PhD-FSTC-2015-4
The Faculty of Sciences, Technology and Communication

DISSERTATION

Defense held on 22/01/2015 in Luxembourg

to obtain the degree of

DOCTEUR DE L'UNIVERSITÉ DU LUXEMBOURG

EN *Informatique*

by

Donia EL KATEB

Born on 23 June 1981 in Menzel Bourguiba, Tunisia

BALANCING NON-FUNCTIONAL REQUIREMENTS IN CLOUD-BASED SOFTWARE: AN APPROACH BASED ON SECURITY-AWARE DESIGN AND MULTI-OBJECTIVE SOFTWARE DYNAMIC MANAGEMENT

Abstract

Beyond its functional requirements, architectural design, the quality of a software system is also defined by the degree to which it meets its non-functional requirements. The complexity of managing these non-functional requirements is exacerbated by the fact that they are potentially conflicting with one another. For cloud-based software, i.e., software whose service is delivered through a cloud infrastructure, other constraints related to the features of the hosting data center, such as cost, security and performance, have to be considered by system and software designers. For instance, the evaluation of requests to access sensitive resources results in performance overhead introduced by policy rules evaluation and message exchange between the different geographically distributed components of the authorization system. Duplicating policy rule evaluation engines traditionally solves such performance issues, however such a decision has an impact on security since it introduces additional potential private data leakage points. Taking into account all the aforementioned features is a key factor to enhance the perceived quality of service (QoS) of the cloud as a whole. Maximizing users and software developers satisfaction with cloud-based software is a challenging task since trade-off decisions have to be dynamically taken between these conflicting quality attributes to adapt to system requirements evolution.

In this thesis, we tackle the challenges of building a decision support method to optimize software deployment in a cloud environment. Our proposed holistic method operates both at the level of 1) Platform as a service (PaaS) by handling software components deployment to achieve an efficient runtime optimization to satisfy cloud providers and customers objectives 2) Guest applications by making inroads into the design of applications to enable the design of secure systems that also meet flexibility, performance and cost requirements. To thoroughly investigate these challenges, we identify three main objectives that we address as follows:

The *first objective* is to achieve a runtime optimization of cloud-based software deployment at the Platform as a service (PaaS) layer, by considering both cloud customers and providers constraints. To fulfill this objective, we leverage the models@run.time paradigm to build an abstraction layer to model a cloud infrastructure. In a second step, we model the software placement problem as a multi-objective optimization problem and we use multi-objective evolutionary algorithms (MOEAs) to identify a set of possible cloud optimal configurations that exhibit best trade-offs between conflicting objectives. The approach is validated through a case study that we defined with EBRC[1], a cloud provider in Luxembourg, as a representative of a software component placement problem in heterogeneous distributed cloud nodes.

The *second objective* is to ameliorate the convergence speed of MOEAs that we have used to achieve a run-time optimization of cloud-based software. To cope with elasticity requirements of cloud-based applications, we improve the way the search strategy operates by proposing a hyper-heuristic that operates on top of MOEAs. Our hyper-heuristic uses the history of mutation effect on fitness functions to select the most relevant mutation operators. Our evaluation shows

[1]http://www.ebrc.com/

that MOEAs in conjunction with our hyper-heuristic has a significant performance improvement in terms of resolution time over the original MOEAs.

The *third objective* aims at optimizing cloud-based software trade-offs by exploring applications design as a complementary step to the optimization at the level of the cloud infrastructure, tackled in the first and second objectives. We aimed at achieving security trade-offs at the level of guest applications by revisiting current practices in software methods. We focus on access control as a main security concern and we opt for guest applications that manage resources regulated by access control policies specified in XACML[2]. This focus is mainly motivated by two key factors: 1) Access control is the pillar of computer security as it allows to protect sensitive resources in a given system from unauthorized accesses 2) XACML is the de facto standard language to specify access control policies and proposes an access control architectural model that supports several advanced access requirements such as interoperability and portability. To attain this objective, we advocate the design of applications based on XACML architectural model to achieve a trade-off between security and flexibility and we adopt a three-step approach: First, we identify a lack in the literature in XACML with obligation handling support. Obligations enable to specify user actions that have to be performed before/during/after the access to resources. We propose an extension of the XACML reference model and language to use the history of obligations states at the decision making time. In this step, we extend XACML access control architecture to support a wider range of usage control scenarios. Second, in order to avoid degrading performance while using a secure architecture based on XACML, we propose a refactoring technique applied on access control policies to enhance request evaluation time. Our approach, evaluated on three Java policy-based systems, enables to substantially reduce request evaluation time. Finally, to achieve a trade-off between a safe security policy evolution and regression testing costs, we develop a regression-test-selection approach for selecting test cases that reveal faults caused by policy changes.

To sum up, in all aforementioned objectives, we pursue the goal of analysing and improving the current landscape in the development of cloud-based software. Our focus on security quality attributes is driven by its crucial role in widening the adoption of cloud computing. Our approach brings to light a security-aware design of guest applications that is based on XACML architecture. We provide useful guidelines, methods with underlying algorithms and tools for developers and cloud solution designers to enhance tomorrow's cloud-based software design.

Keywords: XACML-policy based systems, Access Control, Cloud Computing, Trade-offs, Multi-Objective Optimization.

[2]https://www.oasis-open.org/committees/xacml

ACKNOWLEDGEMENTS

The PhD experience goes beyond research, experimentation and papers writing. It is indeed a challenging human experience. Thank you God for giving me the health and the strength to get through all the difficult moments.

The accomplishment of this challenging experience would have never been possible without the support of my supervisor Professor Yves Le Traon. Dear Yves, I would like to thank you and to express my deepest gratitude to you for never stopping in believing in me and encouraging me. I keep learning from your rigorous scientific guidance as a researcher and from you positive, motivating and open-minded attitude as a team leader.

I would like also to thank my two co-advisors Dr. François Fouquet and Dr. Tejeddine Mouelhi. Dear François, thank you for introducing me into the world of multi-objective optimization, for sharing your passion about several challenging research topics. I learnt a lot from your rigorous feedback and advice. Every brainstorming was a source of enjoyment and inspiration for me.

Dear Tej, thank you for enlightening my steps along the tough road to getting published. Exploring several challenges related to the access control research area was a great experience that would have never been possible without your encouragement and consistent support during all these 4 years.

My special thanks go also to the members of my dissertation committee: Professor Antonia Bertolino, Professor Pascal Bouvry and Professor Benoit Baudry. Dear Professors, thank you for investing time to review my work and for providing interesting and valuable feedback.

Thanks to all the members of the SerVal group. You have been a great source of joy and support during over the last four years. Thank you Chris, Mike and Jabier for your constructive feedback and interesting discussions. Thank you Thomas for the time you spent proof-reading carefully this whole dissertation. Thank you Claudia for all you kind assistance in all the administration steps of the thesis defense procedure.

I would also like to extend my thanks to my external co-authors, especially JeeHyun Hwang and Nicola Zannone for their help, feedback and valuable inputs.

I am grateful to all my friends all over the globe. The ones that I met before and during my PhD journey. Thank you for cheering me up whenever needed, for your care, support and love. My special thanks go to the three girls with whom I spent unforgettable moments during my PhD experience. Thank you Dalia for all the wise advices and for all the kind support. Thank you Wided and Lamia for caring and for the great time we shared.

My PhD experience was a great enriching cultural experience of a girl coming from overseas to another continent. I enjoyed and I am still enjoying my stay in the Grand Duchy of Luxembourg, a great a place to live, work and to meet international people. I am thankful to Luxembourgish institutions and particularly to the the University of Luxembourg for all the facilities that made my stay pleasant and Joyful. My kind and nostalgic thoughts go also to my home country, Tunisia where I did most of my studies and where I have been working in the Computer Science engineering field for 4 years. I am thankful to all the institutions where I have conducted my studies and where I have been working.

Finally, I want to express my deepest gratitude to my lovely family. Baba Moncef and Mama Saïda, I am so blessed to have you as parents, you have been always my role models in this life. The words will never be enough to thank you as you deserve. This piece of work is dedicated to both of you, for you unconditional love, care and support. Thank you for staying connected and close to your daughter despite the geographical distance between us. Dear sisters, Sarra and Mahassen, Dear brother, Allala, thank you for the joy you bring to my life on a daily basis, I love you all!

Dedication

To my parents: mama Saïda and Baba Moncef

For their unconditional love and endless support...

Table of Contents

List of Tables

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

This thesis was submitted to the University of Luxembourg, in partial fulfilment of the requirements to obtain a PhD degree in Computer Science. The work that is presented in this dissertation was carried out in the years 2011-2014. This first chapter introduces the structure of the overall manuscript and it is organized as follows. Section 1.1 gives a brief introduction of the context and the key motivating issues of this work. Section 1.2 outlines the approach adopted in this research. Section 1.3 presents the objectives of this thesis and finally, section 1.4 presents an overview of the remainder of this thesis.

Contents

1.1 Context and problem statement

With the spread of ICT companies, software services that range from web services to social network services have revolutionized our lives. For most of these ICT companies, managing the life-cycle of applications is still a tedious task causing most of these firms to be reluctant to perform dreading negative return on investment (ROI).

According to the NIST definition, cloud computing [MG11] is a computing paradigm that enables cloud providers to provision dynamically their customers with configurable hardware and software resources through network access. Cloud computing offers hosting capabilities that enable cloud customers to get rid of the burden of maintaining their own services by leveraging cloud hardware and virtual resources. Cloud paradigm enables to reduce deployment costs by relying on fees that are changing every month instead of a fixed license. Besides its cost efficiency, cloud computing offers unlimited storage capacity of information, backup and data recovery facilities and dynamic on-demand resources provisioning to cope with load variations.

According to the TechTarget's Cloud Adoption Index[1], public and private clouds have both reached 25% adoption rate within IT. These numbers reveal that the process of migrating applications to the cloud is slow despite the promising solutions offered by the cloud industry to the IT companies.

Cloudsourcing [KV10] introduces several design and deployment challenges. A major example of these challenges is handling non-functional requirements trade-offs for software services that have to be revisited in the context of virtualized environments. Balancing these non-functional requirements is an important factor in improving software quality and in increasing cloud customers satisfaction. The difficulty related to the management of cloud-based software trade-offs arises mainly from two main major concerns:

1. Software systems deployed in the cloud have to cope with constraints that are related to the intrinsic features of cloud computing such as elasticity [ILFL12], resource heterogeneity, variable deployment costs, etc. Besides, cloud computing introduces new complexity by introducing several stakeholders such as cloud providers, cloud third parties and cloud tenants [GSH+07] which can share the same pool of resources. The dynamicity of cloud stakeholders and resources introduces new challenges related to the optimization mechanisms that have to be considered in cloud-based software.

2. As stated in [SP12], deploying an application in a cloud environment will not enhance its software quality attributes if these attributes have not already been taken into consideration at the early stages of the software development cycle. Software services have to take benefit of software engineering methods to develop cloud applications that meet users expectations in terms of quality of service, before moving these applications to the cloud. This implies that software engineers have to maximize non functional requirements satisfaction when building cloud applications. When designing a software system, security is one important objective to consider between other non functional requirements.

   Security poses new challenges for cloud environments. Cloud customers lose control over their data when they host their applications in cloud infrastructures. Security often conflicts with performance, flexibility or other software quality attributes. However, these non-functional requirements are often highly interdependent and it is not easy for software engineers to identify these trade-offs at the design time or to handle them dynamically when the system is in an operational state. Therefore, building a secure cloud-based software that exhibits trade-offs between security and other objectives is a challenging task.

[1]http://searchcloudcomputing.techtarget.com/feature/Cloud-computing-adoption-numbers-dont-add-up-to-vendor-noise

In this thesis, we propose a decision support to address such trade-offs for guest applications (i.e., software that is delivered through a cloud infrastructure). First, we analyse cloud environment constraints and we propose a decision support method to achieve an optimal software provisioning in a cloud environment to satisfy simultaneously trade-offs between several conflicting objectives. Secondly, to improve trade-offs satisfaction at the level of guest applications, we focus on the access control security at the level of guest applications and we propose an approach to maintain secure application design while reducing detrimental effects on other software quality attributes such as flexibility and performance. The next section details our approach.

1.2 Approach description

We leverage software engineering methods and cloud computing paradigm to improve trade-offs in cloud-based software. Software engineering methods enable to provide guidelines to develop applications that meet functional and non functional requirements. Cloud computing promises a delivery model that enables to improve the deployment of these applications through virtualized infrastructures. In this thesis, we provide a decision support to handle software architecture trade-offs that combines cloud computing and software engineering paradigms. Our approach is described as follows:

1) ***Optimization of software deployment in a cloud environment:***

In this part, we consider software systems as black box components and we focus on the constraints related to the cloud environment such as elasticity and variable costs. We model provider and customer constraints as a multi-objective optimization problem and we leverage models@run.time and search-based engineering techniques to provide optimal software placement alternatives. Our optimization approach results in cloud configurations that exhibit trade-offs between conflicting objectives. In the second step, we have enhanced the search-based method to cope with cloud elasticity constraints by operating on the search process that we have used previously to find acceptable cloud configurations. Our goal is to cope with continuous optimization constraints and to reduce the costs of adapting a running system at run-time.

2) ***An approach for the design of guest applications:***

Trade-offs resolution has to be considered at the early stages of applications design. When designing a software application, access control mechanisms have to be enforced to restrict users access to sensitive resources. A leakage is access control security mechanisms might lead to data losses which might have huge impact on organizations business and reputation. In this thesis, we mainly consider access control as a security concern and we propose an approach to design applications so that access control security does not come at the detriment of other quality attributes such as flexibility and performance. We focus on systems that are built upon XACML [2] as an architectural model and as a language to encode access control policies. XACML

[2] https://www.oasis-open.org/committees/xacml

is a de facto standard to encode access control policies and to design a system that enforces access control mechanisms. XACML enables to encode standard policies that can be reusable in heterogeneous platforms and defines a conceptual model of an access control architecture. XACML-based systems are built based on the separation between a security policy that regulates user access to sensitive resources in the system and the underlying interacting software system. This separation contributes to achieve better flexibility in terms of policy and system management. For such systems based on XACML architectural design, we have used three Java policy-based systems, regulated by XACML policies and we have:

- Extended the current XACML model to manage user actions in addition to authorizations in the objective to enhance the support of XACML to handle complex usage control scenarios.

- Reduced time overhead that is inherent from request evaluation time in XACML-based systems to achieve a certain trade-off between security and performance.

- Proposed an optimal security policy safe evolution by reducing test costs through test regression techniques to test efficiently the software system when the security policy evolves.

We aimed at proposing a software architecture design that secures user access without having a negative impact on other quality attributes such as cost and performance. We believe that performance and cost considerations have to be considered in the very early stages of software development.

1.3 Thesis contributions

This dissertation presents the following concrete outcomes, which can be classified according to the approach organization provided in section 1.2.

Optimizing software deployment in a cloud infrastructure: We designed a framework embedded with decision capabilities to optimize software deployment in a cloud environment:

A) Chapter 4 entitled *"Cloud-based software management: A multi-objective optimization problem"*: We propose a generic approach to build a cloud-based software optimization layer by combining architectural modeling and search-based paradigms. Our approach is two-fold:

- We use a models@run.time approach to have an abstraction layer of a cloud configuration that represents cloud intrinsic parameters like cost, load information, etc.

- We use a multi-objective algorithm to navigate through cloud candidate configuration solutions in order to resolve software deployment trade-offs in the cloud.

We validate our approach based on a case study that we defined with a cloud provider partner EBRC[3] as representative of a dynamic management problem of heterogeneous distributed cloud nodes. The prototype enables to find possible cloud configurations in reasonable time. The prototype is flexible since it enables an easy reconfiguration of the cloud customer optimization objectives.

B) Chapter 5 entitled *"Optimizing Multi-Objective Evolutionary Algorithms to enable quality-aware software provisioning"*: We propose an hyper-heuristic [HK03] that operates on top of multi-objective algorithms to improve the search methods to cope with cloud environment runtime constraints. Our hyper-heuristic leverages the history of mutation efficiency to select the most relevant mutations to perform. We evaluate our hyper-heuristic on a SaaS engine, which drives on-demand provisioning while considering conflicting performance and cost objectives. We conduct experiments to highlight its significant performance improvement in terms of resolution time.

