application/Service	Application specific charges, e.g. Social networking bundles, emails, etc.	Movistar (Colombia), Maxis, Celcom, SingTel.
Family-based Plans	Data plans with common data pool and sharing	T-Mobile (USA).

Table 2.1: Key use-cases listed by *Elitecore Technologies* in [14].

2.2 Broadband communications technologies

Conflicts among policy rules within a particular plan are the core focus of this thesis. However, the interaction between technologies and the offered services should be well understood. Essentially, all plans are built on top of technologies.

In order to offer value-added services, network operators need powerful tools to inspect, measure and analyze data traffic in more sophisticated ways that they have done in the past.

Several leading vendors including Comverse, Sandvine, Openet, Allot, Huawei and Amdocs have seized this market opportunity and offer comprehensive data traffic management solutions that combine:

- Policy Control (PCRF/PCEF).
- Deep Packet Inspection (DPI).
- On-line Data Rating and Charging.
- Bandwidth Control and QoS Management.
- Content Filtering, Caching and Compression.
- Business Intelligence and Analytics.

In the past, many operators used technologies like policy management or DPI to solve specific problems with network congestion, assure fair usage and satisfy regulatory requirements, but they rarely used them to generate new revenue streams [2].

Operators are showing early interest in using a combination of DPI and policy management to offer value-added services on top of their existing basic packages.

2.3 Model Checking

A model can be seen as a simplification of reality. We build models to better understand things, and most importantly, to describe part of a system from a particular perspective. Additionally, a model let us describe unambiguously the system itself or its properties. In this section, we focus on modeling languages and how can we prove or disprove the correctness of it with regards to a specification.

A modeling language is any artificial language that can be used to express information or knowledge or systems in a structure that is defined by a consistent set of rules. One of the key benefits of using a modeling language, is that it will let us verify it formally against a given specification.

Formal verification is the process of applying a manual or automatic formal technique for establishing whether a given system satisfies a given property or behaves in accordance to some abstract description (formal specification) of the system. This verification is done by software tools as model checkers. A model checker thoroughly explores the state space to decide whether the system satisfies the set of properties.

Figure 2.2 below, illustrates the overall process of Model Checking [18]. In a first step, which is called modeling, the system description is converted into the system model. The requirements have to be manually formalized because they are mostly given in natural language. The result of this formalization is the formal specification given as formulas in a temporal logic such as LTL (Linear Temporal Logic).

As shown below, the model and the specification are inputs given to the model checker. The model checker uses an exhaustive search over all reachable states of the model to check whether the model satisfies the formula. In the end, it returns a result. The result may be that the model satisfies the formula or that the model does not satisfy the formula.

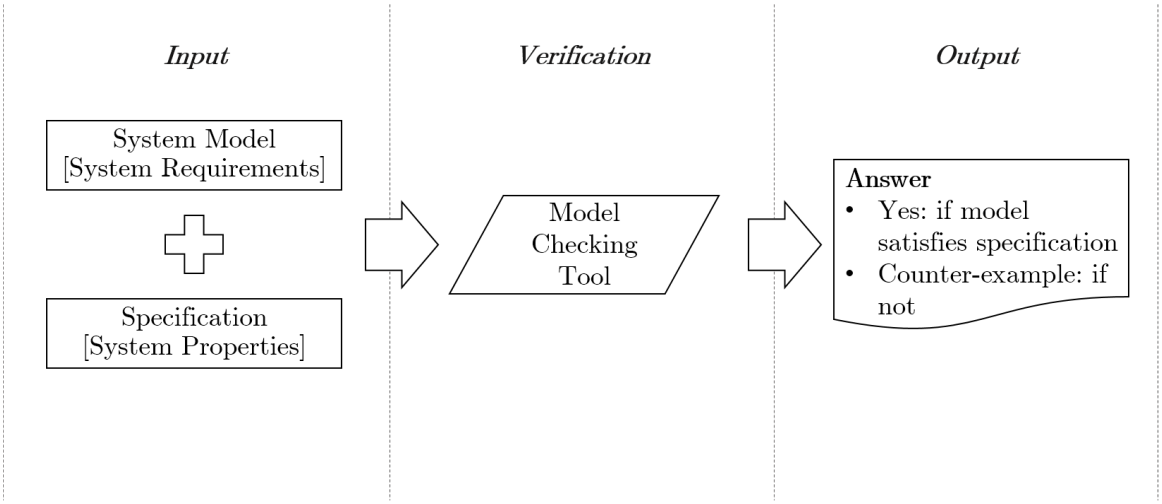

Figure 2.2: Model checking process

2.3.1 Spin Model Checker and PROMELA

Spin and most of its predecessors are implementations of standard automata-based reachability analyzers, with primary emphasis on performance. It provides an efficient implementation of a classic reachability analyzer with a firm and well-understood theory for LTL model checking [19] - or also known as the VardiWolper framework for automata-theoretic verification [20]. It supports the use of LTL (linear temporal logic) for the specification of correctness properties.

Spin accepts specifications written in a meta-language named Promela (Process Meta Language). Promela is a verification modeling language introduced by Gerard J. Holzmann. The language allows for the dynamic creation of concurrent processes to model, for example, concurrent or distributed systems. The three main types of objects that can be manipulated are:

- processes,
- channels, and
- variables.

Spin represents the system as a finite state machine. It visits each reachable state explicitly using Nested DFS and using partial order reduction. If a property is specified in LTL, Spin will first convert the LTL formula into an equivalent never claim, using the procedure outlined in [21]. Formally, the never claim specifies a Buchi automaton (a specific type of omega automaton) and the acceptance conditions from this automaton are evaluated on the global systems execution graph [19]. The efficiency of the Spin system comes from its on-the-fly procedure for performing this check that requires only small amounts of memory to be consumed for each state that is reached. The details of this so-called nested depth-first search algorithm are documented in [22].

Data Types

In Promela, there are seven predefined integer data types: bit, bool, byte, pid, short, int, and unsigned . There are also constructors for user-defined data types, and there is a separate predefined data type for message passing channels. Variables of type bit and bool are stored in a single bit of memory, which means that they can hold only binary, or boolean values.

Typedef declarations can be used to introduce user-defined data types. User-defined data types can use any predefined integer data types.

2.3.2 Linear Temporal Logic - LTL

Spin, as any other model checker, needs a number of system properties to verify the System Model (expressed in Promela). Without the system properties, the model checker cannot really tell whether or not the System Model is correct. In Promela, Linear Temporal Logic is used for specifying the correctness of requirements.

Linear temporal logic (LTL) is a modal temporal logic with modalities referring to time. In LTL, one can encode formulas about the future of paths, e.g. a condition will eventually be true, a condition will be true until another fact becomes true, etc. It is a fragment of the more complex CTL*, which additionally allows branching time and quantifiers.

LTL is built up from a finite set of propositional variables, logical operators (i.e. $\neg$, $\wedge$, $\vee$, $\Longrightarrow$, $\Longleftrightarrow$, *true* and *false*), and four additional temporal modal operators.

LTL Formula	**Description**	**Semantics Meaning**
$\Box\ \phi$	Always	ϕ is satisfied at every state.
$\Diamond\ \phi$	Eventually	ϕ is satisfied at some state.
ψ U ϕ	Until	ψ has to hold at least until ϕ, which holds at the current or a future position.
X ϕ	Next	ϕ has to hold at the next state.

Table 2.2: Semantics for the temporal operators

An important taxonomy of properties, as given below, is found in many books and papers [23]. Safety and liveness properties described below, are of our particular interest; as most of our specifications are safety and liveness properties.

Safety properties state that for all computations of the system, and for all instances of time, some property will invariantly hold. Colloquially, Safety properties assert that something "bad"never happens. For example, a safety property could be defined as: *Facebook traffic will never be blocked.*

Liveness properties state that some desired state of the system can eventually be reached. Colloquially, a Liveness property makes sure that something "good"eventually happens. An example of a liveness property could be: *If any other traffic is generated, it will eventually be blocked.*

Persistence properties are related to the stabilization of certain properties. In general, a persistence property describes that for all possible computations, there is a point of time when a certain property will always hold afterwards. For example: there is a point where rule A is always enabled.

Fairness properties state that some property will hold infinitely often. For example: No process is ignored infinitely often by an O.S.

2.4 Related Work

Model checking, or property checking, has been widely used in many other domains to exhaustively and automatically check whether a given model meets a given specification. Our primary interest is the use of Promela as the Model Checking tool to detect conflicts, via LTL formulas, in rules-based plans.

One of the main contributions for this Thesis is the research done by Zhiping Duan in 2003 [24]. In his work, he provides an approach to automatically detect feature interactions (i.e. conflicts) in telecommunications systems, which are not essentially rules-based. Even though, Duan uses LTL and FOL (First-Order-Logic) to prove their systems, he proposed an implementation of a FOL prover in λProlog. We think it is more intuitively to use Promela for our particular domain, as Promela is a verification modeling language which directly allows LTL formulas for its verification.

In 2007, Antoniou, et al, integrate several aspects of policy specification languages (i.e. rule-based reasoning), in a common framework. One key contribution from his research to our Thesis, is the use of priorities to resolve conflict among rules. In his research, the implementation of a conflict checker was not addressed.

Between 2010 and 2011, several requirements frameworks for business process compliance management have been proposed. In [25, 26], the authors formulate requirements for compliance rules. The requirements address the issues of lifetime compliance. The focus is also on the requirements to languages expressing compliance rules, on the rules priority, and on validation of process models against rules during design time and runtime.

In 2012, Gawanmeh, et al, proposed a novel algorithm for detecting conflicts in firewall rules [27]. Even though his novel approach, using domain restriction, is interesting; its proposed algorithm doesn't consider rules with multiple types of conditions.

The review of the related work above, illustrates that the problem of conflicts among rules exists in different domains and, more importantly, Model-Checking is a proven approach for detecting the conflicts efficiently. In this Thesis, we will use Promela to detect conflicts in a particular area of telecommunications: rule-based Internet Plans.

2.5 Summary of the chapter

The literature review is summarized below:

- Network Operators are basically moving from unlimited or volume-based models to value-added data offerings. These value-added services comprise a number of custom rules to form a Plan.
- Because all of these services are built on top of technologies, there are a number of technical, commercial, regulatory or integration impacts, that needs to be considered when designing these value-added services.
- Model Checking can perfectly be used to exhaustively and automatically verify whether a model meets a given specification. In our case the rule-based plans will be modeled using Promela and the specification will be defined in LTL.
- The problem of conflicts among rules has been studied in many other domains and, more importantly, Model-Checking is a proven approach for detecting the conflicts efficiently.
- In this Thesis, Promela will be used to detect conflicts in a particular area of telecommunications: rule-based Plans.

In the next chapter, we will introduce the model proposed for specifying the rule-based plans and will expand on the LTL properties to verify the proposed model.

CHAPTER 3

Promela Model and LTL Properties

The purpose of this section is to introduce the model proposed for specifying the rule-based plans and to expand on the LTL properties to verify the model.

3.1 Rule-based Plans Model

3.1.1 Overview

The key element to model is the specification of the rule-based plans. For that purpose, we declared a number of user-defined data types (i.e. typedef data structures), which are shown in a typedef diagram below in figure 3.1.

