Chapter

6

Sustainability-Oriented Policies applied to Network Management - A practical view of refinement and application of sustainability-oriented policies

Tereza C. M. B. Carvalho, Catalin Meirosu, Ana C. Riekstin, Marcelo C. Amaral, Guilherme C. Januário, Carlos H. A. Costa, Charles C. Miers, Denis Gabos, Enlai L. Cheng, Lucas B. Figueiredo

Abstract

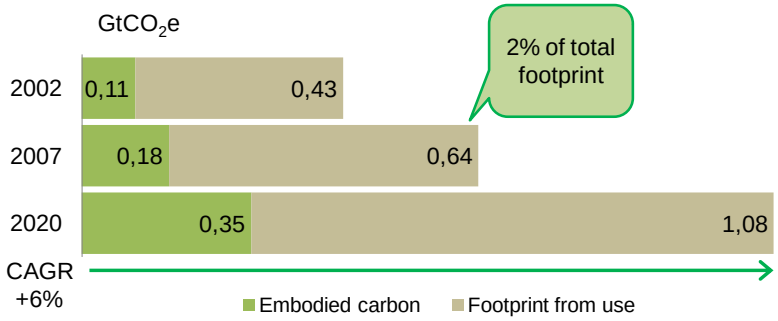
This course aims to present, from a theoretical and a practical point of view, the application of sustainability policies to network management and the available techniques and mechanisms for managing and refining such policies, considering quality of service and energy efficiency parameters. The sections are organized as follows: (i) introduction and concepts; (ii) policy-based network management and policy refinement regarding sustainability; (iii) policy frameworks, policy refinement methods, and related works; (iv) practical point of view and a case study; and (v) final considerations, including a course summary and future challenges.

6.1. Introduction

Network service providers (NSPs) have been facing increasing challenges to deal with the complexity incurred by an unprecedented demand for connectivity. To support new generation network infrastructures for the rapidly growing customer demand, NSPs constantly integrate new equipment, architectural solutions, and protocols [13]. As a consequence, NSPs have been developing sophisticated architectures and increasingly complex infrastructures that entail dealing with heterogeneous resources. This heterogeneity comes from the fact that most vendors have produced their own network management strategy, and almost all devices supply their own management facilities [38]. To deal with such problems, NSPs require more and more skilled and experienced human capital and the development of intelligent network management tools.

Another problem NSPs are facing is related to the world's growing concern with sustainability. Most of the time, NSPs over-provision resources to try to ensure QoS, thus

wasting computing resources and energy. According to Gartner, the total carbon footprint of the Information and Communication Technology (ICT) sector in 2007 was 830 MtCO2e, which corresponds to 2% of the estimated total emissions from human activities. Furthermore, due to the world's industries being increasingly reliant on large-scale data centers and outsourced services, this contribution is expected to grow at 6% yearly by 2020, as depicted in Figure 6.1. Of this total footprint, 10% is produced by data centers, 70% results from the networks, and the remaining 20% is due to other sources, such as offices, shops, etc. [13]. This work is concerned with network infrastructures, which is the sector with the highest contribution to carbon emissions.



*ICT includes PCs, networks, devices, printers, and data centers.
**CAGR = Compounded Annual Growth Rate

Figure 6.1. The global ICT footprint, including PCs, networks, devices, printers, and data centers [32].

The consequences of failing to address sustainability issues and complex infrastructures will lead to different problems, such as high electricity bills and greenhouse gas emissions, as well as dependency on highly skilled people and even constant failures, consuming ever larger chunks of ICT investment [79]. To address these challenges, it is necessary to develop network management tools along with sustainability measures. Future networks must be more flexible and able to autonomically reorganize themselves with the introduction of new types of equipment and services that reduce the necessity of human intervention [3], such as the use of a policy-based network management system.

According to Mazin Yousif [79], one of the most important efforts to reduce energy consumption is to improve the efficiency of the ICT infrastructure by using energy-efficient equipment and resources and by deploying autonomic power optimizations. In this scenario, the policy-based network management (PBNM) emerges as one of the key approaches, serving as an intelligent network management tool capable of applying energy efficiency through autonomic decision processes, besides simplifying the network management.

A policy is a set of rules that chooses a response to a specific condition in order to enforce behavioral or functional actions, as described in Figure 6.2. Strassner [67] described policy management as the usage of rules to accomplish decisions and cited policy-based network management as a way to define business needs and ensure that the network provides the services that its clients require. Furthermore, policies have been introduced in network management to develop an autonomic network control capable of administrating the network devices and resources based on predefined rules.

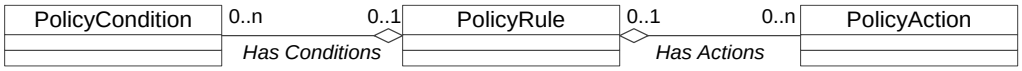


Figure 6.2. A simplistic policy model [67].

The policy behavior consists of three parts: the event part that triggers a specific rule, the condition part that logically analyzes the event and verifies whether the action part is activated or not, and the action part that in turn changes or updates the network. This system is called event-condition-action (ECA), which follows the general syntax "on event if condition do actions" [58]. There are two widely used types of policies. Authorization policies, also called security policies, are intended for access control systems, detection, alarm, notifications, and network administration. Obligation policies, mainly called quality of service (QoS) policies, are applied in traffic management and policing. There are usually three requirements for a policy to be considered as such: first, the policy must endure and be relatively static so that it does not becomes useless after being executed once; second, it must be declarative and so act under some circumstances, thus calling for some operations, but its functionalities must not be changed; and, lastly, it has to be derived from management goals [37].

According to Jude [40], in the first approaches to policy-based network management, it was believed that network applications and users would have the ability to control QoS. PBNM was the ultimate management tool in IT infrastructure management, but early adopters found that it was not as simple as previously thought. Instead of a general and easy-to-use technology, policy-based management proved to be a hard, expensive, and time-demanding approach for the first users.

Even demanding all those efforts to develop and deploy PBNM, the possibilities offered by using this approach for management were many. Any capability that allows controlling QoS, which was the first promise of PBNM, also allows detecting, identifying, and controlling packet flow in the network. These resources lead to security solutions and application performance improvement, and the fact that the network can be managed actively is attractive for Internet service providers. Also, the flexible provisioning provided by PBNM allows improving operations, enabling new types of services, and reducing costs. The last involves reducing the number of necessary hardware for the network operation and usage, resulting in a decrease in energy consumption and leading to a network operation with higher sustainability and less carbon footprint [39].

Policies can be at different levels of abstraction. The Internet Engineering Task Force (IETF) work group defined a division composed of five layers: business, system, network, device, and instance. This is known as the police continuum. The process of transforming high-level, abstract policy specifications into low-level, concrete ones is called "policy refinement" [54]. The goals of policy refinement are: to determine the necessary resources to satisfy the policy requirements, to transform the high-level policies into low-level enforceable operations, and to verify that the low-level policies meet the requirements specified in the high-level ones [9]. The first objective consists in mapping abstract entities defined in high-level policies into concrete devices. The second objective designates the necessity to ensure that the policies derived by the refinement process are supported by the system. Finally, the third objective

verifies that there is a process responsible for ensuring that the translation always decomposes abstract policies into more concrete ones.

This work evaluates some of the frameworks conceived to describe, manage, enforce, and verify policies, although not all these characteristics are present in all frameworks. The requirements used for evaluation were policy specification, analysis and enforcement; use of languages with formal semantics and extensibility; presence of domains and other forms of grouping; and distributed policy enforcement and meta-policy based on [59]. Sustainability-oriented policies support, policy refinement, and content availability for download are the other criteria taken into consideration in analyzing the frameworks in a sustainability-oriented policies scenario. The standards proposed by the IETF working group for defining a framework, namely, Ponder2, KAoS, Rei, and Procera, are described and compared in the next sections.

Besides the framework evaluation, a comparison is made between existing approaches to policy refinement. The evaluation is done according to the requirements specified in [37]. The refinement processes are evaluated for level of automacy, range of application, translation capabilities between levels of the police continuum, real-time interaction, support for sustainability applications, and availability for download.

An experiment involving three different scenarios using one of the frameworks presented demonstrates the use of the techniques and mechanisms available in a network management context aiming at energy-efficient network usage and sustainability. Section 6.6 shows some of the challenges faced by the authors in using the framework [49].

6.1.1. Course Objectives

This course aims to present, based on a theoretical and practical approach, the concepts related to sustainability policy refinement and, as a proof of concept, how to use the techniques and mechanisms available for sustainability policy refinement. Such an approach must be flexible and efficient, and it must make it easy to manage the ever more complex infrastructures of NSPs.

The following topics are featured: (1) introduction and concepts of policy-based management and policy refinement; (2) introduction to the concept of management focusing on energy efficiency and its challenges; (3) comparison among different frameworks for describing, refining, managing, and enforcing policies; (4) presentation and execution, using one of the presented frameworks, of an experiment on policy-oriented network management that aims at refining QoS and energy efficiency policies; and (5) a summary of the course and future challenges for the policy-based network management focused on sustainability.

6.1.1.1. Chapter organization

The remainder of this chapter is organized as follows.

Section 6.2 presents the state of the art in relation to network management driven by policy. This section describes the requirements and ways to save energy in the network. In addition, it shows how to build a sustainability-oriented policy and the requirements of such policy-building.

Section 6.3 is the core of this work. It describes the concepts of policy framework and policy refinement and presents the main works related to these ideas. These works are evaluated and compared, and their suitability for sustainability-oriented management is discussed.

Section 6.4 further describes a policy-based network management system, known as SustNMS. This section aims to provide a description of a real network management system, which will provide the basis for understanding how to perform policy refinement in such system. In addition, the described system will be used in practical experiments on policy refinement.

Section 6.5 describes the policy framework used in the policy-based network management system (SustNMS) implementation, Ponder2. This framework provides the functionalities to perform policy management and policy refinement from the network level to the instance level. In addition, Ponder2 is used in the practical experiment on policy refinement.

Section 6.6 presents a practical view of policy refinement. The case study is a sustainability-oriented policy refinement in SustNMS using Ponder2. The presented policy refinement is partially manual. We perform manually the refinement of a sustainability policy from the business level to the network level, which Ponder2 can understand and handle. Then, Ponder2 translates the policy into an enforceable version that can be applied in the network. This section also presents the results of the experiment on sustainable policy refinement and applies them in the network using SustNMS. The results show the amount of energy savings according to the described policy.

Section 6.7 presents the summary and final considerations.

6.2. Network management driven by policy: focus on energy efficiency

Due to the growing consumption of electricity and other resources by networks, it is important to define a sustainability-oriented strategy for network management. There are proposals for network management systems focused on energy efficiency that could take advantage of policy-based network management and policy refinement.

To define this sustainability-oriented strategy, it is important to understand the state of the art of policy-based network management, different policy levels, how sustainability can be applied to networks, and the related metrics and requirements. All these concepts, definitions, and descriptions are necessary to understand the *"policy refinement problem"* presented in Section 6.3.

6.2.1. Policy-based network management and network management systems

According to [81], one of the most important efforts to reduce energy consumption is to improve the efficiency of the ICT infrastructure by using energy-efficient (EE) equipment and resources and deploying autonomic power optimizations. In this scenario, policy-based network management (PBNM) emerges as one of the key approaches, serving as an intelligent network management tool capable of applying energy efficiency through autonomic decision processes, besides simplifying the network management. The general architecture of the PBNM includes four important modules [62]:

1. The management console acts as a user interface where it is possible to create and adjust policies. The language used for manipulating policies is called Policy Definition Language (PDL).

2. The policy repository is where the policies are stored.

3. The policy decision point (PDP) controls the conditions of policies and verifies whether any enforcement should be taken.

4. The policy enforcement point (PEP) is responsible for assuring that the instructions issued by the PDP are applied.

Figure 6.3 illustrates the PBNM architecture according to the Internet Engineering Task Force (IETF).

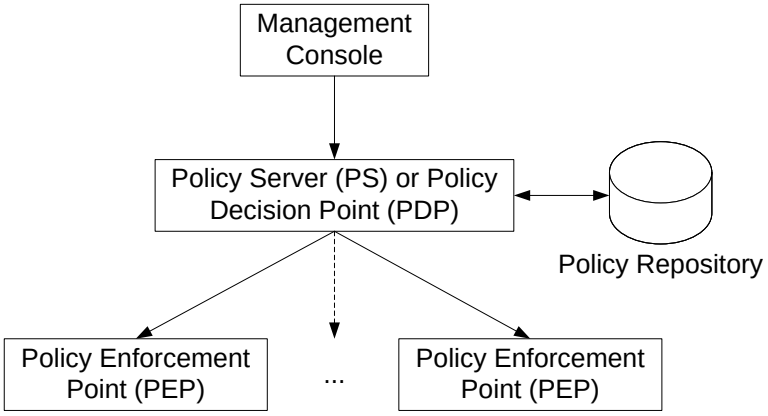


Figure 6.3. IETF Policy Architecture

The figure also presents the module organization as conceived for PBNM. PBNM emerged as a new approach to allow network service providers to have more control over the network usage and the services they provide. Its creation (in the late 1990s) was intended to provide QoS management for network applications and users on IT infrastructures [40]. However, instead of a general and easy-to-use technology, policy-based management turned out to be a hard, expensive, and time-demanding approach. There was a lack of solutions (standardized or proprietary) to turn policies into actions on equipment. Furthermore, network management systems (NMS) already controlled most of the network equipment, thus requiring PBNM to work in conjunction with the already available NMS to apply its policies.

6.2.2. Policy levels and the policy refinement problem

Network management systems consist of several elements, and their efficiency relies on how well these elements are orchestrated. NMS can be configured to provide a wide range of goals (e.g., to configure a single network link, manage a router, manage a data center, etc.). Network management driven by policy enables business rules and procedures to be translated into policies that configure and control networks and services. Usually, the orchestration of all elements is the result of a set of policies defined on high levels, which are enforced on each level until all the elements of the lowest level are reached (Figure 6.4) [67].

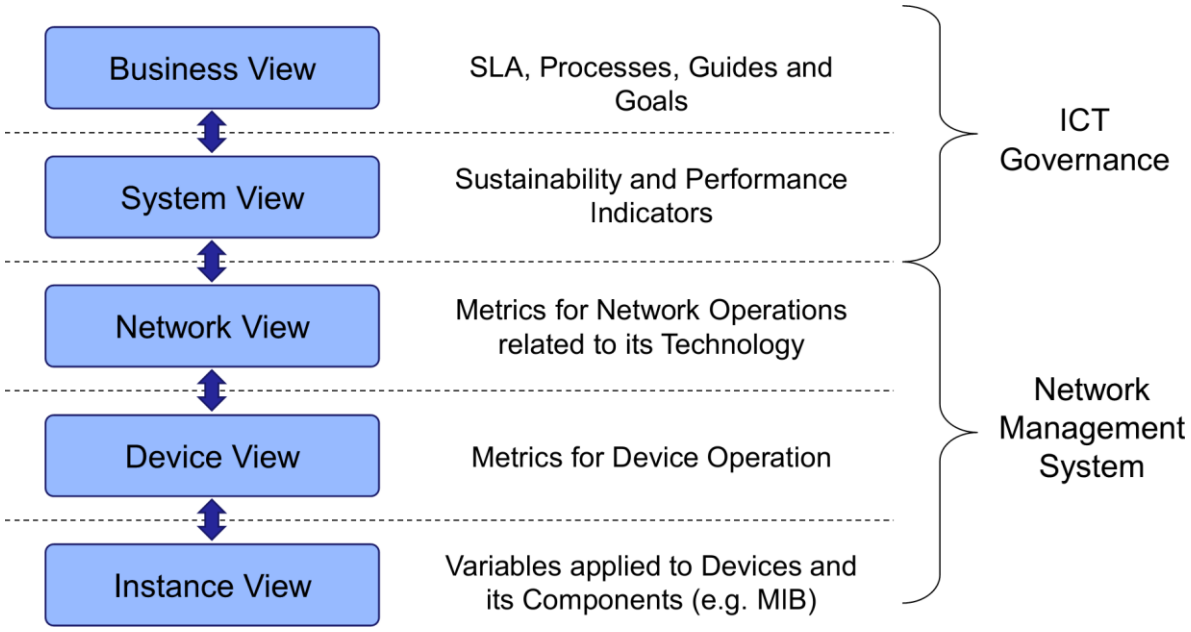


Figure 6.4. Translation between different types of policies [67]

Figure 6.4 illustrates how policies could be organized and their hierarchy from the highest level *(Business view)* to the lowest level *(Instance view)*. This hierarchy is known as the "policy continuum." The process of transforming a high-level, abstract policy specification into a low-level, concrete policy specification is called *"policy refinement."* Each policy view should be described on a satisfactory level of details according to its layer. Moreover, each policy has a defined number of states that can be assigned while the network management is being performed. Figure 6.5 presents an example of a policy life cycle model.

Network management policies can be composed to achieve efficiency in several ways, such as increased reliability, QoS, enhanced security, energy, etc. [4, 81]. Thus, network management driven by policy also poses as an approach to provide sustainability-oriented features to computer networks [77].

Sustainability-oriented goals are frequently measured in terms of energy efficiency and CO_2 emissions [34]. Because the most of the current efforts in sustainable computer networks are related to energy consumption [2, 14], and to keep the storyline easy to understand, this tutorial focuses on energy efficiency.

6.2.3. Sustainability and network management

Information and communication technologies contribute to global warming because the number of computers and the resources necessary to satisfy consumers increase year after year. Furthermore, there are issues related to the sustainability not only of the hosts (servers, computers, smartphones, tablets, etc.) attached to the network but also of the networks themselves [77]. To provide sustainability features to all these hosts is a giant and complex task because they belong to billions of different owners (i.e., companies, governments, end users, etc.) [60]. Sustainability features on computer networks can be achieved in different ways with different implications. Computer networks provide a wide range of services, implying different

kinds of requirements (security, reliability, QoS, etc.) according to each application demand. The addition of sustainability requirements should not imply compromising the service/application performance. The following taxonomy [14] provides the approaches undertaken to achieve energy efficiency in wire-line networks:

- Reengineering: Consists in optimizing or creating new components to obtain more sustainable hardware.

– Energy-efficient silicon/hardware: New processors that consume less energy or demand less cooling effort, for instance.

– Complexity reduction: Simplifying the hardware to demand less power and less raw material.

- Dynamic adaptation: Relies on controlling the network equipment resources (e.g., link bandwidth, equipment capacities, etc.) according to traffic load or policy constraints.

– Performance scaling: Some equipment allow power management on some components (e.g., tuning the clock frequency, setting the network interface packet transfer rate) as a way to manage its power consumption.

– Idle logic: Disabling those resources that are not necessary during certain periods also contributes to energy savings. Here it is also necessary to take into account the changes in the system reliability due to the equipment wake-up time.

- Smart Sleeping: Consists in the network devices supporting at least one power-saving mode and being able to exchange and interpret specific control messages, such as requests to put a device to sleep.

– Network proxying: This category assumes that there is a different class of equipment assigned to the task of changing the states of the managed equipment (wake-up, standby, or sleeping), only when it is needed.