Guest application design: an application to policy-based systems: To design secure XACML-based solutions while maintaining non functional requirements such as flexibility, performance and cost, we have tackled the following challenges and we have addressed them as follows:

C) Chapter 6 entitled *"Towards a full support of obligations in XACML"*: Security is one of the major concerns that have to be considered when designing a software system. Access control is one important aspect of computer security that defines the actions that the different users can/cannot perform on the the different resources in a given system. Architecting a software system using XACML design enables to build a secure system, which allows only eligible users to access the authorized information. XACML is built upon the separation of the security policy that defines the user access and the different components that are involved in the access such as the Policy Decision Point (PDP), which specifies the access decision (i.e., access/deny) by fetching the policy and the Policy Enforcement Points (PEPs) that enforce the access decision at the level of the software system. A solution that is built based on XACML provides a certain level of trade-off between security and flexibility: The update of the system that interacts with a security policy is not needed whenever the security policy evolves. XACML is based on an attribute-based access control model that enables the developers to encode fine-grained access control policies. Chadwick[4] *et al.*, have pointed out that while authorizations are well captured by authorizations in XACML, users duties, also called obligations, are not well managed by XACML architecture. The current version of XACML lacks (1) well-defined syntax to express obligations and (2) an unified model to handle decision making w.r.t. obligation states and the history of obligations fulfillment/violation. To extend XACML to improve its support to obligations, we have:

- Proposed an extension of the XACML reference model that takes into consideration the history of obligation states (permission, violation) when building the access decision.

[3] http://www.ebrc.com/

[4] http://www.w3.org/2009/policy-ws/papers/Chadwick.pdf

- Extended XACML language and architecture for a better obligation support and have shown how obligations are managed in our proposed extended XACML architecture: OB-XACML.

D) Chapter 7 entitled *“Improving performance in XACML-based systems”*: Software systems can become very complex with an increasing number of users and resources in a given system. This results in long and complex policies, which introduce a time overhead in access control request processing and contributes to downgrade the performance of the overall system. To improve performance in XACML-based systems:

- We propose to refactor access control policies by splitting a policy into smaller policies according to a defined splitting criteria that are based on XACML intrinsic parameters.
- We have conducted an evaluation on three subjects of real-life Java systems, each of which interacts with access control policies. Our evaluation results enable to substantially reduce request evaluation time for most splitting criteria.

E) Chapter 8 entitled *“Selection of regression system tests for security policy evolution”*: Testing is an important phase in a software life cycle. Regression testing has to be conducted when the system evolves to ensure that there is no errors that have been introduced by the system evolution or maintenance. Regression testing is a very crucial task, however, it is also very expensive as it can account for a huge amount of the total cost of a software system. To achieve a trade-off between regression testing costs and security policy evolution:

- We developed a regression-test-selection approach, which selects every system test case that may reveal regression faults caused by policy changes.
- We conducted experiments on three policy-based systems and we showed that our test-selection approach reduces a substantial number of system test cases efficiently.

We published the following papers in the context of this thesis. Some of these papers will be presented in the chapters that describe the contribution:

- **Donia El Kateb**, Tejeddine Mouelhi, Yves Le Traon, JeeHyun Hwang, and Tao Xie. Refactoring access control policies for performance improvement. In Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering (ICPE), pp 323-334, 2012.
- JeeHyun Hwang, Tao Xie, **Donia El Kateb**, Tejeddine Mouelhi, and Yves Le Traon. Selection of regression system tests for security policy evolution. In Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering (ASE), pp 266-269, 2012.

- **Donia El Kateb**, Yehia ElRakaiby, Tejeddine Mouelhi, and Yves Le Traon. Access control enforcement testing. In Proceedings of the 8th International Workshop on Automation of Software Test (AST), pp 64-70, 2013.
- **Donia El Kateb**, François Fouquet, Grégory Nain, Jorge Augusto Meira, Michel Ackerman, and Yves Le Traon. Generic cloud platform multi-objective optimization leveraging models@run.time. In Proceedings of the 29th Annual ACM Symposium on Applied Computing (SAC), pp 343-350, 2014.
- **Donia El Kateb**, Nicola Zannone, Assaad Moawad, Patrice Caire, Grégory Nain, Tejeddine Mouelhi, and Yves Le Traon. Conviviality-driven access control policy. Requirements Engineering Journal, 2014.
- Antonia Bertolino, Said Daoudagh, **Donia El Kateb**, Christopher Henard, Yves Le Traon, Francesca Lonetti, Eda Marchetti, Tejeddine Mouelhi, and Mike Papadakis. Similarity testing for access control. Information and Software Technology Journal, 2014.
- **Donia El Kateb**, François Fouquet, Johann Bourcier, and Yves Le Traon. Optimizing multi-objective evolutionary algorithms to enable quality-aware software provisioning. In Proceedings of the 14th International Conference on Quality Software (QSIC), pp 85-94, 2014.
- **Donia El Kateb**, Yehia ElRakaiby, Tejeddine Mouelhi, Iram Rubab and Yves Le Traon. Towards a Full Support of Obligations In XACML. In Proceedings of the 9th International Conference on Risks and Security of Internet and Systems (CRiSIS), pp 213-221, 2014.
- Said Daoudagh, **Donia El Kateb**, Francesca Lonetti, Eda Marchetti, Tejeddine Mouelhi. A Toolchain for Model-Based Design and Testing of Access Control Systems. In Proceedings of the 3rd International Conference on Model-Driven Engineering and Software Development, 2014 (To appear).

1.4 Thesis structure

This thesis starts by a chapter that introduces the major problems that we have tackled in this thesis followed by a background chapter that presents the relevant key concepts used throughout our work. The main contributions, that have been overviewed in 1.3 are structured in 5 main chapters presented between chapter 4 and chapter 8. The conclusion and future work chapters conclude and recapitulate the main contributions of our work. We also outline the limitations of our current work and discuss our future research directions.

CHAPTER 2

CONTEXT AND PROBLEM ANALYSIS

In the last decade, cloud computing has been gaining momentum by leveraging hardware and software resources to provide IT services as utility services [BYV08]. Besides its impact on companies' business models, the shift to cloud computing has a substantial impact on software delivery and development. In this section, we introduce cloud-based software and we highlight the challenges behind designing software that exhibits trade-offs between conflicting objectives.

Contents

2.1 Cloud-based software: definition and specific features

In this section, we first introduce the key factors that lead to the software evolution towards the cloud computing paradigm and then highlight the main specific features of cloud-based software.

2.1.1 Software evolution

The era of cloud computing appears today as a natural evolution of emerging technologies in the last few decades. This evolution is analysed in [BBG10]. In the early 60's and 70's, mainframes have revolutionized the IT industry taking the advantages of cheap microprocessors to provide intensive scientific computing. Distributed client/server architecture has emerged in the 1980's anticipating services delivery through service oriented architecture (SOA) [Erl08] supported by open standards to deliver and publish web services. Grid computing [FZRL08] has emerged to provide distributed services through the aggregation of large-scale clusters. To provide an efficient service usage as an alternative to flat rates, resources have been provided as utility services such as gaz and electricity. Utility computing [BVB08] aims at adjusting prices to customers consumption, which can vary from the number of used memory or the number of running virtual machines per hour/day. Autonomous computing appears as a computing paradigm that describes systems that are able to operate in their environments, without human intervention and with self-* properties such as self-configuration, self-optimization, self-healing and self-protection. In this paradigm, self adaptive systems describe systems that are able to adjust their behaviour to adapt to changes in their underlying environment. The MAPE-K adaptation control loop (Monitor, Analyse, Plan, Execute, Knowledge) presented by IBM [IBM] specifies the main architectural aspects of autonomous systems: The *monitor* function collects data and tracks changes that are are needed by the *analyse* function to perform reasoning tasks. The *plan* function uses data analysis results to perform a set of objectives that are executed by the *execute* function. Cloud computing emerged as a convergence of these concepts leveraging virtualization techniques to provide on-demand computing resources. According to the *NIST* definition, cloud computing [MG11] takes its definition from the following features:

1. *On-demand self-service*: Cloud services are provisioned automatically under cloud customer demands and delivered with minimal intervention of cloud providers.

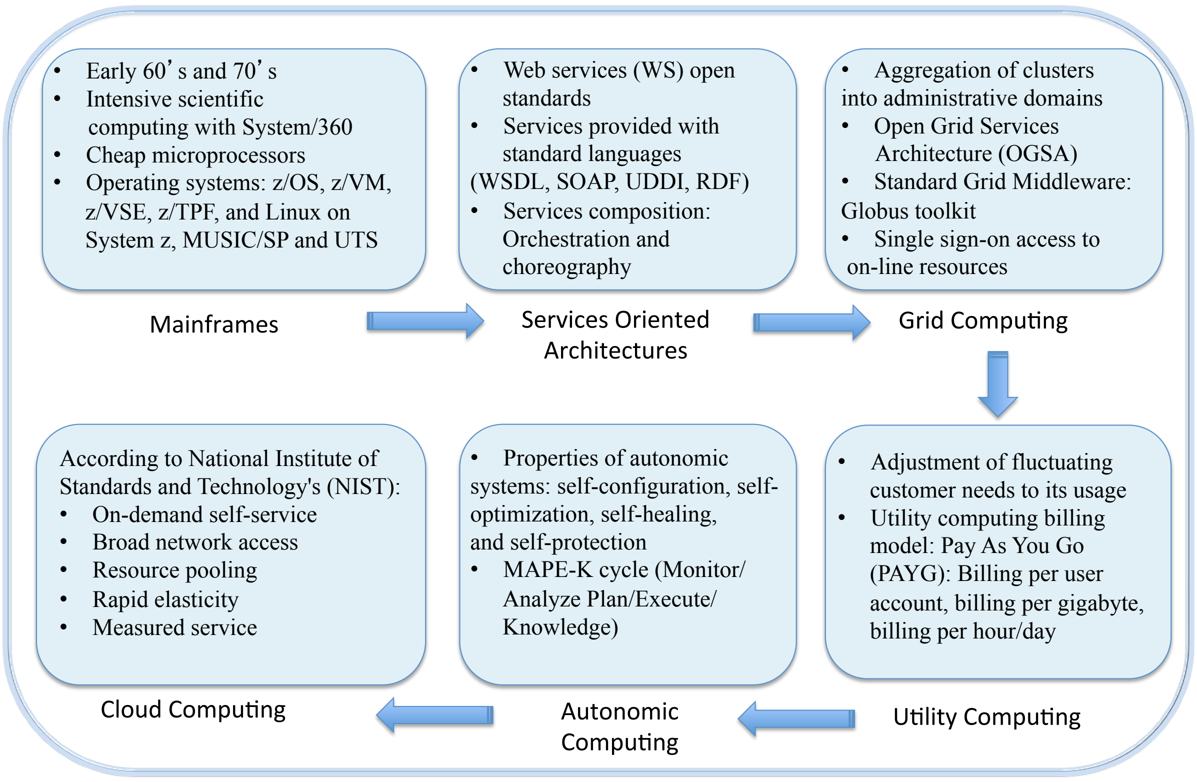

Early 60's and 70's
Intensive scientific computing with System/360
Cheap microprocessors
Operating systems: z/OS, z/VM, z/VSE, z/TPF, and Linux on System z, MUSIC/SP and UTS
Mainframes
Web services (WS) open standards
Services provided with standard languages (WSDL, SOAP, UDDI, RDF)
Services composition: Orchestration and choreography
Services Oriented Architectures
Aggregation of clusters into administrative domains
Open Grid Services Architecture (OGSA)
Standard Grid Middleware: Globus toolkit
Single sign-on access to on-line resources
Grid Computing
Adjustment of fluctuating customer needs to its usage
Utility computing billing model: Pay As You Go (PAYG): Billing per user account, billing per gigabyte, billing per hour/day
Utility Computing
Properties of autonomic systems: self-configuration, self-optimization, self-healing, and self-protection
MAPE-K cycle (Monitor/ Analyze Plan/Execute/ Knowledge)
Autonomic Computing
According to National Institute of Standards and Technology's (NIST):
On-demand self-service
Broad network access
Resource pooling
Rapid elasticity
Measured service
Cloud Computing

Figure 2.1: Evolution towards cloud paradigm

2. *Broad network access*: Cloud customers can access cloud resources through heterogeneous devices, machines and platforms over the network.

3. *Resource pooling*: In the cloud computing paradigm, resources are scalable and adjusted to cloud tenants. The cloud provider manages the sharing of resources between the different cloud customers called “tenants” in a transparent manner.

4. *Rapid elasticity*: Cloud elasticity refers to the capacity of the cloud to adapt its resource provisioning to the variable resource demand through a scale out and scale in operations.

5. *Measured Service*: Cloud providers use a set of tools and mechanisms to monitor resources provisioning, deployment cost plans, bandwidth resource allocation, security enforcement mechanisms, etc.

We summarize the main features of the aforementioned paradigms in Figure 2.1. Cloud computing is a layered model that provides a stack, which is shown in Figure 2.2. This stack reflects the following delivery models:

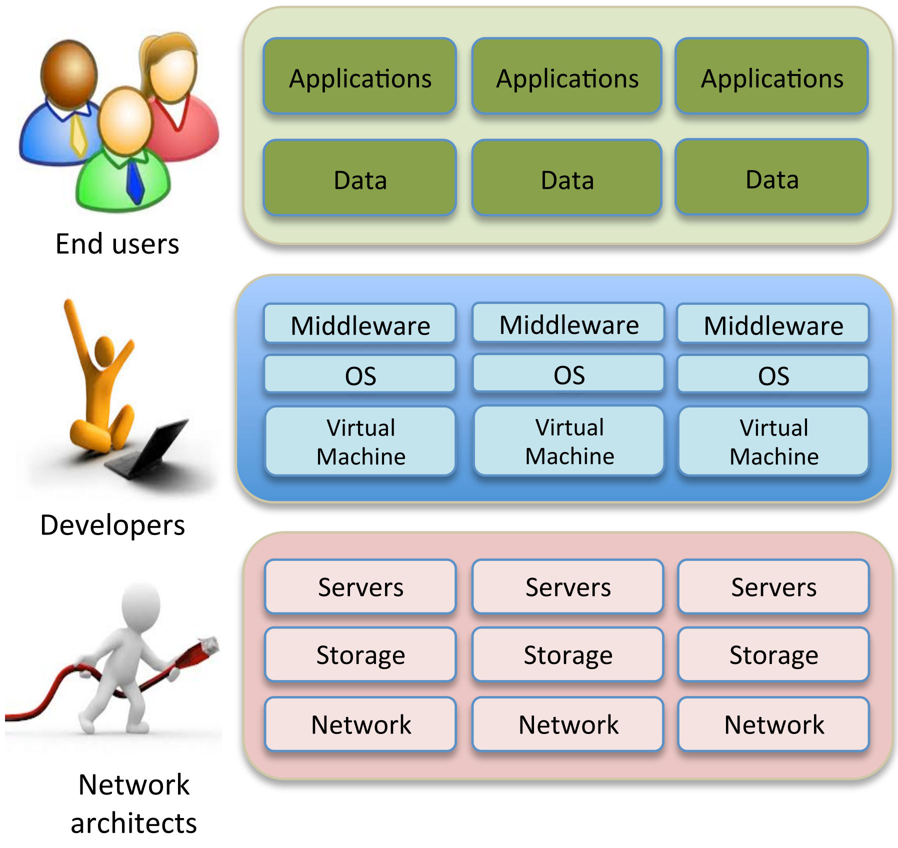


Figure 2.2: Cloud layered stack

- *Infrastructure as a Service (IaaS)*: This layer includes the hardware and the network services that are delivered by the provider so that the customer can host its running operating system and applications.
- *Platform as a Service (PaaS)*: Provides platforms and facilities that can be used by the customers to host their applications.
- *Software as Service (SaaS)*: In this delivery model, customers use the applications that are hosted in the cloud environment.

2.1.2 Cloud-based software features

In the remainder of this manuscript, we refer to cloud-based software as any software that is delivered through a cloud infrastructure. This includes both the software that is hosted in a cloud environment and SaaS [XL08] applications. The former denotes the applications that are owned by some users and hosted in the cloud to take advantages from cloud computing facilities whereas the latter refers to used services through SaaS layer. Cloud-based software is characterized by the following features:

- Cost reduction: Cloud solutions enable to avoid high upfront investments due to the following reasons:
  - *Pay as you use* – Cloud-based software deployment is based on an on-demand pricing model that enables customers to pay for their provisioned resources based on adjustable fees instead of a fixed license. Cloud software services are delivered through virtualization techniques [WVLY+10] and customers pay generally based on the number of virtual machines, which changes over time to adjust to variable demands.
  - *Ubiquitous access* – By hosting a software in a cloud environment, customers can benefit from services that are widely accessible, from anywhere and on any device. Furthermore, IT business customers are able to reduce the costs of hardware maintenance and of software release updates.
- *Elasticity* – Cloud elasticity refers to the capacity of the cloud to adapt its resource provisioning to the variable resource demands by a given service while maintaining its conformance to Service Level Agreements (SLAs) that specify the terms of the service between cloud customers and cloud providers [Bre12].
- *Multi-tenancy* – Multi-tenancy [BZP+10] is a key characteristic of cloud computing that allows cloud customers (tenants) to share one single application instance with customized front-ends and databases that fit each tenant's business logic. Multi-tenancy also reduces costs by enabling providers to share the hardware and the software resources between several customers. Multi-tenancy can be set up through different levels such as separated databases, separated tables or shared tables between tenants. It introduces several security considerations due to the sharing of services and resources, which increases the risk of failure that can be engendered by the misbehaviour of one tenant.