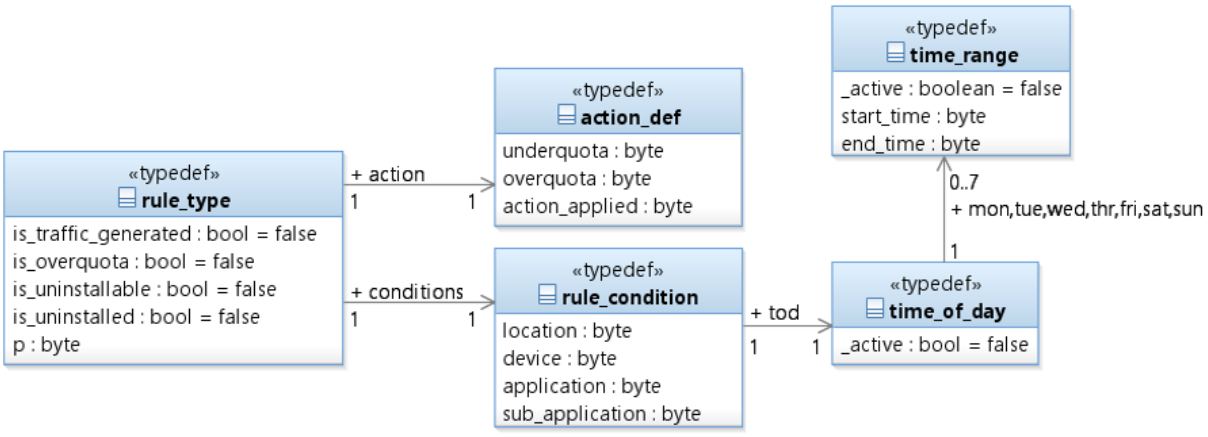

Figure 3.1: Typedef diagram

Rule Type

This data type allows us to specify a rule with its attributes described in table 3.1 below. Additionally, each rule, declares one *action_ def* variable and one *rule_ conditions* variable, described respectively in tables 3.2 and 3.3 below.

Attribute	Type	Description	Example
is_ generated_ traffic	bool	If traffic is generated for this particular rule, *is_ generated_ traffic* attribute is set to true.	true | false
is_ overquota	bool	If this attribute is set to true, the current state of the rule is *overquota*. If it is set to false, then it is *underquota*.	true | false
is_ uninstallable	bool	If this is set to true, this rule will eventually be uninstalled when its quota is reached.	true | false
uninstalled	bool	If this is set to true, this rule has been uninstalled and it is no longer active.	true | false
p	byte	*p* stands for the *priority* of the rule. Valid values are from 0 to 255.	5

Table 3.1: *Rule_ Type* Attribute definitions.

Action Def

This data type allows us to specify the actions to be applied for each state of the rule.

Attribute	Type	Description	Example
underquota	byte	The action to be applied when the rule is in the underquota state.	0:block 1:allow
overquota	byte	The action to be applied when the rule is in the overquota state.	0:block 1:allow
action_ applied	byte	To store the current action applied.	0:block 1:allow

Table 3.2: *Action_ Def* Attribute definitions.

Rule Conditions

This data type allows us to specify the conditions for the rule described in table 3.3 below. Additionally, each rule condition, declares one *time_ of_ date* variable for specifying the different time conditions for each day. Whenever the conditions are met, the actions specified are performed.

Attribute	**Type**	**Description**	**Example**
location	byte	The location condition for applying the rule actions.	0:local 1:roaming
device	byte	The device condition for applying the rule actions.	0:Mobile 1:Tablet 2:Smart TV
application	byte	The application condition for applying the rule actions.	0:Social Networks 1:Video Streaming
subapplication	byte	The subapplication condition for applying the rule actions.	For Social Networks 0:Facebook 1:Twitter 2:Instagram

Table 3.3: *Rule_ Conditions* Attribute definitions.

Time of Day and Time Ranges conditions

As shown in table 3.4 below, the time_of_day data type allows us to specify the time conditions for each day.

Attribute	**Type**	**Description**	**Example**
_ active	bool	Determines whether or not there is a time condition defined.	true | false
mon, tue, wed, thr, fri, sat, sun	time_range	Time conditions for each day.	See table 3.5 below

Table 3.4: *Time_ of_ day* Attribute definitions.

As shown below in table 3.5, each time-day condition compromises a start time and end time.

Attribute	Type	Description	Example
_active	bool	Determines whether or not there is a time condition defined for this particular day.	true | false
start_time	byte	Start time condition.	08
end_time	byte	End time condition.	20

Table 3.5: *Time Range* Attribute definitions.

3.1.2 Process Definition

Two processes, explained in detailed below, have been defined to model rule-based plans: Priority Checker and OverQuota.

Priority Checker

The Priority Checker process randomly chooses two rules, and executes the rule with higher priority. Listing 3.1 below, shows an informal description of the PriorityChecker process.

The process starts in line 2, by randomly selecting two-rule indexes "x" and "y", and initializing a bool local variable local_overquota to store whether or not the rule is over-quota.

In line 10 below, the PriorityChecker process verifies that the two random indexes are different and that the rule with the index "x" has not been uninstalled yet by the OverQuota process. If the verification passes, then the rule "x" is flagged with the traffic generated attribute (line 12). If not, no traffic is generated for that rule (line 28).

From line 11 to 24 below, the PriorityChecker process atomically checks if the conditions of the rules "x" and "y" are different, or if rule "x" has a higher priority assigned.

If any of those scenarios are met, then the underquota action configured for rule "x" is applied and the process waits for the overquota state (lines 15 and 20). If not, no action is applied for rule "x".

Finally, from line 25 to 28 below, the overquota action configured for rule "x" is applied if the local overquota is true. If not, no action is applied for rule "x".

Listing 3.1: PriorityChecker high-level algorithm

```
1   proctype PriorityChecker () {
2     // Pick two random rules x and y.
3     int x;
4     select (x : 0 .. (num_R-1));
5     int y;
6     select (y : 0 .. (num_R-1));
7
8     bool local_overquota=false;
9
10    if (x !=y and !Rules[x].is_uninstalled) then
11      atomic{
12        Rules[x].is_traffic_generated = true;
13        if (conditions of Rules[x] != conditions of Rules[y]) then
14          Rules[x].action.action_applied = (Rules[x].action.underquota);
15          (Rules[x].is_overquota == true);
16          local_overquota = true;
17        else
18          if (Rules[x].p > Rules[y].p) then
19            Rules[x].action.action_applied = (Rules[x].action.underquota);
20            (Rules[x].is_overquota == true);
21            local_overquota = true;
22          else
23            Rules[x].action.action_applied=NO_ACTION;
24      }
25      if local_overquota then
26        Rules[x].action.action_applied = (Rules[x].action.overquota);
27      else
28        Rules[x].action.action_applied=NO_ACTION;
29    else
30      Rules[x].is_traffic_generated = false;
31  }
```

Refer to the Appendix A to review the Promela Code for the PriorityChecker process, detailed in Listing A.1.

OverQuota

The OverQuota process iterates through the array of defined rules, and change each state from underquota to overquota over time. Additionally, if the rule is flagged as uninstallable, the is_uninstallable attribute is set to true as well.

Refer to the Appendix A to review the Promela Code for the OverQuota process, detailed in Listing A.2.

3.1.3 Example of rule-based plans

A mobile operator, would like to offer free local facebook to its subscribers. The following Promela code would represent that rule in our model.

Listing 3.2: Free Facebook Rule.

```
1    rule_type freeFB
2    freeFB.conditions.application = 0; //Facebook
3    freeFB.conditions.location = 0; //local
4    freeFB.action.underquota = 1;
5    freeFB.action.overquota = 1;
6    freeFB.p = 1;
```

Additionally, the same mobile operator may also need to offer a limited amount of bytes that could be consumed in a monthly basis. As shown in the listing 3.3 below, traffic is allowed (1) for any type of traffic generated when underquota. Once overquota, the action is blocked (0). 255 is used as the wildcard for specifying any application and any location.

Listing 3.3: Free Navigation Rule.

```
1    rule_type navRule
2    navRule.conditions.application = 255;
3    navRule.conditions.location = 255;
4    navRule.action.underquota = 1;
5    navRule.action.overquota = 0;
6    navRule.p = 0;
```

3.2 LTL Properties

3.2.1 Macros Definitions

As seen before, Linear-Temporal-Logic formulas will be used to verify if the model satisfies the specification. If it is not satisfied, the system requirements, may not be well defined.

Listing 3.4 below, enumerates the macros we have defined for each rule to be used in the LTL formulas to prove the predefined System properties.

Listing 3.4: Macros for each rule

```
1   #define generated_rule_X (Rules[X].is_traffic_generated)
2   #define allowed_rule_X (Rules[X].action.actionApplied==ALLOWED_ACTION)
3   #define blocked_rule_X (Rules[X].action.actionApplied==BLOCKED_ACTION)
4   #define overquota_rule_X (Rules[X].is_overquota)
```

A further description of each macro is presented in table 3.6 below.

Macro	**Description**	**Expression**
generated_rule_{index}	Whenever the Priority Checker process generates traffic for this particular rule, this macro will be evaluated to true.	Rules[{index}].is_traffic_generated
allowed_rule_{index}	Whenever the Priority Checker process applies the allowed action ("1"), this macro evaluates to true.	Rules[{index}].action.action_applied== ALLOWED_ACTION
blocked_rule_{index}	Whenever the Priority Checker process applies the blocked action ("0"), this macro evaluates to true.	Rules[{index}].action.action_applied== BLOCKED_ACTION
overquota_rule_{index}	Whenever the Overquota process sets the current state of the rule as overquota, this macro evaluates to true.	Rules[{index}].is_overquota

Table 3.6: Macros Definitions

3.2.2 Properties Definitions

Based on the macros described above, the following LTL properties will be used to verify the different rule-based plans.

Id	Property Description	LTL	Classification
PT01	Rule that always allows or blocks the traffic satisfied by the rule condition.	□(generated_rule_{index} ⟹ [allowed|blocked]_rule_{index})	Safety
PT02	Rule that allows the traffic satisfied by the rule condition and then eventually blocks it when all the quota is consumed.	□((generated_rule_{index} ⟹ allowed_rule_{index}) ⟹ ◊blocked_rule_{index})	Liveness

Table 3.7: Properties Definitions

These two properties are very useful and, when used combined, will let us prove a number of rule-based plans. The LTL for PT01 above, can be translated to: *Every* time traffic is generated for rule X, then its traffic will be allowed. While the LTL for PT02 above, can be translated to: *Every* time traffic is generated for rule X, then its traffic will be allowed and then will *eventually* be blocked.

In addition to the properties described above, we defined other properties considering the state of the rule, or an interaction with other rules, using the until operator (U), as shown in table 3.8 below.

Id	Property Description	LTL	Classification
PT03	Consider PT01 only until some new state is reached.	□((generated_rule_{index} ⟹ allowed_rule_{index}) U overquota_rule_{index}) — OR — □((generated_rule_{index} ⟹ allowed_rule_{index}) U !overquota_rule_{other_index})	Liveness
PT04	Consider PT02 only until some new state is reached.	□(((generated_rule_{index} ⟹ allowed_rule_{index}) ⟹ ◊blocked_rule_{index}) U overquota_rule_{index}) — OR — □(((generated_rule_{index} ⟹ allowed_rule_{index}) ⟹ ◊blocked_rule_{index}) U !overquota_rule_{other_index})	Liveness

Table 3.8: Extended Properties Definitions

The LTL for PT03 above, can be translated to: *Every* time traffic is generated for rule X, then its traffic will be allowed *until* rule X reaches its quota (or until any other predefined rule does). While the LTL for PT04 above, can be translated to: *Every* time traffic is generated for rule X, then its traffic will be allowed and then will *eventually* be blocked *until* rule X reaches its quota (or until any other predefined rule does).

Essentially, the until operator allow other rule(s) to take precedence only after some state of that rule, or any other predefined rule, changes.

3.2.3 Conflict verification through properties

The LTL formulas described above, provide a powerful set of properties that will be used to prove a number of desired behaviors on each rule-based plan. In addition to the verification of each property, there is an implicit property, not so intuitively, which is also verified when the properties are combined; this is the conflict-free property

The always operator ($\Box$) surrounding each property guarantees that the property must be satisfied in all the possible cases – regardless the pair of rules chosen by the Priority-Checker process. That is, the Model Checker by definition, will explore every possible combination of pair of rules, to make sure that the property is satisfied.

For instance, if a wrong priority were assigned to a rule and the traffic that meets the condition of the rule was generated, the model checker will eventually pick the other rule with similar conditions but with a higher priority; in which case the action of the rule with the wrong priority will not be applied. This scenario would prevent the Model Checker to verify the system with the given LTL property. In other words, a conflict would have been detected by the Model Checker and the verification would have failed.

3.3 Summary of the chapter

In this chapter, our Promela model was introduced to specify the rule-based Internet plans. The proposed model allows to specify a number of conditions such as application or device used, current location, time of day; and actions to be applied whenever there is quota or not. Additionally, four LTL properties were introduced to verify the proposed model.