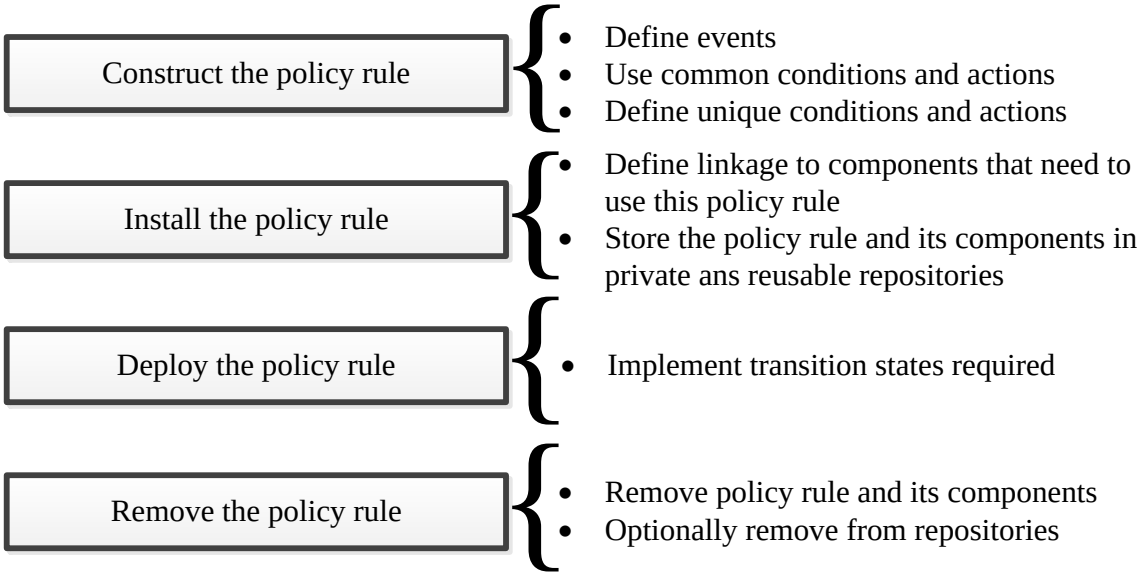


Figure 6.5. Example of a life cycle model of a policy rule [67]

Although there are several approaches to dealing with energy consumption management and verifying the performance evaluation, most of the standard equipment available today does not have most of the features that could allow implementing such approaches. There is the possibility to get some power consumption data on the equipment by attaching specific power measurement devices to it. However, the actions of the network management system applied to the equipment are limited to a few procedures, which can compromise the granularity and dynamicity of the network management system. On the other hand, the measurements will provide some power management to equipment that did not previously have such.

There is also equipment providing information that contributes to the NMS achieving a good level of dynamicity. The information on network equipment is usually provided by:

- Single Network Management Protocol (SNMP): The standard Management Information Base (MIB) provides access and configuration (SNMPv2 and higher) to several resources of the equipment. Unfortunately, there is no standard MIB for power management on network devices. There is a draft proposal of a power- and energy-monitoring MIB [17] that could be adopted if the equipment is compliant with the draft.
- Equipment proprietary interface. Due to the lack of a standardized MIB for power and energy monitoring, some manufacturers provide access to this kind of resource through specific command line requests.

There are also network layer protocols designed to contribute to energy consumption reduction. The IEEE 803.3az [18], called the energy-efficient Ethernet (EEE), is another approach to achieving a more sustainable network. However, its efficiency relies on the adoption of the protocol in a large amount of network equipment.

Analyzing the available resources, it is possible to identify that there are different approaches to obtain information on the power management and energy consumption of network devices. Furthermore, some equipment may provide more management and information than others with limited network management resources. Some information could be used to compose sustainability-oriented policies. These policies can take into account even CO_2 parameters when the information is provided by any of the specified resources. This course considers only energy efficiency, but CO_2 could be an additional or the main parameter (business rules and the available information allow one to choose the best option). Finally, it is clear that there is a significant amount of information that can be gathered from the equipment. Such information should be computed based on some metric to get the measurable results.

6.2.4. Green Metrics

The identification and definition of metrics suitable for evaluating the sustainability of computer networks are the focus of several researches related to green ICT [8, 25]. These metrics are called green metrics and can be organized according to different criteria. One classification proposal defines two main types of green metrics [11]:

- Equipment level: The amount of power/resources per equipment, including the rates for equipment with specific features.
- Facility level: The amounts and rates for consolidated systems (e.g., data centers) composed of several devices.

There is a considerable amount of information that can be used to elaborate the criteria and define the metrics (Section 6.2.3). However, the lack of standardization leads to a complex task of identifying which information on each equipment is available in order to verify if the requested metric information is satisfied. Table 6.1 presents a set of metrics already identified [77].

The columns of Table 6.1 present the equations for obtaining each metric, as well as its target. It is relevant to note that some metrics do not provide a specific unit (ratio and percentage). Thus, it is necessary to identify baseline values in order to evaluate the values of these metrics.

The metrics presented in Table 6.1 can be grouped according to the types defined in [11]:

- Equipment level: Energy Consumption Rating (ECR), Consumer Consumption Rating (CCR), Telecommunications Energy Efficiency Ratio (TEER), Telecommunications Equipment Energy Efficiency Ratio (TEEER), Normalized Power Consumption (NPC), and Power Consumption per Line of Broadband (P_{BBline}).
- Facility level: Power Usage Effectiveness (PUE), Data Center infrastructure Efficiency (DCiE), and Data Center Productivity (DCP).

Green metrics are an important component of network management driven by sustainability-oriented policies because the measured values can be used to evaluate the efficiency of policies. Each metric indicates a set of information that will be required for its use and to pass the results to higher levels. The adoption of green metrics can also provide a useful contribution to network equipment vendors and NSPs in identifying which information should be provided in each category of equipment.

6.2.5. Requirements for using sustainable policies

The requirements of sustainability-oriented policy aim at defining some boundaries and conditions in which the aforementioned approaches to energy efficiency should be used. The definition of these conditions as boundaries depends on a complex set of elements that changes its status dynamically, and it requires the analysis of each scenario to identify the aspects that must be considered.

Table 6.1. Green metrics [77]

Metric	Full name	Creator	Targets	Computation	Units	Remarks
PUE	Power Usage Effectiveness	Green Grid	Data Center	$PUE = \frac{Total\ Facility\ Power}{IT\ Equipment\ Power}$	Ratio	Ranging from 1 to infinite
DCiE	Data Center infrastructure Efficiency	Green Grid	Data Center	$DCiE = \frac{1}{PUE} = \frac{IT\ Equipment\ Power}{Total\ Facility\ Power} \times 100\%$	Percentage	Ranging from 0 to 100%
DCP	Data Center Productivity	Green Grid	Data Center	$DCP = \frac{Useful\ Work}{Total\ Facility\ Power}$	Ratio	Ranging from 1 to infinite
ECR	Energy Consumption Rating	ECRInitiate, (ISIA, Juniper)	ISP, ICT enterprises	$ECR = \frac{Energy\ Consunsumption}{EffectiveSystemCapacity}$	Watt/Gbps	Energy normalized to capacity
ECRW	ECR Weighted	ECRInitiate, (ISIA, Juniper)	ISP, ICT enterprises	$ECRW = \frac{0.35 \times E_f + 0.4 \times E_h + 0.25 \times E_i}{T_f}$	Watt/Gbps	E_f, E_h, and E_i are the energy consumption in full-load, half-load, and idle modes respectively. T_f is the effective throughput
TEER	Telecommunications Energy Efficiency Ratio	ATIS	General, Server, Transport	$TEER = \frac{\sum_{i=1}^{n} D_i}{\sum_{j=1}^{m} \left(\frac{P_{0j} + P_{50j} + P_{100j}}{3} \right)}$	Gbps/Watt	D_i is the data rate of each interface i; P_{0j}, P_{50j}, and P_{100j} are the power of the module j at data utilization of %, 50%, and 100% respectively
TEEER	Telecommunications Equipment Energy Efficiency Ratio	Verizon NEBS	Transport, Switch, Router, Access, Amplifier	$TEEER = log\left(\frac{0.35 \times P_{MAX} + 0.40 \times P_{50} + 0.25 \times P_{sleep}}{Throughput} \right)$	log(Watt/Gbps)	Referring ECRW and TEER, P_{MAX},P_{50}, and P_{sleep} are the power consumption at 100%, 50%, and 0% load utilization. (formula may change base on different types of devices)
CCR	Consumer Consumption Rating	Juniper	Consumer Network devices	$CCR = \frac{E}{\sum A(j)}$	rad (dimensionless)	E is power rating of a consumer network device; A is energy allowance per function. j is the set of all allowances claimed. Value 1 matches an average device
EPI	Energy Proportionality Index	HP Labs	Network Devices	$EPI = \left(\frac{PM - PI}{PM} \right) \times 100\%$	Percentage	PI is the power consumption at idle mode, PM is the power consumption at maximum workload
WattsPerVLL	Watts Per VLL	Ericsson	IP Networks	$WattsPerVLL = \frac{Power\ Consumption}{Number\ of\ VLLs}$	Watt/line	Used for Virtual Leased Line (point-to-point Ethernet-line) services based on the *Number of VLLs*
WattsPerMAC	Watts Per MAC	Ericsson	IP Networks	$WattsPerMAC = \frac{Power\ Consumption}{Number\ of\ MAC\ ports}$	Watt/port	Used for MAC address (multipoint Ethernet-lan) services based on the *Number of MAC ports*
P_{BBline}	Power consumption per line of Broadband	ETSI	Broadband telecommunication networks equipment	$P_{BBline} = \frac{P_{BBeq}}{Number\ of\ Sub\ Lines}$	Watt/subscriber per line or Watt/port	P_{BBline} is the Power consumption of fully equipped broadband equipment, *Number of Sub Lines* is the maximum number of subscriber lines supported
NPC	Normalized Power Consumption	ETSI	Broadband telecommunication networks equipment	$NPC = \frac{1000 \times P_{BBline}}{Bit\ Rate \times Line\ Length}$	Watt/(Mbps × km)	Normalized power consumption per line for broadband network equipment based on the *LineLength*

From the sustainability perspective, the goal is to expend the lowest amount of energy possible. However, less power usually means less equipment and, consequently, less reliability [25] or performance.

Network operators want to provide available and reliable services to accomplish service level agreements (SLAs) and to avoid fines or losing customers. Network providers usually overprovision their networks to support the traffic demand during peak hours. Furthermore, the expansion of the optical network and the increasing adoption of applications using high-

definition content (video streaming, video games, etc.) are demanding more bandwidth each month (the situation could be more critical if we consider the recent commercial launch of ultra-HD screens and TVs for home users). As a consequence, networks' energy consumption could become a constraint to network operators [45].

The aforementioned perspectives present a trade-off between energy consumption, reliability, and QoS. Moreover, the presented facts also indicate that network operators may be motivated to reduce the energy consumption of their network infrastructures. This is initially due to the desire to keep the network infrastructure running with lower costs, but it could also be accomplished with other sustainability-oriented policies to improve the benefits. In this scenario, the clients usually have minor participation in the optimization of the network usage to obtain lower power consumption.

The trade-off between performance and energy consumption poses as a complex equation with several variables and requirements. The dark region in Figure 6.6 presents the energy performance curve for mobile networks.

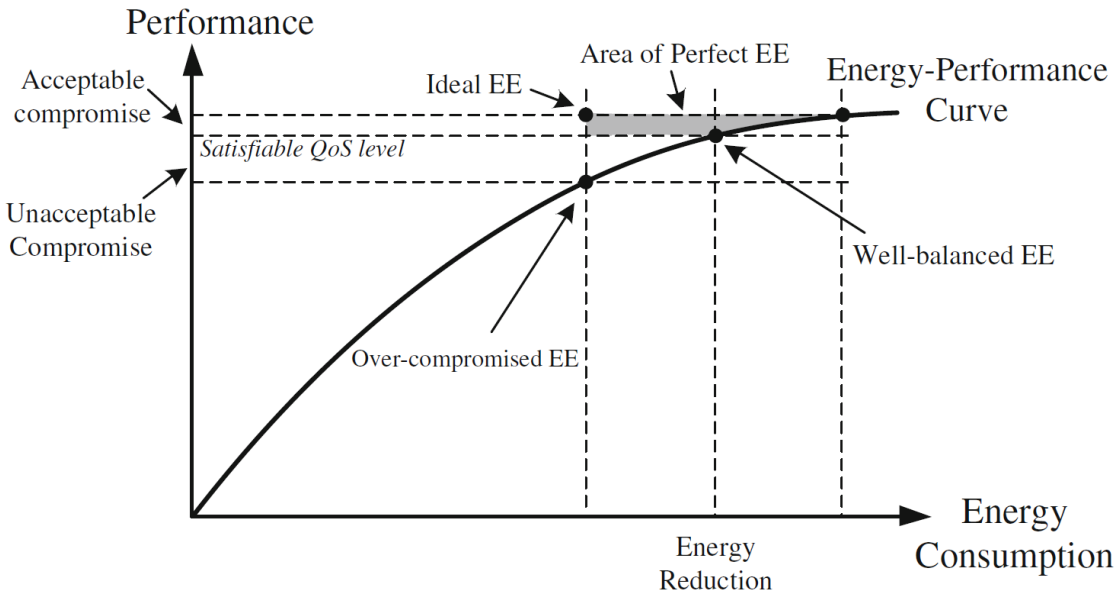


Figure 6.6. Typical energy performance curve for mobile networks [77]

Figure 6.6 shows, in the shaded portion, the ideal energy efficiency at which performance and QoS are on acceptable levels while still saving energy. However, the energy performance curve for each network medium (optical, copper, satellites, power line, hybrids, etc.) presents different results.

It is clear that the first requirement for using sustainability-oriented policies is related to getting information about the network equipment as well as to applying some configurations. These information and configuration requirements could include:

- information of the life span of the network equipment and accessories;
- the capacity to adjust the equipment power consumption based on its performance;
- the equipment features that allow putting some components in sleep or wake-up mode;

- the available equipment resources for measuring the power consumption and additional information that contributes to the sustainability aims; and
- the capacity to configure the equipment parameters to operate in different configurations.

The first requirement is related to the lower levels of the policy continuum (Figure 6.4). It allows obtaining information and modifying the network equipment on the lower levels according to the policies defined on the higher levels.

The second requirement concerns the policy specification and definition on the higher levels. Because there is a trade-off related to energy consumption vs. QoS and performance, the information related to this trade-off needs to be gathered from the lower levels (first requirement) and processed on higher levels to return the right action to the lowest levels. The main goal is to identify the satisfactory levels of reliability and QoS while applying the sustainability-oriented policies in order to avoid undesirable behavior of the provided services. NSPs usually provide a wide range of services, each with different requirements of reliability and QoS. Thus, an NSP needs to determine which criteria to use in defining policies. This definition can be oriented to the network flows or to specific services. The definition based on network flows considers the amount of traffic being transferred between network hosts from the source to the destination. The application data that compose the flow are not considered individually. On the other hand, the criteria based on specific services aim to identify each service, and the result of the optimization should be the intersection of the sustainable practices suitable for all services.

Analyzing these two options, it is possible to understand that the criteria based on specific services allow fine-tuning of the necessary resources and that sustainability-oriented policies can explore this. However, analyzing all services may be a complex and expensive task if the system supports a lot of services and their usage is dynamic. Thus, this type of criteria is suitable for systems with well-defined services, such as IPTV. The criteria based on network flows do not require the same amount of effort to identify each service of the former type but demands dynamicity to quickly identify the network flows and apply the right policies. Thus, the criteria based on network flows are more suitable for dealing with networks that provide different types of services.

As a way to "put all the requirements together," it is important to understand how to manage policies and methods in order to refine them, as will be presented in the next section.

6.3. Policy framework, policy refinement, and related works

As discussed in previous sections, the network operation is governed by business logic, which specifies and enforces policies regulating access control, service level agreement (SLA), and dynamic resource provisioning in applications [59]. Policy-based management is a suitable approach to dynamically change the behavior of a managed system according to the requirements of a changing context without modifying the system implementation. A policy should facilitate the network management and make it more flexible, being the key to applying sustainability-oriented decisions to the network.

A policy framework is commonly used to describe, manage, monitor, enforce, and verify the policies. The framework should facilitate the work of the network manager by providing a way to describe policies in a constrained natural language [48]. However, a policy described in a

high-level language, as a constrained natural language, must be refined in a language readable to the framework and network devices.

Hang et al. [80] pointed out the increasing importance of the development of tools for policy refinement. Policy refinement is the automated decomposition from a high-level, device-independent language to the device-specific language of the various enforcement points, that is, the concrete objects and actions to be performed [23]. A policy refinement framework should allow policy administrators to write policies in high-level languages and provide automatic or semiautomatic tools for refining the high-level policies into versions that can be enforced by the existing policy systems [48].

The main objectives of policy refinement, according to Moffet [54], are: (1) to define the resources needed to satisfy the requirements of the policies, (2) to translate high-level policies into operational policies that a system can enforce, and (3) to verify whether the low-level policies are in accordance with the high-level ones. The main challenge in policy refinement is the translation complexity, for which, although progress has been made, no standard solution is available yet [22]. Moreover, policy-based management has not yet gained a wide range of applicability because the policy refinement process is still considerably complicated [37] and it is not completely automated. However, there are approaches that claim to solve some part of the policy refinement problem. While some of them rely on manual operations in the policy refinement process, these works have implemented useful methods of policy refinement and are worth studying.

Neither the management nor the refinement of sustainability-oriented policies is completely understood; however, the existing approaches seem to fulfill the requirements of specific applications. The majority of policy frameworks do not include policy refinement, which is frequently a separate process. It is necessary to perform refinement (which is not necessarily automatic) for a high-level policy and then deliver the result in a language that the policy framework is able to interpret, as shown in Figure 6.7.

Figure 6.7 presents a general architecture for policy management and refinement. When the manager writes a high-level (or business) policy, this policy is stored and sent to the policy refinement module (PRM). The PRM decomposes the high-level policy into a low-level one that the policy framework module (PFM) can handle. The PFM receives the policy as well as indexes about the network. These indexes may be related to QoS, security, sustainability, and so on. Based on these indexes, the PFM makes decisions related to the manager's goal, which was described in the business policy. The decisions are sent, as events, to the policy enforcement module, which generates the machine-executable command (instance-level policy) to either apply the changes or not. In the following, we present and compare the existing policy frameworks and refinement methods. We also describe the criteria used to compare them.

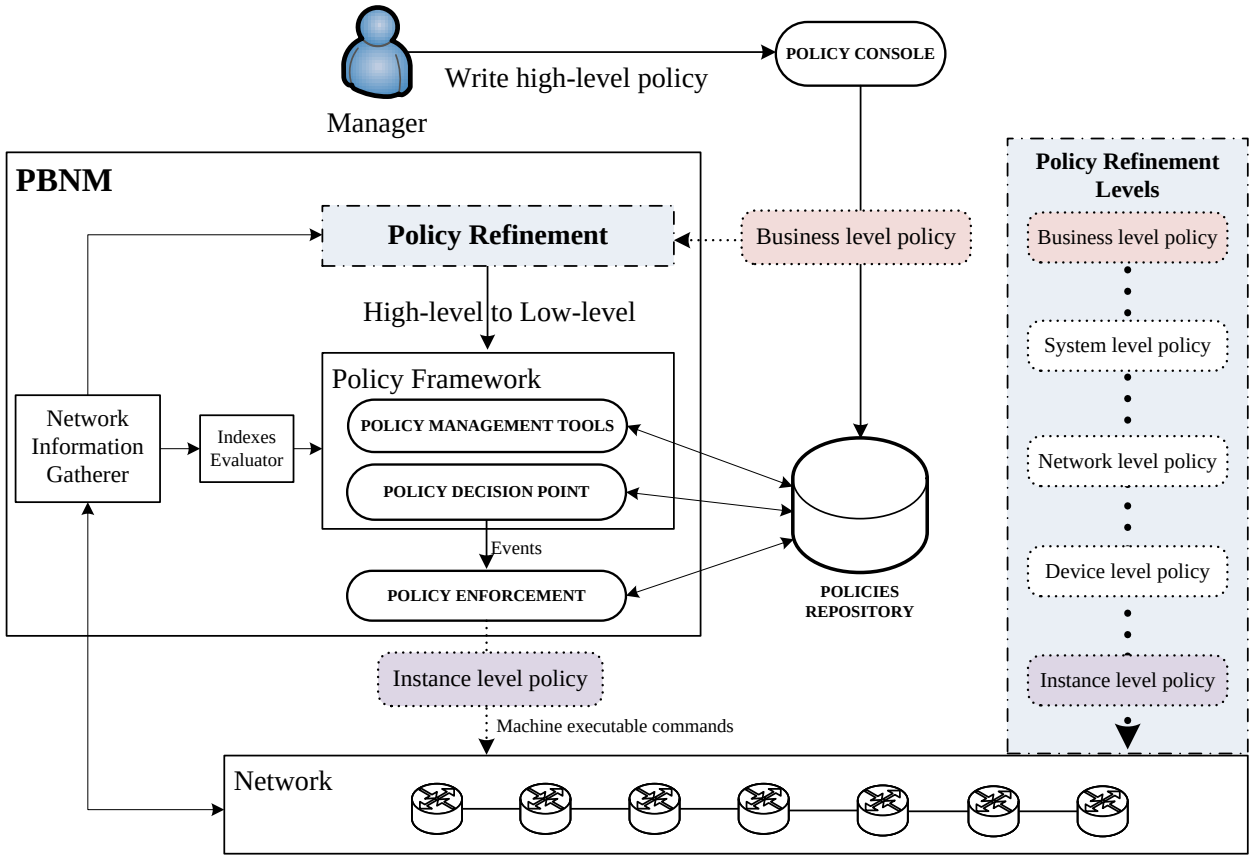


Figure 6.7. General architecture for policy refinement

6.3.1. Evaluation criteria

The criteria for analyzing the policy frameworks and refinement processes are based on [59] and [37]. The first describes a set of general requirements for policy frameworks. The second classifies the methods of policy refinement. In addition, we included in the analysis other requirements related to policy refinement and sustainability-oriented policies, which are related to the goals of this work.