2.2 Cloud-based software deployment

A cloud infrastructure is a dynamic environment, that is driven by on-demand provisioning and pay-as-you-go model, in which customers and providers are expressing evolving requirements towards cost saving, minimization of energy consumption, QoS evolution, etc. A cloud infrastructure as shown in Figure 2.3 can be abstracted by a set of Physical Machines (PMs) that are organized in clusters. Each physical machine hosts one or many virtual machines (VMs). Virtualization enables to mitigate the effects of hardware resource constraints [UNR+05]. Enacted through a software layer called the hypervisor, virtualization enables to emulate a guest operating system within a hosting operating system allowing to run more than one operating system in a single

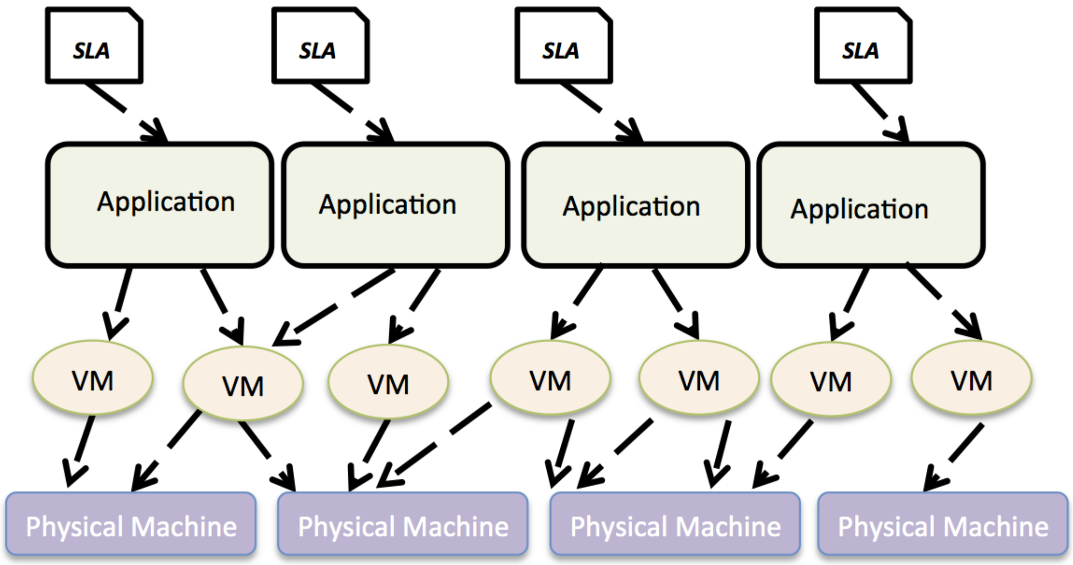


Figure 2.3: Abstraction of a cloud infrastructure

physical machine. The cloud provider is an entity that owns computing resources and makes those resources available to the customers. Customers benefit from the virtualization technology to host their data/applications. Applications hosting in the cloud is governed by terms of contracts between the cloud customer and provider. These terms are commonly referred to as Service Level Agreements (SLAs) [PRS09], which are used to define constraints related to service parameters such as cost, performance, reliability, etc.

2.2.1 Trade-offs analysis in cloud-based software deployment

Nowadays, cloud computing offers services that go beyond web services to include storage, data processing and big data services. The adoption of cloud services is tied to software services quality. The quality of a software system is often defined by the degree to which it satisfies its non-functional requirements, referred to as software quality attributes [BDE+05], such as performance, security, availability, etc. Since these attributes can often conflict in practice, software designers aim at achieving a trade-off among the quality attributes to maximize user satisfaction with the system. Making a trade-off between interdependent attributes requires to balance these attributes or to improve some of them on the detriment of others, depending on the design priorities that can be configured by the system designer.

Hauck *et al.* [HHK+10] have discussed the different trade-off decisions inherent in cloud computing and have pointed out the relevancy of considering these design trade-offs in cloud-based software, to enhance the perceived quality of service (QoS) of the cloud as a whole. However, the different conflicts among the quality attributes makes this endeavour a complex and tedious

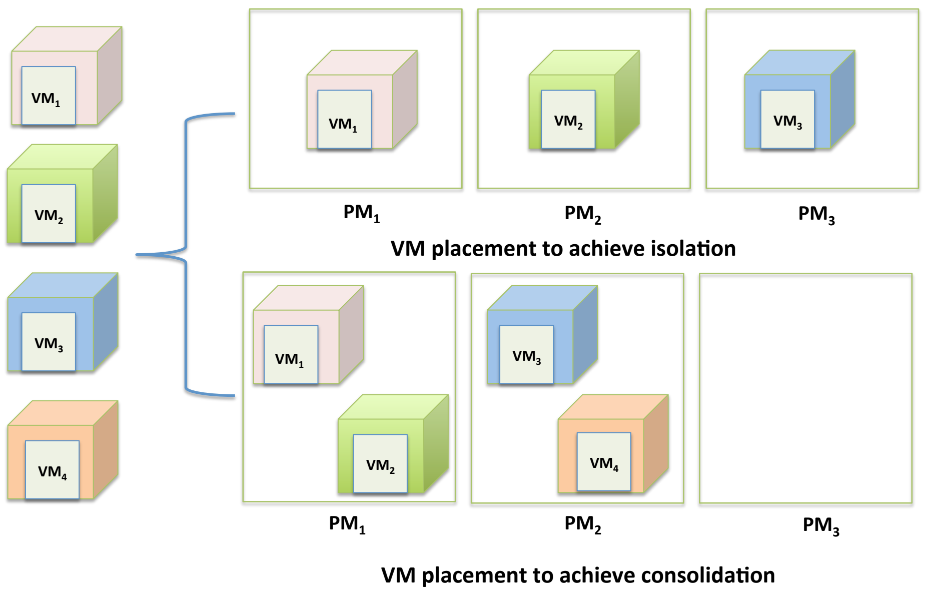

VM$_1$
VM$_2$
VM$_3$
VM$_4$
PM$_1$
PM$_2$
PM$_3$
VM placement to achieve isolation
VM placement to achieve consolidation

Figure 2.4: Virtual machines placement trade-offs

one. For instance, *performance* can be compromised by the setup of *security* mechanisms: access control mechanisms for checking that only eligible users can access a sensitive resource in a given system may introduce performance overhead due to the number of operations to verify user access rights. Cloud computing offers an elastic production environment, in which various trade-offs such as security and performance, have to be considered. An illustrating example of these trade-offs is the one related to isolation and energy consumption [LZX13]. For security reasons, it is better to isolate applications in different virtual machines to avoid security attacks that can result from VMs interference. To achieve high availability, software components have to be dispatched in separate physical machines to improve high availability. In this case, the failure of one server will not result in the failure of the overall system. However, from the provider perspective, it is desirable to reduce the number of physical machines that are active to reduce the overall energy consumption in the data center. Figure 2.4 illustrates this scenario. In what follows, we give ample details about the challenging issues behind managing trade-offs in cloud-based software.

2.2.2 Cloud-based software trade-offs challenges

This section discusses current challenges in resolving cloud-based software trade-offs:

Design of guest applications: Software applications that are migrated to a cloud platform are commonly referred to as “guest applications” [SP12]. It should be noted that migrating to the cloud does not improve e.g., the security or performance of an application if these quality attributes were not taken into account during the design of the application. To maximize the satisfaction on the quality attributes of an application, its whole structure as well as its dependencies with the environment and other systems must be explored. A holistic optimization approach that considers trade-offs at the early stages of software development and considers aspects such as energy saving and cost beyond security and performance has to be designed and set up for a software that operates in the cloud environment. The literature proposes a number of trade-off analysis methodologies to handle trade-offs at the early stages of software development. The Architecture Trade-off Analysis Method (ATAM) [KKB+98, KBK+99] is one of the pioneering trade-off analysis methods that is based on risk assessment and which evaluates software quality attributes, their interactions and their impact on a complex software system. The usage of trade-off analysis methodologies such as ATAM has to be explored for the design of cloud-based software to maximize software quality attributes satisfaction.

Specific trade-offs for each application feature: Optimizing trade-offs of the different features of applications have to be taken into consideration to build cloud-based software. Nowadays, applications are composed of different features. The complexity of trade-off management arises from the fact that each feature might require optimization in different quality attributes. Nallur *et al.* [NBY09], have highlighted this challenge through the case study of a social networking website, myChaseBook, which provides various functionalities including mobile access to photos, music and friends. The authors have discussed the quality attributes required for each feature to achieve a better application usage, and showed that e.g., performance requirements are most crucial in the feature of tracking friends location while security requirements are most relevant for the feature of buying on-line music tracks. The identification and prioritization of software quality attributes that are relevant for each cloud-based software feature is a required step in the development of cloud-based software.

Multi-tenancy: Security, synchronization and coordination between tenants are major factors to consider to avoid security leakages that result from interference between tenants. Isolation mechanisms have to be set up with appropriate costs. Krebs *et al.* [KMK12], have proposed an extensive discussion on the architectural decisions that must be taken to achieve a cost effective multi-tenancy.

Service Level Agreements: SLAs reflect several trade-offs that are agreed upon beforehand. Customers may agree to pay additional costs to improve the performance of their running applications. Another illustrative example is the trade-off between isolation and energy consumption: for security reasons, customers aim at loading their applications in dedicated clusters while providers prefer aggregating workloads from several customers in the same physical cluster to reduce energy consumption [VTM09]. Cloud-based software managers have to resolve these trade-offs dynamically since the cloud infrastructure and the SLAs are continuously evolving. Cloud-based software managers have to handle the trade-offs of several customers, simultaneously and set up priority policies on each customer service quality attributes.

Cloud layered stack: Cloud parameters lead to serious complex state representation. In many real case studies, we observe an explosion of the number of elements to configure. For instance, cloud architectures manipulate virtual machines, and applications hosted on top of them. Additionally, each level could have several parameters that are relevant to control performance. This could lead to a complex data structure that is very difficult to reflect with only events, conditions, actions concepts.

Uncertainty: Uncertainty criteria refer to parameters which are unpredictable or subject to undefined changes [Nid96]. These changes result from several factors including changes in the environment or in the stakeholders requirements. For cloud-based software development, uncertainty criteria are related to the classification of relevant quality attributes for a cloud-based software, the hosting environment, system users, SLAs, third parties, etc. To improve software quality attributes, it is strongly required to define models which identify uncertainty parameters in cloud-based software and are able to analyze their impact on the software behaviour. A research initiative that takes uncertainty parameters into consideration has been presented by Chaisiri *et al.* [CLN12]. For their model, the authors have proposed an approach for optimizing cloud resource provisioning under cloud prices and demand uncertainty.

Elasticity: Elasticity mechanisms are based on cloud resource adaptations (growing and shrinking operations) according to fluctuations in the workload. Elasticity brings new challenges to the monitoring and analysis mechanisms to resolve software quality trade-offs in a reasonable time frame and to provide on-demand resources scaling in/out in a seamless manner to the cloud customers. To ensure elastic services, adaptation time should not impact the quality of service of running services.

Service location: Increasingly, a variety of applications deployed in the cloud require extensive processing and data analysis capabilities. To improve data processing services, data stores and processors have to be placed in specific clusters, or geographically close clusters, to reduce the performance overhead that results from data queries and network overhead. Typically, for applications that maintain resources regulated by security policies and which are built based on XACML architecture, the placement of (1) Policy Enforcement Points (a.k.a PEPs, i.e. the components intercepting XACML requests and encoding them in a XACML format), of (2) the Policy Decision Point (a.k.a PDP, i.e. the decision engine that formulates the access decision (i.e. permitting or denying the resource) by evaluating an access request against applicable policies) and of (3) the policy repository has to be efficient to reduce the performance overhead of message exchange among these XACML components. This can be achieved by placing the policy repository and the PDP in the same geographic location to speed up the processing of the decision engine that has to fetch the policy engine for each request to access a service regulated by a security policy.

Optimization frameworks: Current cloud managers operate based on software load balancers which focus on one or two optimization axes. There is however a lack of commercial solutions that handle several optimization axes simultaneously. Optimization frameworks have to be able to aggregate, at runtime, information from the environment, and to provide optimized cloud software deployment configurations that effectively maximize cloud software quality attributes. Such frameworks must also be efficiently implemented to compute optimized configuration states in a reasonable time, in order to cope with runtime optimization constraints.

Cloud bursting: Cloud bursting [NPD+10] refers to exporting part of private cloud resources and workloads to a public cloud. Cloud bursting aims at reducing inherent costs in a setting based on private cloud model by adopting an hybrid cloud solution. Cloud bursting enables to provide an improved scalability of deployed cloud-based software. Rackspace[1], is a cloud-based company that offers cloud bursting services to maximize the scalability of its customers' websites when they face an increase in their traffic. To automatically move cloud customers workloads to suitable providers that are able to maximize software quality attributes, there is a need to implement efficient cloud brokering services [TMMVL12]. These brokers will operate to guide the cloud customer towards the best cloud provider which better fits his requirements.

Platform implementations: Cloud platforms have different features that impact the quality attributes of guest applications. Sodhi and Prabhakar [SP12] have provided a taxonomy of these features based on the domain of their impact. We note from this taxonomy that a cloud platform that has the ability to take a snapshot of virtual machines status will be able to restore data in case of an incident, and thus has an impact on applications reliability. Similarly, the heterogeneity of virtual machines in terms of data transfer time and in terms of service initialization time may lead to different impacts on service performance. In consequence, the heterogeneity of cloud platforms requires to develop usable interfaces that guide cloud customers to select most suitable cloud platforms that better fit the needs of their deployed software.

Containers development: Software containers [SH06] provide an isolated environment for software components. With the emergence of cloud-based software, containerization approaches have been proposed to host and execute cloud customers services in separate containers and to manage their dependencies. Linux containers [Hel09] have emerged recently as lightweight and fast virtual machines with supporting open sources (i.e, Docker [2]). Linux containers provide an isolation layer which allows only an application to interact with other applications through specified messaging services. For a better support of dockers, management tools have to be designed and implemented to assist developers in assessing the performance of containers, selecting the most suitable containers according to the requirements of their applications, composing several containers to build complex applications and optimizing the deployment of their applications inside containers. Cloud-based software managers have to be empowered with security isolation mechanisms to avoid security leaks across applications sharing same dockers.

[1] http://www.rackspace.com/cloud/cloud_bursting/

[2] https://www.docker.com/

Cloud load balancers development: To improve service fault tolerance capabilities, PaaS platforms have to provide load balancing mechanisms to monitor traffic, distribute workloads across geographically distributed clusters and to control in a fine-grained manner auto-scaling operations. To cope with a changing workload, a considerable effort has to devoted to develop customizable configuration interfaces that enable to set up traffic forwarding rules based on clusters localization and server types, to set up thresholds that trigger resources provisioning/de-provisioning operations and to schedule monitoring tasks. With the complexity of applications, it is not always possible to expect all possible states that are necessary to describe the behaviour of such load balancers. This motivates the need to design feedback-based learning mechanisms that are able to update the load balancing rules based on execution states collected during effective runtime execution. Load balancers have also to perform requests forwarding to the most appropriate physical clusters within acceptable time frame to cope with elasticity requirements in terms of response time.

PaaS & standardization: In the last few years, several PaaS tools have been proposed from several PaaS market players. Some of these tools are based on standard programming languages however they still suffer from the lack of standard APIs in terms of interfaces and application connectors exposing customers to vendor lock-in risks. With the lack of interoperability between platforms, customers workload migration from one cloud provider to a new cloud provider becomes a tedious task since the customer has to adapt its migrated applications to the proprietary components of the new provider's platform each time he decides to migrate his workloads. To ensure high interoperability and portability between PaaS, standardization initiatives have to be developed to provide standard application dependencies mechanisms and common interfaces to ease the transfer of applications between heterogeneous PaaS platforms.