In the next chapter, we will present the rule-based Internet plans, which will be verified using the Promela Model described in this chapter.

CHAPTER 4

Case Study: MexCom Use Cases

In this chapter, we present the rule-based plans, which will be verified using the Promela Model described in the previous chapter.

We begin this chapter by introducing "MexCom", a not real company we created to illustrate the number of use-cases described below. We build the use-cases, based on the rules-based plans found in the Public Domain. We also considered some not-yet-existing rules; which, based on the literature review, are quite reasonable to predict.

In the next chapter, MexCom plans will be modeled using our proposed Promela Model and verified through the LTL formulas previously studied.

4.1 MexCom Overview

MexCom is a new Mobile Network Operator, which is eager to start operations in Mexico soon. MexCom will target low-income subscribers, who cannot afford an unlimited mobile plan.

The next sections describe the evolution of the service plans MexCom will offer to gain market share. It is important to mention that a plan will consist of one or more rules described below.

4.2 Core Rules

MexCom wants to start operations in Mexico by offering an initial set of monthly limited usage rules, which are listed below in table 4.1

Core Rule ID	Capacity	Monthly Cost (MXN)
CORE01	100MB	$75.00
CORE02	200MB	$100.00
CORE03	400MB	$150.00
CORE04	1GB	$350.00
CORE05	2GB	$600.00

Table 4.1: Core Limited Usage Rules.

To make the initial offer more attractive to its future subscribers, MexCom decides to offer certain commonly-used applications without charging their subscribers to their limited usage rule. This is known as Zero-Rated applications, offered at a very low-monthly cost.

Rule Bolt-on ID	Application(s)	Monthly Cost (MXN)
BOLT01	Unlimited Local Facebook	$10.00
BOLT02	Unlimited Local Twitter	$10.00
BOLT03	Unlimited Local Whatsapp	$10.00

Table 4.2: Bolt-on Rules

The bolt-on rules above, essentially allow subscribers to form a low-cost plan by acquiring a basic limited usage rule, with any bolt-on rule. For example, a subscriber can acquire a limited monthly usage plan of 100MB with unlimited access to Facebook for $85.00 MXN monthly (CORE01 + BOLT01).

4.3 Roaming Bolt-on Rules

MexCom now wants to offer roaming bolt-on rules to their current offer of limited usage rules, previously described.

Rule Bolt-on ID	Application(s)	Monthly Cost (MXN)
BOLT04	Unlimited Roaming Twitter	$40.00
BOLT05	Unlimited Roaming Whatsapp	$40.00
BOLT06	Unlimited Roaming Email	$40.00

Table 4.3: Roaming Bolt-on Rules

Roaming data rates are usually much more expensive than local traffic rates. Most of the mobile devices, run a number of background applications which constantly consume traffic without the subscriber noticing it. With roaming bolt-on rules shown in table 4.3 above, MexCom will allow subscribers to choose which applications shall only be allowed while roaming. This will prevent background applications from consuming part of a quota.

4.4 Family Rules

MexCom decides to extend their offer by including family plans. Family plans let a subscriber share a monthly quota between multiple family members or devices. A family plan consists of a quota, big enough to be shared amongst family members, listed in table 4.4, and individual bolt-on/constraint rules per device listed in table 4.5 below.

Family Rule ID	Capacity	Monthly Cost (MXN)
FAM01	10GB	$450.00
FAM02	15GB	$600.00
FAM03	20GB	$800.00

Table 4.4: Family Shared Rules

Family Rule Bolt-on/Constraint ID	Bolt-on/Constraint	Monthly Cost (MXN)
FAMC01	Unlimited Facebook	$10.00
FAMC02	Video Streaming HD Blocked	$0.00
FAMC03	Social Network Blocked	$0.00
FAMC04	Video Streaming Optimized	$50.00

Table 4.5: Family Shared Rules - Bolt-ons/Constraints

The above set of rules let a subscriber form its own family plan. For instance, the head of a family can acquire the FAM01 rule for all its family members; and additionally, acquire a constraint rule for blocking Social Networks to their kids device (FAMC03) or acquire a Video-Streaming-Optimization rule for an Smart TV device (FAMC04).

4.5 Parental-Control Rules

Parental-Control let a subscriber add a number of constraint rules to a rule-based plan. The constraints are usually applied during the day time and are released at night. This type of plans, prevent the subscriber to access certain applications during the day, for example during class hours.

Table 4.6 below, describes the rule constraints that can be added to the initial set of plans previously listed in the table 4.1, to form a Parental Control plan.

Rule Constraint ID	Traffic Condition	Time of Day	Action	Monthly Cost (MXN)
CON01	Social Networking	Weekdays 0800-1800	Block	$0.00
CON02	Video Streaming	Weekdays 0800-1800	Block	$0.00
CON03	Gaming	Weekdays	Block	$0.00

Table 4.6: Rule Constraints

These plans, let a head of a family, acquire a Limited Usage rule and on top of that, add a rule constraint to block a set of applications, during a predefined time of a day.

4.6 Limited Bolt-on Rules

Limited bolt-on rules are similar to the bolt-on rules seen before, except that they are not unlimited. That is, there is a usage limit. For example, a subscriber may purchase a plan based on the unlimited usage rules previously described in table 4.1, and only enhance it during a particular weekend, when an special sport event would take place.

In order to satisfy the need described above, MexCom decides to offer the limited bolt-on rules below, so subscribers can add them to their current plans at any time.

Limited Rule Bolt-on ID	**Limited Bolt-on**	**Cost (MXN)**
LADD01	1GB Video Streaming Optimized	$15.00
LADD02	2GB Video Streaming Optimized	$25.00
LADD03	1GB Gaming Optimized	$10.00
LADD04	1GB Free Usage	$15.00

Table 4.7: Limited Bolt-on Rules

4.7 Summary of the chapter

In this chapter, we introduced MexCom and presented the rule-based plans based on real and future use-cases scenarios found in the public domain.

The spectrum of possible combinations of rules to form a plan is huge. We will see in the next chapter the significance of determining the right priority for each of the above rules, and most importantly, the value of an automated system that verifies the rule-based plans against its specification and that finds any possible conflict among the rules.

CHAPTER 5

Case Study: Detecting Conflicts on MexCom Use Cases

In this chapter, we verify the the use-cases studied in the previous chapter using our Promela model introduced in Chapter 3. The main purpose of this chapter is to prove the hypotheses introduced in chapter 1:

- **H1:** Rule-based Internet plans can be abstracted and specified in a verification modeling language.
- **H2:** Model Checking can be used to verify whether the model meets the specification of the plan and to detect conflicts between the rules within the plans.

In the sections below, each use-case is modeled and verified using our proposed Promela Model.

5.1 Core Rules

5.1.1 Promela Model

The model of the Core Rules and unlimited bolt-ons is shown in Listing 5.28. Given that a plan could consist of one or more rule instances, the worst case scenario was modeled; that is, one instance of each rule was selected to form the plan below.

Listing 5.1: Core-Rules-based Plan Model

```
1   // CoreRule
2   Rules[0].conditions.application = 255;
3   Rules[0].conditions.location = 255;
4   Rules[0].action.underquota = 1; //Allow Traffic
5   Rules[0].action.overquota = 0; //Block Traffic
6   Rules[0].p = 0;
7
8   // Unlimited local Facebook Bolt-on
9   Rules[1].conditions.application = 0;
10  Rules[1].conditions.location = 0;
11  Rules[1].action.underquota = 1; //Allow Traffic
12  Rules[1].action.overquota = 1; //Allow Traffic
13  Rules[1].p = 1;
14
15  // Unlimited local Twitter Bolt-on
16  Rules[2].conditions.application = 1;
17  Rules[2].conditions.location = 0;
18  Rules[2].action.underquota = 1; //Allow Traffic
19  Rules[2].action.overquota = 1; //Allow Traffic
20  Rules[2].p = 1;
21
22  // Unlimited local Whatsapp Bolt-on
23  Rules[3].conditions.application = 2;
24  Rules[3].conditions.location = 0;
25  Rules[3].action.underquota = 1; //Allow Traffic
26  Rules[3].action.overquota = 1; //Allow Traffic
27  Rules[3].p = 1;
```

5.1.2 System Properties

The key rule of this plan, is the Rule[0] above; it dictates how most of the traffic should be handled. Basically, we are specifying no condition for that rule and indicating to apply the block action when the quota is reached. Please note that the amount of bytes of the quota of the Rule[0] is irrelevant as it is assumed that eventually the quota will be consumed regardless of the quota allowance.

Additionally there are three more rules, each one to always allow a different application. As it has been discussed, these additional rules shall have a higher priority, so the

action to allow these particular applications (i.e. facebook, twitter and whatsapp), takes precedence over the Rule[0].

If any of the additional rules has a lower priority assigned, the Rule[0] will take precedence over that rule and consequently, the desired behavior of allowing that particular application will not be met.

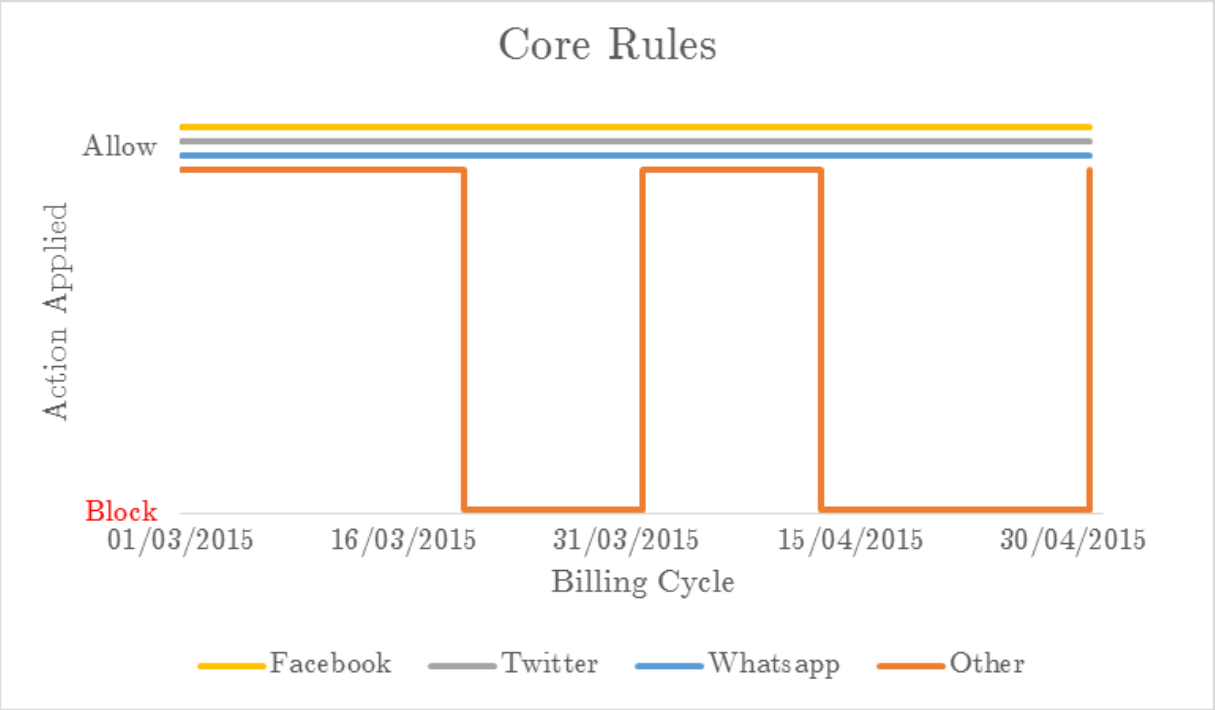

Figure 5.1: Core Rules - Linear Temporal Logic

Figure 5.1 above, represent the expected behavior of this rule-based plan. Facebook, Twitter and Whatapp traffic shall always be allowed; while the rest of the traffic will be allowed and then eventually be blocked.

In order to verify the expected behavior expressed above, the LTL below will be used. As seen in chapter 3, Linear-Temporal-Logic formulas are used to prove the property desired.