6.3.1.1. Policy framework requirements

In the following, we describe the requirements that a policy framework should fulfill based on [59].

Policy specification, analysis, and enforcement. A complete policy framework should consist of five parts: *(i)* a *policy specification language*, that is, a set of expressions used to describe a policy; *(ii)* a *policy deployment model* for distributing the policies to the enforcement points; *(iii)* a *policy analysis mechanism* for making the right policy enforcement decisions; *(iv)* a *policy monitoring and enforcement mechanism* that identifies and executes the policy actions; and *(v)* a *policy verification* method that includes a consistency analysis of redundancy, resolution, and detection of conflicts to ensure the correctness and quality of the policies specified.

In addition to these five parts, we argue that a complete policy framework should also include *(vi)* a *policy refinement model* that allows the description of a high-level policy and the transformation of the policy to all the continuum levels.

Languages with formal semantics and extensibility. Performing automatic analysis and refinement is easier for formal policies, such as description logic and event calculus, than for informal ones. In addition, a policy language should be easy to extend in order to support new expressions (i.e., to include expressions related to sustainability).

Domains and other forms of grouping. To simplify the policy management, it is desirable for a policy framework to support the grouping of elements. The organization of domains allows the hierarchical grouping of objects. If the policy framework has grouping structures, the management actions can be performed on a set of similar equipment rather than on individual ones.

Distributed policy enforcement. A policy framework that has distributed policy enforcement is suitable for addressing flexibility and scalability constraints. In addition, the overload can be minimized and fault tolerance may be increased when there are many policy enforcement points.

Meta-policy. Meta-policy is a policy about policies; it allows defining rules and constraints on the policies themselves, facilitating the policy analysis and conflict resolution.

Moreover, the policy framework should include requirements for describing, managing, and refining sustainability-oriented policies.

Sustainability-oriented policies support. Sustainability-oriented policies specify the actions that must be performed when certain events occur and provide the ability to respond to changes. For instance, sustainability-oriented policies define which actions must be specified when the network load changes and who must execute those actions. Sustainability-oriented policies may be described as obligation policies, but with less priority than pure obligation ones. Another sustainability requirement is that the policy enforcement points must know how to perform sustainability actions, such as setting a device to sleep mode. This depends on the vendor implementation.

Policy refinement. This makes the policy description easier by permitting the usage of a constrained natural language and translating this into a low-level language for executing commands in the network devices.

Downloadable. We include this as an additional requirement that specifies whether it is possible to access (or download) the framework. For many of these frameworks, we tried to get the source code or an executable file, but our attempts were not successful; some are not available on the Internet, and others have complicated licensing requirements that prevented us from gaining access to them.

6.3.1.2. Policy refinement requirements

The policy refinement approaches should include the requirements specified by Hu et al. [37], which are *automatic* and *non-domain-specific* refinement. In addition, they should refine policy from the *business level*, perform *online (real-time) transformations,* and support *sustainability*-oriented policy refinement.

Automatic. Performing automatic policy refinement, which consists in decomposing and operationalizing the policy from a high to a low level without manual intervention, is complex. Automatic refinement may leave semantic gaps because experts usually perform policy refinement manually by translating downward from a higher level and writing the lower levels based on the meaning of the higher levels [61].

Non-domain-specific. Some policy refinement techniques may be specific to a domain, for example, the refinement of only authority or performance policies, or of only policies related to software-defined networking (SDN) or robotics contexts. Ideally, a policy refinement approach should not be domain-specific.

Business level. A complete policy refinement approach should start from the business level, for instance, a constrained natural language, and go down to the instance level, that is, a set of machine-executable commands.

Online transformation. The policy transformation may be real-time, in which there is an online monitoring component verifying the behavior of its agents to ensure that the objectives are being met [10]. It gathers online system information and translates the policy according to the current system status. Offline or static transformation uses static data and predetermined transformation rules to translate the policies.

Sustainability. The refinement process should be flexible enough to be used in the sustainability domain. It may include rules that understand the sustainability features described in the policy. In addition, the low level that represents machine-executable commands needs to be in accordance with the energy-efficiency features of each vendor specification.

Downloadable. As described in the policy framework requirement, we also included this requirement, which represents whether the implementation of the approach is feasible or not.

6.3.2. Existing approaches

In the following, we present and compare the existing approaches to policy framework and refinement considering the aforementioned criteria. There are many policy frameworks for different application scenarios, and we analyzed the most complete ones according to [59]: the IETF (as baseline) [67], Ponder2 [49], KAoS [31], and Rei [70]. We also included the Procera [76] language in the review, it being a very important work related to policies in software-defined networking (SDN). For the policy refinement, we used as basis the classification made by [37] and selected one work related to the classification, such as Craven et al. [23], Rubio-Loyola et al. [63], Beigi et al [10], Udupi et al. [72], KAoS [31], and Uszok et al. (KAOS) [73].

6.3.2.1. Policy frameworks

In the following, the previously cited policy frameworks are presented and compared, as shown in Table 6.2.

Table 6.2. Features comparison between policy frameworks

Policy Frameworks		IETF	Ponder2	KAoS	REI	Procera
General Characteristics	Policy specification, analysis, and enforcement	*partial*	✓	✓	*partial*	✓
	Languages with formal semantics and extensibility	✗	✓	✓	✓	✓
	Distributed policy	✓	✓	✓	*partial*	✓
	Meta-policy	✗	✗	✗	✓	✗
Specific Carachteristics	Sustainability requirements support	✓	✓	✓	✓	✓
	Policy refinement	✗	✗	✓	✗	✓
	Downloadable	✗	✓	✗	*partial*	✓

(1) IETF

The IETF policy framework is vendor-independent and proposes the representation, management, sharing, and reuse of policies and policy information [65]. It includes the grouping of policy terminologies, a model for the policy, and a policy architecture meta-model. The most flexible part of the framework of the IETF is its definition language, called Policy Framework Definition Language (PFDL) [69], which is a network administrator's tool for expressing various kinds of network policies, such as regarding security, QoS, restrictions, etc. Nevertheless, the PFDL was discontinued due to the IETF suspending work on policy definition language because there was no consensus [56].

The IETF conceptual model of a policy-based system consists of a policy console, a policy repository (PR), a policy decision point (PDP), and a policy enforcement point (PEP). The policy manager controls the PR through a PC and is responsible for creating and adjusting policy objects. These policy objects are stored and classified in the policy repository. The PDP, also called the policy server, analyzes the environment to verify which policies should be triggered. Lastly, the PEPs enforce the policies chosen by the PDP. Figure 6.8 shows the IETF's policy deployment model.

Because the IETF has no specific policy languages, it proposes the Common Information Model (CIM) [1] and the Policy Core Information Model (PCIM) [29] as general representations of a managed system and as representations of policy-related information. In those models, systems, services, and users are defined as object-oriented classes.

All IETF policies follow the same "if conditions then actions" paradigm. An example given in [65] illustrates a high-level business rule: "On any interface on which these rules apply, guarantee at least 30% of the interface bandwidth to UDP flows, and at least 40% of the interface bandwidth to TCP flows." This is translated to: "If (IP protocol is UDP) THEN (guarantee 30% of available BW) If (IP protocol is TCP) THEN (guarantee 40% of available BW)."

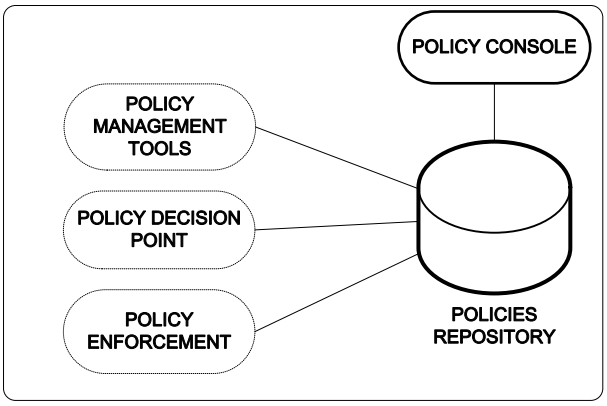


Figure 6.8. IETF Policy Deployment Model

In the IETF, the managed objects are not deployed to a domain, and policies cannot be applied to a domain. Instead, the elements are managed through the use of roles. A set of roles is assigned to each managed element so the policies are applied to the roles. Despite the fact that IETF policies are managed objects, and as such are also applicable to policies, the IETF does not support meta-policies.

Although its primary focus has been on representing QoS and IP security policies, the IETF is capable of developing any kind of policies [29]. For this reason, it is possible to propose sustainability-oriented policies through object-oriented representation.

The conditions and actions are stored separately; thus, the combination of different policies and actions can generate multiple rules. The enforcement of some policies may interfere in actions related to other policies, causing conflicts. To solve this problem, it is possible to assign a priority for each policy so that policies with a higher priority will be enforced over others. Although the abstract model of the IETF may be very convenient, the lack of a specified language complicates the process of policy verification and refinement, making the framework not very applicable [59].

(2) Ponder2

Ponder2 is a declarative, object-oriented, runtime policy management framework. It supports event-based runtime controls used to specify management and security policies in networks and distributed systems. Ponder2 has two types of policies: authorization and obligation. Authorization policies, being essentially access control policies, allow or deny message exchange among objects and determine which activities a domain member can execute over which objects. Obligation policies, which include ECA (Event-Condition-Action) policies, specify which actions the agent must perform..

Ponder2 is based on the self-managed cell (SMC) concept. An SMC is defined as a set of hardware and software components that constitute an administrative domain able to work autonomously and manage itself [71]. Ponder2 can be seen as the policy service for the SMC, which must also have an event bus service and a discovery service. The Ponder2 policy service is constituted by: a domain service (a structure for managing/organizing objects), an obligation policy interpreter, a command interpreter (able to read PonderTalk, a high-level language used to

control Ponder2), and an authorization enforcement mechanism (to deal with positive and negative authorization policies). Because Ponder2 has an obligation policy interpreter, it can support sustainability-oriented policies.

Ponder2 acts as a Policy Decision Point and, according to [59], each Policy Enforcement Point needs to implement a policy enforcement interface. The operation of ECA policies (obligation) down to the device and instance levels must also be implemented because Ponder2 only decides on "what to do" at the network level and only implements authorization enforcement. There is no explicit mention of meta-policies in Ponder2, but the domains can contain some information about policies. Ponder2 partially supports business-oriented policy specification [59] and works at the network level of the policy continuum; thus, the translation of the previous levels must be executed methodologically, such as in [15], in a semiautomated way, as in [62], or in other ways, as in [23]. As such, the editor is expected to have a deep understanding about the system access control and operation. Later in this chapter, Ponder2 is discussed in detail to serve as a proof of concept.

(3) KAoS

KAoS (Knowledgeable Agent-oriented System), developed at the Florida Institute for Human and Machine Cognition (IHMC), is a policy services framework able to run in different environments. It allows policy description in a human-expressible language and, at the same time, a machine-enforceable form. The framework has a graphical interface for defining policies called the KPAT (KAoS Policy Administration Tool). KAoS also has a domain service that enables hierarchical grouping of actors and equipment to make the policy administration easier [59]. The concepts in KAoS are defined using an ontology system and represented by the Web Ontology Language (OWL). KAoS supports four policy types: PositiveAuthorization, NegativeAuthorization, PositiveObligation, and NegativeObligation, but it can be extended to consider other types [59], such as sustainability-oriented policies. Complex policy constructs are built using the four basic policy types rather than through specific mechanisms, such as meta-policies [70]. The policies defined using OWL are compiled in a format suitable for monitoring and enforcement [74]. Figure 6.9 illustrates the KAoS policy service architecture.

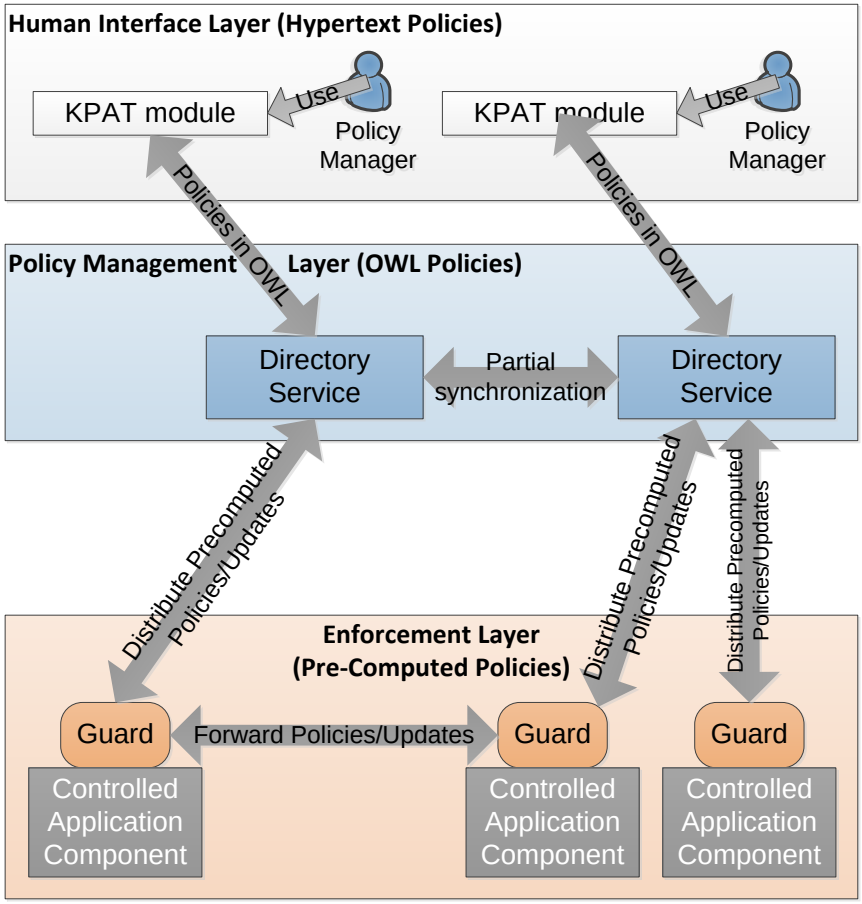


Figure 6.9. KAoS Policy Service Architecture [74]

(4) Rei

Rei is a policy framework developed by Hewlett-Packard that integrates support for policy specification, analysis, and reasoning in pervasive computing applications [70]. Its language, based on OWL-Lite, provides a few constructs founded on deontic principles, allowing users to express and represent concepts of rights, prohibitions, obligations, and dispensations. These constructs, however, are general enough to cover a range of policies by allowing the behavior of an entity to be modified. This simple policy language is not tied to any specific application, thus allowing different elements of a pervasive environment to understand and interpret Rei policies in the correct way. Domain-dependent and application-dependent ontology can also be plugged into Rei's existing hierarchy as subclasses of predefined classes. Rei has a policy engine that accepts policies in first-order logic (FOL) and the Resource Description Framework (RDF) [41].

The policy deployment model for Rei relies on an engine that reasons about the policy specifications and replies to queries. However, the engine does not provide an enforcement model; it was not designed for this and does not support policy refinement. The policy control is decentralized. The framework does not provide a graphical policy editor, making policy specification less user-friendly [59].

The engine has some capabilities for conflict resolution among policies, such as marking two policies that have conflicting modalities or an overlap in subject, target, and action. To resolve conflicts, Rei makes use of meta-policies that define priorities on policies and/or sets precedence relations between policy modalities that regulate conflicting policies statically. The two main types of meta-policies supported by Rei are meta-policies for "defaults," which describe the default behavior of the policy, and meta-meta policies, which are intended for conflict resolution [59]. The policy engine core of this framework has been implemented in SICStus Prolog and uses a Java wrapper. The parsing of the Resource Description Framework (RDF) is done by Jena, a Java RDF API (application programming interface) for RDF model and syntax specification [52].

The Rei framework is suitable for applications regarding sustainability-oriented policies because its ontology system can support changes in the application target or in the policies themselves by the addition of new ontology classes into the model. Its support for obligation policies and the use of meta-policies for conflict resolution constitute a good approach to setting precedence and priority between QoS and energy efficiency policies. However, the lack of an enforcement model is notable, and the use of Rei for management aimed at sustainability requires implementing this feature or adopting a management system capable of performing sustainability actions that uses Rei frameworks to drive its actions.

This is a defunct project that started in April 2002 and ended in May 2005. Nevertheless, there are still some contents on OWL available for download, such as examples, ontologies, and specifications [33].

(5) Procera

Procera is a high-level control architecture intended to offer more expressiveness to software-defined networking (SDN) policies. It is an alternative to conventional methods that require a low-level and vendor-specific configuration and are therefore complex and error-prone. Procera includes a declarative policy language inspired by functional reactive programming. The architecture allows operators to express different kinds of policies, from intrusion detection triggering actions, to load balancing systems choosing servers, to access denial after the occurrence of complex temporal bandwidth conditions.

The aforementioned policies, which involve user authentication, time of day, bandwidth, or server load, are triggered by events that are not inherent to OpenFlow. Procera is reactive because it reacts to dynamic changes in the network, a property that is highly desirable for many realistic policies [76].

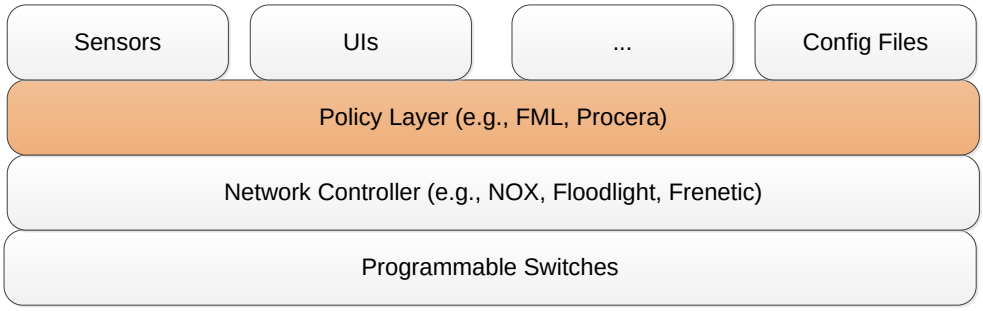


Figure 6.10. Architecture of a Procera system [76]

Figure 6.10 depicts the architecture of a Procera system. A typical implementation of the system makes use of OpenFlow programmable switches on the lowest level. OpenFlow switches provide information to and are controlled by the network control layer; Procera lies on the layer above the network control. The network administrator or analyst interacts with Procera through the interactive upper layer. This layer is not mandatory, but because Procera itself includes a policy definition language, the definition of policies can still be simplified by the layer above it.

The policy layer provides flexible and efficient directives to the network control layer. Such directives help keep the network control simple enough to be efficient and sufficiently abstract to be flexible. The complexity of implementing a policy layer similar to that described demands that the layer be as complex as an engine.

The developers of Procera point out as an advantage of a declarative and reactive language the fact that through a language that lacks such attributes, one can define neither recursion nor arithmetic constraints or functions. Another key advantage of the language used by Procera over other policy languages, e.g., Flow Management Language (FML), is its ability to define, rather than merely observe, policies that affect the network state. The FML is a declarative policy language for managing the configuration of networks [36]; as stated, it does not provide the capability to configure the network. In addition, Procera enhances the programming logic without substantially complicating it, as frame calculus [53] and event calculus [44] do.

By using functional reactive programming (FRP), the network administrator can describe parameters whose values depend on event histories or on the history of the values of other parameters. FRP also distinguishes Procera from FML by its inherent ability to invoke arbitrary functions, thus granting Procera support for arithmetic constraints. As expected, the Procera language is inspired by other systems [76], from which the following key features were aggregated:

- a core language based on functional reactive programming, according to Yampa [21];
- event comprehension to manipulate event streams, inspired by the Haskell language;
- windowing and aggregation signal functions, inspired by streaming database systems [7]; and
- the use of function values to represent high-level policy.