PaaS & security: PaaS hosting security issues arise from both guest applications and the hosting PaaS platforms as discussed in [HRFMF13]. The security of the platform is related to the runtime environment including the security issues inherited from databases, connectors, network and message exchange between third-party services. Besides, PaaS hosting inherits the security issues related to guest applications that have to manipulate outsourced and distributed data in PaaS platforms. For wider adoption of PaaS platforms, research effort in securing PaaS has to encompass these two dimensions: Ensuring security in all the phases of software development of guest applications and ensuring the security of hosting platforms. Besides, PaaS platforms have to be empowered with policy enforcement and compliance checking mechanisms with regards to standards and regulations to reduce organizational IT risks.

2.3 Requirements to build an optimization layer for cloud-based applications

In this thesis, we aim at empowering cloud-based software with capabilities to adjust itself dynamically to fit several conflicting, environmental constraints (e.g., the current workload, energy consumption). This section discusses the challenges to achieve this goal.

2.3.1 Foreseen challenges

The challenge of optimizing cloud-based software breaks down into several targets analyzed below:

- *Abstraction and problem modeling:* To reason about trade-offs, an abstraction layer is needed to have an overview of the different cloud node elements and their interactions. An abstraction layer enables to reason about a specific problem and to hide the low level details related to the complexity of cloud-based software deployment in a real cloud environment. In this thesis, we will answer the following questions:
  - What are the specific features of these abstraction layers?
  - How they are able to capture the complexity of cloud heterogeneous resources?
  - How they are able to take decisions upon evolution in their surrounding context?
- *Need for an optimization layer:* A PaaS instance leverages operating systems, application servers and databases to allow cloud customers to deploy their applications. Cloud managers are platform tools that operate on top of PaaS instances to deploy applications and to govern resource provisioning. Most of the current cloud elasticity managers embedded in PaaS are limited to a specific cloud usage and thus focus mainly on performance optimization. Companies have thus to explore other objectives such as security, cost and fault tolerance. We will explore this challenge by answering the following question:
  - How to design and build a cloud optimization layer on the top of PaaS to optimize cloud-based software deployment from both cloud customers and providers perspectives?
- *Resolution methodology:* To resolve cloud-based software trade-offs, we will answer the following research questions:
  - How to achieve satisfactory trade-offs between customers and providers conflicting objectives to reduce applications cost and to increase applications security and performance?
  - What are the candidate resolution methods to resolve these trade-offs?

  - Given a set of cloud configurations that exhibit acceptable trade-offs between conflicting objectives, how to explore and select efficiently best candidate cloud configurations?
  - What are the metrics that can be used to compare cloud configurations candidate solutions according to the requirements stated in the SLAs?

- *Optimization improvement:* Continuous evolution of cloud stakeholders is a motivating factor to set up a continuous optimization approach. An optimization method has to be efficient to cope with the requirements of cloud elasticity. This requirement brings up the following research question:

  - How to resolve cloud-based software deployment trade-offs within an appropriate execution time frame to cope with runtime optimization requirements?

- *Guest applications design:* In [HHK+10], the authors have discussed the trade-off decisions that are introduced by cloud computing paradigm at the level of software systems with a particular focus on specific software quality attributes such as security, performance and energy consumption. They have pointed out that cloud computing intrinsic parameters such as elasticity and cost imply to revisit software methods with the goal of enhancing software architectures design. As pointed out in [SP12], cloud computing platforms can improve the quality attributes of hosted applications. For example the availability of a given application can be improved if the hosting platform provides redundancy capabilities that improve application fault tolerance. However the design of hosted applications has to be improved to maximise software quality attributes. This requires to recur to software engineering methods. Some of these methods are not novel but they are more critical in the context of cloud computing paradigm [YA11]. This raises the following research question:

  - How can we improve these quality attributes in all the phases of software life cycle design?

2.3.2 Addressing the challenges

Throughout this section, we will detail some key concepts to address the challenges that we have raised in the previous section with the objective to meet the requirements of improving quality attributes of cloud-based applications. In what follows, we adopt a three-fold approach: (1) We take advantage of models@run.time approach to provide a cloud abstraction layer for the discretization of cloud states to reason about cloud-based software deployment multi-objective optimization problem. (2) Leverage search based methods to resolve these trade-offs. (3) Leverage software engineering methods to improve the quality attributes at the level of guest applications.

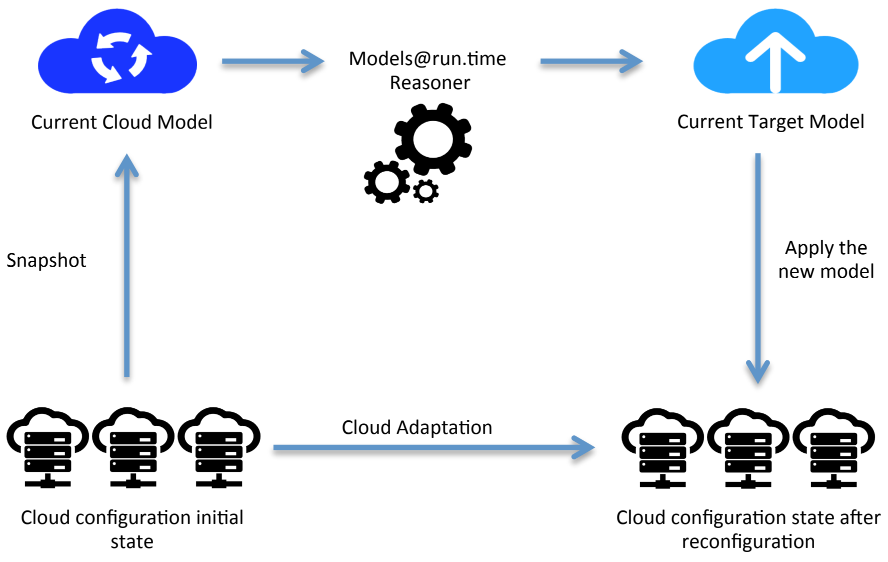


Figure 2.5: Cloud adaptation

2.3.2.1 Dynamic adaptive systems and models@run.time

Dynamic Adaptive Systems (DAS) [BCZ05] are characterized by their reconfiguration capabilities at runtime, to adjust themselves to unpredictable conditions, in their operating environment. A cloud infrastructure can be considered as a dynamic adaptive system (DAS) since it maintains variable parameters and architectural elements to manage an elastic hosting. It continuously adapts itself according to the request load or to the hosting price. Studying dynamic evolution of cloud implies to define mechanisms to extract cloud states like a snapshot in order to evaluate this snapshot against requirements. Besides, cloud users and software designers need an abstraction to analyse and evaluate these snapshots. In this thesis, we will leverage models@run.time [BBF09b] approaches which provide tools to capture a system snapshot while it is in a running state. This snapshot can be used for a deep system analysis with the objective to perform cloud multi-objective optimization. A cloud state represented as a model, is considered as a common representation to reason about a cloud configuration. Models@run.time approaches enable to synchronize on demand a new model with a real cloud infrastructure by using a causal link [MBJ$^+$09], thus the abstract model can directly impact the running system. The models@run.time approach is illustrated in Figure 2.5.

2.3.2.2 Multi-objective optimization resolution methods

Search Based Software Engineering (SBSE) research [Har07] aims at offering a certain level of automation to software engineering through several search methods. Their applicability in the context of cloud computing has been advocated in recent research initiatives [HLS+12]. Genetic algorithms [GC00] are metaheuristic search techniques that maintain a pool of solutions, called a population, which evolves from one generation to another. The selection of best individuals that serves for producing a new generation is done based on a fitness function that determines which possible solutions are selected into the next generation of solutions. In each generation, the genetic algorithm constructs a new population using the genetic operators. The population is normally randomly initialized. As the search evolves, the population includes fitter and fitter solutions, and eventually it converges. Search-based evolutionary approaches are suitable candidate for multi-objective optimization problems resolution. In this thesis, we leverage search-based algorithms and particularly multi-objective evolutionary algorithms (MOEAs) to resolve trade-offs inherent in the context of cloud-based software deployment.

2.3.2.3 Guest applications design

In [YA11], the authors have discussed the different types of synergies between cloud computing and software service development by discussing how each paradigm is able to impact the other. While applications do not focus on service delivery, cloud computing enables to achieve effective service delivery through virtualization and enables to improve applications quality of services through on-demand resource allocation. Nowadays, cloud computing still suffers from some issues related to standardization, these issues can be addressed by software engineering methods, which can develop interoperable interfaces between different services. As highlighted in [HHK+10], some non functional requirements related to the development of distributed applications have to be reconsidered in the context of cloud computing paradigm. Software quality attributes related to the design of guest applications have been discussed in [KMK12]. In this thesis, we focus on security trade-offs for the design of guest applications. Our contribution demonstrates how to achieve trade-offs between security and other quality attributes such as performance and cost. Next section motivates our key motivating factors behind our focus on security and presents our proposed contributions at the level of the guest applications that have been used in this thesis.

2.4 Security trade-offs

Security is one of the major concerns that is required to build a trustworthy cloud computing environment. Zhou *et al.* [ZZX+10] have surveyed security issues in cloud computing and have pointed out that the prosperity of cloud computing is closely tied to the resolution of security challenges. These challenges arise from resource sharing, virtualization, security threats and multi-tenancy. According to the ranking that has been conducted by IDC IT group [New11] in August 2009, security is one of the major concerns as shown in Figure 2.6. Among security aspects, this thesis explores access control security.

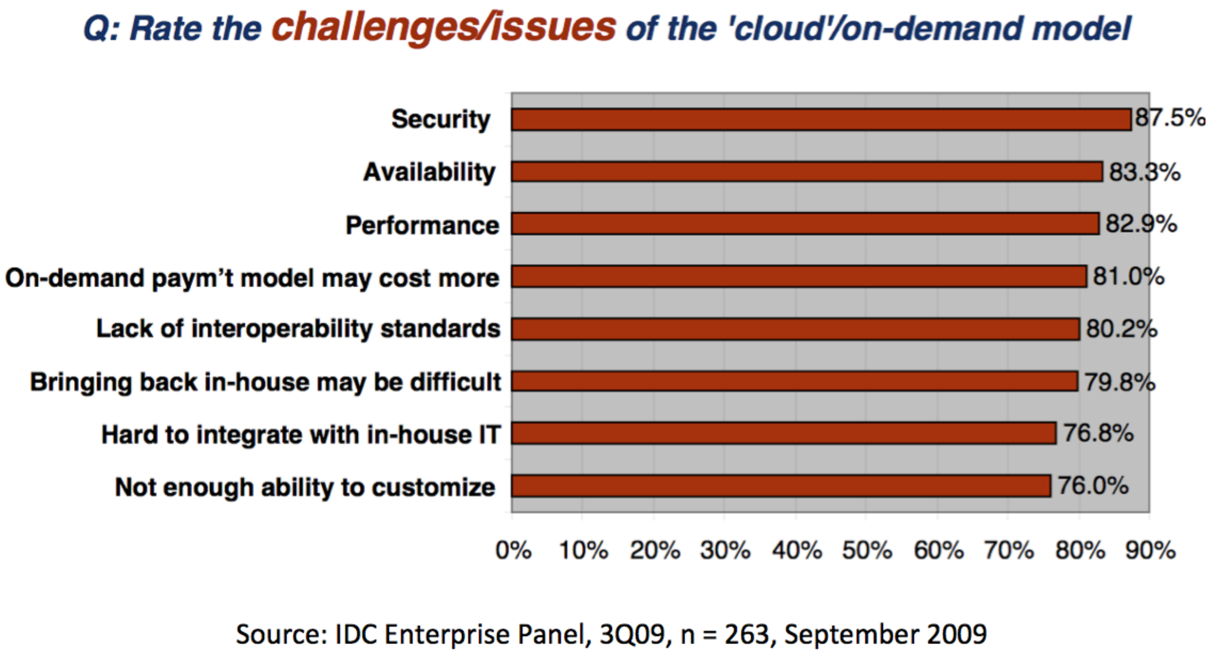


Figure 2.6: Cloud concerns ranking

2.4.1 Access control security in cloud computing

Access control [SS94] ensures that only eligible users are able to access protected resources in a given system. The versatility of access control scenarios has lead to the evolution of access control and to the appearance of usage control concepts [LMM10]. UCON [PS] is a reference model for usage control, which illustrates two important concepts: the *continuity of decision* and *mutability of attributes*. The former concept is related to the dynamic nature of the decision, which can be revoked in the case of a policy violation even when the access has already been granted. The latter describes the changes in the attributes of the different entities in the policy. Usage control encompasses authorizations to handle actions that have to be carried out before the access (i.e., the user has to authenticate before accessing a web site), or during the access (i.e., the user has to keep an open window while he is accessing a web site), or after access (i.e., the user has to submit a form after his access). In the domain of security policies, those actions are usually referred to as obligations [HBP05a].

There are several key parameters that present challenging issues behind setting an access control/usage architecture in the cloud. In [AYKM14], A. Younis *et al.*, have presented a comprehensive analysis to capture the new requirements of access control for cloud computing paradigm:

- *Interoperability*: A cloud infrastructure spans over several administrative domains. Each domain includes several authorization models and leverages different security policies languages. There is a need to develop standard interfaces to ensure interoperability between heterogeneous administrative domains and to ease the inter-communication between different cloud providers [Jos04].

- *Flexibility*: Service heterogeneity and the different requirements of stakeholders [TJA10b] imply to use access control frameworks that are able to adapt easily to rapid changes in the cloud environment, changes in the services and in stakeholders requirements. Access control policies have to capture complex and fine-grained authorizations scenarios where permissions can be configured according to user and resource attributes.

- *Multi-tenancy*: Several recent research contributions have pointed out that multi-tenancy adds new challenges to existing access control models, policies and implementation mechanisms [PYK+10, LSGM10, TLS13, YLL13]. New policies have thus to be defined to express requirements related to isolation, sharing mechanisms, rate-limiting between different tenants.

- *Authorization management*: The management of authorizations has to allow access control administrators to define features such as delegation and context-aware access control. It has also to support the configuration and the enforcement of delegation rules [CK08] that enable access control administrators to substitute user rights to other users to handle absence situations. The support of contextual permissions [JCH+05] is very crucial in access control management frameworks for cloud computing [ZWH+13]. Context-aware access control enables to take access control decisions based on some conditions such as user localisation or the request evaluation time.

2.4.2 Security versus other quality attributes

There are several definitions that refer to non functional requirements [CdPL09]. Non functional requirements or quality attributes in software architecture paradigms [BKLW95] are concerns that have to be taken to improve the quality of a software system. As mentioned previously, in this thesis, we give specific focus to the security quality attribute. "Security is about Trade-offs, not Absolutes" [San03]. Ravi Sandhu has discussed the challenges behind good enough security [San03]. Several lines of research have explored the problem of designing software systems aiming to achieve a trade-off between security and other quality attributes. In [EY07], Elahi *et al.*, have used a requirements model that extends i* notation and a goal-oriented framework to analyse security trade-offs. A trade-off analysis method was proposed based on a qualitative evaluation of the impact of security mechanisms on the satisfaction of the goals related to the actors in the system. In [FBEK11], Raza *et al.*, have applied ATAM method [KKB+98] on a web application to identify security risks at the early stages of software development. Their approach aims at reducing costs that can result from security incidents at later stages of software development process. In [Yee04], the authors have discussed the tension between usability and security. They have provided guidelines to design secure interaction interfaces by deriving security requirements from user tasks. In [WRWR05], the authors have presented an approach that exhibits trade-offs between flexibility and security in adaptive process management systems. They have particularly focused on authorizations management and extended RBAC [SCFY96a] access control model [FCK95] to handle access control rights in a flexible manner.

2.4.3 XACML and access control in the cloud

XACML[3] policy specification language defines access control policies in an XML format and defines a standardized way to exchange requests/responses. It relies on an abstract architecture consisting in abstract components interacting with each other to handle a decision making process. XACML relies on a standardized encoding since it enables to encode a policy independently from the underlying platform, to make it thus interoperable with heterogeneous platforms. In XACML architecture, the policy is externalized from the application code and from the decision engine. This eases the maintenance of software systems since the update of the policy, usually a frequent task, can be done without changing the system's implementation. XACML architecture and language have several features that make them suitable in cloud access control scenarios:

- Support of fine-grained access control scenarios: XACML supports Attribute Based Access Control (ABAC) [YT05]. ABAC extends RBAC [SCFY96b] to allow permissions to be provided based on subjects, resources, and environment attributes instead of only the role attribute. This permits to handle fine-grained access control scenarios. One illustrating example is to configure permissions based on a subject's age to restrict a minor's access to certain adult websites. In cloud access control scenarios, users, providers resources and business relationships are in a continuous change, thus the definition of permissions can not be just configured based on predefined roles like in RBAC models. XACML is based on the ABAC model and thus it is able to capture the versatility of cloud access control scenarios [CSK12]. In [DWD12], Dinh *et al.*, have extended XACML to support fine grained access control scenarios. These scenarios are characterized by complex data handling features such as aggregation of data that ranges in a certain interval or the collection of approximated data.