Listing 5.2: Core-Rules LTL

```
1   ltl Core_Rules_LTL {
2     [](
3         ((generated_rule_0 -> allowed_rule_0) -> <>blocked_rule_0) &&
4         (generated_rule_1->allowed_rule_1) &&
5         (generated_rule_2->allowed_rule_2) &&
6         (generated_rule_3->allowed_rule_3)
7       )
8   }
```

Listing 5.2 above can be translated to natural language as *every* time traffic that meets the conditions of rules number 1, 2 and 3 is generated, then it will be allowed. Additionally, *every* time traffic that meets the condition of rule number 0 is generated, then it will be allowed and *eventually* will be blocked.

5.1.3 SPIN Verification

At this point we have intuitively specified the first rule-based plan with our Promela model and we have determined which LTL formulas will be used to verify the model. Listing 5.3 below, presents the result obtained in Spin, with 0 errors shown in line 7.

Listing 5.3: Core-Rules Plan Verification

```
1  (Spin Version 6.3.2 -- 17 May 2014) + Partial Order Reduction
2  Full statespace search for:
3         never claim  + (Core_Rules_LTL)
4         assertion violations + (if within scope of claim)
5         acceptance cycles  + (fairness enabled)
6         invalid end states - (disabled by never claim)
7  State-vector 324 byte, depth reached 55, *** errors: 0 ***
8        60 states, stored
9        19 states, matched
10       79 transitions (= stored+matched)
11       48 atomic steps
12 hash conflicts: 0 (resolved)
13 Stats on memory usage (in Megabytes):
14    0.019 equivalent memory usage for states (stored*(State-vector + overhead))
15    0.143 actual memory usage for states
16   64.000 memory used for hash table (-w24)
17    0.069 memory used for DFS stack (-m2000)
18   64.195 total actual memory usage
```

Exceptions Verification

Our first results were expected, as the right priorities were assigned to each rule. For the next test, the priority of the key rule will be increased from 0 to 2, as shown below:

Listing 5.4: Increased priority of the Core Rule

```
1    // CoreRule
2    Rules[0].conditions.application = 255;
3    Rules[0].conditions.location = 255;
4    Rules[0].action.underquota = 1; //Allow Traffic
5    Rules[0].action.overquota = 0; //Block Traffic
6    Rules[0].p = 2;
7    ...
```

The updated model above, will not meet the LTL specification. The rules number 1, 2 and 3 will not be executed, because facebook, whatsapp and twitter traffic will be handled by the rule 0, which has a higher priority assigned. SPIN model basically identifies that when facebook traffic is generated, it is not always allowed.

Listing 5.5: Invalid Core-Rules Plan - Verification

```
1  (Spin Version 6.3.2 -- 17 May 2014) + Partial Order Reduction
2  Warning: Search not completed
3  Full statespace search for:
4          never claim  + (Core_Rules_LTL)
5          assertion violations + (if within scope of claim)
6          acceptance cycles  + (fairness enabled)
7          invalid end states - (disabled by never claim)
8  State-vector 324 byte, depth reached 57, *** errors: 1 ***
9        162 states, stored (244 visited)
10       100 states, matched
11       344 transitions (= visited+matched)
12       202 atomic steps
13 hash conflicts: 0 (resolved)
14 Stats on memory usage (in Megabytes):
15     0.053 equivalent memory usage for states (stored*(State-vector + overhead))
16     0.143 actual memory usage for states
17    64.000 memory used for hash table (-w24)
18     0.069 memory used for DFS stack (-m2000)
19    64.195 total actual memory usage
```

5.2 Roaming Bolt-on Rules

5.2.1 Promela Model

The model of the Roaming Bolt-on Rules is shown in Listing 5.6. Given that a plan could consist of one or more rule instances, the worst case scenario was modeled; that is, one instance of each rule was selected to form the plan below, including all the roaming bolt-on.

Listing 5.6: Core-Rules-based Roaming Plan Model

```
1   // CoreRule
2   Rules[0].conditions.application = 255;
3   Rules[0].conditions.location = 255;
4   Rules[0].action.underquota = 1; //Allow Traffic
5   Rules[0].action.overquota = 0; //Block Traffic
6   Rules[0].p = 0;
7
8   // Unlimited local Facebook Bolt-on
9   Rules[1].conditions.application = 0;
10  Rules[1].conditions.location = 0;
11  Rules[1].action.underquota = 1; //Allow Traffic
12  Rules[1].action.overquota = 1; //Allow Traffic
13  Rules[1].p = 1;
14
15  // Unlimited local Twitter Bolt-on
16  Rules[2].conditions.application = 1;
17  Rules[2].conditions.location = 0;
18  Rules[2].action.underquota = 1; //Allow Traffic
19  Rules[2].action.overquota = 1; //Allow Traffic
20  Rules[2].p = 1;
21
22  // Unlimited local Whatsapp Bolt-on
23  Rules[3].conditions.application = 2;
24  Rules[3].conditions.location = 0;
25  Rules[3].action.underquota = 1; //Allow Traffic
26  Rules[3].action.overquota = 1; //Allow Traffic
27  Rules[3].p = 1;
28
29  // Unlimited Email Roaming Bolt-on
30  Rules[4].conditions.application = 3;
31  Rules[4].conditions.location = 1;
32  Rules[4].action.underquota = 1;
33  Rules[4].action.overquota = 1;
```

```
34    Rules[4].p = 1;
35
36    // Unlimited Twitter Roaming Bolt-on
37    Rules[5].conditions.application = 1;
38    Rules[5].conditions.location = 1;
39    Rules[5].action.underquota = 1;
40    Rules[5].action.overquota = 1;
41    Rules[5].p = 1;
42
43    // Unlimited Whatsapp Roaming Bolt-on
44    Rules[6].conditions.application = 2;
45    Rules[6].conditions.location = 1;
46    Rules[6].action.underquota = 1;
47    Rules[6].action.overquota = 1;
48    Rules[6].p = 1;
```

As it was seen in the first test result, rules that have exclusive conditions may have the same priority as there is no conflict between them. However, and again, the Rule[0] above, shall have a higher priority assigned to meet the specification.

5.2.2 System Properties

As with the previous test, the key rule of this plan, is the Rule[0] above; it dictates how most of the traffic should be handled. Basically, we are specifying no condition for that rule and indicating to apply the block action when the quota is reached.

In addition to the local bolt-on rules seen before, there are three more roaming bolt-on rules, each one to always allow a different application while the subscriber is roaming. As it has been discussed, these additional rules shall have a higher priority, so the action to allow these particular applications (i.e. email, twitter and whatsapp roaming), takes precedence over the Rule[0].

If any of the bolt-on rules has a lower priority assigned, the Rule[0] will take precedence over that particular bolt-on rule, and consequently the desired behavior will not be met.

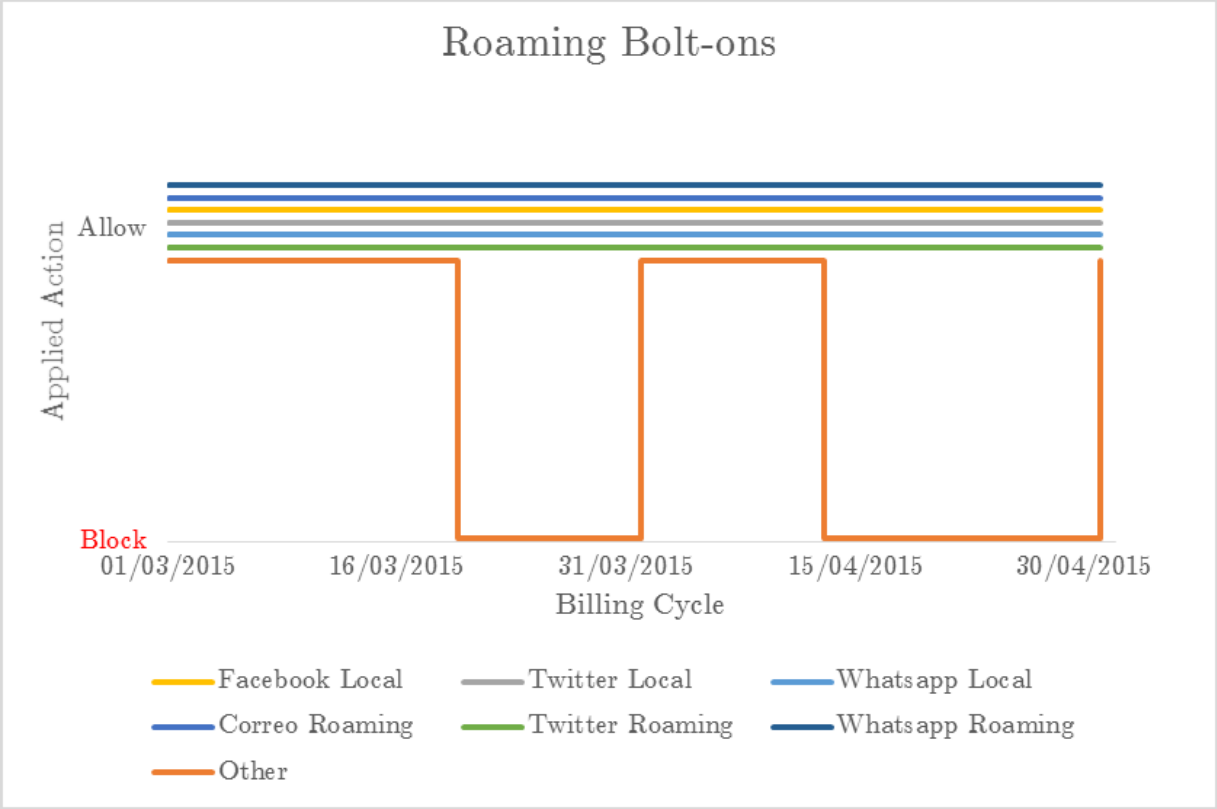

Figure 5.2: Roaming Rules - Linear Temporal Logic

Figure 5.2 above, represent the Model shown in Listing 5.6. As it is, local traffic of Facebook, Twitter and Whatapp applications would always be allowed. Additionally, roaming traffic of Email, Twitter and Whatsapp applications would always be allowed too. The rest of the traffic would have being allowed and then eventually would have being blocked, when the quota is reached.

The model in Listing 5.6, as it is, has a flaw. Other roaming traffic would eventually consume usage from the rule[0]. This is a problem for the subscribers, as they generally cannot control which applications should be allowed and which should not from the mobile devices. That is, background applications running on their mobiles, would consume their main usage while they are on roaming without subscribers even noticing it. To solve this flaw, one additional rule needs to be considered, as shown in Listing 5.7 below.

Listing 5.7: Core-Rules-based Roaming Plan Model - II

```
1    ...
2
3    // Unlimited Email Roaming Bolt-on
4    Rules[4].conditions.application = 3;
5    Rules[4].conditions.location = 1;
6    Rules[4].action.underquota = 1;
7    Rules[4].action.overquota = 1;
8    Rules[4].p = 2;
9
10   // Unlimited Twitter Roaming Bolt-on
11   Rules[5].conditions.application = 1;
12   Rules[5].conditions.location = 1;
13   Rules[5].action.underquota = 1;
14   Rules[5].action.overquota = 1;
15   Rules[5].p = 2;
16
17   // Unlimited Whatsapp Roaming Bolt-on
18   Rules[6].conditions.application = 2;
19   Rules[6].conditions.location = 1;
20   Rules[6].action.underquota = 1;
21   Rules[6].action.overquota = 1;
22   Rules[6].p = 2;
23
24   // Block other Roaming
25   Rules[7].conditions.application = 255;
26   Rules[7].conditions.location = 1;
27   Rules[7].action.underquota = 0;
28   Rules[7].action.overquota = 0;
29   Rules[7].p = 1;
```

Rule[7] was added in order to block always the traffic from the roaming location, specified as location 1. Note that in order to meet the desired behavior, rules [4], [5] and [6] require a higher priority than the priority of the rule [7]. When this model is verified, the priorities of the rule [7] will be updated, to show that model allow to detect that kind of wrong priority specification.

Figure 5.3 below, represent the expected behavior of this rule-based plan described previously in Listing 5.6 and 5.7. Note that other Roaming traffic shall always be blocked.