Procera is not just inspired by the Haskell language but is also an embedded domain-specific language within it. Therefore, Procera can make use of the data types and constructs of Haskell. As a last remark about Procera, we point out that with this architecture, one can filter, transform, merge, or even join event streams. A useful way to conveniently express such operations is by event comprehension, a concept that is very close to list comprehension.

From the above information, we can see that Procera is highly applicable to domains of sustainability-oriented policies. A network manager stands to benefit much from the flexibility of the framework. Unfortunately, the framework does not allow the existence of policies that modify policies. Finally, we note that the Procera solution, as illustrated in Figure 6.10, interacts

with the programmable switches through the network control layer. Therefore, this framework is adequate for performing enforcement.

6.3.2.2. Policy refinement

In this section, we describe, classify, and compare the most important works related to policy refinement approaches. First, we consider (1) a methodological approach that describes how a refinement method should behave. Then, based on Hu et al. [37], we classify the approaches into five types: (2) goal-oriented policy refinement methods (divided into two parts: (I) event calculus-based policy refinement method and (II) policy refinement using linear temporal model checking technology), (3) classification-based approach to policy refinement, (4) ontology-based policy refinement, (5) three policy transformation approaches, and (6) recipe-based policy refinement (called prescription-based policy refinement by Hu et al. [37]). We present one example for each classification, additionally including the UML-based (7) classification approach to policy refinement. We describe these approaches in the following and compare them in Table 6.3.

Table 6.3. Features comparison between policy refinement approaches

Policy refinement approach	Method. [15]	KAOS [73]	Rubio-Loyola [63]	Udupi [72]	KAoS [31]	Beigi [10]	Craven [23]	Liao [47]
Automated	✘	✘	*partial*	*partial*	✓	✓	✓	✓
Not Domain-specific	✓	✓	✓	✓	✓	✓	✓	✓
Business-level	✓	✓	✓	✓	✓	✓	*partial*	*partial*
Online transformation	✘	✘	*partial*	✓	✓	✓	✘	✓
Sustainability	✓	✓	✓	✓	✓	✓	✓	✓
Downloadable	✘	✘	✘	✓	✘	✘	✘	✘

(1) Methodological approach

Carvalho et al. [15] proposed a methodological approach to sustainability-oriented policy refinement from the business down to the instance level. The approach is not domain specific. Figure 6.11 illustrates the example by which the authors detail their method. The process starts at the business level, which is translated into the system level by incorporating some key performance indicators (KPIs). This level is then operationalized on the network level by using the *"on Event if Condition then Action"* sequence of the Ponder2 framework.

The policies described in Ponder2 must be understandable to the devices in the network. The devices must support power modes, such as sleep and power on. Figure 6.12 provides an example of a device-level policy. The refinement down to the instance level is done by using SNMP commands; the instances apply the actions and provide information to the upper layers. Figure 6.13 provides an example of an instance-level policy.

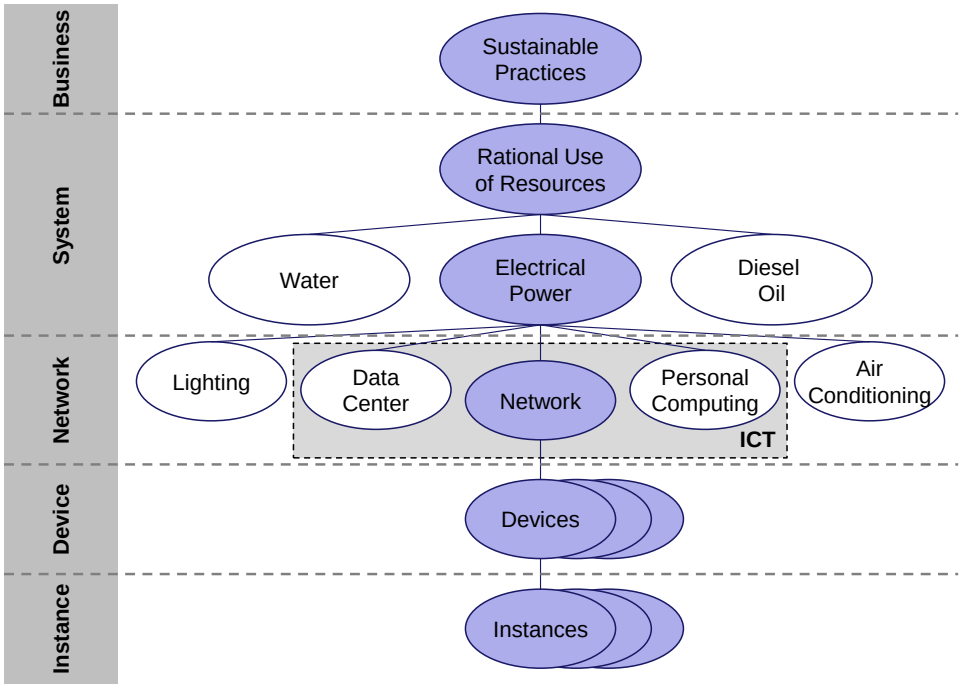


Figure 6.11. Sustainability-Oriented Policy mapping [15]

(2) Goal-oriented policy refinement methods

(I) Event calculus-based policy refinement method

KAOS is both an approach to requirements engineering [35, 30] and a goal-oriented method [26, 46]. Given a desired system and some initial goals, the method of KAOS allows one to refine goals into other, more specific goals. The goals obtained through the refinement are subgoals of the higher-level goals, which are closer to the management level. The higher-level goals describe the desired properties of the system and therefore help in understanding the essence of the subgoals.

```
       //Example of device level policy
       Not dependent on Operating System
functionalities;
       SNMP protocol support
       MIB support: RFC1213-MIB, TN3270E-RT-MIB,
POWER-MONITOR-MIB
       Power states support: Sleep, Low e High
```

Figure 6.12. Example of a device-level policy

```
        snmpget [options] [-Cf] [OID]
        Options: protocol version, device address,
MIB name
        -Cf : try to correct errors
        OID: variables to be read
        snmpset [options] [-Cf] [OID] [type] [value]
        Options: protocol version, device address,
MIB name
        -Cf : try to correct errors
        OID: variables to be set
        type: i (integer), t (time), a (ip address),
 s (string), D (floating point), ...
```

Figure 6.13. Example of an instance-level policy

The subgoals, in turn, are defined in such a way as to identify the requisites, objects, agents, and actions of the system [35]. Thus, the subgoals are closer to the instance level than are the goals from which they are refined.

The next step in the KAOS method is identifying AND, OR, or XOR relations among the goals and developing a goal graph for a given domain [35]. In the graph, the conjunction or the (exclusive) disjunction of subgoals reduces a previously existing goal. Once equipped with the network of concepts represented by the goal graph, an analyst is able to trace conflicting relations between the goals. The network of concepts makes up a semantic network in which the nodes are concepts and the connections between nodes are associations of concepts [35]. The subgoals that cannot be further reduced embody the leaves of the goal graph. They represent requisites and can be assigned as a responsibility of individual agents.

The meta-level of KAOS is a rich ontology that guides the identification of requirements, as well as provides means for capturing and modeling the relationships between the requirements. When deriving and comparing subgoals, obstacles to the execution and deployment of a goal may appear. Notwithstanding, the graph can then be reformulated, and the specification of the system can be enhanced by the analyst. A possible solution to the reformulation and enhancement is the introduction of new goals or the adoption of a different refinement approach, that is, the use of a different reduction path.

After the reduction of the identified goals, the analyst can further identify the requisites of the system-to-be. The next step in the KAOS method is identifying objects from the goal descriptions. Objects can be used as the input and output of actions that operationalize a requisite. The use of objects is done by agents, which monitor and control them. The third and final step is assigning the requisites, objects, and actions to the agents.

Despite being the most widely cited approach [35], large KAOS models may result in incomplete or complex requirement models [30]. Some approaches, such as those described in [35], help to understand and maintain such models. KAOS has the advantage of not being domain-specific, making the method capable of supporting sustainability goals. Nonetheless, the method has the drawbacks of depending on manual operation by an administrator and having non-online transportation.

(II) Policy Refinement Using Linear Temporal Model Checking Technology

Rubio-Loyola [62] proposed a partially automated approach to policy refinement and management. The approach is generic and can thus be used in any domain. The author used requirements engineering techniques to translate goals and then refine them in an automated way. The method starts with goal definition and selection considering the application. These steps are performed by using KAOS (Knowledge Acquisition in autOmated Specification), the most relevant approach combining semiformal and formal specifications of goals [62]. These activities are performed manually and, according to the author, do not invalidate the method because the policy refinement process is mainly an offline process.

After these activities, the system requirements are formulated using linear temporal logic (LTL), a way to put together logic and temporal operators, regulated by some refinement patterns. LTL, along with the system behavior documentation (in UML format, for instance), produces a step-by-step trace required to meet the previously defined goals. This step is executed with support from the SPIN search engine, which provides a plan of the transitions to be executed by the model entities. This step also produces a trace, that is, the input to the next one, that understands the required policies and codifies them using Ponder. The policies are then stored for future use. Figure 6.14 illustrates the method.

The method proposes a semiautomated refinement of high-level business-oriented policies, specified as a set of goals, into Ponder by using goal elaboration. Because this is a generic method, sustainability-oriented policies could be used, defined as goals on the business level. According to the author, even with the framework depicted in the figure above, *we are still far from proposing a generic solution that bridges the gap between SLA fulfillment and the formulation of high-level goals.*

(3) Classification-Based Approach to Policy Refinement

Udupi et al. developed a policy refinement method and implemented it in the Rice University Bidding System (RUBIS)[1]. The method consists in performing policy refinement by statistically classifying attributes to generate policies. In this approach, the end user specifies high-level goals, and then, during the test-and-development phase, the management system generates the low-level policies that are relevant and that may be used to monitor and assess the system behavior. The high-level policies are described based on the SLA, which includes metrics such as availability, response time, throughput, security and so on. The SLA is composed of a set of service-level objectives (SLOs) that specifies how to consider a service acceptable and define the constraints. An example of an SLA is "service response time < 85 ms between 8 AM and 5 PM." The low-level policies need to be in accordance with the SLA in order to not violate it; thus, there is a need to derive low-level policies from high-level SLAs.

[1] http://rubis.objectweb.org/

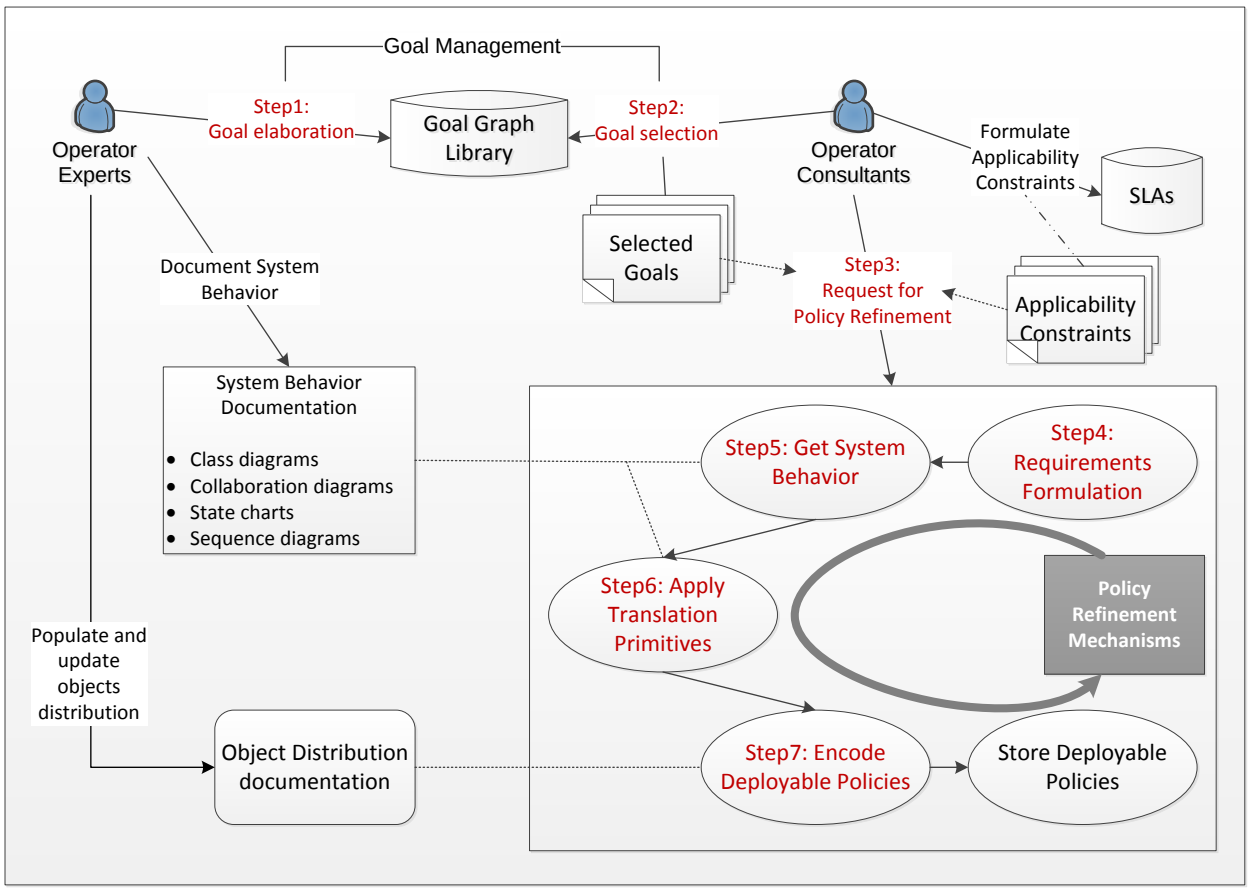


Figure 6.14. Policy Refinement Framework proposed by [62]

In this way, Udupi et al. developed a refinement method that identifies all the important low-level attributes and their policies and then refines them to specify the most optimal constraint thresholds for the relevant metrics. The policy refinement process is applied in three phases as follows:

(a) Test and Development Phase

i. Create an ad hoc system on the basis of the given high-level SLA goals.
ii. Run the workload around the target high-level SLA goal.
iii. Collect the low-level attribute metrics and SLA values to form a data set.

(b) Classification Phase

i. Apply a classification algorithm on the data set collected. This can be performed by a decision tree classification on the data set with the high-level SLA goals as target.
ii. Obtain a decision tree with TRUE and FALSE leaves, depending on whether the high-level SLA goals are satisfied or not.
iii. Select all the paths leading to the TRUE leaves, providing the required rules on low-level metrics.

(c) Policy Derivation and Refinement Phase

i. All TRUE paths are converted into policies [?] conjunctions of inequalities on selected attributes that were picked in the classification phase.

ii. Obtain distribution statistics, such as the MIN and MAX of the attributes appearing in the TRUE paths (A refinement strategy that is applied at this level uses the distribution statistics of the attributes on these TRUE paths).

iii. Aggregate the above two to obtain REFINED TRUE policies. The inequalities of attributes that appear on the TRUE policies are further refined by appending the MINIMUM and MAXIMUM values, which give definite bounds for the attributes, resulting in the required TRUE REFINED policies.

This approach is relatively easy to implement in performing autonomic transformation between high-level and low-level policies. The limitation is that it uses static rules predefined by experts, which are hard to apply in a dynamic environment. In addition, the case based on the policy transformation method may be difficult to apply in instances that lack usable cases at the initial stage.

This approach is flexible to supporting sustainability-oriented policy refinement because the low-level rules may be whatever the experts have determined, which can be the sustainability rules.

(4) Ontology-Based Policy Refinement

An ontology defines a common vocabulary that includes formal, explicit descriptions of the concepts in a domain of discourse, the properties and attributes of each concept, the restrictions on slots, and the relations among the concepts [57]. As previously described, the policy framework KAoS (Knowledgeable Agent-oriented System) uses an ontology system based on the Web Ontology Language (OWL) to describe concepts. This language can be extended to include sustainability-oriented policies by introducing new ontology classes. Because KAoS works with policy refinement, it is a policy framework as well as a policy refinement method starting at the business level; it is also autonomic and online.

(5) Three Policy Transformation Approaches

The proposal of [10] adopts three approaches to refine business goals into machine-readable commands: transformation by using static rules, by policy table lookup, and by applying case-based reasoning. In the static rule transformation method, static rules are predefined. The method describes how to refine high-level policies through definitions specified in the rules in light of some system-understandable configuration parameters. The policy table lookup approach makes use of a table of policies; the system administrator queries the module with configuration parameters to obtain goals according to the input. The case-based reasoning technique uses the knowledge acquired in previous system behaviors to predict present and future behaviors. In this case, it stores previous cases in the database so some translating processes can be completely or partially reused [37].

Although the author makes no declaration about any specific type of policy, it is possible to imply that his framework is capable of dealing with sustainability-oriented policies. None of the three proposals suggested, namely, through static rules, table lookup, and case-based reasoning, impose any opposition to the creation of any kind of policies. In addition, the proposals permit transformation through both offline and online methods. Figure 6.15 shows an adaptive policy-based system management architecture with static policy transformation.

The author's proposal also includes a framework for policy management based on the IETF framework. It follows the main principles of the IETF, such as centralization. This means attributing the device configuration and policies to the same point and business level abstractions, with the idea of making the network management easier. This is because the framework allows the administrator only to create business objectives, without necessarily having to understand low-level networking terminologies. It also uses a similar policy architecture model.

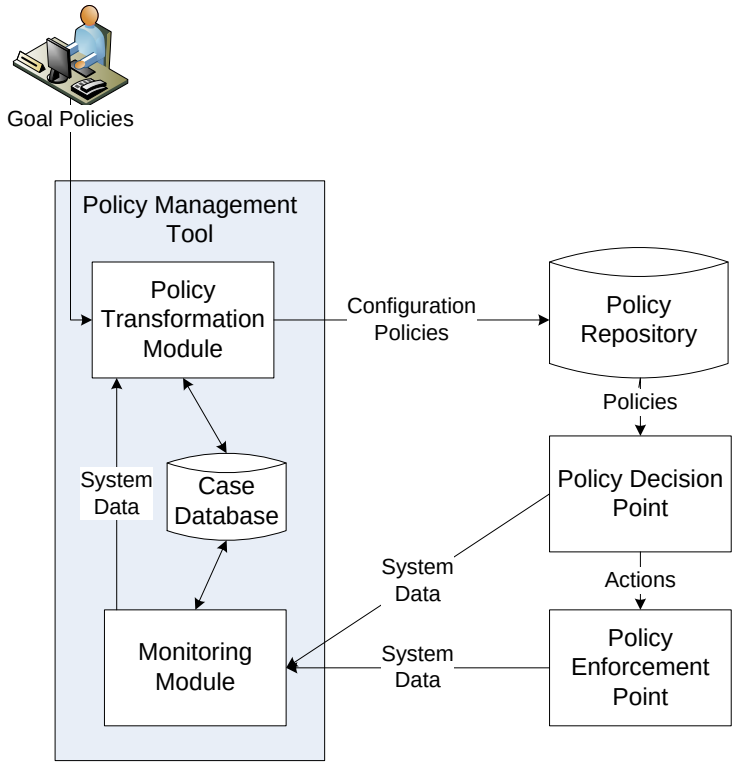


Figure 6.15. Policy Refinement Framework proposed by [10]

(6) UML-Based Approach to Policy Refinement

Craven et al. [24] described a complex policy refinement method, the core of which involves decomposition, operationalization, deployment, and re-refinement. Like other formalists, the authors made use of diagrams of the well-established UML for logical formalization. The focus of the discussed work is a study about how to automate distributed policy refinement for obligation and authorization policies.

The starting point of the method is threefold: a UML model describing the objects over which the policy will rule, a high-level authorization or obligation policy, and rules relating high-level actions and objects to low-level ones. Such are the decomposition rules, whose description language is based on constraint logic programs and concepts from data integration. The integration of decomposition and operationalization into a single, complete method for refining authorization and obligation policies is the focus of the work under discussion. The UML model contains a description of the class structure, the possible associations, the operations applicable to the classes, and an instance repository, that is, a set of records about the existing objects of the domain and about the relations among them.