- Multi-tenancy support: In a multi-tenant context, access control policies have to be able to isolate in a fine-grained manner the resources related to the different tenants. It has to support in a flexible way the addition or the removal of tenants and to map each tenant request to the right resource. The standard attributes in XACML specification can be used to refer to the different tenants[4]. In [TLS13], Tang *et al.*, have modeled and specified a multi-tenant collaboration using XACML policies.

[3] https://www.oasis-open.org/committees/xacml/

[4] http://securesoftwaredev.com/2012/08/20/xacml-in-the-cloud/

Table 2.1: XACML and access control in cloud paradigm

Contributions	**Feature**	**Summary**
[DXX09, LMMM12]	Cloud services based on UCON model	An access control framework based on UCON model is proposed to protect accesses to cloud resources
[CF12]	Sticky policies	An access control model based on XACML is proposed. In this model, policies travel with the data to enable data to be verified when it moves between cloud providers.
[DWD12]	Data sharing in the cloud	A framework based on XACML is proposed to allow defining fine grained constraints on data sharing between Clouds.
[NMDdL12]	Exchange between several cloud providers	A model to regulate accesses in an IaaS environment is proposed. The model defines dynamic establishment of trust between different entities involved in the IaaS.
[LBB13]	Authorization as a Service	An authorization as a service model, based on XACML architecture, is proposed.

- Powerful administration features: In [Lan10], Lang has explained the motivation behind the need to build authorization management frameworks that are flexible to meet cloud requirements such as fine-grained policies, stakeholders interactions, evolving resources and subjects. In [DNdL+11], the authors have used XACML to build an authorization management framework. Their model supports the establishment of dynamic trust relations between different cloud stakeholders. In [TJ12], Takabi *et al.*, have proposed a framework, based on XACML architecture, to overcome the heterogeneity issues related to the management of distributed security policies, that regulate resources across several platforms.

  To conclude, XACML model is a powerful model to handle resources accesses in the cloud. Several recent approaches have used XACML in different access control scenarios related to authorizations in the cloud. We summarize some of these recent approaches in Table 2.1. For all aforementioned XACML features, we choose XACML-based systems as an architectural model for the guest applications that will be considered in this thesis.

2.4.4 Policy-based systems challenges

Policy-based systems are applications that interact with a security policy to regulate resource distribution. We have motivated earlier the advantages of building applications based on the XACML architectural model. We believe that policy-based system built upon XACML architecture enable to meet flexibility requirements of authorizations systems [LPL+03]. In this thesis, we take the example of policy-based systems as a sample of guest applications. We identified some limitations in the design of XACML policy-based systems. In what follows, we describe these limitations and we explain how they will be addressed in the thesis.

2.4.4.1 Improving obligations support in XACML

In [LMMM12], the authors have presented several usage control scenarios that are relevant in handling a cloud infrastructure. Some scenarios are related to virtual machine management in a cloud infrastructure. Access to virtual machines is adaptive based on user reputation or resources workload. For instance, access to virtual machines can be revoked based on the change of users reputation. Usage control scenarios can also be defined in the requirements mentioned in the SLAs. To manage in a fine-grained manner usage control requirements of XACML policy-based systems, a comprehensive support of obligations is required. However, the current version of XACML lacks:

- Well-defined syntax to express standard obligation elements (Standard obligation elements can ease the interoperability between heterogeneous distributed systems).

- A model to handle decision making w.r.t. obligation states and the history of obligation fulfilment/violation (Taking into consideration obligation states at the decision making time enables to handle usage control scenarios based on user behavior tracking).

In this thesis, we will answer the following research question:

- How to extend XACML language and architecture to improve obligations support and how obligations can be managed in the extended XACML architecture?

2.4.4.2 XACML & performance improvement

The number of cloud stakeholders, resources and the complexity of business processes, are major factors that lead to the increase of XACML policy size. The growth of policy size has an impact on access control request processing time and slows down the performance of the overall policy-based system. To enhance the performance of policy-based systems and to achieve a trade-off between security and performance at the level of these policy-based systems, we will answer the following research questions:

- How to refactor access control polices to improve the performance of policy-based systems?

- How to assess the impact of this refactoring?

2.4.4.3 Policy evolution & regression testing costs

Regression testing [RUCH01] aims at retesting the software system after any code changes. It aims at retesting the new introduced functionalities and at verifying that the new changes do not alter existing functionalities. For policy-based systems, when the security policy evolves, we need to perform regression testing at the code level to ensure the correctness of security mechanisms. Usually, the costs of regression testing account for a considerable percentage of the costs of the overall software development process. The challenge in regression testing is to select a minimal set of test cases that have to be rerun to test the new system behaviour and to maximise fault detection capabilities. This enables to reduce regression testing costs. In the context of security policy evolution, we will reply to the following research questions:

- What are the techniques for regression testing selection in the context of security policy evolution?
- Are these techniques effective to reduce regression testing costs?

2.5 Conclusion

This thesis delves into the matter of cloud-based software trade-offs by putting forward a synergy between cloud computing paradigm and software engineering to tackle the challenges of cloud-based software quality attributes. By proposing new approaches, algorithms and methods, our main objective is to improve cloud-based software quality attributes to enhance the perceived quality of the cloud as a whole. To achieve this goal, we adopt a two step approach: 1) In the first step, applications are considered as black-box components and we only focus on the constraints related to the software deployment. 2) In the second step, we focus on improving the quality attributes related to guest applications. In this chapter, we describe the context by presenting the challenges of improving software quality attributes. We also give an insight into the resolution methods that will be described in the remainder of the manuscript. In the next chapter, we will describe the key concepts and basic definitions that will be used throughout this thesis.

Chapter 3

Background

This chapter introduces the basic definitions that are used throughout the thesis. We organize the background section as follows. We first start by analyzing the current landscape in resolving cloud-based software quality attributes trade-offs. We further present the key concepts related to policy-based systems and multi-objective optimization research. Finally, we present background information about models@run.time and model-driven optimization.

Contents

3.1 Current research status on cloud-based software optimization

In this section, we survey existing approaches in optimizing cloud-based software trade-offs. We provide an overall overview of (1) recent academic research initiatives, (2) international projects which tackled these issues, and (3) emerging commercial products dedicated to manage cloud-based software deployment.

3.1.1 Academic research in optimizing cloud-based software

When moving their applications to the cloud, cloud customers take advantage of an elastic environment in which resources are provisioned/deprovisioned automatically to adapt to variable workload. Cloud elasticity leverages a set of actions that are responsible to move cloud configurations from one state to another. This leads to a wide set of potential candidate solutions, and consequently to a wide domain to explore in order to find at a given time the best cloud configuration. To maximize the satisfaction of cloud-based software quality attributes, a wide body of research has explored optimization methodologies and techniques as an initiative to improve cloud-based software acceptance. This effort has been put into the elicitation of requirements and the development of PaaS environments, including services to develop and deploy cloud-based software. The proposed approaches are distinguishable by the optimization axis that they consider. For instance some of the approaches aim at being cost effective, some other approaches are oriented towards performance improvement or security hardening, while others are oriented towards achieving an eco-friendly, green, cloud by dispatching software components in virtual machines in a way that reduces the number of active virtual machines. There are also approaches that aim at achieving a holistic optimization by considering simultaneously several optimization axes. A few optimization approaches aim at optimizing cloud-deployed software from the customer perspective to basically reduce its allocated resource costs. Some approaches focus on the cloud provider perspective by focusing on reducing the energy deployment costs. Finally, the literature describes approaches that aim at achieving a global optimization by considering both customers and providers perspectives. In what follows, we give an overview about these proposed optimization research contributions approaches and we classify them based on two main features: the *Optimization Criteria* and the *Resolution Techniques*. Table 3.1 summarizes these approaches.

Frey *et al.* [FFH13a], have identified most suitable configurations in terms of cost and response time by comparing cloud deployment options (CDOs) which represent runtime reconfiguration rules for resource scaling. Their approach aims at finding the most suitable cloud providers to migrate workloads and has been validated through an open source software CDOXplorer that uses Amazon EC2 and Microsoft Windows Azure. Mao *et al.* [MLH10], have proposed mechanisms to start-up/shutdown virtual machines based on the evaluation of performance and budget constraints. Most suitable virtual machines in terms of price are started-up to finish workloads with respect to their respective deadlines. The approach has been evaluated through simulation on a real scientific cloud application in Windows Azure platform.

Chaisiri *et al.* [CLN12], have defined an optimal cloud resource provisioning (OCRP) algorithm to reduce cloud customer costs under the uncertainties that derive from customer requirements in terms of resource and provider costs. Jung *et al.* [JHJ+10], have presented Mistral that can optimize power consumption, performance, and adaptation costs for distributed applications. The approach has been validated using up to eight machines that host servlet version of a RUBiS benchmark emulating an eBay-like auction. Raj *et al.* [RNSE09], have provided an approach validated on an Hyper-V virtualization platform to overcome the cache sharing in multicore systems that can lead to interference and performance impacts in virtual machines. Yusoh *et al.* [YT12], have used evolutionary algorithms to optimize performance and resources usage of composite SaaS (SaaS composed of software components that are geographically repartitioned in separate physical servers). They have provided an optimal initial placement of components into physical servers by taking into considerations components characteristics and inter-dependencies rather than virtual/physical machines characteristics. The authors have conducted simulations that are based on the characteristics of the nodes attributes[1].

Wada *et al.* [WSYO11], have proposed and evaluated E^3-R algorithm that aims at obtaining cloud deployment configurations exhibiting trade-offs between QoS objectives. Their approach has been validated based on a model of a loan processing application.

Guzek *et al.* [GPDB14], have used multi-objective evolutionary algorithms to find an optimal schedule to allocate tasks to the resources available in distributed systems. Their approach aims at reducing execution time and energy consumption. They have assessed the impact of using new configurations of genetic operators on the problem resolution. Several variants of genetic algorithms have been used and their performance has been compared.

Sharma *et al.* [SSSS10, SSSS11b, SSSS11a], have leveraged migration and replication to reduce provisioning resources and to reduce time overhead that results from switching to new cloud configurations. The validation platform, they have proposed, is based on laboratory-based private Xen cloud, public Amazon EC2 and OpenNebula cloud. Pandey *et al.* [PWGB10], have

[1]http://www-07.ibm.com/storage/au/

proposed a model to map resources to tasks and have used Particle Swarm Optimization (PSO) as a resolution technique to minimize execution costs. The authors have used JSwarm[2] package to implement algorithms for Particle Swarm Optimization, Amazon CloudFront[3] for the simulation of the cost of unit data transfer between resources and Amazon EC2's pricing policy for virtual machine instances management[4].

kllapi *et al.* [KSTI11], have explored the optimization of data-flows scheduling in terms of cost and time using several optimization algorithms on a set of families of data flows. They have used four families of data flows (Montage, Ligo, Cybershake and Lattice) and uniform containers for validation. Li *et al.* [LCWL09], have presented an approach for applications optimization in a cloud infrastructure by using optimization techniques based on network flow model (NFM) and a performance model built upon a layered queuing network (LQN) that takes into account resource contention effects.

3.1.2 International projects

Several European research projects in the context of the seventh framework programme (FP7) have been recently exploring trade-offs of cloud-based software[5]. *OPTIMIS*[6] is an FP7-ICT that aims at achieving an optimal deployment of cloud services for cloud providers and cloud customers. *OPTIMIS* is based on scenarios that describe how the *OPTIMIS* framework can be used for software migration, all aspects of services development life cycle and automatic scaling of cloud customers infrastructure. *NESSoS*[7] aims at considering security concerns at the very early stages of software development life cycle.

ASCETIC[8] aims at developing a model to achieve an energy-aware cloud throughout all the phases of software development. *CACTOS*[9] project aims as well at achieving an energy-aware cloud by taking into consideration the heterogeneity of hardware resources. *BigFoot*[10] aims at designing PaaS solutions to optimize large volume of data incoming from several applications and contributes to the Apache Hadoop project. *Broker@Cloud*[11] aims at improving continuously software quality attributes of cloud-based software and at tracking the satisfaction of the service requirements that are stated in the Service Level Agreements (SLAs). *Coherent PaaS*[12] leverages complex event processing techniques to develop optimized tasks, data and workloads. Beyond

[2]http://jswarm-pso.sourceforge.net/
[3]http://aws.amazon.com/cloudfront
[4]http://aws.amazon.com/ec2/
[5]http://www.sucreproject.eu/node/40
[6]http://www.optimis-project.eu/project
[7]http://www.nessos-project.eu/
[8]http://ascetic-project.eu/
[9]http://www.cactosfp7.eu/
[10]http://www.bigfootproject.eu/
[11]http://www.broker-cloud.eu/
[12]http://coherentpaas.eu/

Table 3.1: State of the art in optimizing cloud-based software

			References
Optimization Criteria	Security	Isolation	[RNSE09]
		SLA Violation	[FFH13a]
	Cost	VM Instances	[CLN12]
		Provided Services & Resources	[LCWL09], [KSTI11] [SSSS10], [PWGB10] [SSSS11b], [SSSS11a] [FFH13a], [YT12] [WSYO11],
		Uncertainty of price and demands	[CLN12]
		Transient adaptation costs	[JHJ+10], [SSSS10] [SSSS11b], [SSSS11a]
		Search Operations	[JHJ+10]
	Performance	CPU Usage	[MLH10], [WSYO11]
		Response Time	[LCWL09], [KSTI11] [SSSS10], [SSSS11b] [SSSS11a], [FFH13a] [YT12], [JHJ+10] [RNSE09], [WSYO11] [MLH10]
		Storage	[MLH10]
		Network Usage	[MLH10]
		Power Costs	[JHJ+10]
Optimization Technique	Genetic Algorithms	Multi-Objective Evolutionary Algorithms (MOEAs)	[YT12] [WSYO11] [GPDB14]
		CDOXplorer	[FFH13a]
	Integer Programming	[LCWL09], [CLN12] [SSSS10], [SSSS11b] [SSSS11a]	
	Greedy, Probabilistic and Exhaustive Optimization	[KSTI11]	
	Heuristic Bin-packing Algorithm	[JHJ+10]	
	Cache hierarchy aware core assignment and page coloring	[RNSE09]	
	Particle Swarm Optimization (PSO)	[PWGB10]	

FP7-ICT projects, the *TOOM* project[13] aims at enhancing quality attributes such as testing robustness and reliability of cloud computing services. TOOM aims at empowering existing testing techniques to handle aspects such as scalability, to cope with the huge amount of data and cloud nodes continuous changes. *TOOM* leverages large-scale stress loads to validate the overall resilience and reliability of cloud services.

3.1.3 Industrial emerging platforms

To design cloud-based applications, cloud oriented frameworks such as Athena Framework[14] enable to design a cloud-based dedicated application with features such as multi-tenancy. In the last years, several cloud managers tools have been proposed as an initiative to help companies optimize and manage their resources in a cost-effective manner, offering reasoning capabilities that enable to maximize software quality attributes satisfaction. Cloud Cruiser[15] has provided recently some existing cloud managers tools. PuppetLabs[16], Chef[17], Salt[18], Cloudify[19], Juju[20] are pioneering products that share common features such as instance management, container provisioning, dynamic services discovery, rapid deployment of new applications and resources scale in/out to handle on-demand provisioning variations. These tools provide on-demand provisioning capabilities through monitoring services that provide data analysis and enable administrators to set thresholds parameters that are used to launch automatically scaling in/out operations.

3.1.4 Summary

In the last few years, a plethora of research has addressed cloud engineering optimization problems. In this section, we have surveyed existing approaches that have proposed resolution techniques to improve trade-offs at the level of cloud-based software. Our approach is distinguishable from existing work by its reasoning layer which is based on a models@run.time paradigm. This will be investigated in chapter 4 and chapter 5. Models@run.time provide automated reasoning capabilities which enable to provide abstraction from implementation details and to reason about the system at a high level of abstraction. These reasoning capabilities can be performed at the level of a snapshot of the running system. The optimization of the snapshot configuration can be achieved automatically without prior knowledge of optimization techniques and without the presence of optimization experts. Models@run.time enable to enact a feedback loop since the model is adapted with the actions of the real system and new optimizations can be performed at the level of the model to enhance the behavior of the real system. Recall that the problems

[13]http://wwwen.uni.lu/snt/research/research_projects2/toom
[14]http://athenasource.org/java/
[15]http://www.cloudcruiser.com/media/16-of-the-most-useful-cloud-management-tools/
[16]http://puppetlabs.com/
[17]http://docs.opscode.com/
[18]http://salt-cloud.readthedocs.org/en/latest/
[19]http://getcloudify.org/
[20]https://juju.ubuntu.com/

Figure 3.1: XACML commercial and open source products

raised by models synchronization with the real system and the validation of the chosen model, at a given time, are not in the scope of this thesis. Our approach aims also at optimizing cloud-based trade-offs by investigating the design of guest applications. This will be explored in chapters 6, 7 and 8.