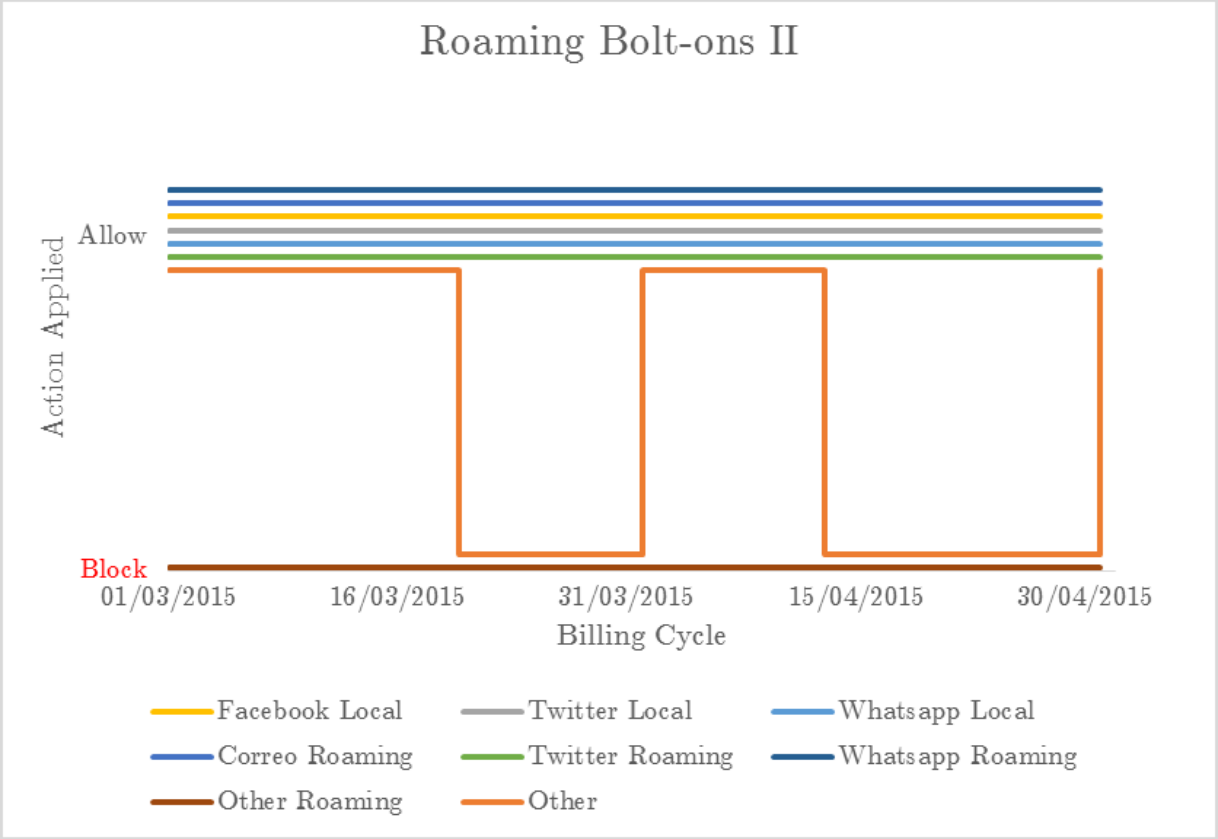

Figure 5.3: Roaming Rules - Linear Temporal Logic II

In order to verify the expected behavior expressed above, the LTL below will be used. As seen in chapter 3, Linear-Temporal-Logic formulas are used to prove the property desired.

Listing 5.8: Roaming-Rules LTL

```
1   ltl Romaing_Rules_LTL {
2     [](
3         ((generated_rule_0 -> allowed_rule_0) -> <>blocked_rule_0) &&
4         (generated_rule_1->allowed_rule_1) &&
5         (generated_rule_2->allowed_rule_2) &&
6         (generated_rule_3->allowed_rule_3) &&
7         (generated_rule_4->allowed_rule_4) &&
8         (generated_rule_5->allowed_rule_5) &&
9         (generated_rule_6->allowed_rule_6) &&
10        (generated_rule_7->blocked_rule_7)
11      )
12  }
```

Listing 5.12 above can be translated to natural language as *every* time traffic that meets the conditions of rules number 1 to 6 is generated, then it will be allowed. Additionally, *every* time traffic that meets rule number 7 is generated, it will be blocked. Finally, *every* time traffic that meets the condition of rule number 0 is generated, then it will be allowed and *eventually* will be blocked.

5.2.3 SPIN Verification

At this point we have intuitively specified the roaming rule-based plan with our Promela model (Listing 5.6 and 5.7 above) and we have determined which LTL formulas will be used to verify the model (Listing 5.12 above). Below it is presented the result obtained in Spin, with 0 errors shown in line 7.

Listing 5.9: Core-Rules Plan Verification

```
1  (Spin Version 6.3.2 -- 17 May 2014) + Partial Order Reduction
2  Full statespace search for:
3          never claim  + (Romaing_Rules_LTL)
4          assertion violations + (if within scope of claim)
5          acceptance cycles  + (fairness enabled)
6          invalid end states - (disabled by never claim)
7  State-vector 612 byte, depth reached 94, *** errors: 0 ***
8       2924 states, stored (3106 visited)
9       1299 states, matched
10      4405 transitions (= visited+matched)
11      3703 atomic steps
12 hash conflicts: 0 (resolved)
13 Stats on memory usage (in Megabytes):
14     1.751 equivalent memory usage for states (stored*(State-vector + overhead))
15     1.800 actual memory usage for states
16    64.000 memory used for hash table (-w24)
17     0.069 memory used for DFS stack (-m2000)
18    65.855 total actual memory usage
```

The memory used is also relevant to note. In this test execution, the number of rules was doubled from 4 to 8. The memory required for verifying the model increased in 1.66Mb, from 64.195 Mb to 65.855 Mb. The memory used is found in the last line of each execution.

Exceptions Verification

The results obtained previously were expected as the right priorities were assigned to verify the model. Empirical tests are performed below to validate invalid models, with wrong priorities assigned.

Listing 5.10: Increased priority of the Roaming Rule

```
1    ...
2    // Unlimited Whatsapp Roaming Bolt-on
3    Rules[6].conditions.application = 2;
4    Rules[6].conditions.location = 1;
5    Rules[6].action.underquota = 1;
6    Rules[6].action.overquota = 1;
7    Rules[6].p = 2;
8
9    // Block other Roaming
10   Rules[7].conditions.application = 255;
11   Rules[7].conditions.location = 1;
12   Rules[7].action.underquota = 0;
13   Rules[7].action.overquota = 0;
14   Rules[7].p = 3;
```

In the updated model above, the priority of roaming rule[7] was increased from 1 to 3. The action of the rule[7], is to block all the roaming traffic. With that priority, the other roaming applications that are supposed to be always allowed, will eventually be blocked, and consequently the model will not meet the LTL specification. That is, the rules [4], [5] and [6] will not be executed, all the roaming traffic will be handled by the rule 7, which has a higher priority assigned. SPIN model basically identifies that when whatsapp roaming traffic is generated, it is not always allowed.

As seen in Listing 5.11 below, an error is shown by the SPIN model checker. Also note that the number of states reached by the Model Checker is usually much lower when an error is found, as SPIN stops the state exploration as soon as an error is found.

Listing 5.11: Invalid Roaming-Rules Plan - Verification

```
1  (Spin Version 6.3.2 -- 17 May 2014) + Partial Order Reduction
2  Warning: Search not completed
3  Full statespace search for:
4          never claim  + (Romaing_Rules_LTL)
5          assertion violations + (if within scope of claim)
6          acceptance cycles  + (fairness enabled)
7          invalid end states - (disabled by never claim)
8  State-vector 612 byte, depth reached 88, *** errors: 1 ***
9       1861 states, stored (2043 visited)
10       865 states, matched
11      2908 transitions (= visited+matched)
12      2348 atomic steps
13 hash conflicts: 0 (resolved)
14 Stats on memory usage (in Megabytes):
15     1.115 equivalent memory usage for states (stored*(State-vector + overhead))
16     1.213 actual memory usage for states
17    64.000 memory used for hash table (-w24)
18     0.069 memory used for DFS stack (-m2000)
19    65.270 total actual memory usage
```

Another way for finding an error is having the right Model but using the wrong properties to verify it. Listing 5.13 below shows the result of verifying the original Roaming Model found in Listing 5.6 and 5.7, with the invalid LTL below.

Listing 5.12: Roaming-Rules LTL Invalidad

```
1    ltl Romaing_Rules_LTL_Invalid {
2      [](
3          ((generated_rule_0 -> allowed_rule_0) -> <>blocked_rule_0) &&
4          (generated_rule_1->allowed_rule_1) &&
5          (generated_rule_2->allowed_rule_2) &&
6          (generated_rule_3->allowed_rule_3) &&
7          (generated_rule_4->allowed_rule_4) &&
8          (generated_rule_5->allowed_rule_5) &&
9          (generated_rule_6->allowed_rule_6) &&
10         (generated_rule_7->allowed_rule_7)
11       )
12   }
```

The LTL expression in line 10 above, was changed. Now the roaming traffic is expected to be always allowed. Since that is not the case, an error is shown per the output below.

Listing 5.13: Invalid Roaming-Rules Plan - Verification II

```
1  (Spin Version 6.3.2 -- 17 May 2014) + Partial Order Reduction
2  Warning: Search not completed
3  Full statespace search for:
4          never claim  + (Romaing_Rules_LTL_Invalid)
5          assertion violations + (if within scope of claim)
6          acceptance cycles  + (fairness enabled)
7          invalid end states - (disabled by never claim)
8  State-vector 616 byte, depth reached 93, *** errors: 1 ***
9       2592 states, stored (2774 visited)
10      1163 states, matched
11      3937 transitions (= visited+matched)
12      3283 atomic steps
13 hash conflicts: 0 (resolved)
14 Stats on memory usage (in Megabytes):
15     1.562 equivalent memory usage for states (stored*(State-vector + overhead))
16     1.596 actual memory usage for states
17    64.000 memory used for hash table (-w24)
18     0.069 memory used for DFS stack (-m2000)
19    65.660 total actual memory usage
```

5.3 Family Rules

5.3.1 Promela Model

Below is presented three Promela models, which represent let us represent the plan model of each family member.

Family Member #1

The family plan specified in the Model below, can be used by a Family member, who wants to prevent its Smartphone to consume HD streaming videos. It consists on the Family Rule specified as the Rule[0], which is shared by all the family members, and two additional Familiy Bolt-ons rules to allow unlimited social networks (Rule [1]) and always block Streaming HD videos (Rule [2]).

Listing 5.14: Family Member#1 Model

```
1    // Family Rule
2    Rules[0].conditions.application = 255;
3    Rules[0].conditions.location = 255;
4    Rules[0].action.underquota = 1;
5    Rules[0].action.overquota = 0;
6    Rules[0].p = 0;
7
8    // Unlimited Social Networks
9    Rules[1].conditions.application = 2; // Social Networks
10   Rules[1].conditions.sub_application = 255; // All Social Networks
11   Rules[1].conditions.location = 0;
12   Rules[1].action.underquota = 1;
13   Rules[1].action.overquota = 1;
14   Rules[1].p = 1;
15
16   // HD Blocked
17   Rules[2].conditions.application = 1; // Video Streaming
18   Rules[2].conditions.sub_application = 1; // Only HD Streaming
19   Rules[2].conditions.location = 0;
20   Rules[2].action.underquota = 0;
21   Rules[2].action.overquota = 0;
22   Rules[2].p = 1;
```

Family Member #2

The family plan specified in the Model below, can be used by a Family member, who wants to have blocked Social Networks and HD Streaming Videos traffic. It consists of the same Family Rule specified as the Rule[0], shared by all the family members, and two additional Familiy Bolt-ons rules to always block social networks (Rule [1]) and always block Streaming HD videos (Rule [2]).