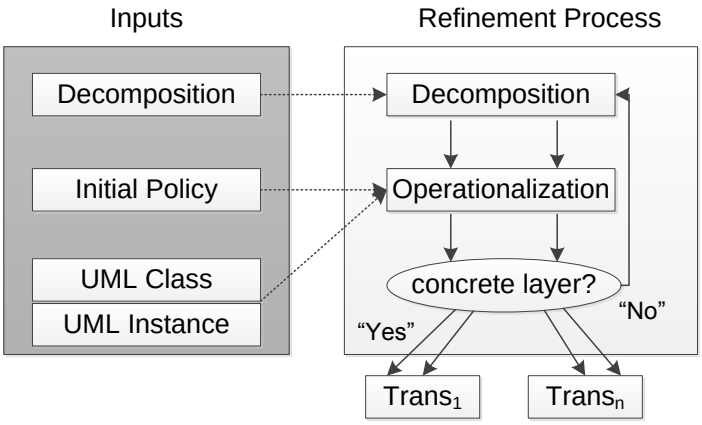


Figure 6.16. Overview of the refinement process as depicted in [24]

Figure 6.16 depicts the refinement process adopted by Craven et al. The first step is the operationalization of a high-level policy in order to find every resource to which the policy is applicable. One can see that the operationalization phase depends on the existing objects such that this phase requires as input a sequence of semi-refined policies provided by a decomposition, the associated sequence of information, and the current instance repository of the domain. The instance repository is compared with the policy conditions to determine the objects that could satisfy the conditions. Next, the enforcement points that understand the policies are searched. The ones found receive a specific policy language description for implementation (e.g., Ponder). During the lifetime of a system, the destruction, creation, or change of objects might occur. Such events probably invalidate the instance repository, giving rise to the necessity for re-refinement and update of old policies with new, consistent ones.

The next phase of the refinement is decomposition. During this phase, abstract elements are assigned to components and implementations of a more concrete level. The policy language adopted is expressive enough to let other renowned policy languages be translated into it.

The method is highly automated, disposing of many algorithms. For enforcement, the authors propose the use of Ponder2. Similar to the other generic methods discussed, the proposal by Craven et al. is applicable to sustainability-oriented policies. Notwithstanding, the authors described only the refinement of authorization and obligation policies. Further works in the area could make the method even more adequate for domains of sustainability.

(7) Recipe-Based Policy Refinement

The goal of Liao et al.'s [47] approach is to perform autonomic policy refinement that is based on policy refinement templates (recipes). The policy refinement should be context-aware because of the dynamic nature of virtual organization. The mechanism should be flexible enough to reflect the variations in the environment. In addition, it should be simple and efficient to ensure that the process takes place within a time limit.

This approach is inspired by the ways that humans treat complex problems. For instance, when a person deals with a complex problem, he/she should first have knowledge about how to solve it. This knowledge is dynamic; it is not a specific program to be performed but a template that defines possible plan steps. The plan steps are chosen at runtime depending on the real-time conditions. Hence, in this work, the recipes are templates of policy refinement that define the steps in abstract policy refinement. The refinement of an abstract policy performs decompositions according to a recipe and results in concrete or enforceable policies. Enforceable policies are distributed to specific agents, which are required to comply with them. Concrete policies describe the duties and rights of the roles that are applied to agents. In this way, the outcomes of the refinement not only meet the goal of abstract policy but also reflect the status of the underlying environment.

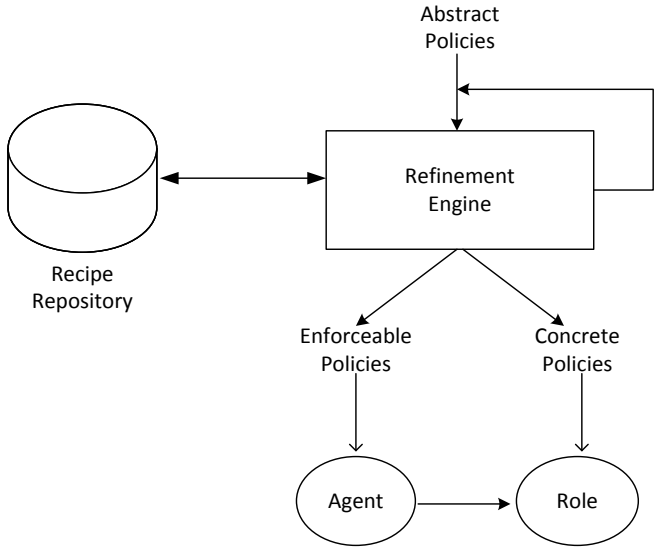


Figure 6.17. Policy Refinement Framework proposed by [47]

Figure 6.17 shows the recipe-based refinement mechanism proposed by Liao et al. [47]. The refinement engine has an algorithm that checks the conditions of each step described in the recipe. The algorithm is described in Algorithm 2, and an abstract policy (p) is refined into several sub-policies. In this algorithm, A is the list of ongoing refinement policies, NA is the list of non abstract policies, and T is the tree that records the refinement scheme of an abstract policy.

Algorithm 1: Liao et al. policy refinement

Step1 Let A=p, NA=Ø, and T.root=p.

Step2 Take p out of the list A, and let $p_0 = p$.

Step3 If p_0 is not an abstract policy, push it into the list NA and GOTO Step7.

Step4 If $\exists r_i \in$ **Set of available recipe**, such that r_i.AbstractPolicyID=p_0, then Input(r_i); otherwise, Return error and GOTO Step9.

Step5 If $\exists c_k \in r_i.PlanSet\ p.Condition = TRUE$, then Output ($c_k$, PolicySet); otherwise, Return error and GOTO Step9.

Step6 Let the policies of PolicySet from Step5 be p_0' Children, adding to the tree T, and meanwhile, push them into the list A.

Step7 Take the first element (denoted as f) out of A. If A= Ø, then let $p_0 = f$ and GOTO Step3; otherwise, GOTO Step8.

Step8 Output T.

Step9 Stop.

Policy-oriented management can be used to apply sustainability-oriented policies to a network operation. Policy refinement is used in this case to determine domain-specific actions at multiple levels of the network operation. To understand this approach, it is necessary to consider in detail: (i) the network management system driven by sustainable policies, as provided in section 6.4; (ii) the framework for managing policies, as presented in section 6.5; and (iii) the practical point of view regarding the refinement of a sustainability-oriented policy, as discussed in section 6.6.

6.4. SustNMS - Network Management System driven by sustainable policies

As described previously, energy efficiency can be achieved in wire-line networks by: (i) reengineering (energy-efficient hardware), (ii) dynamic adaptation (managing the network to performance scaling or idle logic), and (iii) smart sleeping (power-saving or sleep mode) [13]. Reengineering is not related to decisions based on policies and is not our focus. Dynamic adaptation and smart sleeping can be performed by decisions from the network management or by green routing algorithms driven by policies. The advantage of network management is that it provides a high-level overview of the network, by which a management system may analyze the entire network instead of just making a local decision, as occurs in routing algorithms, which could not result in the best decision.

In this way, our focus is on network management. Thus, we describe in this section a network management system driven by sustainability-oriented policies, the SustNMS [19]. This system is used to orchestrate the network according to policies, such as sustainability-oriented

ones, which may describe how to save energy in the network by specifying when and which device may be set to sleep.

Figure 6.18 shows the SustNMS architecture. SustNMS is based on the policy-based network management system defined by the IETF, which is represented by the (i) policy management framework (PMF) in Figure 6.18. SustNMS extends the policy-based network management architecture defined by the IETF by including four modules: the (ii) model repository (MR), the (iii) quality of service monitor (QoSM), the (iv) sustainability monitor (SM), and the (v) device updater (DU).

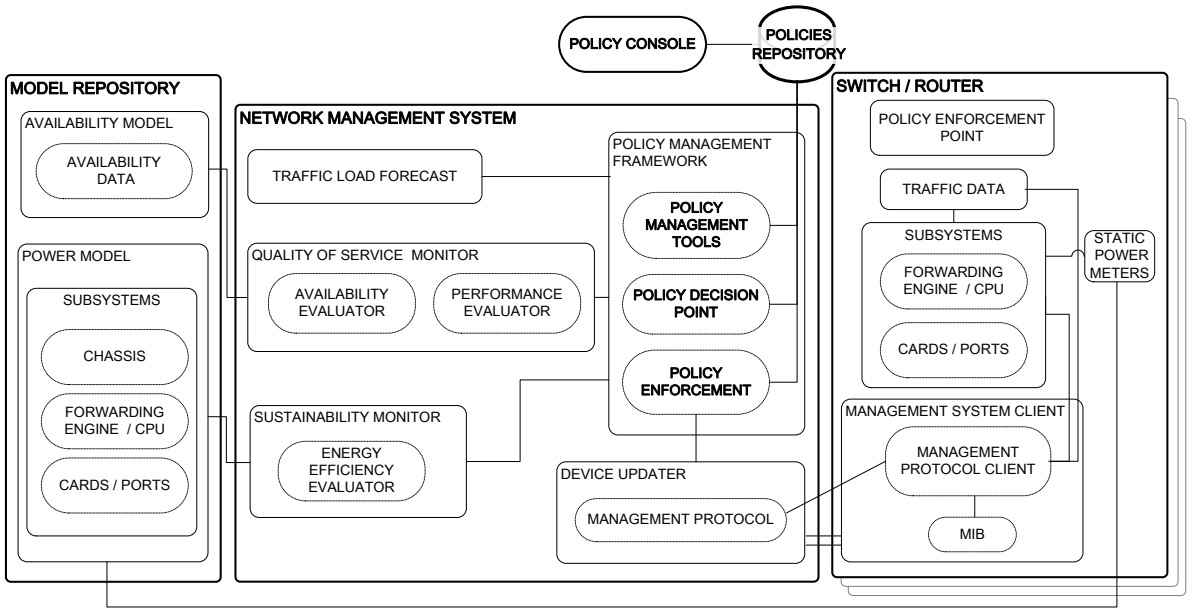


Figure 6.18. Sustainability-Oriented Policy-Based Network Management (Sust-NMS) Architecture [20]

i. The **PMF** is responsible for managing, describing, checking, storing, and enforcing the policies. Thus, it consists of four sub-modules according to [78]: *(1) policy management tools, (2) policy decision point, (3) policy enforcement point, and (4) policy repository*.

   1. The *policy management tools* (PMT) consist of a set of policy rules and may edit, translate, and validate functions. This sub-module enables interaction between a user and the policy-controlled system.

   2. The *policy decision point* (PDP) "is composed by trigger detection and handling, rule location and applicability analysis, network and resource-specific rule validation, and device adaptation functions" [78]. It is an aggregate of the algorithms that consider which policies must be applied in a particular operational situation. It also determines the actions to be performed to comply with these policies. A policy framework like Ponder2 is used for this sub-module.

   3. The *policy enforcement* point (PEP) "is responsible for the execution of actions, and may perform device-specific condition check and validation" [78].

4. The *policy repository* (PR) provides storage for the set of policies defined for the network system.

ii. The **MR** consists of two sub-modules: (1*) power models* and (2) *availability models*.

1. *The availability model* is composed of information on failure and repair rates for each network device type, such as mean time to failure (MTTF) and mean time to repair (MTTR).
2. The *power model* describes the parameters that define a power profile, which is used to calculate the energy efficiency of a device. A power profile would be determined by a function f (load), which defines how power consumption varies with the load handled by the device. An example of a function f (load) for a router following a linear power consumption scaling is [6]:

$$P\big(state, f(load)\big) = \begin{cases} P_{standby} & \text{if } state = standby \\ f(load) & \text{otherwise} \end{cases}$$

$$f(load) = P_{idle} + \frac{load}{capacity_{max}}\big(P_{full} - P_{idle}\big) \qquad (1)$$

iii. The **QoSM** includes two modules for the network evaluation: (1) availability evaluator and (2) performance evaluator.

1. The availability evaluator dynamically evaluates the network availability every time the network state changes, i.e., when a device is set to sleep or fails. One approach to evaluating the availability in a sustainable network was described by Amaral et al. [5], who evaluated the network in a hierarchical way using Markov chain modeling that considers the time it takes for a device to wake up from sleep mode and calculates the network availability through the cut and tie set method.
2. The performance evaluator analyzes the network information, calculating the amount of packet loss, delay, and jitter. The indicators used depend on the management approach and on the requirements defined in the SLAs.

iv. The **SM** is the module responsible for energy efficiency evaluation.

The energy efficiency evaluator assesses the power consumption of each device in the network. The evaluation may be performed in different ways. On the one hand, a network management protocol can be used to gather information about the instant power consumption of the devices. On the other hand, if such feature is not available in the device, the module retrieves the proper power model from the MR. If a power model that matches the network device cannot be located in the MR, a pessimistic power model is loaded from the power model repository using a predefined lower bound for the power consumption variables. Then the total power consumption, along with the load handled by the device, is used to evaluate the energy efficiency ratio (W/bps).

v. The **DU** collects data from the network and applies changes according to each instance of the policy enforcement point (PEP). The DU collects data by using a network management protocol to probe the devices. For configuration deployment, a network

management protocol (e.g., SNMP, NETCONF) or a command-line interface (CLI) command is used.

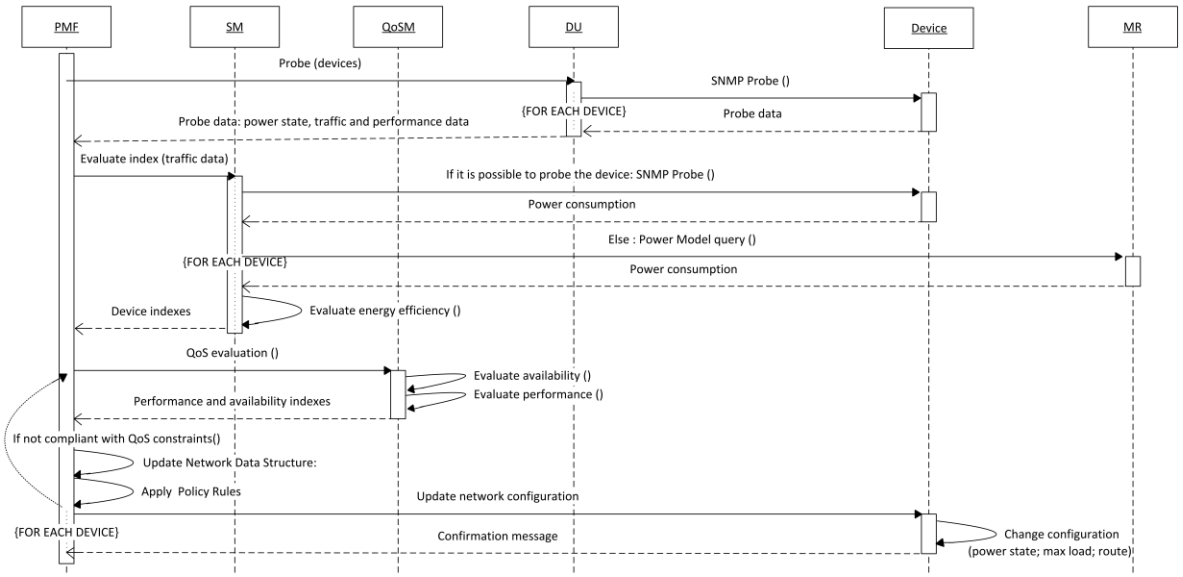


Figure 6.19. SustNMS sequence diagram

The sequence diagram in Figure 6.19 describes the system behavior and each module interaction. The system operates by probing the network at a frequency defined by the operator. Each probe request is an action triggered by the **PMF**. The **DU** then receives a request to collect network statistics. For each device in the network, an SNMP probe based on the MIB-II (RFC1213) standard is issued, collecting traffic, state, and performance data. The data are stored in a graph that represents the network topology. The graph is then sent to the requester.

After gathering information from all devices, the **PMF** sends a request to the **SM** to evaluate the efficiency indexes. For devices that support the energy monitoring MIB [17], the instant power consumption is obtained through the sustainability monitor data identifiers. For the remaining devices, the power consumption is evaluated through device-specific power models stored in the MR, a database accessed through Simple Query Language (SQL) queries, defining the static parameters. The traffic and state data in each node are used to evaluate the power consumption function in Equation (1). The network energy consumption rate (milliwatts per Mbps) [51] is evaluated for each switch based on the traffic and power consumption of the nodes of the graph. The graph is updated to include the calculated indexes for each node.

Immediately after the energy efficiency evaluation, the **PMF** makes a request to the **QoSM** to evaluate the network availability and performance indexes. For the availability evaluation, the MTTF and MTTR of each router are retrieved from the MR and written in the respective node. These data, along with the state of each node, are used in a dynamic availability evaluation [5]. The network availability is stored in a specific variable. Then, the performance indexes are obtained through SNMP probes using the MIB-II data identifiers for packet loss statistics for each switch. The delay and jitter are obtained through Internet Control Message Protocol (ICMP) request. The graph is then updated with the calculated indexes.

The **PMF** handles the network status. The *policy decision point* (**PDP**) defines the actions that must be performed in the network, according to the defined policies. The PMF module then checks whether the actions can be applied by comparing the new configuration against the QoS constraints. As a result, it might be necessary to recalculate the network configuration to fulfill availability requirements, or the quality of service constraints are relaxed to improve energy efficiency. This decision is defined in the policies. The output of the module is a new configuration for the network.

The **PEPs** enforce the decision on an updated network configuration decided by the **DU**. They translate the configuration request in each node into device-specific commands and set the power state for each device or update switching-related data to enforce the prioritization of the most efficient paths. Multi-Protocol Label Switching (MPLS) is used to direct data among network nodes. It defines how to commute data by checking a small tag on each data packet. MPLS characterizes the routers as a label switch router (LSR), which only commutes packets, or as a label edge router (LER), which classifies a flow by adding an MPLS tag. In this way, a PEP sends CLI commands for the LERs to determine the flows. The action is concluded by a confirmation on the updated configuration that is sent to the PMF.

The entire process is always repeated for each probe made by the SustNMS. The accuracy of the measurements is defined by the rate at which these actions occur. With a long interval, the system may not identify the network changes, and thus, it may not apply the best decisions. Defining an adequate probing rate should consider that an increase in the probe rate would generate an increase in messages related to management operations. This may imply an increase in the network overhead and energy consumption because an increased network load consumes more energy.

6.5. Policy framework: Ponder2

In this tutorial, we focus on developing a case study using the policy framework Ponder2. This framework is an open source framework, which provides full access to the source code, allowing any modification. It has a complete manual, as well as a tutorial that facilitates learning how to configure the framework. In addition, Ponder2 is widely used in many applications. The framework is described in the following.

Ponder2, developed at the Imperial College, London, is a declarative, object-oriented language for specifying security and management policies in networks and distributed systems [50]. It comprises a general-purpose object management system with message passing between objects [71] and is used in many projects [62].

The first version, Ponder, became associated not only with the language for defining policies but also with the entire tool kit. It was designed for general network and systems management. The second version, Ponder2, is a significant redesign and re-implementation of Ponder. It can be used at different levels of scale, from small devices to complex services [49].

Ponder2 is based on the self-managed sell (SMC) concept. An SMC is defined as a set of hardware and software components that constitute an administrative domain able to work autonomously and manage itself [71]. Ponder2 combines a distributed object management system with:

- a domain service, which provides a structure for managing objects, similar to directories;
- an obligation policy interpreter, which interprets Event-Condition-Action (ECA) rules;
- a command interpreter, which accepts commands compiled from PonderTalk, a high-level language used in Ponder2; and
- authorization enforcement, which deals with positive and negative authorization policies, besides structuring the domain for conflict resolution regarding authorization [49].

6.5.1. Self-Managed Cell (SMC)

An SMC, as previously stated, is a set of hardware and software components that constitute an administrative domain able to work autonomously and manage itself [71]. Figure 6.20 illustrates the SMC architecture. A number of services constitute the core functionality of the SMC and must always be present. These include the event bus, a discovery service, and a policy service [28]. In this work, we use Ponder2 as the policy service for the SMC. The event bus and the discovery service are based on the Ponder2 tutorial and are implemented in Java.