3.2 Policy-based systems

Policy-based systems are applications that maintain resources whose access is regulated by security policies. In this thesis, we advocate the usage of policy-based systems built upon XACML model to design guest applications. Our motivation behind this decision is strongly related to the flexibility of the architectural design provided by XACML policy-based systems. This flexibility is advocated for the management of authorizations in cloud-based software as highlighted in section 2.4.3. In this section, we give ample details about XACML language and architecture.

3.2.1 Access control as a security concept

Unauthorized access to sensitive data is one of the main challenging issues in IT security. Access control is a security mechanism that regulates the actions that users can/cannot perform on system resources. Access control policies are used to manage authorizations in the objective to protect sensitive data. Access control research spans mainly over access control policies models [SdCdV01], access control policies enforcement mechanisms [SCZ03, LBW] and access control policies languages definition [DDLS01a]. Access control policies are defined based on several access control models such as Role-Based Access Control (RBAC) [FSG+01], Mandatory Access Control (MAC) [BLP76], Discretionary Access Control (DAC) [Lam71], and Organization-Based Access Control (OrBAC) [KBM+03]. Access control policies are specified in various policy specification languages such as the eXtensible Access Control Markup Language (XACML) and Enterprise Privacy Authorization Language (EPAL) [AHK+03]. An access control policy is composed of authorization rules that regulate the access to data and services. At the decision making time, a request to access a service/resource is evaluated against the rules in the policy. A response is then sent to the user, that authorizes/prohibits her/him to/from access/accessing the requested resource.

3.2.2 Security policies

In the last few years, XACML (eXtensible Access Control Markup Language) has gained momentum as a standard to develop security solutions. Several commercial and open source solutions, as highlighted in Figure 3.1, have been developed to help build access control systems.

3.2.2.1 XACML model

XACML proposes a conceptual model of an access control architecture and defines the interactions between the components in this conceptual model. It also defines an access control policy language and a protocol for access control requests and responses. XACML policy-based systems rely on the separation of concerns, by implementing independently a software system and its associated security policy. Such separation eases policy management and increases the degree of policy interoperability with heterogeneous platforms. It also limits potential risks arising from incorrect policy implementation or maintenance when the policy is hard-coded inside the business logic. In the context of policy-based systems, a Policy Enforcement Point (PEP) is located inside an application's code (i.e., business logic of the system). Business logic describes functional algorithms to govern information exchange between access control decision logic and a user interface (i.e., presentation). To determine whether a user can access which resources, a request is formulated from a PEP located in an application code. Given a request, a Policy Decision Point (PDP) evaluates the request against an access control policy and returns its access decision (i.e., permit or deny) to the PEP.

XACML architecture is based on the following components:

- Policy Administration Point (PAP): It is the policy repository which sends policies to the Policy Decision Point (PDP).

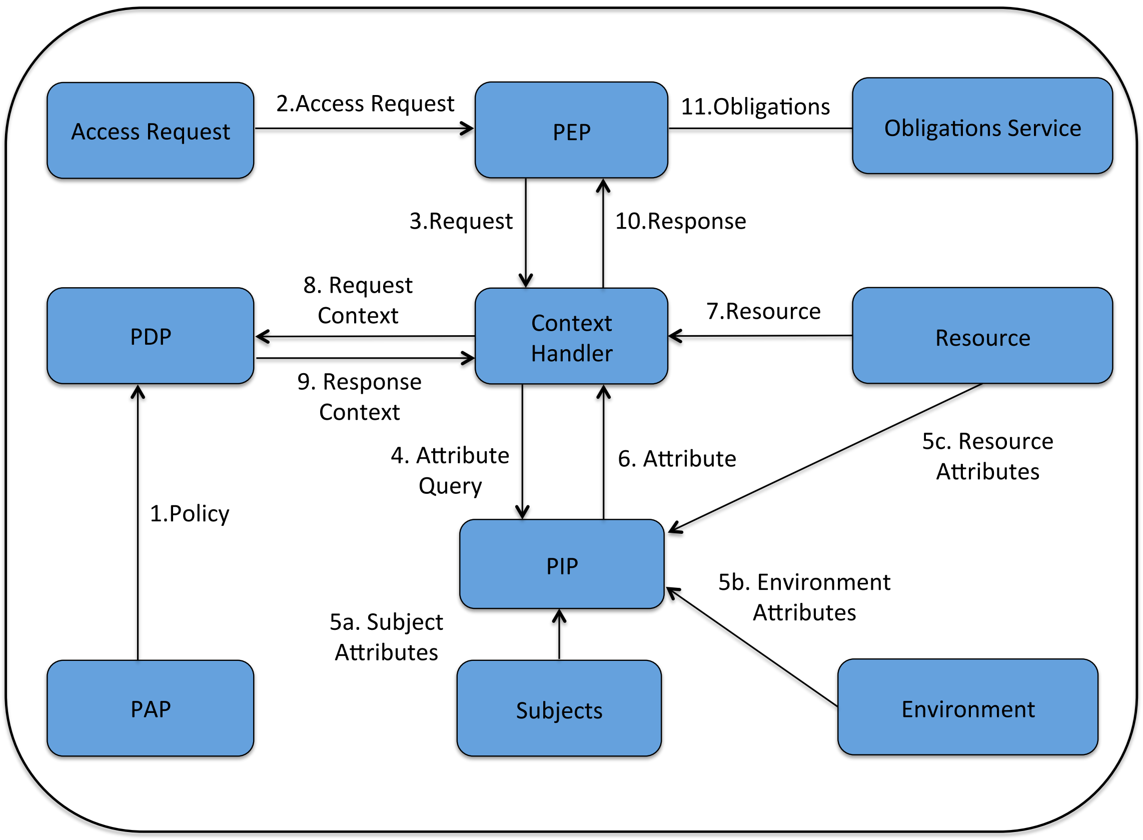


Figure 3.2: XACML data flow

- Policy Decision Point (PDP): The PDP is responsible for making decisions based on the collected information from other actors.
- Policy Enforcement Point (PEP): It receives an access request whenever a service which is regulated by a security policy is called, sends a request in an XACML format to the PDP and enforces the decision sent by the PDP.
- Policy Information Point (PIP): The PIP retrieves necessary attributes from external resources (i.e, LDAP).
- Context Handler: It transforms requests/responses in an XACML format.

Figure 3.2 presents the interactions between the different components to handle an access control request: 1) Policies are written and managed in the PAP. The PDP will fetch the policies in the PAP in each access control request evaluation. 2) The PEP is triggered whenever a service which is regulated by a security policy is called. 3) The PEP sends the request to a context handler that transforms the native format of the request into an XACML format. 4) The context handler sends a request to the PIP to collect attributes. 5) The PIP obtains the requested attributes from subject, resource, environment and 6) The PIP sends the attributes to the context handler. 7) The context handler may include the resource in the context. 8) The context handler sends an XACML request to the PDP for evaluation. 9) The request is evaluated against the policies in the PAP. 10) The context handler returns the response to the PEP in a format that can be interpreted by the PEP. 11) If some obligations are returned with the decision, the PEP has to discharge those obligations.

3.2.2.2 XACML policies

XACML policy has a hierarchical structure as shown in Figure 3.3. On the top level of the this structure, a *policy set* can contain one (or more) *policy set*(s) or *policy* elements. A *policy set* (a *policy*) consists of a target, a set of rules and a rule combining algorithm. The target specifies the subjects, resources, actions and environments on which a policy can be applied. If a request satisfies the target of the *policy set* (*policy*), then the set of rules of the *policy set* (*policy*) is checked, else the *policy set* (*policy*) is skipped. A rule is composed by: a target, which specifies the constraints of the request that are applicable to the rule; a condition, which is a boolean function evaluated when the request is applicable to the rule. If the condition is evaluated to true, the result of the rule evaluation is the rule effect (*Permit* or *Deny*), otherwise a *NotApplicable* result is given. If an error occurs during the application of a request to the policy, *Indeterminate* is returned. The rule combining algorithm enables to resolve conflicts when there is more than one rule that can be applicable to a given request. For instance, if the *permit-overrides* algorithm is used:

- If there is one single rule that is evaluated to permit, the permit decision takes precedence regardless of the result of the evaluation of other rules.
- If one rule is evaluated to Deny and all other rules are evaluated to NotApplicable, the final decision is Deny.
- If there is an error in the evaluation of a rule with Permit effect and the other policy rules with Permit effect are not applicable, the Indeterminate result is given.

The access decision is given by considering all attribute and element values describing the subject, resource, action and environment of an access request and comparing them with the attribute and element values of the policy. An XACML request is composed of four elements: a subject, a resource, an action and an environment. The values and types of these four elements should be among the values and types defined by the policy rules or targets. Listing 3.2 illustrates an XACML request in which a student requests to borrow a book. Listing 3.1 illustrates an XACML policy with one rule. The rule (lines 26-58) states that a student can borrow and return books from the library.

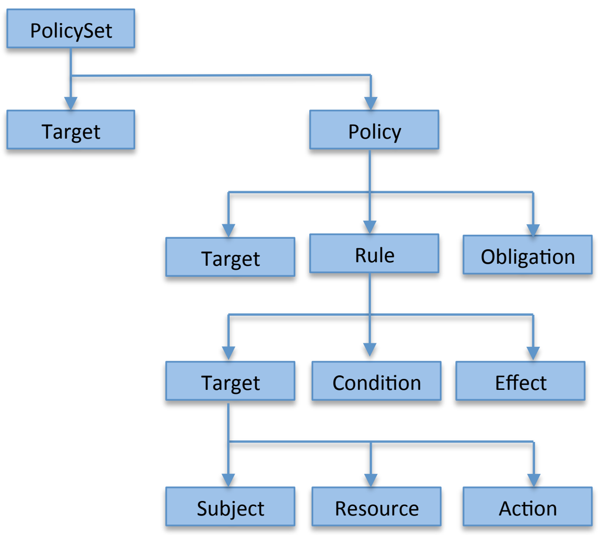


Figure 3.3: XACML policy structure

3.2.3 Beyond access control: usage control

Usage control extends the notion of access control to consider what can happen to the data in the future [PHB06]. A security policy reflects usage control concepts [PS02a] when it includes some actions that have to be carried out before the access (i.e., the user has to authenticate before accessing a web site), or during the access (i.e., the user has to keep an open window while he is accessing a web site), or after access (i.e., the user has to submit a form after his access). In security policies paradigm, those actions are usually referred to as (pre, ongoing, post) obligations [HBP05b] [Zha06]. They accurately allow to extend the notion of access rights with related duties, called obligations. Obligations, which have been introduced in [ML85], are considered as an important research direction in the domain of usage control [IYW06]. A complete security policy should encompass both rights and duties, both access authorizations and obligations. Usage control model (UCON) [PS02b, PS04, DXX09] is a popular model that is built based on the following concepts that we illustrate in Figure 3.4:

- *Continuity of decision:* Access control is verified before and during the access. Access to resources can be revoked after it has been granted due to a change of some object or subject attributes.

- *Mutability of attributes:* Subject's or object's attributes can be mutable (i.e., subject name) or immutable (i.e., resource cost). Immutable attributes can be updated before, during or after the access.

```
<PolicySet xmlns="xacml:2.0:policy:schema:os" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 PolicyCombiningAlgId="first-applicable" PolicySetId="LibrarySet">
    <!--                THE POLICY SET TARGET                                    -->
 <Target>
  <Resources>
   <Resource>
    <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
     <AttributeValue DataType="string">Book</AttributeValue>
     <ResourceAttributeDesignator AttributeId="resource-id" DataType="string"/> </ResourceMatch>
   </Resource>
  </Resources>
 </Target>
 <Policy PolicyId="Library" RuleCombiningAlgId="first-applicable">
     <!--               THE POLICY TARGET                                    -->
  <Target>
   <Subjects>
    <Subject>
     <SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
      <AttributeValue DataType="string">Student</AttributeValue>
      <SubjectAttributeDesignator AttributeId="subject-id" DataType="string"/>
     </SubjectMatch>
    </Subject>
   </Subjects>
  </Target>
  <!--              THE POLICY RULES                                   -->
  <Rule Effect="Permit" RuleId="Rule1">
      <!--             RULE 1 TARGET: SUBJECTS, RESOURCES AND ACTIONS                  -->
   <Target>
    <Subjects> <Subject>
      <SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
       <AttributeValue DataType="string">Student</AttributeValue>
       <SubjectAttributeDesignator AttributeId="subject-id" DataType="string"/>
      </SubjectMatch>
     </Subject>
    </Subjects>
    <Resources><Resource>
      <ResourceMatch
       MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
       <AttributeValue DataType="string">Book</AttributeValue>
       <ResourceAttributeDesignator AttributeId="resource-id" DataType="string"/>
      </ResourceMatch>
     </Resource>
    </Resources>
    <Actions><Action>
      <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
       <AttributeValue DataType="string">Borrow</AttributeValue>
       <ActionAttributeDesignator AttributeId="action-id" DataType="string"/>
      </ActionMatch>
     </Action>
     <Action>
      <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
       <AttributeValue DataType="string">Return</AttributeValue>
       <ActionAttributeDesignator AttributeId="action-id" DataType="string"/>
      </ActionMatch>
     </Action>
    </Actions>
   </Target>
  </Rule>
 </Policy>
</PolicySet>
```