Listing 5.15: Family Member#2 Model

```
1    // Family Rule
2    Rules[0].conditions.application = 255;
3    Rules[0].conditions.location = 255;
4    Rules[0].action.underquota = 1;
5    Rules[0].action.overquota = 0;
6    Rules[0].p = 0;
```

```
7
8     // Blocked Social Networks
9     Rules[1].conditions.application = 2; // Social Networks
10    Rules[1].conditions.sub_application = 255; // All Social Networks
11    Rules[1].conditions.location = 0;
12    Rules[1].action.underquota = 0;
13    Rules[1].action.overquota = 0;
14    Rules[1].p = 1;
15
16    // HD Blocked
17    Rules[2].conditions.application = 1; // Video Streaming
18    Rules[2].conditions.sub_application = 1; // Only HD Streaming
19    Rules[2].conditions.location = 0;
20    Rules[2].action.underquota = 0;
21    Rules[2].action.overquota = 0;
22    Rules[2].p = 1;
```

Family Member #3

The family plan specified in the Model below, can be used by a Family member, who wants to have an optimized experience in Streaming Videos. It consists of the same Family Rule specified as the Rule[0], shared by all the family members, and one additional rule to optimize Streaming Video (Rule [1]).

Listing 5.16: Family Member#3 Model

```
1     // Family Rule
2     Rules[0].conditions.application = 255;
3     Rules[0].conditions.location = 255;
4     Rules[0].action.underquota = 1;
5     Rules[0].action.overquota = 0;
6     Rules[0].p = 0;
7
8     // Optimize Streaming Video
9     Rules[1].conditions.application = 1; // Video Streaming
10    Rules[1].conditions.sub_application = 255; // All Video Streaming
11    Rules[1].conditions.location = 0;
12    Rules[1].action.underquota = 1;
13    Rules[1].action.overquota = 1;
14    Rules[1].p = 1;
```

5.3.2 System Properties

The key rule of all the family plans above, is the Rule[0]; it dictates how most of the traffic should be handled by a shared quota. As before, we are specifying no condition for that rule and indicating to apply the block action when the quota is reached.

Additionally, each plan includes different bolt-on rules, to either allow or block certain applications.

In order to verify the expected behavior of each Family-member plan, the following LTL formulas will be used. As previously studied, Linear-Temporal-Logic formulas are used to prove the desired properties. In this case, no new property is being used to verify this Family Plan composed by three family members.

Listing 5.17: Family Member#1 LTL

```
1   ltl ltl_family_member_N1 {
2     [](
3         ((generated_rule_0 -> allowed_rule_0) -> <>blocked_rule_0) &&
4         (generated_rule_1->allowed_rule_1) &&
5         (generated_rule_2->blocked_rule_2)
6       )
7   }
```

Listing 5.18: Family Member#2 LTL

```
1   ltl ltl_family_member_N2 {
2     [](
3         ((generated_rule_0 -> allowed_rule_0) -> <>blocked_rule_0) &&
4         (generated_rule_1->blocked_rule_1) &&
5         (generated_rule_2->blocked_rule_2)
6       )
7   }
```

Listing 5.19: Family Member#3 LTL

```
1   ltl ltl_family_member_N3 {
2     [](
3         ((generated_rule_0 -> allowed_rule_0) -> <>blocked_rule_0) &&
4         (generated_rule_1->allowed_rule_1)
5       )
6   }
```

5.3.3 SPIN Verification

Listings 5.20, 5.21 and 5.22 below, presents the result obtained in Spin. Per the previous results obtained, it is clear that no error will be shown.

Listing 5.20: Family Member#1 Plan Verification

```
1  (Spin Version 6.3.2 -- 17 May 2014) + Partial Order Reduction
2  Full statespace search for:
3         never claim  + (ltl_family_member_N1)
4         assertion violations + (if within scope of claim)
5         acceptance cycles  + (fairness enabled)
6         invalid end states - (disabled by never claim)
7  State-vector 252 byte, depth reached 54, *** errors: 0 ***
8       259 states, stored (316 visited)
9       109 states, matched
10      425 transitions (= visited+matched)
11      261 atomic steps
12 hash conflicts: 0 (resolved)
13 Stats on memory usage (in Megabytes):
14     0.066 equivalent memory usage for states (stored*(State-vector + overhead))
15     0.144 actual memory usage for states
16    64.000 memory used for hash table (-w24)
17     0.069 memory used for DFS stack (-m2000)
18    64.195 total actual memory usage
```

Listing 5.21: Family Member#2 Plan Verification

```
1  (Spin Version 6.3.2 -- 17 May 2014) + Partial Order Reduction
2  Full statespace search for:
3         never claim  + (ltl_family_member_N2)
4         assertion violations + (if within scope of claim)
5         acceptance cycles  + (fairness enabled)
6         invalid end states - (disabled by never claim)
7  State-vector 252 byte, depth reached 54, *** errors: 0 ***
```

```
8       259 states, stored (316 visited)
9       109 states, matched
10      425 transitions (= visited+matched)
11      261 atomic steps
12  hash conflicts: 0 (resolved)
13  Stats on memory usage (in Megabytes):
14      0.066 equivalent memory usage for states (stored*(State-vector + overhead))
15      0.144 actual memory usage for states
16     64.000 memory used for hash table (-w24)
17      0.069 memory used for DFS stack (-m2000)
18     64.195 total actual memory usage
```

Listing 5.22: Family Member#3 Plan Verification

```
1   (Spin Version 6.3.2 -- 17 May 2014) + Partial Order Reduction
2   Full statespace search for:
3           never claim  + (ltl_family_member_N3)
4           assertion violations + (if within scope of claim)
5           acceptance cycles  + (fairness enabled)
6           invalid end states - (disabled by never claim)
7   State-vector 180 byte, depth reached 46, *** errors: 0 ***
8       105 states, stored (137 visited)
9        42 states, matched
10      179 transitions (= visited+matched)
11       84 atomic steps
12  hash conflicts: 0 (resolved)
13  Stats on memory usage (in Megabytes):
14      0.020 equivalent memory usage for states (stored*(State-vector + overhead))
15      0.144 actual memory usage for states
16     64.000 memory used for hash table (-w24)
17      0.069 memory used for DFS stack (-m2000)
18     64.195 total actual memory usage
```

Exceptions Verification

The results obtained previously were expected as the right priorities were assigned to verify the model. Empirical tests are performed below to validate invalid model specifications.

If by any chance, the rule[1] of the Family Member #2 plan shown in Listing 5.16 (Block Social Networks), were added to the Family Member #1 plan; an error would be shown regardless the priorities assigned to this new rule.

An error will be always shown regardless the priorities assigned to the rules [1] of the Family Member #1 and #2 plans, is due the conditions of the two rules are the same. That is, they can't coexist without overriding one of them.

Listing 5.23: Invalidad Family Member#1 Model

```
1    // Family Rule
2    Rules[0].conditions.application = 255;
3    Rules[0].conditions.location = 255;
4    Rules[0].action.underquota = 1;
5    Rules[0].action.overquota = 0;
6    Rules[0].p = 0;
7
8    // Unlimited Social Networks
9    Rules[1].conditions.application = 2; // Social Networks
10   Rules[1].conditions.sub_application = 255; // All Social Networks
11   Rules[1].conditions.location = 0;
12   Rules[1].action.underquota = 1;
13   Rules[1].action.overquota = 1;
14   Rules[1].p = 2;
15
16   // HD Blocked
17   Rules[2].conditions.application = 1; // Video Streaming
18   Rules[2].conditions.sub_application = 1; // Only HD Streaming
19   Rules[2].conditions.location = 0;
20   Rules[2].action.underquota = 0;
21   Rules[2].action.overquota = 0;
22   Rules[2].p = 2;
23
24   // Blocked Social Networks
25   Rules[3].conditions.application = 2; // Social Networks
26   Rules[3].conditions.sub_application = 255; // All Social Networks
27   Rules[3].conditions.location = 0;
28   Rules[3].action.underquota = 0;
29   Rules[3].action.overquota = 0;
30   Rules[3].p = [1,2,3];
```

Regardless the priority assigned to Rule[3] above (e.g. 1, 2 or 3), the following error is shown as the rules [1] and [3] cannot coexist in the same plan:

Listing 5.24: Invalidad Family Member#1 Plan Verification

```
1  (Spin Version 6.3.2 -- 17 May 2014) + Partial Order Reduction
2  Warning: Search not completed
3  Full statespace search for:
4          never claim  + (ltl_family_member_N1_invalid)
5          assertion violations + (if within scope of claim)
6          acceptance cycles  + (fairness enabled)
7          invalid end states - (disabled by never claim)
8  State-vector 328 byte, depth reached 62, *** errors: 1 ***
9        435 states, stored (517 visited)
10       190 states, matched
11       707 transitions (= visited+matched)
12       488 atomic steps
13 hash conflicts: 0 (resolved)
14 Stats on memory usage (in Megabytes):
15     0.143 equivalent memory usage for states (stored*(State-vector + overhead))
16     0.241 actual memory usage for states
17    64.000 memory used for hash table (-w24)
18     0.069 memory used for DFS stack (-m2000)
19    64.293 total actual memory usage
```

The issue of two rules with exact the same conditions is addressed below, in the section "Limited Bolt-on Rules" of this chapter, by using a different LTL formula.

5.4 Parental-Control Rules

5.4.1 Promela Model

The model of the Parental-Control Rules is shown in Listing 5.25. Given that a plan could consist of one or more rule instances, the worst case scenario was modeled; that is, one instance of each rule was selected to form the plan below; including one from the Core Rules.

The main difference with this model, is that it requires time conditions; specified through the time_of_day (tod) typedef.

Listing 5.25: Parental-Control-Rules-based Plan Model

```
1   // CoreRule
2   Rules[0].conditions.application = 255;
3   Rules[0].conditions.location = 255;
4   Rules[0].action.underquota = 1;
5   Rules[0].action.overquota = 0;
6   Rules[0].p = 1;
7
8   // Block Social Networks
9   Rules[1].conditions.application = 1;
10  Rules[1].conditions.location = 255;
11  Rules[1].conditions.tod._active = true;
12  Rules[1].conditions.tod.mon.start_time = 8;
13  Rules[1].conditions.tod.mon.end_time = 18;
14  Rules[1].conditions.tod.mon._active = true;
15  Rules[1].conditions.tod.tue.start_time = 8;
16  Rules[1].conditions.tod.tue.end_time = 18;
17  Rules[1].conditions.tod.tue._active = true;
18  Rules[1].conditions.tod.wed.start_time = 8;
19  Rules[1].conditions.tod.wed.end_time = 18;
20  Rules[1].conditions.tod.wed._active = true;
21  Rules[1].conditions.tod.thr.start_time = 8;
22  Rules[1].conditions.tod.thr.end_time = 18;
23  Rules[1].conditions.tod.thr._active = true;
24  Rules[1].conditions.tod.fri.start_time = 8;
25  Rules[1].conditions.tod.fri.end_time = 18;
26  Rules[1].conditions.tod.fri._active = true;
27  Rules[1].action.underquota = 0;
28  Rules[1].action.overquota = 0;
29  Rules[1].p = 2;
30
31  // Block Gaming
32  Rules[2].conditions.application = 2;
33  Rules[2].conditions.location = 255;
34  Rules[2].conditions.tod._active = true;
35  Rules[2].conditions.tod.mon.start_time = 8;
36  Rules[2].conditions.tod.mon.end_time = 18;
37  Rules[2].conditions.tod.mon._active = true;
38  Rules[2].conditions.tod.tue.start_time = 8;
39  Rules[2].conditions.tod.tue.end_time = 18;
40  Rules[2].conditions.tod.tue._active = true;
41  Rules[2].conditions.tod.wed.start_time = 8;
42  Rules[2].conditions.tod.wed.end_time = 18;
43  Rules[2].conditions.tod.wed._active = true;
44  Rules[2].conditions.tod.thr.start_time = 8;
45  Rules[2].conditions.tod.thr.end_time = 18;
46  Rules[2].conditions.tod.thr._active = true;
```

```
47    Rules[2].conditions.tod.fri.start_time = 8;
48    Rules[2].conditions.tod.fri.end_time = 18;
49    Rules[2].conditions.tod.fri._active = true;
50    Rules[2].action.underquota = 0;
51    Rules[2].action.overquota = 0;
52    Rules[2].p = 2;
53
54    // Block Streaming
55    Rules[3].conditions.application = 4;
56    Rules[3].conditions.location = 255;
57    Rules[3].action.underquota = 0;
58    Rules[3].action.overquota = 0;
59    Rules[3].p = 2;
```

5.4.2 System Properties

The System Properties to be verified are quite similar to the previous ones used, where rule[0] has no condition so all the traffic is eventually blocked when the quota is reached. Additionally there are three more rules, each one to always block a different application when the rule conditions, including the time conditions, are met.