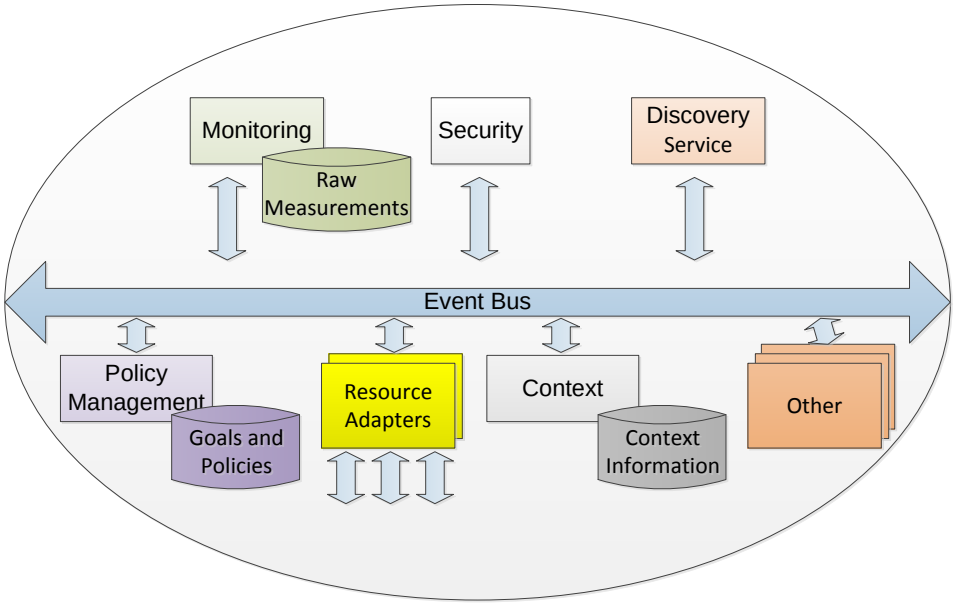


Figure 6.20. Self-Managed Cell Architecture [43]

For larger systems or complex environments, it is possible to implement a composition of several SMCs [42]. The interactions between the SMCs can be made through an event bus or other communication paradigms, such as point-to-point messages or remote invocations [28] by using UDP or other protocols [43]. It is also possible to scale the SMC architectural pattern by considering that the managed resources are themselves SMCs and then by composing these SMCs [28].

Policies define how the system should adapt in response to events (failures or changes in the requirements). The SMC implements a local feedback control loop where changes of state in the managed objects or resources trigger an adaption that affects the state of the system [43]. The policy service is consists of: (i) an event interface, which receives notifications; (ii) an invocation interface through which external invocations are received; and (iii) an action interface that sends instructions to external objects, as depicted in Figure 6.21.

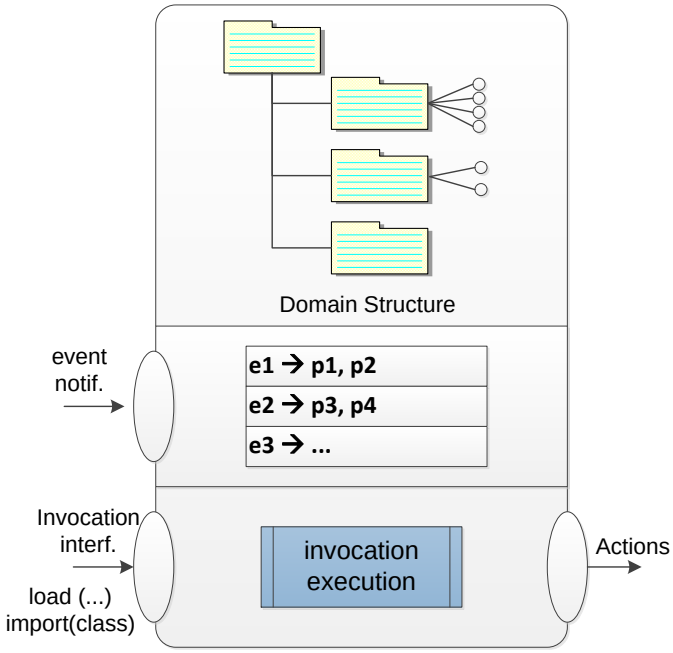


Figure 6.21. Policy Service Architecture

6.5.2. Managed Objects

Everything in Ponder2 is a managed object. Ponder2 objects include events, policies, and domains. These are first loaded into the SMC, and then it is possible to send messages to them to create new instances of the desired managed object (just as classes and instances operate in object-oriented programming). Ponder2 managed objects are written in Java (Ponder version 1 used XML as configuration and control language). Domains are managed objects that act as containers for other managed objects. They are like the directories of conventional operating systems but with an important difference: a managed object may belong to several domains. Domains are used to group objects so that the same policy can apply to sets of devices rather than to individual ones [50]. They are managed objects in which actions can be performed, e.g., adding or removing an object from a domain [43]. Managed objects can also be written in Java by the administrator. A managed object must implement an interface called ManagedObject. This interface tells the Java compiler that the object will be a Ponder2 managed object and that it must start creating a Java adaptor code to perform the required mappings from PonderTalk to Java methods. If we want to send a message to a managed object, for instance, to set the name and age of a person, this can be done in PonderTalk as: myobject name: Fred age: 24. The Java code would be as described in Algorithm 2.

Whereas @annotation allows PonderTalk messages to be mapped to methods within the managed objects, a managed object must be loaded into the SMC from a library, producing a factory-managed object (like a Java class). This factory-managed object receives instructions to create new instances of managed objects that will work on the system (like instances of classes in object-oriented programming) [49].

Algorithm 2: Java code sample to translate PonderTalk code

```
#@Ponder2op("name:age:") //"Annotation" - interprets the PonderTalk
command
public void setInfo(String name, int age){
this.name = name;
this.age = age;
}
```

6.5.3. Events

Events are notifications with named attributes, created by managed objects. They are either internal or external, captured by a monitoring service [50], and are also managed objects. The events trigger ECA policies and are able to integrate with one or several external event systems through adapter objects. There are many publish/subscribe event services, such as XMLBlaster, but it is also possible to implement your own service for use in a specific situation [28].

6.5.4. Discovery Service

The discovery service detects new devices in the SMC and, based on policies and information on the device profile and authentication, determines in which domain these will be allocated [64]. In this case, policies are also used to decide which adapters should be created for the new components. Being assigned to a domain, the object will receive all messages sent to that domain. As for the event bus service, it is possible to implement your own device discovery service, although various protocols already exist for both fixed and ad hoc networks [28]. In Algorithm 3, the solution proposed by [43] is described as an example. The discovery service is expected to work with any kind of policy because it is a complimentary service to the policy service.

6.5.5. Policies

As defined previously, a policy is a set of rules used to manage and control access to a set of ICT resources and services [68]. A policy is created with the policy factory and is also a managed object. Ponder2 has two types of policies: authorization and obligation. Authorization policies, being essentially access control policies, allow or deny message exchange among objects, determining which activities a domain member can execute over which objects. Obligation policies specify which actions the agent must perform. For instance, security policies specify which actions must be performed when violations occur and who must execute them. These policies are ECA (Event-Condition-Action) policies. [71].

6.5.6. PonderTalk

PonderTalk is the language used to configure and control Ponder2. It is based on SmallTalk, but without the definition of classes. It includes the message-passing aspects and some default types and has some syntax modifications. The PonderTalk code is compiled into XML, which is interpreted at runtime. The main difference between SmallTalk classes and Ponder2 managed objects is that managed objects are written in Java and may be held transparently in a remote Ponder2 SMC. Table 6.4 summarizes the differences between PonderTalk and Smalltalk.

Algorithm 3: Example of an algorithm for the discovery service

---

The discovery service broadcasts its identity message at frequency X.
New devices respond to the identity message identifying themselves.
The discovery service queries the device to obtain the device profile and authentication credentials.
The discovery service decides whether to accept the device and to which role it should be assigned.

if** Accepted, **then
    The accepted device is informed, and a component detection event is generated.
    A device-specific communication adapter is also created.

Each existing member device unicasts its identity message to the discovery service at frequency X.

if** The discovery service misses N successive messages from a particular device, **then
    It concludes that the device has left the SMC permanently and generates a corresponding component-left event.
    It also removes the respective adapter.

PonderTalk is a sequence of statements separated by a full stop (period). Each statement consists of a receiver (object) and a command for it. In response to the message, the receiver returns another object or itself. Temporary variables may be created without declaration. To assign a value to a variable, ":=" is used. There are three message types: unary, a single-word command that the object is expected to recognize; binary, which has one argument, generally special characters like "+" and "»" followed by a single data value; and keyword, a function with one or more arguments. In general, everything is processed from left to right, but in case a combination of message types occur, unary messages are performed first, followed by binary messages and keywords, respectively.

A powerful feature of PonderTalk is the block. A block acts as a function or a stored procedure, being a section of code with one or more statements and one or more arguments. A block returns the result of its last statement, and "what happens inside a block stays at the block."

Table 6.4. PonderTalk versus SmallTalk [49]

PonderTalk	SmallTalk	Description
Managed Object	Class	Ponder2 managed objects are the equivalent of Smalltalk Classes
Managed Objects written in Java	Class Objects written in Smalltalk	Managed Objects are "imported" into an SMC. The Smalltalk syntax to create new Classes has not been included in PonderTalk
Managed Objects may be local or remote	Class instances are always local	It is not necessary to know whether a Managed Object is local or remote, Ponder2 is a transparent, distributed system

Table 6.5. PonderTalk Types [49]

Name	Description
Hash	A collection of named objects such as Java Map or Smalltalk dictionary
Number	An integer or real number
Array	An array of objects
String	A string value
Boolean	A true or false value
Nil	The null value
Xml	An XML structure

A factory-managed object is used to create a new managed object. In addition to managed objects, PonderTalk has other basic, built-in objects that can be used as arguments for managed objects. Table 6.5 briefly describes these objects. A more complete description of PonderTalk syntax and types is available at the Ponder2 project Wiki [49].

6.5.7. Extras

Ponder2 has an interactive shell, similar to a Unix shell, able to show the domain structure and to execute PonderTalk statements on the machine running Ponder2 by simply using Telnet <machine> <port>. Ponder2 also has the ability to load all the codes needed on demand.

6.6. Practical point of view: a case study of sustainability-oriented policy refinement in SustNMS using Ponder2

This section presents a case study of sustainability-oriented policy refinement in a network management system driven by sustainability-oriented policies, the SustNMS (Section 6.4). The policy framework used is Ponder2 (Section 6.5). First, we describe the development of the test environment ("testbed"), including the SustNMS implementation and the Ponder2 preparation. Then, we describe the experiments showing the power consumption for different applied policies.

6.6.1. The testbed

The simulation environment is based on Linux software routers. The usage of Linux-based routers is growing [16]. For this reason, the described test environment is applicable to a large set of devices. To emulate different routers, the network is built with virtual machines using VMWare vSphere (ESXi 5.0). The virtual machines are interconnected through VMWare vSwitch (Layer 2 forwarding, VLAN tagging). Disabling the network interface cards (NICs) related to the data plane emulates the standby state. Control plane NICs are never disabled so as to maintain the network presence of the corresponding router. In addition to disabling data plane NICs, the system relies on power profiles to model the energy consumption of standby modes.

The routers run on top of an MPLS data plane using the MPLS-Linux project [55], a software-based approach that supports virtual MPLS tunnels, backup paths, label stacking, recursive lookups, and DiffServ. MPLS is a suitable way to control and regulate traffic trunks in a network by routing them along label-switched paths (LSPs). LSPs can have some QoS properties and a priority or precedence level [12]. The operating system used is the Debian GNU/Linux (Lenny release, Linux kernel 2.6.27.24). The usage of backup paths provides fast rerouting and a low overhead in the control plane [27]. To do this, the paths are set beforehand so that no path calculation is necessary during the network operation.

SustNMS infers information about the tunnels by using SNMP to access RFC1213- MIB variables (if InOctets and if OutOctets). By using command-line interface (CLI) commands through Secure Shell (SSH), the server defines which tunnels the ingress routers should configure for the incoming flows. SNMP and SSH packets constitute the overhead generated by the system.

6.6.1.1. SustNMS implementation

All the SustNMS modules were implemented in Python. The device updater consists of an SNMP Python Library, the pySNMP [66], which allows the usage of SNMP 1.1 to collect information about network load. To configure the routers, we used CLI through SSH connections. The SustNMS machine runs over Linux Ubuntu 11.04. The model repository stores the power and reliability models in a MySQL database.

To manage policies, we chose Ponder2 because it is an open source framework, with full access to the source code, allowing the necessary modifications. In addition, it has a complete manual and a tutorial available in [49]. Ponder2 is a policy service framework in which it is possible to define policies at the network level (if -> then -> else), as stated in Section 6.5.

As previously stated in Section 6.2, there are five levels of abstraction for describing policies: business, system, network, device, and instance levels. A sustainability policy defined at the business level can impact and be mapped in different policies in all levels, down to the instance level [15]. This process of moving from a high-level to a low-level policy is called policy refinement.

We described some existing approaches to policy refinement in Section 6.3. The only solution available for download, [72], requires experts for the input of policies, which is not the objective of this work. Thus, the chosen approach is the methodological one, as described in [15]. The policy refinement in SustNMS is performed in a manual fashion, from business policies to Ponder2 policies (network level) as in [15]. Ponder2 then receives the network

policies and provides enforceable policies to be applied on the network (devices and instances). The Ponder2 action must support power modes, such as sleep and power on (device level). By using SNMP commands, the instances of the device apply the actions and provide information for the upper layers (instance layers) as described in [15]. These actions receive data and are executed by SustNMS. In summary, we deal with all policy continuum levels by using a methodological approach with Ponder2 support for managing policies.

6.6.1.2. Preparing Ponder2

First, to understand "where we are," it is important to note that we are working on the policy decision point, described previously in Figure 6.3. The policy enforcement is made "outside," as explained in Section 6.4. A good way to start preparing Ponder2 is to use the tutorial files available at the Ponder2 project Wiki [49]. Ponder2 is made available under the terms of the GNU General Public License. The file for download at the above-mentioned site comprises the sources, documentation, and binaries for any operating system. Java JDK 1.5 is required. Ponder2 can be run as a stand-alone application or within a Java Virtual Machine.

Ponder2 can be instantiated using Apache Ant, in which the Ant version 1.7 or later is required. It is also possible to run Ponder2 without Ant. Apache Ant is a Java library and command-line tool able to drive processes described in Xml build files. The main known usage of Ant is in building Java applications. Ant supplies a number of built-in tasks that allow compiling, assembling, testing, and running Java applications.

In this course, we used the most recent binary file available at the time of publication. The archive contains the Ponder2 and Ponder2 Communications jar files and all the API and PonderTalk module documentation. You will find an ant build.xml file in the Ponder2 directory, along with the necessary library Jar files. It is also possible to use the source distribution, as indicated in the Ponder2 project Wiki [49].

The Ponder2 directory contains an Ant build.xml file, which can be used to run Ponder2 in a simple manner. To build and run the basic system, "ant run" is used. Testing Ponder2 requires opening a new terminal session and connecting to port 13570 on a computer by using, for instance, "telnet." It is important to note that in Windows 7, telnet is disabled by default, but it is possible to activate the service "Client Telnet."

Ponder2 comprises one or more Java jar files. The main Ponder2 jar file (ponder2. jar) picks managed objects from other jar files on the class path as necessary. The SMC is instantiated by starting the policy service. The policies to be used for self-configuration and management are pre-deployed in the policy service. In the beginning, the SMC is just an empty domain by default, and a domain hierarchy rooted in the directory "/" will be created. A built-in domain factory is used to create new domain objects within the domain hierarchy. All other managed objects are instantiated through factories that can be loaded on demand. The code described in Algorithm 4 shows how to bootstrap Ponder2.

To correctly start using PonderTalk, it is necessary to instantiate the events (an event contain name arguments and their values). Event templates are created from the event factory, and then events are created from the event templates. As previously described, events trigger policies. Algorithm 5 shows how to start using events.

Algorithm 4: Bootstrapping Ponder2 [49]

```
//Bootstrap code for Ponder2
//Import the Domain code and create the default domains
domainFactory := root load: "Domain".
root
at: "factory" put: domainFactory create;
at: "policy" put: domainFactory create;
at: "event" put: domainFactory create.

//Put the domain factory into the factory directory
root/factory at: "domain" put: domainFactory.

//Import event and policy factories
root/factory
at "event" put: ( root load: "EventTemplate" );
at "ecapolicy" put: ( root load: "ObligationPolicy" ).}
```

Algorithm 5: Starting Events [49]

```
//Create template /event/toohigh newevent := root/factory/event.
root/event at: "tooinefficient"
put: (newevent create: #("message" "value")
)
```

To start using policies, the same instructions apply: it is necessary to create them with the policy factory. The policy needs to be associated with an event. Algorithm 6 shows how to start using policies.

Algorithm 6: Starting Events [49]

```
//Create policy /policy/toohigh
newpolicy := root/factory/ecapolicy.

root/policy at: "tooinefficient"
put: (newpolicy create).

root/policy/toohigh
event: root/event/toohigh;
condition: [ :value | value > 10 ];
action: [ :message | root print: "Got event" + message ];
setActive: true.
```

These steps are necessary to set up the policy service, but as previously stated, an SMC has three core functionalities: an event bus, a discovery service, and a policy service. Now, it is necessary to implement the event bus and the discovery service. The code we used for the event bus and the discovery service were based on the Ponder2 tutorial, available at the Ponder2 project Wiki [49]. Figure 6.22 shows the structure used in this tutorial. The tutorial comprises all Ponder2 files and Java classes that implement the RMI communications, the discovery service, and a graphical interface.

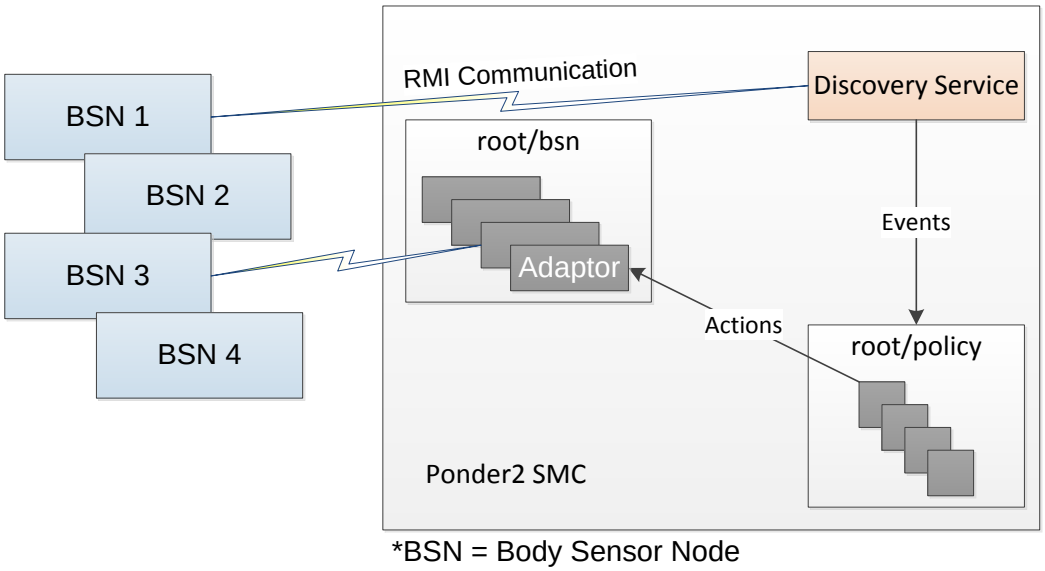


Figure 6.22. Ponder2 Tutorial Architecture [49]

The discovery service issues events when a new measurement panel is detected or lost. A policy creates or removes the appropriate adaptor managed object. Adaptor objects act as a proxy for the measurement panels and can receive commands for them. It is important to note that, in our case, the managed resource is the SustNMS, and some modified Java classes act as an interface between the SMC and SustNMS.

To the best of our knowledge, Ponder2 does not fully support conflicting obligation policies (it supports conflict resolution for authorization policies), which are common when considering energy efficiency in networks. That is why we have two policies, as stated above. This is an open issue we are still working on.

6.6.1.3. Network Topology

Figure 6.23 shows the network topology used in the experiment. The topology consists of four routers connecting two servers to one client. There are redundant paths when all the devices are powered on. A control plane that is parallel to the data plane connects the SustNMS to the routers.