Listing 3.1: XACML policy example

The model is based on a mapping of subjects and objects to access rights. Rights are evaluated based on 1) authorizations which are predicates on subject and object attributes, 2) conditions which represent predicates on environmental attributes and 3) obligations which represent actions that have to be performed before or during access.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<Request>
    <Subject>
        <Attribute AttributeId="subject-id" DataType="XMLSchema#string">
            <AttributeValue>Student</AttributeValue>
        </Attribute>
    </Subject>
    <Resource>
        <Attribute AttributeId="resource-id"
            DataType="XMLSchema#string">
            <AttributeValue>Book</AttributeValue>
        </Attribute>
    </Resource>
    <Action>
        <Attribute AttributeId="action-id"
            DataType="XMLSchema#string">
            <AttributeValue>Borrow</AttributeValue>
        </Attribute>
    </Action>
    <Environment/>
</Request>
```

Listing 3.2: XACML request example

3.2.4 Summary

In this section, we have have introduced key definitions that are necessary to the understanding of our contributions related to the optimization of the design of guest applications. The details that we have provided concerning XACML policy language and architectural model will be used in chapter 6 to explain the limitations that we have identified in the support of XACML to obligations. These concepts will also be used in chapter 7 to explain the problem of performance overhead encountered when processing XACML requests. Finally, we will also refer to the concepts of XACML language to present our proposed approach that aims at reducing regression testing costs in chapter 8.

3.3 State of the art in multi-objective optimization

This section provides background information about existing methods that have been used in the literature to solve multi-objective optimization problems. We conclude the section by analyzing the motivation behind using the resolution method that is adopted in this thesis.

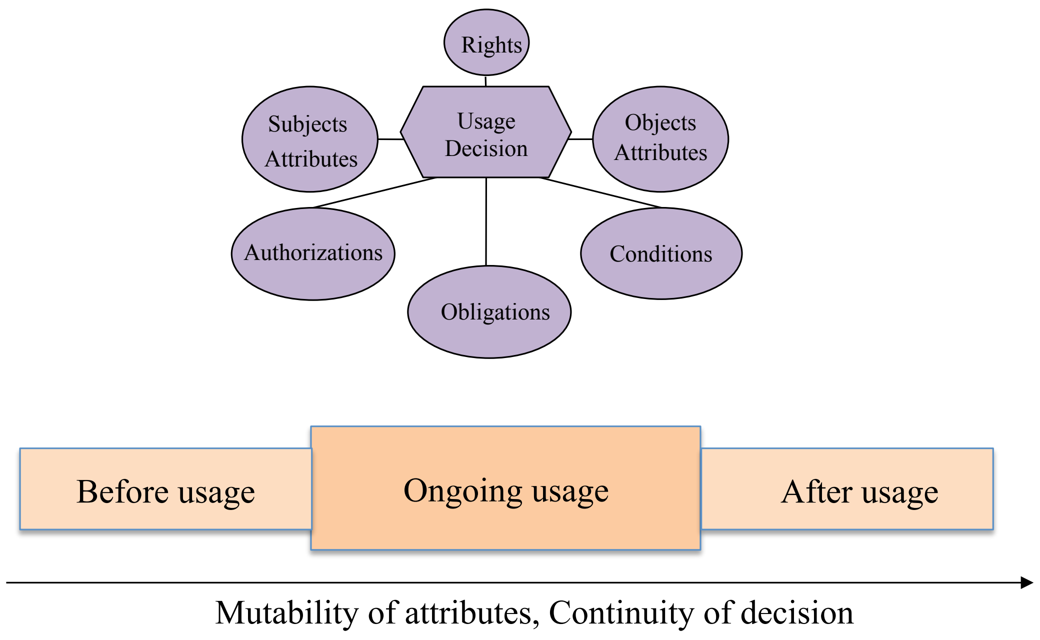


Figure 3.4: UCON model

3.3.1 Definition

In several fields and particularly in software engineering, we are facing several problems in which we aim at satisfying several constraints. Usually, these problems fall into two categories. The first category is the Constraint Satisfaction Problems (CSP) [ER97]. These problems are defined using a set of variables, variables domains and constraints over variables. A CSP resolution aims at finding a variable assignment that satisfies all the constraints. A CSP is defined by the triplet (X, D, C) where:

- $X = \{X_1, ...X_n\}$ is a set of variables,
- $D = \{D_1, ...D_n\}$ is a set of variables domains where $D_i(X_i)$ is the domain of the variable (X_i)
- $C = \{C_1, ...C_n\}$ is a set of constraints,

The second category is the Constraint Optimization Problem (COP) which extends a CSP with mathematical functions that a decision maker aims at maximising or minimizing. These mathematical functions are representative of objectives, that have to be maximized or minimized [Gav02]. In real life situations, decisions have to be taken by considering several objectives. In several contexts, these objectives can be conflicting. Multi-objective optimization [Deb14] refers to the discipline that tackles optimization problems [SD08] with several objectives potentially conflicting. The maximization or the minimization of one objective can degrade other objectives. Multi-objective optimization problems are illustrated by a wide range of real-life problems related to finance [TC07], engineering [MA04], software testing [YH07, LHM07], biology [HKK07], automotive systems [MAB09], etc.

In [AF09], Abbas *et al.*, have leveraged multi-objective optimization to schedule construction projects. The authors consider three axis to optimize: the project financing costs, the projects duration and the amount allowed for developing the project. The uncertainty related to the financing costs and credits has been taken into consideration using fuzzy-sets theory.

In [MAB09], Meedeniya *et al.*, have applied multi-objective optimization to find trade-offs between response time, reliability and cost in the context of redundancy allocation applied to automotive systems. Redundancy allocation is used to improve reliability in automotive systems. However, it introduces an overhead in the response time and increases the products costs. The approach uses ant colony optimisation algorithms to find a set of acceptable design solutions.

In the domain of software testing, Shin *et al.* [YH07], have used multi-objective optimization in the context of regression test selection. Their approach considers the code coverage, fault detection history and the cost in the problem formulation. The results, obtained with greedy algorithms and meta-heuristic search show that meta-heuristic search can perform greedy algorithms in terms of Pareto-optimality.

Formally, a multi-objective optimization problem aims at *minimizing* or *maximizing* a vector function F where F is a vector of n objective functions, i is the number of inequality constraints, j is the number of equality constraints, $x \in \Re_m$ is a vector of m decision variables:

$$\begin{aligned} max/min\ F(x) &= (f_1(x), f_2(x), ..., f_n(x)) \\ subject\ to\ G(x) &= (g_1(x), g_2(x), ..., g_i(x)) >= 0 \\ subject\ to\ H(x) &= (h_1(x), h_2(x), ..., h_j(x)) = 0 \end{aligned}$$

Three main categories of resolution methods to resolve multi-objective optimization problems exist over the literature [WBN07]:

- Aggregated-based methods: These methods transform the multi-objective optimization problem into a mono-objective optimization problem in which all objectives are assigned different weights and grouped into a single function.

- Non-Pareto-based methods, non Aggregated-based methods: Such approaches include the lexicographic method [BT80]. In this method, objectives are classified with respect to the decision maker preferences. The decision maker optimizes the different objective functions according to the classification that is given to the different objectives.

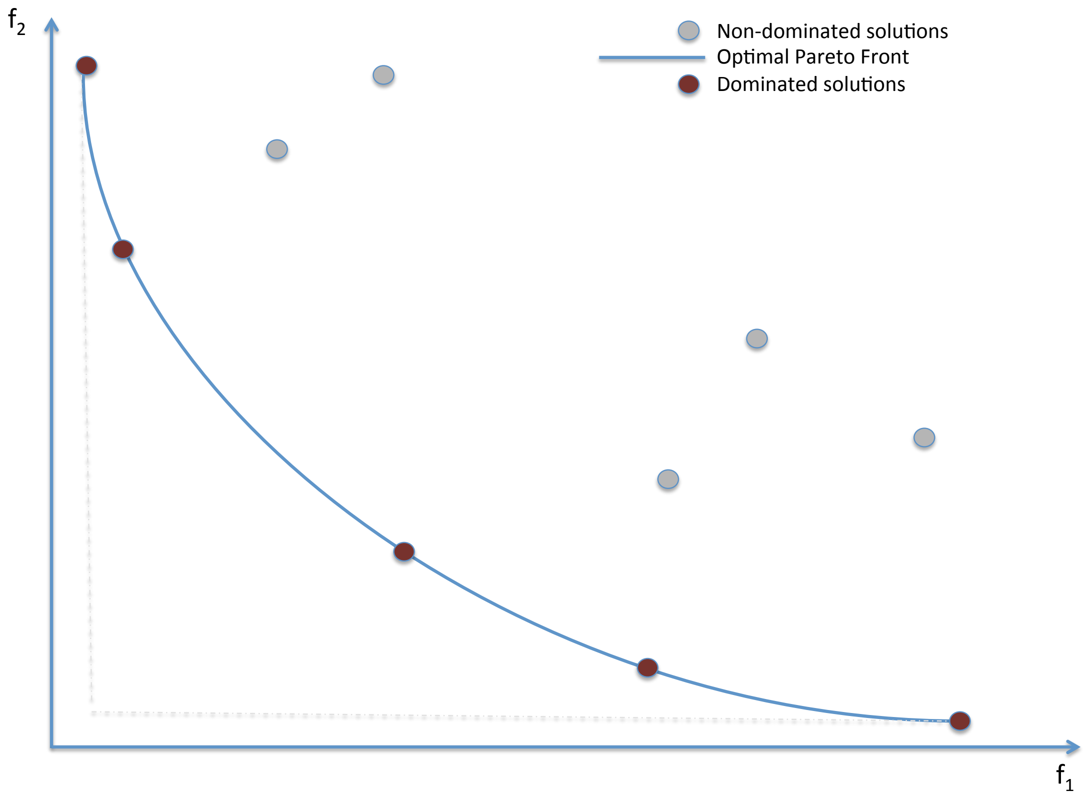


Figure 3.5: Pareto-optimal front

- Pareto-based methods: These methods are based on the concept of dominance as it has been defined by Goldebrg [Gol90]. In what follows, we give more details about Pareto-based methods as they will be used in our approach described in chapter 4 and chapter 5.

Multi-objective optimization based on Pareto-based approaches introduces two important concepts: The *Pareto Dominance* and the *Pareto Optimality* [D+01]:

- *Pareto Dominance*: For a *minimization* problem, given two solution X_1 and X_2, X_1 is said to *dominate* X_2, ($X_1 \succeq X_2$), if $f_i(X_1) \preceq f_i(X_2)$, for all $i = 1,..,n$ and $f_i(X_1) \prec f_i(X_2)$ for at least one objective function $f_i(X)$. This means that if X_1 is at least as good as X_2 in all objectives, and superior to X_2 in at least one objective.

- *Pareto Optimality*: Non dominated solutions are referred to as Pareto-optimal set. These solutions are grouped in a line called Pareto-optimal front. These solutions have at least one objective that is better than other solutions outside the line. Figure 3.5 presents these concepts in the context of a minimization problem including two objectives (f_1,f_2).

The resolution of multi-objective optimization problems aims at guiding the search to find closest points to the Pareto-optimal front. Another important target of the search in multi-objective optimization is to maintain diverse [TB03] set of solutions with uniform coverage of the Pareto-optimal front. By maintaining diversity, we reduce the risk of falling into local optimum which results in suboptimal performance.

3.3.2 Meta-heuristics

Several methods have been proposed in the literature to solve multi-objective optimization problems. These methods can be classified according to the preferences articulation of the decision maker: i) a priori methods are the methods in which the decision maker defines his preferences over the objectives at the beginning of the search ii) a posteriori methods are the methods that aim at finding a set of Pareto-optimal solutions. The decision maker chooses one solution among the solutions in the Pareto-optimal front by articulating his preferences at the end of the search iii) no preferences methods are the methods in which the decision maker does not articulate any preferences [MA04] and attempts to find one single-optimal solution iv) interactive methods are methods in which the decision maker introduces his preferences during the search. For the a priori methods, the decision maker introduces preference information for the different objectives using weighs on each objective and thus all objectives are transformed into a composite objective function. A single solution that satisfies the objective weight vector is produced. Multi-objective resolution methods can be classified into two categories. The first is called exact methods and usually provides one global optimum solution within unlimited time. This time execution limitation prevents the usage of this category in the context of search problems that have to be solved within limited time frame. The second category of resolution methods includes several types of heuristics. Heuristics enable to obtain approximate solution within specified period of time without optimality guarantees. The user may refer to [Meu02] for more ample details about resolution algorithms for multi-objective optimization problems. Figure 3.8 schematizes the classification presented in [Meu02]. Meta-heuristics [OK96] refer to the category of resolution algorithms that adapt their own processing during their execution. The usage of meta-heuristics techniques has been proven to find near-optimal solutions to resolve problems that lead to combinatorial complexity.

Meta-heuristics include a family of population-based techniques which relies on an iterative process and several agents that operate in parallel to find good solutions. At each generation, new solutions are created integrating the features of the parent solutions within a given population, similarly to the natural selection. A population with the same size is generated and the process continues until a stopping criterion is reached. Several parameters, objectives, have to be taken into consideration for the resolution of cloud-based software multi-objective optimization. This makes population-based approaches a suitable candidate for the problem that we address in this thesis. In what follows, we give an overview about the main meta-heuristics used in the literature.

Algorithm 1 Particle swarm optimization pseudo-code

```
Particle Initialization
for all Particle p do
    Calculate fitness value f
end for
repeat
    for all Particle p do
        if f < pbest then
            pbest:=f
        end if
    end for
    set gbest as best fitness function among all particles
    for all Particle p do
        Calculate particle velocity according to equation (1)
        Update particle position based on calculated velocity
    end for
until Stopping criterion is reached
```

Particle swarm optimisation (PSO)

Particle swarm optimisation (PSO) [ES01], has been introduced in 1995 by Kennedy and Eberhart. PSO techniques mimic the moving behaviour of flocks of fishes or birds. The method involves a set of particles that constitute a swarm that explores the search space aiming at finding best solutions. Each particle is a point in a N-dimensional search space. The movement of each particle in the swarm is guided by the information related to the particle's best position value called pbest and to the best value achieved by particles in its neighbourhood best position called gbest value. The search aims at accelerating particle's velocity based on pbest and the gbest values. Thus beyond particle's self experience, the search takes advantage of the swarm social experience. Each particle move can be modeled through the following mathematical function:

$$v_i(k+1) = w \times v_i(k) + c_1 \times rand_1(...) \times (pbest - x_i(k)) + c_2 \times rand_2(...) \times (gbest - x_i(k)) \quad (1)$$

- $v_i(k)$: velocity at iteration k
- $v_i(k+)$: velocity at iteration $k+1$
- c_1, c_1: constants
- rand: random number in the interval 0..1
- $x_i(k)$: current position at iteration k
- pbest: particle's best position value
- gbest: neighbourhood best position value

For a minimization problem, the algorithm is described in the pseudo-code of algorithm 1.

Tabu search (TS)

Tabu search (TS) [Glo89] has been introduced by Fred Glover in 1986. The algorithm maintains a tabu list that contains already visited solutions. At each iteration, the solution that has the highest score is selected if it does not belong to the tabu list. To prevent from falling into local optimum, the solutions in the tabu list can not be revisited for a certain number of iterations. This gives higher priority to the unexplored search areas to be visited. Algorithm 2 presents the pseudo-code of Tabu search algorithm.

Algorithm 2 Tabu Search Pseudo-code

```
Initialize Tabu List TL
current_configuration:=Random(AllSolutions)
while stopping criterion not met do
    Evaluate score(s), $s \in$ Neighborhood(current_configuration) $\cap s \notin$ TL
    best_configuration:=SelectHighestFitness(s)
    current_configuration:=best_configuration
    TL.Add(current_configuration)
end while
```

Simulated annealing (SA)

Simulated annealing (SA) [Kir84] is an optimization technique that mimics the annealing technique in metallurgy which consists in varying a material temperature. A high temperature increases the atoms energy and consequently increases the probability to achieve random moves. Temperature cooling decreases the atoms energy and consequently slows down the atoms moves. The intuition is that the cooling process produces a thermal equilibrium. During this thermal equilibrium, atoms reach a global energy minimum. Simulated annealing (SA) prevents from falling down into a local optimum through random movements that can produce inferior solutions. The search process starts by random initial placement and the initialization of the highest temperature value. A new configuration is obtained by introducing a mutation operation in the current configuration. The new configuration is evaluated based on the evaluation of a fitness function and its acceptance depends on the current probability that depends on the current temperature. In each iteration, the temperature is updated based on a temperature schedule. Algorithm 3 presents the pseudo-code of the Simulated annealing algorithm.

Ant colony optimization (ACO)

Ant colony optimization algorithm (ACO) [DB10] is inspired by ants behaviour to locate food. An ant localizes randomly a source of food and returns back to its nest laying down a pheromone trail. Other ants, searching for food, follow a pheromone trail with a high probability. Effective pheromone are the ones that represent shortest paths between ants and the source of food and

Algorithm 3 Simulated annealing pseudo-code