As it has been discussed, the rules mutually exclusive among themselves requires a higher priority. If any of these rules has a lower priority assigned, the Rule[0] will take precedence over that rule and consequently, the desired behavior of blocking that particular application will not be met.

Listing 5.26: Parental-Control-Rules LTL

```
1   ltl parental_control_LTL {
2     [](
3         ((generated_rule_0 -> allowed_rule_0) -> <>blocked_rule_0) &&
4         (generated_rule_1->blocked_rule_1) &&
5         (generated_rule_2->blocked_rule_2) &&
6         (generated_rule_3->blocked_rule_3)
7       )
8   }
```

Listing 5.26 above can be translated to natural language as *every* time traffic that meets the conditions of rule [0] is generated, then it will be allowed and *eventually* will be blocked. Additionally, *every* time traffic that meets rules [1], [2] and [3] is generated, it will be blocked.

5.4.3 SPIN Verification

Below it is presented the result obtained in Spin, with 0 errors shown in line 7.

Listing 5.27: Parental-Control-Rules Plan Verification

```
1  (Spin Version 6.3.2 -- 17 May 2014) + Partial Order Reduction
2  Full statespace search for:
3         never claim  + (parental_control_LTL)
4         assertion violations + (if within scope of claim)
5         acceptance cycles  + (fairness enabled)
6         invalid end states - (disabled by never claim)
7  State-vector 324 byte, depth reached 62, *** errors: 0 ***
8       504 states, stored (586 visited)
9       217 states, matched
10      803 transitions (= visited+matched)
11      556 atomic steps
12 hash conflicts: 0 (resolved)
13 Stats on memory usage (in Megabytes):
14    0.163 equivalent memory usage for states (stored*(State-vector + overhead))
15    0.241 actual memory usage for states
16   64.000 memory used for hash table (-w24)
17    0.069 memory used for DFS stack (-m2000)
18   64.293 total actual memory usage
```

5.5 Limited Bolt-on Rules

5.5.1 Promela Model

LADD01 and LADD04 Limited Bolt-on rules were selected from Table 4.7 above, and were combined with the unlimited bolt-on Rules seen previously to form a plan.

Listing 5.28: Core-Rules-based Plan Model

```
1   // CoreRule
2   Rules[0].conditions.application = 255;
3   Rules[0].conditions.location = 255;
4   Rules[0].action.underquota = 1; //Allow Traffic
5   Rules[0].action.overquota = 0; //Block Traffic
6   Rules[0].p = 0;
7
8   // Unlimited local Facebook Bolt-on
9   Rules[1].conditions.application = 0;
10  Rules[1].conditions.location = 0;
11  Rules[1].action.underquota = 1; //Allow Traffic
12  Rules[1].action.overquota = 1; //Allow Traffic
13  Rules[1].p = 3;
14
15  // Unlimited local Twitter Bolt-on
16  Rules[2].conditions.application = 1;
17  Rules[2].conditions.location = 0;
18  Rules[2].action.underquota = 1; //Allow Traffic
19  Rules[2].action.overquota = 1; //Allow Traffic
20  Rules[2].p = 3;
21
22  // Unlimited local Whatsapp Bolt-on
23  Rules[3].conditions.application = 2;
24  Rules[3].conditions.location = 0;
25  Rules[3].action.underquota = 1; //Allow Traffic
26  Rules[3].action.overquota = 1; //Allow Traffic
27  Rules[3].p = 3;
28
29  // 1GB Video Streaming Optimized
30  Rules[4].conditions.application = 5;
31  Rules[4].conditions.location = 0;
32  Rules[4].action.underquota = 1; //Allow Traffic
33  Rules[4].is_uninstallable = true;
34  Rules[4].p = 2;
35
36  // 1GB Extra
37  Rules[5].conditions.application = 255;
38  Rules[5].conditions.location = 255;
39  Rules[5].action.underquota = 1; //Allow Traffic
40  Rules[5].is_uninstallable = true;
41  Rules[5].p = 1;
```

Limited Bolt-on rules are specified in Rules [4] and [5] above. Rule[4] optimize Video Streaming; while Rule[5] add more volume to a Plan, for a one-time-usage.

5.5.2 System Properties

The new LTL formulas to verify the properties of Rules [4] and [5] are shown below:

Listing 5.29: Limited Bolt-on Rules LTL

```
1   ltl limitedLTL {
2    [](
3       (generated_rule_1->allowed_rule_1) &&
4       (generated_rule_2->allowed_rule_2) &&
5       (generated_rule_3->allowed_rule_3)
6      ) &&
7    ([](((generated_rule_4 -> allowed_rule_4)) U overquota_rule_4)) &&
8    ([](((generated_rule_5 -> allowed_rule_5)) U overquota_rule_5)) &&
9    ([](((generated_rule_0 -> allowed_rule_0) ->
10         blocked_rule_0)) U (!overquota_rule_5))
11  }
```

Lines 2 to 6 verify the expected behavior of the rules studied previously. From line 7, a new operator is used: Until (U). Line 7 can be translated to natural language as *every* time traffic that meets the conditions of rule [4] is generated, then it will be allowed until rule [4] reaches its quota. Similarly, Line 8, as *every* time traffic that meets the conditions of rule [5] is generated, then it will be allowed until rule [5] reaches its quota. Consequently, line 9 is translated to *every* time traffic that meets the condition of rule [0] is generated, then it will be allowed and *eventually* will be blocked until rule [5] is not overquota.

Given that Rule[0] and Rule[5] are mutually exclusive rules, the until operator in line 10 is required. As seen before in 5.3.3 and 5.4.2, rules that have exactly the same conditions, cannot coexist in a plan. However, the until operator above, let mutually exclusive rules coexist, only if one of the rules is temporary applied (e.g. when the quota is reached). That is, with the Until operator (U), which rule is applied is controlled.

5.5.3 SPIN Verification

We have specified a rule-based plan with a number of unlimited and limited bolt-on rules and introduced the LTL to be used to verify the Promela model of the plan. Listing 5.30 below, shows no error in line 7.

Listing 5.30: Limited Bolt-on-Rules Plan Verification

```
1  (Spin Version 6.3.2 -- 17 May 2014) + Partial Order Reduction
2  Full statespace search for:
3          never claim  + (limitedLTL)
4          assertion violations + (if within scope of claim)
5          acceptance cycles  + (fairness enabled)
6          invalid end states - (disabled by never claim)
7  State-vector 468 byte, depth reached 76, *** errors: 0 ***
8       3725 states, stored (13883 visited)
9      12596 states, matched
10     26479 transitions (= visited+matched)
11     24133 atomic steps
12 hash conflicts: 0 (resolved)
13 Stats on memory usage (in Megabytes):
14     1.719 equivalent memory usage for states (stored*(State-vector + overhead))
15     1.796 actual memory usage for states
16    64.000 memory used for hash table (-w24)
17     0.069 memory used for DFS stack (-m2000)
18    65.855 total actual memory usage
```

Exceptions Verification

No error was shown in Listing 5.30 above, as the Until operator (U) was used. If a different LTL formula, with no Until operator, were used to verify the Model, as below, an error would be shown by Spin Model Checker. The invalid LTL is presented in Listing 5.31 below, where no Until operator is used and the corresponding result is presented in Listing 5.32.

Listing 5.31: Invalid Limited Bolt-on Rules LTL

```
1   ltl InvalidLimitedLTL {
2    [](
3       (generated_rule_1->allowed_rule_1) &&
4       (generated_rule_2->allowed_rule_2) &&
5       (generated_rule_3->allowed_rule_3)
6      ) &&
7    ([](((generated_rule_4 -> allowed_rule_4)))) &&
8    ([](((generated_rule_5 -> allowed_rule_5)))) &&
9    ([](((generated_rule_0 -> allowed_rule_0) ->
10         blocked_rule_0)))
11  }
```

Listing 5.32: Invalid Limited Bolt-on-Rules Plan Verification

```
1  (Spin Version 6.3.2 -- 17 May 2014) + Partial Order Reduction
2  Warning: Search not completed
3  Full statespace search for:
4         never claim  + (InvalidlimitedLTL)
5         assertion violations + (if within scope of claim)
6         acceptance cycles  + (fairness enabled)
7         invalid end states - (disabled by never claim)
8  State-vector 468 byte, depth reached 58, *** errors: 1 ***
9        59 states, stored
10       13 states, matched
11       72 transitions (= stored+matched)
12       33 atomic steps
13 hash conflicts: 0 (resolved)
14 Stats on memory usage (in Megabytes):
15    0.027 equivalent memory usage for states (stored*(State-vector + overhead))
16    0.143 actual memory usage for states
17   64.000 memory used for hash table (-w24)
18    0.069 memory used for DFS stack (-m2000)
19   64.195 total actual memory usage
```

5.6 Result Analysis and Discussion

The main highlights of each verified use-case scenario is summarized in table 5.1 below. Additionally, the maximum concurrent rules verified on each use-case, along with its memory usage and the properties used during the verification, are also detailed.

Use Case	Highlights	Concurrent Rules	Properties
Core Rules	Basic use-case is presented. Always operator (□) is introduced. Eventually operator (◊) is introduced.	4 Rules (64.195 MB)	PT01, PT02
Roaming Bolt-on Rules	Basic use-case is extended with Roaming Bolt-on rules. Significance of having the right priority is illustrated.	7 Rules (65.855 MB)	PT01, PT02
Family Rules	Application and sub-application conditions are considered. An example is shown where rules with the same conditions cannot co-exist with the LTL used so far.	3 Rules (64.195 MB)	PT01, PT02
Parental-Control Rules	Time-of-day conditions are considered.	4 Rules (64.293 MB)	PT01, PT02
Limited Bolt-on Rules	Complex use-case is presented, where temporary bolt-on rules are used. Until(U) operator is introduced.	6 Rules (65.855 MB)	PT01, PT02, PT03, PT04

Table 5.1: Summary of results

As seen in table 5.1 above, most of the use-cases were verified using the properties PT01 and PT02 – that is because the *Until* operator (U) is only required when a rule needs to be applied until some other event occurs. On the other hand, Always (□) and Eventually (◊) operators were always used to verify the use-cases.

Each of the use-cases above, considers a different rule condition or a new property not verified before – this, to let us illustrate the application of each condition or property. More complex plans, not covered in the use-cases above, may be modeled, specified and verified by combining the conditions used with their corresponding properties. For example, Time of Day conditions may be used with the location condition. The device condition was not used on the use-cases, but its application is similar to the other conditions (e.g. location).

Our proposed Promela model, introduced in figure 3.1, can be intuitively extended to support more conditions not considered in this thesis.

The memory required for verifying the use-cases was fairly low. However, as more rules are specified in a plan, more memory would be required. That is, models with many plans and rules may require huge amounts of memory – an exponential relation between the number of rules and properties used, with the required memory is presumed.

CHAPTER 6

Conclusion

6.1 Summary of Contributions

In this thesis, we provided a framework to formally specify and verify rule-based Internet plans. The provided framework includes, a Promela Model to specify the plans and a set of four Linear Temporary Logic (LTL) properties to be used to verify the model.

The proposed Promela model can be used to unambiguously model rule-based Internet plans including a number of conditions such as Location, Application, Device and Time-of Day and different actions to be applied when the rule runs out of quota.

This thesis, identified two major LTL operators, Always ($\Box$) and Eventually ($\Diamond$), which were used in all the use-cases; while the Until (U) LTL operator was applied only in one use-case.

The real and future use-cases scenarios provided based on Internet Plans offered in the public domain, can be used as a future benchmark for other models.

Finally, it was proved that Model Checking is a feasible approach to detect conflicts between rules within Internet plans through the scenarios used.

6.2 Future research

In Chapter 3, we introduced our proposed Promela model including several conditions and actions. Our model may be extended by adding other conditions to support use-cases not considered or offered yet. Similarly, the four LTL properties, may be extended accordingly.