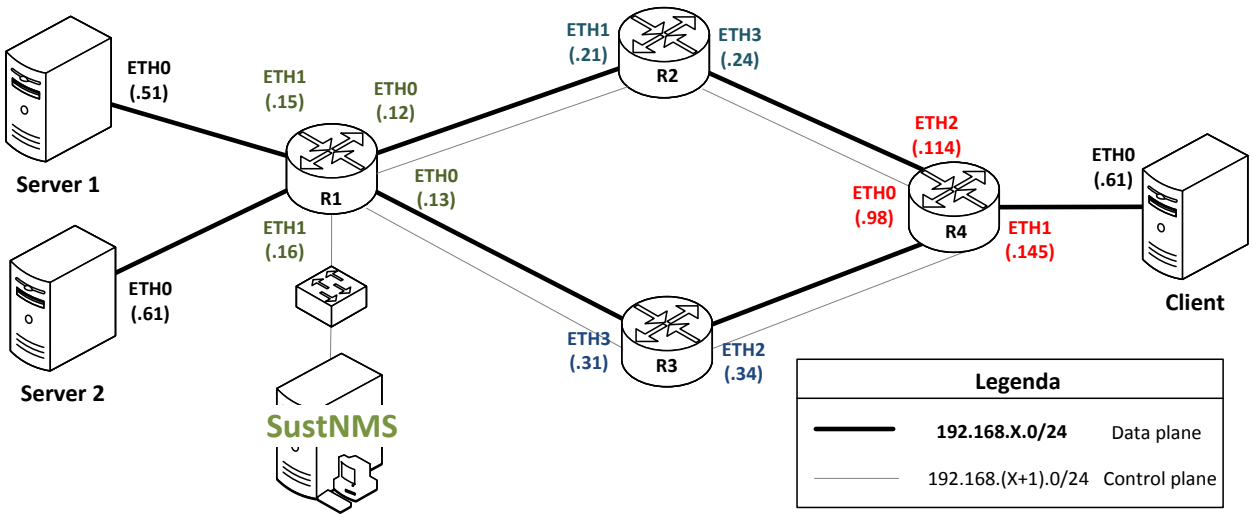


Figure 6.23. Network topology of the experiment

The routers R3 and R4, in Figure 6.23, are always powered on; i.e. such routers never sleep. This is because putting any of them in sleep mode implies network disconnection, which must not occur. On the other hand, the routers R1 and R2 are redundant. Depending on the current traffic, one of them may be placed in sleep mode, but never both to meet the connection requirement. When all routers are powered on, one can enumerate two MPLS tunnels of the topology in Figure 6.23; Tunnel 1 corresponding to R1[?] R3[?] R4 and Tunnel 2 to R1[?] R2[?] R4. If a certain router is set to sleep, one of the paths will not be available.

6.6.1.4. Important data for the testbed execution

To execute the testbed, it is necessary to provide: (i) the equipment power profile, that is, the behavior of each piece of equipment given a data load; and (ii) a traffic profile.

(i) Power profile

The power consumption of each router is calculated based on a predefined power profile. For all experiments, we defined only two types of power profile. One represents a device behavior envisioned for sustainable network, which the power consumption linearly scales with load [6]. The other type of power profile represents the legacy equipment, which does not present a significant variation in power consumption as the load increases [14], thus maintaining an almost constant consumption.

Because the test environment uses a software router-based approach, the system calculates the power consumption proportionally to the maximum capacity of a corresponding physical device. The maximum capacity of the software routers were predefined as 32 Mbps for R1 and R2 and as 40 Mbps for R3 and R4 (see Fig. 1.23). Because every possible stream traverses through R3 and R4, the devices were parameterized so that these routers are able to hold the network maximum traffic. A maximum traffic of 32 Mbps takes place when all streams are active. Power profiles must include a power consumption related to the standby power mode.

Four power profiles are used. Routers R2, R3, and R4 scale linearly and are based on the profiles in [14]. Router R1 has a constant power consumption, which is based on [2]. The following equations are used to indicate power consumption (Watts) as a function of usage (ratio of throughput load l to maximum capacity lmax), and Figure 6.24 presents the power consumption for all routers.

$$P_{R3} = P_{R4} = \begin{cases} 120 & \text{if standby state} \\ 200 + l\left(\frac{500}{l_{max}}\right) & \text{otherwise} \end{cases} \quad (2)$$

$$P_{R1} = \begin{cases} 250 & \text{if standby state} \\ 1000 & \text{otherwise} \end{cases} \quad (3)$$

$$P_{R2} = \begin{cases} 170 & \text{if standby state} \\ 200 + l\left(\frac{1000}{l_{max}}\right) & \text{otherwise} \end{cases} \quad (4)$$

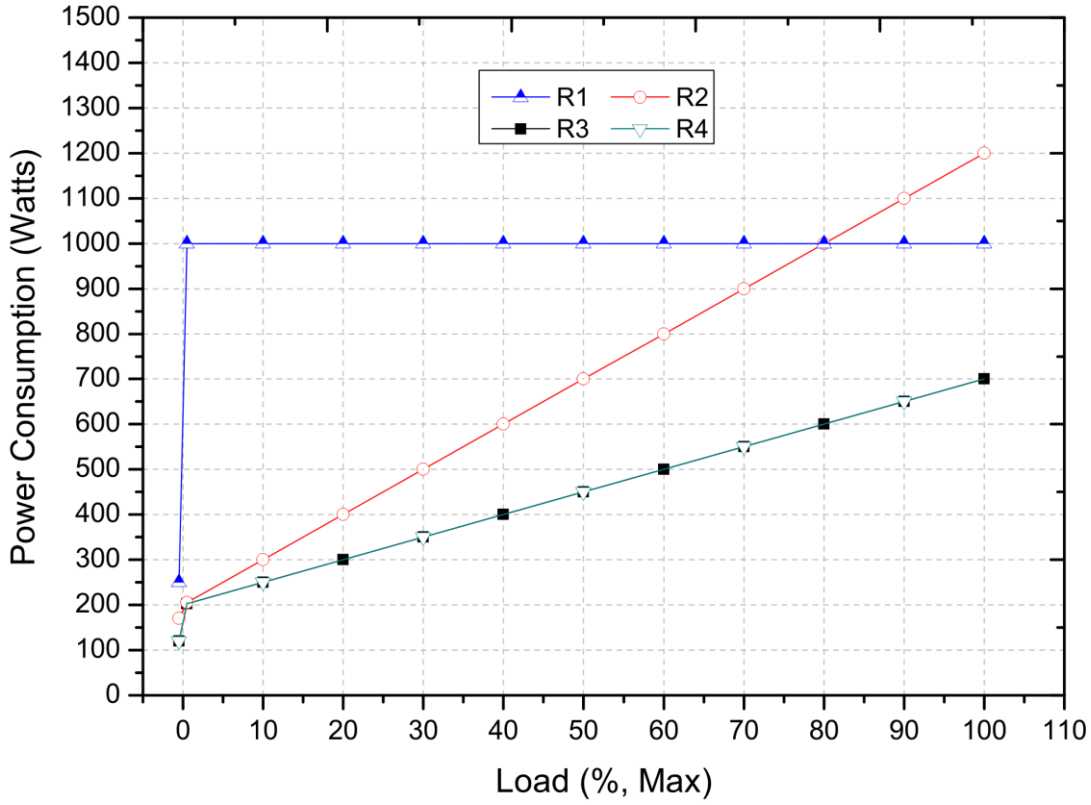


Figure 6.24. The power consumption according to the power profile of each router in the experiment.

(ii) Traffic profile

As a proof of concept, we defined a traffic profile composed of video streaming, which is increasingly becoming a significant component of IP network traffic. The incoming traffic surpassed 30Mbps at two moments of peak usage. The traffic profile used consists of four video

streams that start from the Servers 1 and 2 to the Client 1. The video streams start asynchronously, as shown in Figure 6.25.

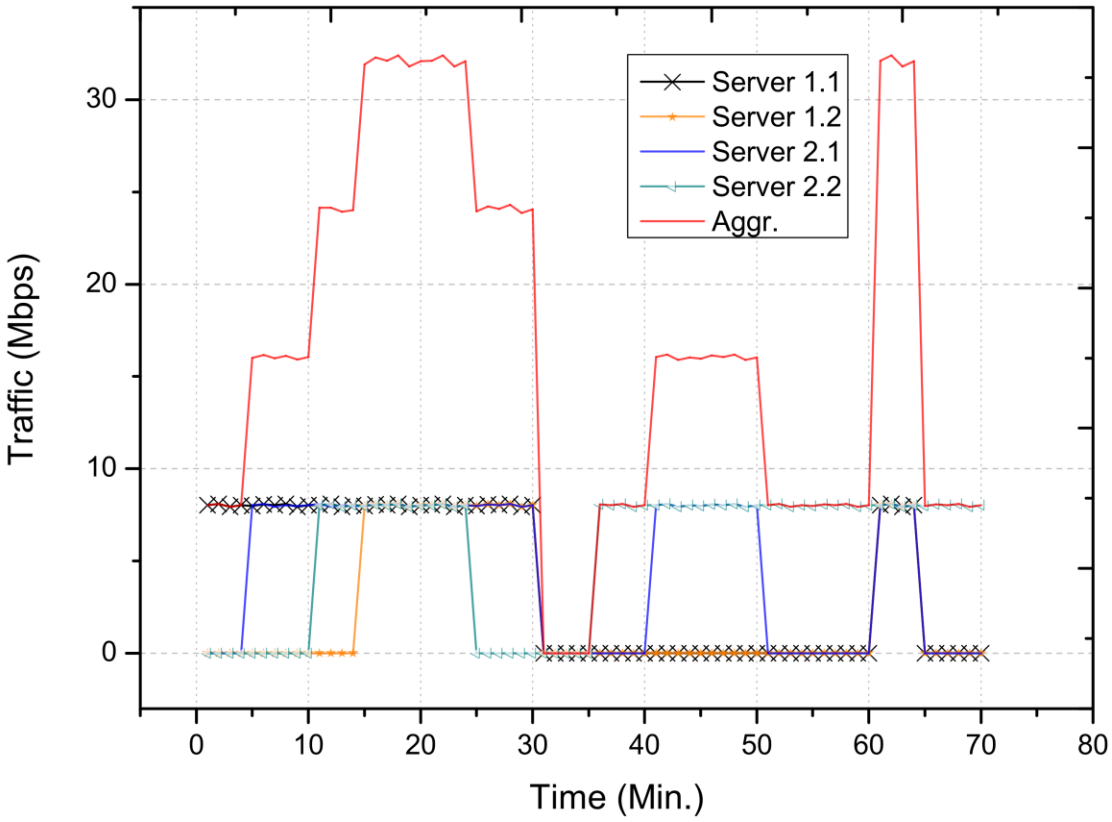


Figure 6.25. Traffic profile: 1.1 = stream 1, 1.2 = stream 2.

6.6.2. The experiments

The goal of the experiments is to explore how a sustainability-oriented policy can be developed and deployed in a network management system by using Ponder2. The sustainability-oriented policy is described at the network level. The network policy is refined until the instance level, which represents the commands performed by the PEP and DU. Three experiments that show how to develop policies and how to adapt Ponder2 to handle these policies are described in the following.

The first experiment (i) describes a constraint-free policy. In this policy, neither energy savings nor QoS constraints are specified. In this experiment, the paths 1 and 2 may be used to allocate traffic originated from Servers 1 and 2. Depending on the load generated from the servers, one or more paths may be used to deliver the data to the client. The devices that are not handling any load are also kept powered on; i.e., one of the devices can sometimes be idle. Having devices in idle mode allows one to perform traffic engineering and to use some devices as a fault protection mechanism. However, this consumes more energy compared to keeping some routers in sleep mode. Given that this is a constraint-free scenario, there is no PonderTalk code in this experiment.

The second experiment (ii) consists of a policy enforcing energy savings and accepting QoS degradation. This experiment presents energy savings, implying that one router will be put in sleep mode. Thus, if router 1 is sleeping, only path 2 is available to deliver the content from Servers 1 and 2 to the client. On the other hand, if router 2 is sleeping, only path 1 is available. Because this experiment accepts QoS degradation, there is no case in which both routers 1 and 2 are on; the network will always be saving energy. In this experiment, the energy savings are expected to be maximum. However, in this case, the network configuration should be less reliable (see [5]). This is because, in case a failure occurs, the chance of the network becoming unavailable is higher when the redundant device takes some time to wake up. Algorithms 7 and 8 describe the PonderTalk code for this experiment.

Algorithm 7: PonderTalk code for Experiment 2 (continues in Algorithm 8)

```
// This line is to ensure that policies will see all events, even if the values do
not change.
root/sustsys filter: false.

//The first rule verifies if the tunnel 1 (the upper tunnel) is not energy efficient.
//That is, the NECR, the index calculated by dividing the power consumption
(Watts) by the load, is bellow some pre-defined baseline.
//The router R1 is a typical example of a not energy efficient device, since it
expends the same amount of energy disregarding the load (Power
consumption always the same when turned on, even using minUtilization).
//The minUtilization load is 0.1Mpbs

//Creates a policy using the ecapolicy template from the factory
policy := root/factory/ecapolicy create.
policy event: root/event/sustValue.

policy condition: [ :name :tunnelId1 :tunnelId2 :NECR1 :NECR2 :on1 :on2 |
name == "Experiment2"
& (NECR1 < 9.7656)//means that the tunnel is not being used - NECR
calculated for minUtilization
& (on1 == 1)//and the router from tunnel1 (R1) is not turned off.
& (on2 == 1)//and if the main router from tunnel2 (R2) is not turned off.
//If yes, R1 can not be turned off, or else, there is no network anymore!].

policy action: [ :tunnelId1 |
root print: "Turning tunnel1 off".
on1 := 0. //This implies turning the R1 off
root/alarm setAlarm: true; show: "turn off "+tunnelId1.].

root/policy at: "P" put: policy.
policy active: true.
```

Regarding the network policy refinement, the device receiving the Ponder2 action must support power modes, such as sleep and power on (device level). By using SNMP commands, the instances of the device apply the actions and provide information for the upper layers (instance layer), as described in Section 6.4.

Algorithm 8: PonderTalk code for Experiment 2 (continued)

```
//The second rule verifies if the tunnel 2 (the lower tunnel) is not energy
efficient.
//That is, the NECR, the index calculated by dividing the power consumption
(Watts) by the load, is bellow some pre-defined baseline.
//The power consumption varies according to the equipment power profile
and, for this policy, is calculated considering the minUtilization load.
//The minUtilization load is 0.1Mpbs

//Creates a policy using the ecapolicy template from the factory
policy := root/factory/ecapolicy create.
policy event: root/event/sustValue.

policy condition: [ :name :tunnelId1 :tunnelId2 :NECR1 :NECR2 :on1 :on2 |
  name == "Experiment2"
  & (NECR2 < 2.1158)//if tunnel2 is not being used
  & (on2 == 1)//and the router from tunnel2 (R2) is not turned off.
  & (on1 == 1)//and if the main router from tunnel1 (R1) is not turned off.
  //If yes, R2 can not be turned off, or else, there is no network anymore!].

policy action: [ :tunnelId2 |
  root print: "Turning tunnel2 off".
  on2 := 0. //This implies turning the R2 off
  root/alarm setAlarm: true; show: "turn off "+tunnelId2.].

root/policy at: "P" put: policy.
policy active: true.
```

The third experiment (iii) consists of a policy that specifies energy savings enforcement but does not accept QoS degradation. Such a policy does not accept packet loss until the network reaches its capacity limit. If the woken up router is not fully loaded, the network configuration will behave similarly to that in experiment (ii), delivering data through paths 1 or 2. However, in case the router is overloaded, the redundant router must be woken up. This is because the policy applied does not accept QoS degradation. Algorithm 9 describes the PonderTalk code for this experiment. The policy refinement must be performed exactly as stated previously: the device receiving the Ponder2 action must support power modes, such as sleep and power on (device level). By using SNMP commands, the instances of the device apply the actions and provide information for the upper layers (instance layer), as described in Section 6.4.

All the experiments have the same aforementioned network topology, traffic profile, and power profile.

Algorithm 9: PonderTalk code for Experiment 3

```
// This line is to ensure that policies will see all events, even if the values do
not change.
root/sustsys filter: false.

//This rule verifies if the tunnel 1 (the upper tunnel) is losing packets, a QoS
metric. //It is verified if the packet loss is more than a "maxLoss" factor. //The
maxLoss here is 0.03

//Creates a policy using the ecapolicy template from the factory
policy := root/factory/ecapolicy create.
policy event: root/event/sustValue.

policy condition: [ :name :tunnelId1 :tunnelId2 :packetloss1 :packetloss2
:on1 :on2 |
name == "Experiment3"
& (packetloss1 > 0.03)//if tunnel1 is losing packets.
& (on2 == 0)//if the main router from tunnel2 (R2) is turned off.

policy action: [ :tunnelId2 |
root print: "Turning tunnel2 ON". //turn the second tunnel on
on2 := 1. //turn R2 on
root/alarm setAlarm: true; show: "turn on "+tunnelId2].

root/policy at: "P" put: policy.
policy active: true.

//This rule does the same, but for tunnel2

//Creates a policy using the ecapolicy template from the factory
policy := root/factory/ecapolicy create.
policy event: root/event/sustValue.

policy condition: [ :name :tunnelId1 :tunnelId2 :packetloss1 :packetloss2
:on1 :on2 |
name == "Experiment3"
& (packetloss2 > 0.03)//if tunnel2 is losing packets
& (on1 == 0)//if the main router from tunnel1 (R1) is turned off.

policy action: [ :tunnelId1 |
root print: "Turning tunnel1 ON". //turn the first tunnel on
on1 := 1. //turn R1 on
root/alarm setAlarm: true; show: "turn on "+tunnelId1].
root/policy at: "P" put: policy.
policy active: true.
```

6.6.3. Experiment results

The evaluation of the aforementioned experiments showed the energy savings for scenarios (ii) and (iii) when compared to scenario (i). In scenario (i), all the devices are always on, whereas in the other scenarios, there are periods in which some devices are in reduced power mode. The experiments were run with a 1-minute polling frequency to update the network data. The overhead introduced by the management system is determined by the polling of traffic counters and the centralized control signaling for switching between pre-configured paths. Figure 6.26 shows the power consumption for each scenario when the experiment is run with the power profile depicted in Figure 6.25.

The first scenario (i) does not achieve energy savings, presenting a total energy consumption of 751.54 MJ after 70 minutes. On the other hand, when a policy that enforces energy savings and accepts some loss of performance is applied (scenario (ii)), the total energy consumption is 506,38 MJ, indicating 33% energy savings when compared to scenario (i). However, real networks usually do not accept performance loss. Therefore, scenario (iii) reflects a more realistic experiment by enforcing energy savings and ensuring high performance constraints. Scenario (iii) has a policy that does not allow performance degradation. Hence, the energy savings achieved are not maximum, albeit representative of an 18% reduction when compared to scenario (i).

6.7. Final considerations

This course discussed the main topics regarding policy-based network management (PBNM), policy levels, and policy refinement, and their applicability to address sustainability issues. The work presented frameworks for policy management and methods of policy refinement. The combination of these two parts constitutes a holistic policy environment considering business policies, automated approaches, and sustainability. The widespread policy-based network management depends on the integration of a consistent and coherent approach to extend policies towards the business, network, and device levels. Thus, it is necessary to define a business language for policies and to translate these policies to the device level to finally use them to configure the network nodes automatically [75]. Among the analyzed methods, we studied Procera, a method intended for use in software-defined networks (SDNs). We believe that this is the next frontier for PBNM and that it brings opportunities for more intelligent and sustainable networks.

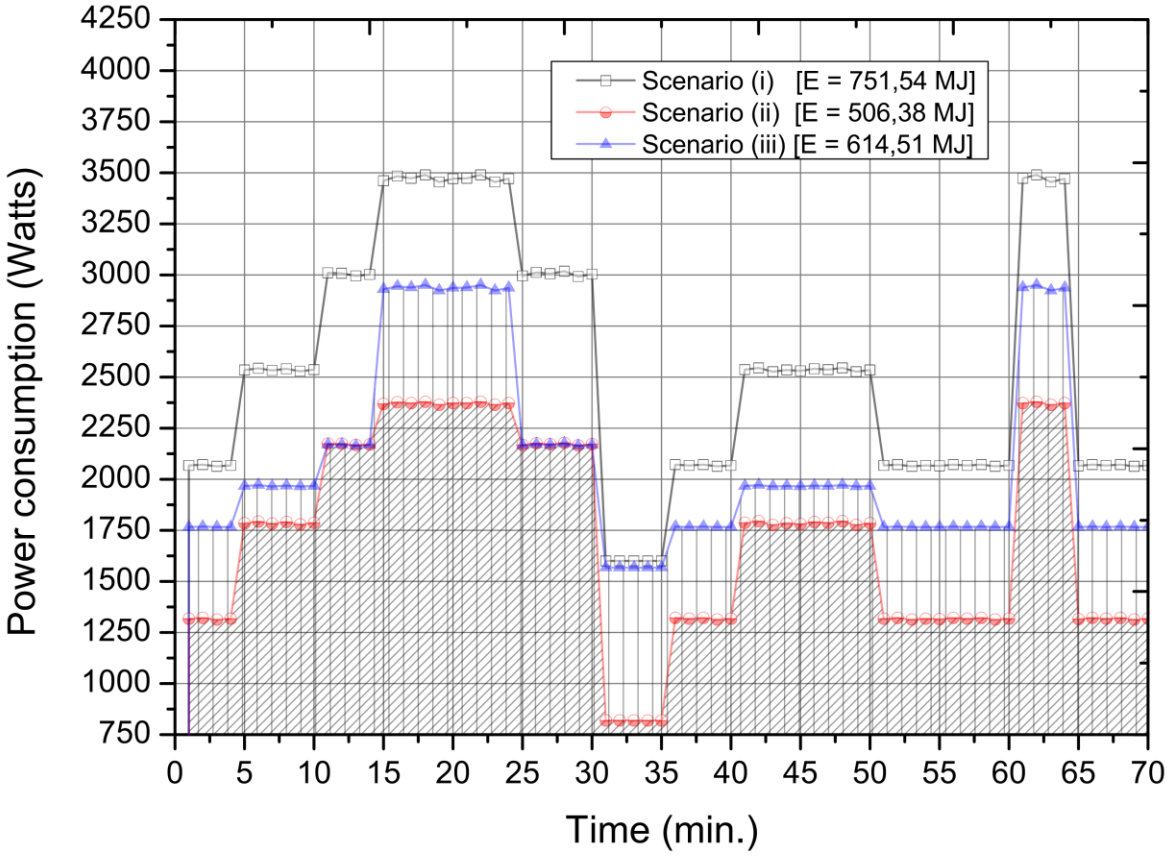


Figure 6.26. Power consumption for each evaluated scenario: (i), (ii), and (iii).

As a proof of concept, we used the Ponder2 framework (the only one available for download) and a methodological approach to policy refinement because the only solution available for download requires the input of experts. Ponder2 is a policy service has been used in many applications since its first version. It provides a domain service, message passing between managed objects, and an object-oriented command interpreter (PonderTalk). The challenges of using Ponder2 start with not finding complete examples available, only the Ponder2 tutorial regarding a body sensor network, with no conflicting policies. The tutorial also does not consider policy refinement. The papers published, perhaps due to the constraints in the number of pages, are not easily understandable and replicable. That is why we proposed this tutorial.

To fully present the case study, we also presented SustNMS, a sustainability-oriented network management system, as well as a test environment to be used in the sustainability context. We showed how to prepare Ponder2 and the policies relating sustainability and QoS described in PonderTalk. To conclude, we ran experiments consisting of policies related to energy efficiency and/or QoS, and the results showed expressive energy savings of 33% when QoS degradation is accepted and of 18% when it is not.

References

[1] Common information model (cim) infrastructure specification v2.3 final. Technical report, Distributed Management Task Force, DMTF, October 2005.

[2] A. Adelin, P. Owezarski, and T. Gayraud. On the impact of monitoring router energy consumption for greening the internet. In *Grid Computing (GRID), 2010 11th IEEE/ACM International Conference on*, pages 298 –304, Oct. 2010.

[3] N. Agoulmine. *Autonomic Network Management Principles: From Concepts to Applications*. Elsevier Science, 2010.

[4] I. Aib, N. Agoulmine, M. S. Fonseca, and G. Pujolle. Network control and engineering for qos, security and mobility ii. pages 26–50, Norwell, MA, USA, 2003. Kluwer Academic Publishers.

[5] M. C. Amaral, C. H. A. Costa, T. C. M. B. Carvalho, and C. Meirosu. REASoN - Reliability and/or availability evaluation for sustainable networking. In *4th Int. Work. on Reliable Networks Design and Modeling (RNDM'12)*, St. Petersburg, Russia, Oct. 2012.

[6] S. Antonakopoulos, S. Fortune, and L. Zhang. Power-aware routing with rate-adaptive network elements. In *GLOBECOM Workshops (GC Wkshps), 2010 IEEE*, pages 1428 –1432, dec. 2010.

[7] A. Arasu, S. Babu, and J. Widom. The cql continuous query language: semantic foundations and query execution. *The VLDB Journal*, 15(2):121–142, June 2006.

[8] S. Avallone and G. Ventre. Energy efficient online routing of flows with additive constraints. *Comput. Netw.*, 56(10):2368–2382, July 2012.

[9] A. K. Bandara, E. C. Lupu, J. Moffett, and A. Russo. A goal-based approach to policy refinement. In *Proceedings of the 5th IEEE Workshop on Policies for Distributed Systems and Networks*, pages 229–239. IEEE Press, 2004.

[10] M. Beigi, S. Calo, and D. Verma. Policy transformation techniques in policy-based systems management. In *Policies for Distributed Systems and Networks, 2004. POLICY 2004. Proceedings. Fifth IEEE International Workshop on*, pages 13 – 22, June 2004.

[11] A. P. Bianzino, K. Raju, and D. Rossi. Apple-to-apple: A framework analysis for energy-efficiency in networks, 2010.

[12] F. Blanchy, L. MÃc lon, and G. Leduc. A preemption-aware on-line routing algorithm for mpls networks. *Telecommunication Systems*, 24:187–206, 2003.

[13] R. Bolla, R. Bruschi, F. Davoli, and F. Cucchietti. Energy efficiency in the future internet: A survey of existing approaches and trends in energy-aware fixed network infrastructures. *Communications Surveys Tutorials, IEEE*, 13(2):223 –244, 2011.

[14] R. Bolla, F. Davoli, R. Bruschi, K. Christensen, F. Cucchietti, and S. Singh. The potential impact of green technologies in next-generation wireline networks: Is there room for energy saving optimization? *Comm. Magazine, IEEE*, 49(8):80 –86, 2011.

[15] T. C. M. B. Carvalho, A. C. Riekstin, M. Amaral, C. H. A. Costa, G. C. Januario, C. K. Dominicini, and C. Meirosu. Towards sustainable networks - energy efficiency policy from business to device instance levels. In L. A. Maciaszek, A. Cuzzocrea, and J. Cordeiro, editors, *ICEIS (3)*, pages 238–243. SciTePress, 2012.

[16] J. Chabarek and P. Barford. Power-awareness extensions for network testbeds. In *Communications Workshops (ICC), 2011 IEEE International Conference on*, pages 1 –6, June 2011.

[17] M. Chandramouli, B. Schoening, J. Quittek, T. Dietz, and B. Claise. Power and energy monitoring mib. Technical report, Network Working Group, IETF, May 2011.

[18] K. Christensen, P. Reviriego, B. Nordman, M. Bennett, M.Mostowfi, and J. Maestro. Ieee 802.3az: the road to energy efficient ethernet. *Communications Magazine, IEEE*, 48(11):50 – 56, November 2010.

[19] C. Costa, M. Amaral, G. Januario, T. Carvalho, and C. Meirosu. Sustnms: Towards service oriented policy-based network management for energy-efficiency. In *Sustainable Internet and ICT for Sustainability (SustainIT)*, 2012, pages 1 –5, oct. 2012.

[20] C. H. A. Costa, M. C. Amaral, G. Januario, T. C. M. B. Carvalho, and C. Meirosu. SustNMS: towards service oriented Policy-Based network management for Energy-Efficiency. In *Second IFIP Conference on Sustainable Internet and ICT for Sustainability (SustainIT 2012) (SustainIT 2012)*, Pisa, Italy, Oct. 2012.

[21] A. Courtney, H. Nilsson, and J. Peterson. The yampa arcade. In *Proceedings of the 2003 ACM SIGPLAN workshop on Haskell*, Haskell '03, pages 7–18, New York, NY, USA, 2003. ACM.

[22] R. Craven, J. Lobo, E. Lupu, A. Russo, and M. Sloman. Decomposition techniques for policy refinement. In *Network and Service Management (CNSM)*, 2010 International Conference on, pages 72 –79, Oct. 2010.

[23] R. Craven, J. Lobo, E. Lupu, A. Russo, and M. Sloman. Policy refinement: Decomposition and operationalization for dynamic domains. In *Network and Service Management (CNSM), 2011 7th International Conference on*, pages 1 –9, Oct. 2011.

[24] R. Craven, J. Lobo, E. Lupu, A. Russo, and M. Sloman. Policy refinement: Decomposition and operationalization for dynamic domains. In *Network and Service Management (CNSM), 2011 7th International Conference on*, pages 1–9, Oct.

[25] F. Cuomo, A. Cianfrani, M. Polverini, and D. Mangione. Network pruning for energy saving in the internet. *Computer Networks*, 56(10):2355 – 2367, 2012. <ce:title>Green communication networks</ce:title>.

[26] A. Dardenne, A. van Lamsweerde, and S. Fickas. Goal-directed requirements acquisition. In *Selected Papers of the Sixth International Workshop on Software Specification and Design*, 6IWSSD, pages 3–50, Amsterdam, The Netherlands, The Netherlands, 1993. Elsevier Science Publishers B. V.

[27] W. Dongmei and L. Guangzhi. Efficient distributed bandwidth management for mpls fast reroute. *Networking, IEEE/ACM Transactions on*, 16(2):486 –495, april 2008.

[28] N. Dulay, S. Heeps, E. Lupu, R. Mathur, O. Sharma, M. Sloman, and J. Sventek. Amuse: Autonomic management of ubiquitous e-health systems. In *Proceedings of the U.K. e-Science All Hands meeting*, 2005.

[29] E. Ellesson, B. Moore, J. Strassner, and A. Westerinen. Policy Core Information Model—Version 1 Specification. RFC 3060, RFC Editor, Fremont, CA, USA, Feb. 2001.

[30] P. Espada, M. Goulao, and J. Araujo. Measuring complexity and completeness of kaos goal models. In *Empirical Requirements Engineering (EmpiRE), 2011 First International Workshop on*, pages 29–32, Aug.

[31] F. I. for Human and M. C. (IHMC). Kaos project.

[32] GeSI. Smart 2020: Enabling the low carbon economy in the information age. Technical report, The Climate Group on behalf of the Global eSustainability Initiative (GeSI), June 2008.

[33] U. E. R. Group. Rei project.

[34] M. Gupta and S. Singh. Greening of the internet. In *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM '03, pages 19–26, New York, NY, USA, 2003. ACM.

[35] W. Heaven and A. Finkelstein. Uml profile to support requirements engineering with kaos. *Software, IEE Proceedings* -, 151(1):10–27, Feb.

[36] T. L. Hinrichs, N. S. Gude, M. Casado, J. C. Mitchell, and S. Shenker. Practical declarative network management. In *Proceedings of the 1st ACM workshop on Research on enterprise networking*, WREN '09, pages 1–10, New York, NY, USA, 2009. ACM.

[37] J. Hu and Y. Fu. Research on policy refinement method. In *Cyberworlds, 2008 International Conference on*, pages 800 –804, Sept. 2008.

[38] B. Janssen. Model-based diagnosis in integrated network management. *ERCIM News*, (12):17, 1993.

[39] G. Januário, C. Costa, M. Amaral, A. C. Riekstin, T. Carvalho, and C. Meirosu. Evaluation of a policy-based network management system for energy-efficiency. In *IM 2013 - TechSessions* (), May 2013.

[40] M. Jude. Policy-based management: beyond the hype, 2001.

[41] L. Kagal. Rei: A policy language for the me-centric project. Technical report, 2002.

[42] S. Keoh, K. Twidle, N. Pryce, E. Lupu, A. Schaeffer Filho, N. Dulay, M. Sloman, S. Heeps, S. Strowes, and J. Sventek. Policy-based Management for Body-Sensor Networks. In *4th InternationalWorkshop on Wearable and Implantable Body Sensor Networks (BSN 2007), Aachen, Germany, 26-28 March 2007. IFMBE Proceedings Vol 13*, pages 92–98, March 2007.

[43] S. L. Keoh, N. Dulay, E. Lupu, K. Twidle, A. E. Schaeffer-Filho, M. Sloman, S. Heeps, S. Strowes, and J. Sventek. Self-managed cell: A middleware for managing body-sensor networks. *Mobile and Ubiquitous Systems, Annual International Conference on*, 0:1–5, 2007.

[44] R. Kowalski and M. Sergot. A logic-based calculus of events. *New Gen. Comput.*, 4(1):67–95, Jan. 1986.

[45] C. Lange, D. Kosiankowski, R. Weidmann, and A. Gladisch. Energy consumption of telecommunication networks and related improvement options. *Selected Topics in Quantum Electronics, IEEE Journal of*, 17(2):285 –295, march-april 2011.

[46] E. Letier. Reasoning about agents in goal-oriented requirements engineering, Phd thesis, 2001.

[47] B. S. Liao and J. Gao. An automatic policy refinement mechanism for policy-driven grid service systems. In H. Zhuge and G. Fox, editors, *Grid and Cooperative Computing - GCC 2005*, volume 3795 of *Lecture Notes in Computer Science*, pages 166–171. Springer Berlin Heidelberg, 2005.

[48] J. Lobo, J. Ma, A. Russo, E. Lupu, S. Calo, and M. Sloman. Logic programming, knowledge representation, and nonmonotonic reasoning. pages 280–299, Berlin, Heidelberg, 2011. Springer-Verlag.

[49] I. C. London. Ponder2 project.

[50] L. Lymberopoulos, E. Lupu, and M. Sloman. An adaptive policy based framework for network services management. In *Journal of Network and Systems Management*, pages 277–303. Plenum Press, 2003.

[51] V. Manral. Benchmarking power usage of networking devices. Technical report, Network Working Group, IETF, May 2010.

[52] B. McBride. Jena: Implementing the rdf model and syntax specification. In *SemWeb*, 2001.

[53] J. Mccarthy. Situations, actions and causal laws. 1963.

[54] J. Moffett and M. Sloman. Policy hierarchies for distributed systems management. *Selected Areas in Communications, IEEE Journal on*, 11(9):1404 –1414, dec 1993.

[55] MPLSLinux. Mpls for Linux project.

[56] J. Nicklisch. A rule language for network policies. In Position Paper. *Policy Workshop. POLICY 2009*, 2009.

[57] N. F. Noy and D. L. Mcguinness. Ontology development 101: A guide to creating your first ontology. Technical report, 2001.

[58] G. Papamarkos, A. Poulovassilis, R. Poulovassilis, and P. T.Wood. Event-condition-action rule languages for the semantic web. In *Workshop on Semantic Web and Databases*, pages 309–327, 2003.

[59] T. Phan, J. Han, J.-G. Schneider, T. Ebringer, and T. Rogers. A survey of policy-based management approaches for service oriented systems. In *Software Engineering, 2008. ASWEC 2008. 19th Australian Conference on*, pages 392 –401, march 2008.

[60] E. Reinhard, B. Champion, N. Schulz, R. Gould, E. Perez, and N. Brown. Computer power consumption and management: "earth, wind, and fire: Sustainable energy for the 21st century"; research experience for undergraduates. In *Power Systems Conference and Exposition (PSCE), 2011 IEEE/PES*, pages 1 –8, march 2011.

[61] R. Romeikat, B. Bauer, and H. Sanneck. Automated refinement of policies for network management. In *Communications (APCC), 2011 17th Asia-Pacific Conference on*, pages 439 –444, oct. 2011.

[62] J. Rubio-Loyola. *Towards the Policy Refinement Problem in Policy-based Management Systems: A synthesis study*. VDM Publishing, 2008.

[63] J. Rubio-Loyola, J. Serrat, M. Charalambides, P. Flegkas, G. Pavlou, and A. Lafuente. Using linear temporal model checking for goal-oriented policy refinement frameworks. In *Policies for Distributed Systems and Networks, 2005. Sixth IEEE International Workshop on*, pages 181 – 190, June 2005.

[64] A. Schaeffer-Filho, E. Lupu, N. Dulay, S. L. Keoh, K. Twidle, M. Sloman, S. Heeps, S. Strowes, and J. Sventek. Towards supporting interactions between self-managed cells. In *Self-Adaptive and Self-Organizing Systems, 2007. SASO '07. First International Conference on*, pages 224 –236, July 2007.

[65] Y. Snir, Y. Ramberg, J. Strassner, R. Cohen, and B. Moore. Policy quality of service (qos) information model, 2003.

[66] Python snmp project.

[67] J. Strassner. *Policy-Based Network Management: Solutions for the Next Generation*. The Morgan Kaufmann Series in Networking. Elsevier Science, 2003.

[68] J. Strassner. *Policy-Based Network Management: Solutions for the Next Generation*. Morgan Kaufmann, 1st edition, Sept. 2003.

[69] J. Strassner and S. Schleimer. Policy framework definition language. Internet Draft draft-ietf-policy-framework-pfdl-00.txt, Nov 1998.

[70] G. Tonti, J. M. Bradshaw, R. Jeffers, R. Montanari, N. Suri, and A. Uszok. Semantic web languages for policy representation and reasoning: A comparison of kaos, rei, and ponder. pages 419–437. Springer, 2003.

[71] K. Twidle, E. Lupu, N. Dulay, and M. Sloman. Ponder2 - a policy environment for autonomous pervasive systems. In *Policies for Distributed Systems and Networks, 2008. POLICY 2008. IEEE Workshop on*, pages 245 –246, June 2008.

[72] Y. B. Udupi, A. Sahai, and S. Singhal. A classification-based approach to policy refinement. In *Proc. of The Tenth IFIP/IEEE IM*, 2007.

[73] A. Uszok, J. Bradshaw, R. Jeffers, N. Suri, P. Hayes, M. Breedy, L. Bunch, M. Johnson, S. Kulkarni, and J. Lott. Kaos policy and domain services: Toward a description-logic approach to policy representation, deconfliction, and enforcement. In *Proceedings of the 4th IEEE International Workshop on Policies for Distributed Systems and Networks*, POLICY '03, pages 93–, Washington, DC, USA, 2003.

[74] A. Uszok, J. Bradshaw, J. Lott, M. Breedy, L. Bunch, P. Feltovich, M. Johnson, and H. Jung. New developments in ontology-based policy management: Increasing the practicality and comprehensiveness of kaos. In *Policies for Distributed Systems and Networks, 2008. POLICY 2008. IEEE Workshop on*, pages 145 –152, June 2008.

[75] S. van der Meer, A. Davy, S. Davy, R. Carroll, B. Jennings, and J. Strassner. Autonomic networking: Prototype implementation of the policy continuum. In *Broadband Convergence Networks, 2006. BcN 2006. The 1st International Workshop on*, pages 1 –10, April 2006.

[76] A. Voellmy, H. Kim, and N. Feamster. Procera: a language for high-level reactive network control. In *Proceedings of the first workshop on Hot topics in software defined networks*, HotSDN '12, pages 43–48, New York, NY, USA, 2012. ACM.

[77] X. Wang, A. V. Vasilakos, M. Chen, Y. Liu, and T. T. Kwon. A survey of green mobile networks: Opportunities and challenges. *Mob. Netw. Appl.*, 17(1):4–20, 2012.

[78] G.Waters, J. Wheeler, A. Westerinen, L. Rafalow, and R. Moore. Policy framework architecture. Technical report, Network Working Group, IETF, Feb. 1999.

[79] M. Yousif. Keynote - towards green ict. *ERCIM News*, (79):3, October 2009.

[80] H. Zhao, J. Lobo, A. Roy, and S. Bellovin. Policy refinement of network services for manets. In *Integrated Network Management (IM), 2011 IFIP/IEEE International Symposium on*, pages 113 –120, May 2011.

[81] A. Y. Zomaya and Y. C. Lee, editors. *Energy-Efficient Distributed Computing Systems*. July 2012.