We are able to conclude that Model Checking is a feasible and powerful tool to verify rule-based Internet plans and identify conflicts between their rules, per the experimental results. But since we have not tried our Model on a Plan with many rules, we have not challenged our model from a stressful verification perspective – as more than 20 rules within only one plan would be unrealistic for now. Our future work includes more thorough experiments with numerous rules.

Other Model Checking tools could be considered to compare time or space requirements between them.

References

[1] S. Sen, C. Joe-Wong, S. Ha, and M. Chiang, "A survey of smart data pricing: Past proposals, current plans, and future trends," *ACM Comput. Surv.*, vol. 46, no. 2, pp. 15:1–15:37, Nov. 2013. [Online]. Available: http://doi.acm.org/10.1145/2543581.2543582

[2] K. Kimbler and M. Taylor, "Value added mobile broadband services innovation driven transformation of the smart pipe," in *Intelligence in Next Generation Networks (ICIN), 2012 16th International Conference on*, Oct 2012, pp. 30–34.

[3] D. D. Clark, "Policy routing in internet protocols," M.I.T. Laboratory for Computer Science, RFC 1102, 1989. [Online]. Available: https://tools.ietf.org/rfc/rfc1102.txt

[4] M. Sloman and E. Lupu, "Policy specification for programmable networks," in *Active Networks*, ser. Lecture Notes in Computer Science, S. Covaci, Ed. Springer Berlin Heidelberg, 1999, vol. 1653, pp. 73–84. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-48507-0_7

[5] G. Stone, B. Lundy, and G. Xie, "Network policy languages: a survey and a new approach," *Network, IEEE*, vol. 15, no. 1, pp. 10–21, Jan 2001.

[6] A. Albaladejo, F. de Gouveia, M. Corici, and T. Magedanz, "The pcc rule in the 3gpp ims policy and charging control architecture," in *Global Telecommunications Conference, 2008. IEEE GLOBECOM 2008. IEEE*, Nov 2008, pp. 1–5.

[7] T. Grgic and M. Matijasevic, "An overview of online charging in 3gpp networks: new ways of utilizing user, network, and service-related information," *International Journal of Network Management*, vol. 23, no. 2, pp. 81–100, 2013. [Online]. Available: http://dx.doi.org/10.1002/nem.1816

[8] D. Evans, "The internet of things. how the next evolution of the internet is changing everything," 2011. [Online]. Available: http://www.cisco.com/web/about/ac79/docs/innov/IoT_IBSG_0411FINAL.pdf

[9] M. Cheboldaeff, "Service charging challenges in converged networks," *Communications Magazine, IEEE*, vol. 49, no. 1, pp. 118–123, January 2011.

[10] Sandvine, "Success story: A million reasons to select vox telecom," 2014. [Online]. Available: https://www.sandvine.com/downloads/general/success-stories/a-million-reasons-to-select-vox-telecom.pdf

[11] Hk.chinamobile.com, "Lite data service plans," 2015. [Online]. Available: https://www.hk.chinamobile.com/en/services_plan/Lite_Data/

[12] Z. Ezziane, "Charging and pricing challenges for 3g systems," *Communications Surveys Tutorials, IEEE*, vol. 7, no. 4, pp. 58–68, Fourth 2005.

[13] J. Hart and R. Brown, "What lte policy control features can operators execute to differentiate themselves from ott players?" in *Intelligence in Next Generation Networks (ICIN), 2013 17th International Conference on*, Oct 2013, pp. 16–22.

[14] E. Technologies, *Whitepaper: A Hand-book of emerging Policy Management use cases*, 2012. [Online]. Available: http://techlibrary.informationweek.com/whitepaper/asset/detail/110443

[15] Sandvine, "Increasing revenue with innovative consumer services: Overview," 2014. [Online]. Available: https://www.sandvine.com/downloads/general/solutions/revenue-generation/consumer-services-overview.pdf

[16] Alcatel, "Monetize over-the-top mobile applications," 2014. [Online]. Available: http://www.tmcnet.com/tmc/whitepapers/documents/whitepapers/2014/10718-monetize-over-the-top-mobile-applications.pdf

[17] A. Communications, "Delivering digital lifestyle services," 2014. [Online]. Available: http://47maz93acasm1hwicm2d3sw7.wpengine.netdna-cdn.com/wp-content/uploads/Service_Provider_Use_Case_Booklet_12-2014.pdf

[18] W. Chan, R. J. Anderson, P. Beame, S. Burns, F. Modugno, D. Notkin, and J. D. Reese, "Model checking large software specifications," *IEEE Trans. Softw. Eng.*, vol. 24, no. 7, pp. 498–520, Jul. 1998. [Online]. Available: http://dx.doi.org/10.1109/32.708566

[19] G. Holzmann, E. Najm, and A. Serhrouchni, "Spin model checking: an introduction," *International Journal on Software Tools for Technology Transfer (STTT)*, vol. 2, no. 4, pp. 321–327, 2000.

[20] M. Y. Vardi and P. Wolper, "An automata-theoretic approach to automatic program verification," *1st Symposium in Logic in Computer Science (LICS)*, pp. 322–331, 1986.

[21] R. Gerth, D. Peled, M. Y. Vardi, R. Gerth, D. D. Eindhoven, D. Peled, M. Y. Vardi, and P. Wolper, "Simple on-the-fly automatic verification of linear temporal logic," in *In Protocol Specification Testing and Verification*. Chapman and Hall, 1995, pp. 3–18.

[22] G. J. Holzmann, D. Peled, and M. Yannakakis, "On nested depth first search," 1996.

[23] K. Schneider, *Verification of reactive systems*. Springer-Verlag, 2004.

[24] Z. Duan, "Proof search algorithms for detecting interactions in telecommunication features," 2003.

[25] A. Awad, M. Weidlich, and M. Weske, "Consistency checking of compliance rules," in *Business Information Systems*, ser. Lecture Notes in Business Information Processing, W. Abramowicz and R. Tolksdorf, Eds. Springer Berlin Heidelberg, 2010, vol. 47, pp. 106–118. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-12814-1_10

[26] F. Hantry, M.-S. Hacid, and R. Thion, "Detection of conflicting compliance rules," in *Enterprise Distributed Object Computing Conference Workshops (EDOCW), 2011 15th IEEE International*, Aug 2011, pp. 419–428.

[27] A. Gawanmeh and S. Tahar, "Novel algorithm for detecting conflicts in firewall rules," in *Electrical Computer Engineering (CCECE), 2012 25th IEEE Canadian Conference on*, April 2012, pp. 1–4.

APPENDIX A

Promela Model Algorithms

Listing A.1 shows the Promela code for the PriorityChecker process.

Listing A.1: PriorityChecker Process - Promela Code

```
1  proctype PriorityChecker()
2  {
3    // Pick two rules.
4    int x;
5    select (x : 0 .. (num_R-1));
6    int y;
7    select (y : 0 .. (num_R-1));
8    bool local_overquota=false;
9
10   if
11   :: (x != y && !Rules[x].is_uninstalled) ->
12     atomic{
13       Rules[x].is_traffic_generated = true;
14       if
15       :: (
16          (Rules[x].conditions.location != Rules[y].conditions.location &&
              Rules[y].conditions.location !=255) ||
17          (Rules[x].conditions.application != Rules[y].conditions.application &&
              Rules[y].conditions.application !=255) ||
18          (Rules[x].conditions.application == Rules[y].conditions.application &&
              Rules[x].conditions.sub_application !=
              Rules[y].conditions.sub_application &&
              Rules[y].conditions.sub_application!=255)
19        ) ->
20          Rules[x].action.action_applied = (Rules[x].action.underquota);
21          (Rules[x].is_overquota == true)
22          local_overquota = true;
```

```
23        :: (
24           (Rules[x].conditions.tod._active && Rules[y].conditions.tod._active) &&
25           !(Rules[x].conditions.tod.mon._active &&
                Rules[y].conditions.tod.mon._active &&
                (Rules[x].conditions.tod.mon.start_time<Rules[y].conditions.tod.mon.end_time
                &&
                Rules[x].conditions.tod.mon.end_time>Rules[y].conditions.tod.mon.start_time))
                &&
26           !(Rules[x].conditions.tod.tue._active &&
                Rules[y].conditions.tod.tue._active &&
                (Rules[x].conditions.tod.tue.start_time<Rules[y].conditions.tod.tue.end_time
                &&
                Rules[x].conditions.tod.tue.end_time>Rules[y].conditions.tod.tue.start_time))
                &&
27           !(Rules[x].conditions.tod.wed._active &&
                Rules[y].conditions.tod.wed._active &&
                (Rules[x].conditions.tod.wed.start_time<Rules[y].conditions.tod.wed.end_time
                &&
                Rules[x].conditions.tod.wed.end_time>Rules[y].conditions.tod.wed.start_time))
                &&
28           !(Rules[x].conditions.tod.thr._active &&
                Rules[y].conditions.tod.thr._active &&
                (Rules[x].conditions.tod.thr.start_time<Rules[y].conditions.tod.thr.end_time
                &&
                Rules[x].conditions.tod.thr.end_time>Rules[y].conditions.tod.thr.start_time))
                &&
29           !(Rules[x].conditions.tod.fri._active &&
                Rules[y].conditions.tod.fri._active &&
                (Rules[x].conditions.tod.fri.start_time<Rules[y].conditions.tod.fri.end_time
                &&
                Rules[x].conditions.tod.fri.end_time>Rules[y].conditions.tod.fri.start_time))
                &&
30           !(Rules[x].conditions.tod.sat._active &&
                Rules[y].conditions.tod.sat._active &&
                (Rules[x].conditions.tod.sat.start_time<Rules[y].conditions.tod.sat.end_time
                &&
                Rules[x].conditions.tod.sat.end_time>Rules[y].conditions.tod.sat.start_time))
                &&
31           !(Rules[x].conditions.tod.sun._active &&
                Rules[y].conditions.tod.sun._active &&
                (Rules[x].conditions.tod.sun.start_time<Rules[y].conditions.tod.sun.end_time
                &&
                Rules[x].conditions.tod.sun.end_time>Rules[y].conditions.tod.sun.start_time))
32           ) ->
33             Rules[x].action.action_applied = (Rules[x].action.underquota);
34             (Rules[x].is_overquota == true)
35             local_overquota = true;
36         :: else ->
```

```
37          if
38          ::(Rules[x].p > Rules[y].p) ->
39              Rules[x].action.action_applied = (Rules[x].action.underquota);
40              (Rules[x].is_overquota == true)
41              local_overquota = true;
42          :: else ->
43              Rules[x].action.action_applied=NO_ACTION;
44          fi
45       fi
46     }
47     if
48     :: local_overquota -->Rules[x].action.action_applied =
           (Rules[x].action.overquota);
49     :: else -> Rules[x].action.action_applied=NO_ACTION;
50     fi
51   :: else -> Rules[x].is_traffic_generated = false;
52   fi
53 }
```

Listing A.2 shows the Promela code for the OverQuota process.

Listing A.2: OverQuota Process - Promela Code

```
1  proctype OverQuota()
2  {
3    int count = 0;
4         do
5         :: (count < num_R) ->
6        atomic {
7          Rules[count].is_overquota = true;
8          if :: Rules[count].is_uninstallable -> Rules[count].is_uninstalled =
                true;
9             :: else -> skip;
10         fi
11         count++;
12           }
13        :: else ->
14              break;
15        od
16 }
```

Vitae

Alvaro Maza was born in Lima, Peru on April 15th, 1987. He received his Bachelor in Computer and Systems Engineering degree at "Universidad de San Martín de Porres" in Lima in 2009.

After university, Alvaro worked in the software development and telecommunication industry for four years for IBM and Avantica Technologies. In August 2013, he started his graduate studies in "Instituto Tecnológico y de Estudios Superiores de Monterrey" in Jalisco, Mexico and obtained an academic scholarship from the Organization of American States (OAS).

During his graduate studies, he did a semester abroad in Australia, taking 4 graduate courses in the University of Queensland. By December 2015, he finished his graduate studies and received his Master of Computer Science degree.

Currently, Alvaro is still working in the telecommunication industry for Sandvine, as a Solutions Architect.