```
Initialization: Temperature $T_{start}$, $T_{end}$, Temperature schedule Initialization Schedule(T),
Random initial placement
**while** current T > $T_{end}$ **do**
    $new\_configuration$ = Mutate($current\_configuration$);
    $\Delta score = score(new\_configuration) - score(current\_configuration)$
    **if** $\Delta score < 0$ **then**
        $current\_configuration := new\_configuration$
    **else**
        with probability $P = e^{-\Delta score/currentT}$
        $current\_configuration := new\_configuration$
    **end if**
    current T=Schedule(T)
**end while**
```

consequently will be followed by more ants. The pheromone that are not followed by few ants will disappear. Ant colony optimization represents optimization problems that can be modeled like finding the shortest path in a given graph. In a multi-objective optimization problem, a number of ant colonies and a number of pheromones are provided as inputs in the algorithm. A colony of ants is mapped to one objective. At each iteration, pheromone trails are updated based on the paths followed by ants. The reader may refer to [ASG07] for more details about ant colony optimization (ACO) defined for multi-objective optimization problems.

Genetic algorithms

As depicted by [ZQL$^+$11], genetic based approaches are suitable candidate for scheduling and planning problems. Genetic Algorithms (GA) are driven by elitism rules that favor the survival of strongest species (best candidate solutions) in analogy to natural selection [VVL00]. They are based on an iterative search process, which involves a population composed of a certain number of individuals. These individuals are randomly selected and mutated (or mixed using crossover operator) in each iteration to constitute the next *generation* population. The initial population can be generated randomly. Offspring generation can also be performed to achieve certain goals such as maximizing the population, maintaining diversity or favouring a certain objective among the set of considered objectives. *Fitness functions* [BMB93] are used to evaluate a population with regards to a specific optimization problem, in analogy to natural selection in which species qualities are evaluated according to their surrounding context.

Genetic algorithms introduce changes in the population to create new individuals called offspring through the following operators [VV99]:

- The *crossover* operator generates an offspring by a genetic recombination of the two selected parents. The resulting offspring maintains some features from each parent, thus maintaining population diversity. There are three types of crossover [BC95]. The *one-point* *crossover* is based on a random selection of a crossover single point. Based on the position of this point, the gene pool of the two individuals is switched. The *two-point crossover* differs from *one-point crossover* by the selection of two crossover points. The *uniform crossover* is based on an uniform switch of all the genes composing two individuals.

- The *mutation* operator introduces perturbations in the population and aims at preventing a convergence towards local optimal convergence. Mutation operator aims at preserving the diversity in the population [Mah95] by performing new changes in the individuals structure. Figure 3.6 illustrates the mutation and crossover operators for two individuals with a numerical encoding.

- The *selection* operator selects a fixed number of fittest offspring for the next generation to maintain a fixed population size. The literature identifies five techniques of individuals selection [GD91]:

  - *Roulette wheel selection*: The principles of this selection mimic the traditional roulette wheel which is used in casinos. Each individual is assigned a probability of selection which is proportional to its fitness improvement. By random drawing of lots, individuals who have higher probability have more chance to be selected.
  - *Rank selection*: All individuals are ranked based on their fitness. Best individuals, based on this ranking, are selected with higher probability.
  - *Tournament selection*: The *binary tournament* is the simplest version of tournament selection. It is based on random selection of two individuals and on maintaining the best one as parent. The *Probabilistic binary tournament* differs from binary tournament selection by choosing the two individuals with a probability p instead of a random selection. Another version of tournaments is referenced in the literature and it is based on the selection of $n > 2$ individuals for comparison instead of the selection of two individuals.
  - Steady state selection: This selection technique maintains most individuals in the population. A number n of best individuals is selected and a number n of worst individuals is discarded and most individuals remain in the next population.
  - Elitist selection: This type of selection ensures the selection of the best individuals. While the fittest individuals in the rank section are selected wit a probability $p < 1$, the elitist selection strategy selects the fittest individuals with a probability of selection $p=1$.

Genetic algorithm workflow is described in Figure 3.7. The process starts with an initial population that contains n individuals. The individuals that are selected based on selection operators, are mutated or crossed over to compose the new population P'. If the stopping criteria is reached, best individuals are selected. If the stopping criterion is not reached, another iteration

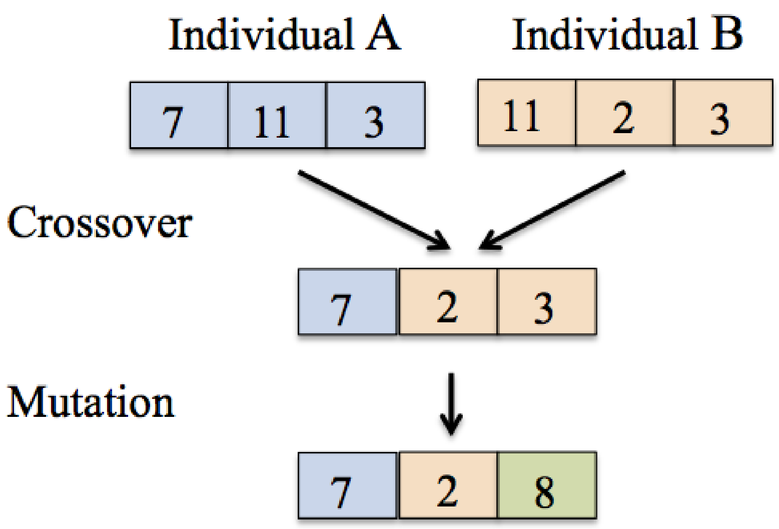


Figure 3.6: Mutation and crossover operators

of the workflow is repeated. This iteration starts by the evaluation of the current solution and the selection of n best individuals.

The table 3.2 synthetizes the different characteristics of the different meta-heuristics that have been presented above.

3.3.3 MOEAs and cloud engineering problems

Genetic algorithms are advocated to resolve problems which are characterized by a wide search space. The variant of genetic algorithm that is based on non-dominated ranking is referred to as Multi-Objective Evolutionary Algorithms (MOEAs) and it has been introduced by Goldberg in 1989 [Gol89]. MOEAs have been successfully applied in many domains such as finance, logistics, test cases optimization [BFJLT05] and recently in cloud engineering problems. In [CL04], the authors have provided a taxonomy of the different application domains of MOEAs. Some of these applications have to be used in a run-time context where dynamic parameters adjustment is required such as running vehicles guidance. In [HLS+13], the authors have motivated the use of search-based approaches for the resolution of cloud engineering problems. Multi-objective Evolutionary Algorithms (MOEAs) [D+01, VVL00] is a class of search based approaches that addresses problems in which a decision maker aims at finding a solution that optimizes several conflicting objectives. MOEAs simulate population evolution to produce solutions exhibiting trade-offs between conflicting objectives such as grid jobs scheduler [FFH13b] as they are able to automate a set of configurations exploration [CL04]. MOEAs offer generic and reusable domain exploration capabilities which make them suitable candidate for self-adaptive systems. Self-adaptive systems require run-time corrective actions [KC03], that will be made after the identification/selection

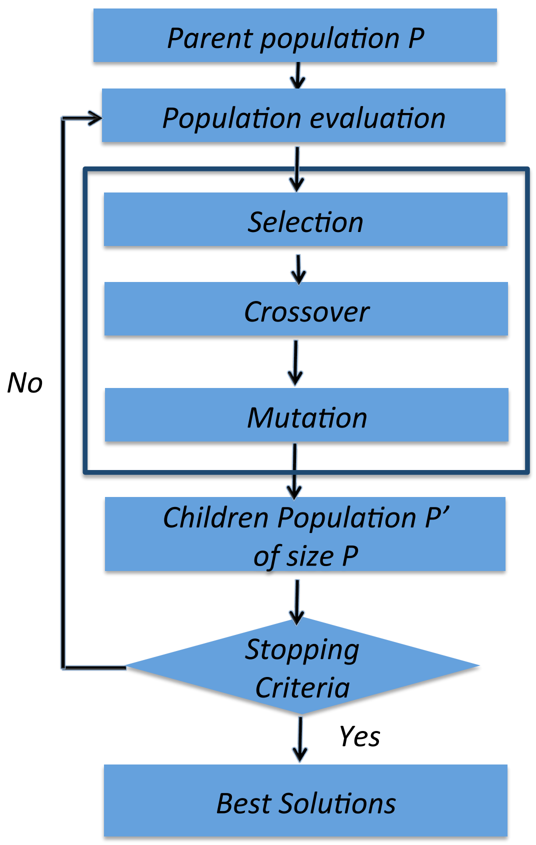


Figure 3.7: Genetic algorithm workflow

of the best fitted configuration. A cloud infrastructure can be abstracted by a set of software resources that run on top of virtual machines (VMs) dynamically starting/stopping in physical machines. MOEAs are thus used nowadays in several design case studies [FFH13b], such as self-adaptive cloud scheduling problems, to maintain conflicting quality characteristics [D$^+$01, VVL00] such as system performance, cost and safety. Beyond their applicability for cloud optimizers, MOEAs offer the following advantages to set-up autonomous self-adaptive engines working "at run-time": (i) no need for predefined solutions, (ii) the incremental optimization process can be stopped on-demand, (iii) operating multi-objective optimization and finding trade-offs.

3.3.4 Quality criteria

Several metrics have been proposed over the literature to evaluate MOEAs. Some of these metrics evaluate the convergence achieved by MOEA by measuring how close are the solutions from the Pareto optimal front. Other metrics evaluate the solutions diversity by evaluating the uniform distribution across the Pareto optimal set while the third category of metrics evaluates both the convergence and the diversity of the obtained solutions. The table 3.3 presents these quality

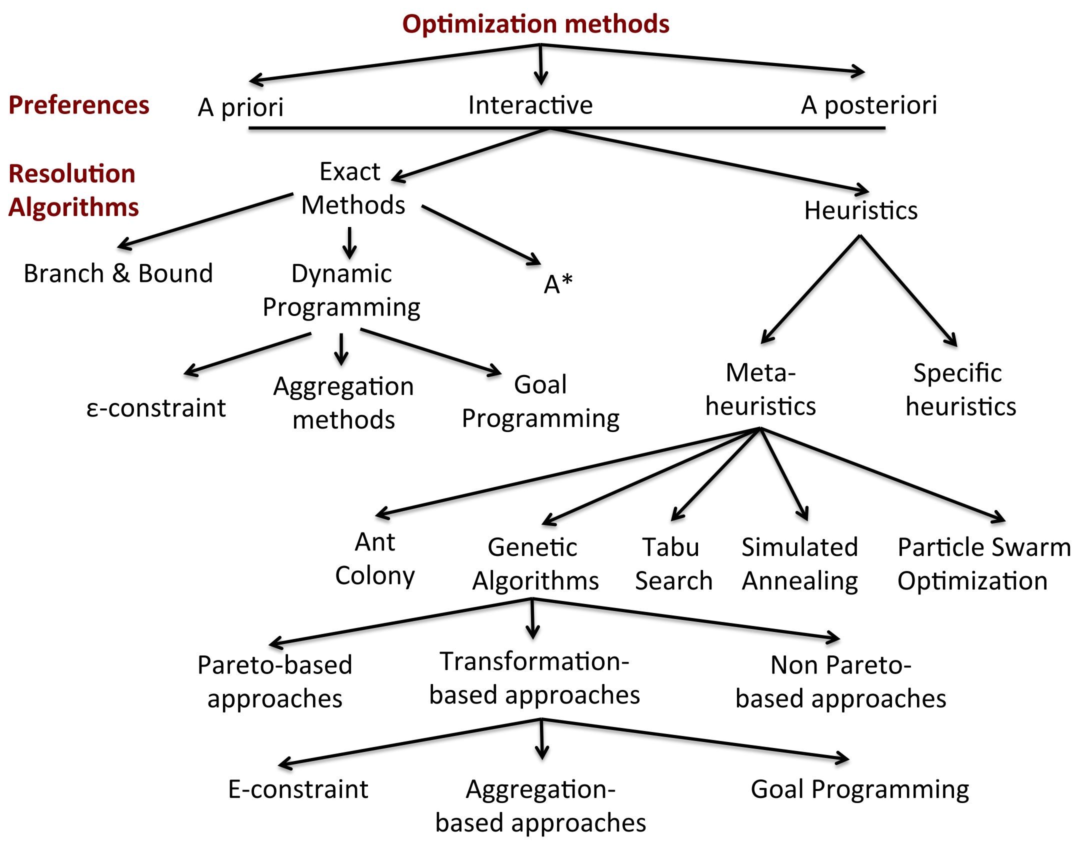


Figure 3.8: Classification of multi-objective optimization resolution algorithms, Source: [Meu02]

criteria. In what follow, we only detail the hypervolume criterion, which is the criterion that we have chosen in our work.

The hypervolume indicator was proposed by Zitzler [ZT99] as a quantitative indicator to evaluate multi-objective evolutionary algorithms. It was defined as the n-dimension volume covered by a set of pareto-optimal solutions where n is the number of objectives. The hypervolume is calculated based on a reference point. The illustration of the hypervolume calculation is shown in Figure 3.9.

3.3.5 Summary

In this section, we have explored resolution techniques that can be used to solve multi-objective optimization problems. We have discarded a priori methods from our resolution methods due to the difficulty that arises from setting preferences over objectives. We have selected Pareto-based approaches to take into consideration all the different objectives that will be defined in chapter 4. Pareto-based approaches enable to provide several acceptable solutions in one single run which enables decision makers to have high flexibility in terms of selection of Pareto-optimal solutions.

Table 3.2: Meta-heuristics comparison

	Swarm optimization	Ant colony	Tabu search	Simulated annealing	Genetic algorithms
Stochastic search	x	x	x	x	x
Near optimal solutions	x	x	x	x	x
Parameters tuning	Generation number Population size Operators	Generation number Ants number Operators	Neighbourhood structure Tabu operators Size of tabu list	Initial temperature Annealing schedule temperature Acceptance probability function Energy function	Generation number Population size Operators and their probability
Population-based	x	x	x		x
Memory-usage			x		

Table 3.3: Quality metrics classification

Metric Objective	Quality Indicator
Convergence	Generational distance [VVL98] ϵ-indicator [FKTZ05]
Diversity	Generalized Spread [NLA+08] Spread [D+01]
Convergence and Diversity	hypervolume [ZT99] Inverted Generational Distance [ZTL+03]

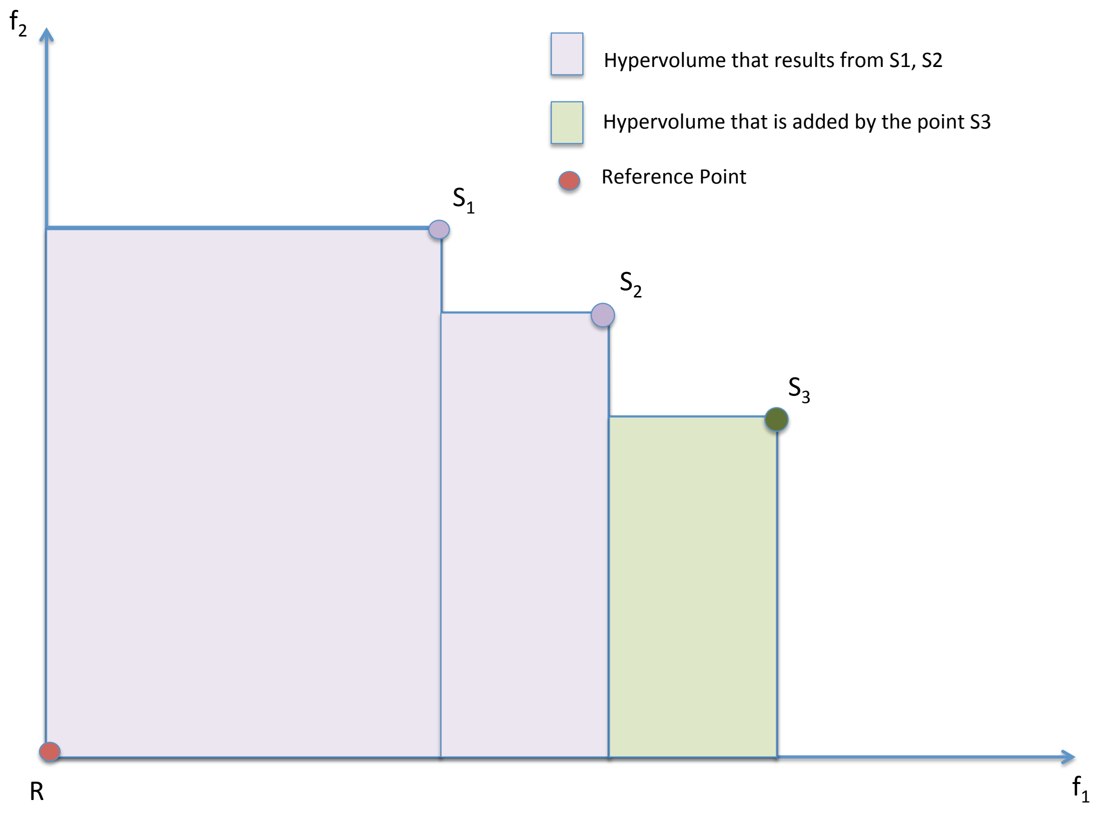

f_2
Hypervolume that results from S1, S2
Hypervolume that is added by the point S3
Reference Point
S_1
S_2
S_3
R
f_1

Figure 3.9: Hypervolume indicator

In the context of cloud-based software, a selected solution can be a solution that represents a cloud configuration that is the most similar to the configuration that corresponds to the initial population. The intuition behind choosing an optimal configuration with small similarity distance with regards to the original configuration is to reduce the costs of changing the configuration of the real system.

3.4 Architectural modeling

In this section, we motive the use of models@run.time for cloud abstraction and we describe existing approaches in architectural optimization. The concepts that are introduced in this section are relevant to an overall understanding of chapter 4 and chapter 5

3.4.1 Models@run.time for abstraction of cloud infrastructures

Model Driven Engineering (MDE) [Béz04] is a paradigm that promotes models as the central key element of all the phases of software development. MDE enable to build an abstraction layer to handle complex software systems and to automate various tasks. Model Driven Architecture (MDA) illustrates a normalization of MDE that has been proposed in 2000 by OMG (Object Management Group) [S^+00]. MDA offers several techniques that allow defining families of languages and supporting tools. As an example, MOF (Meta-Object Facility) [Poo01] provides a language for defining meta-models. MDA allows to define mappings between platform-independent models (PIM) and platform-specific models (PSM). EMF (Eclipse Metamodel Framework) is an MDA implementation that enables to support metamodels and to generate application code. The limitations of EMF to support a runtime context, in terms of memory and computational power, have been discussed in [FNM^+14b]. Kevoree Modeling Framework (KMF) [21] is an alternative to EMF that aims at decreasing memory footprint, defining different operations on models (i.e, loading, saving, model elements query). KMF enables to improve the support of runtime models for large distributed systems.

Several approaches have used model driven engineering [Ken02] for cloud abstraction. Recently, Chatziprimou *et al.* have advocated the use of MDE for selecting optimal cloud optimizations [CLZ13]. They have built an automated model that helps extracting available resources in the cloud as a first step to select optimal cloud configurations. In [FACDN13], the authors have proposed a model driven approach to reason about cloud costs and performance. The approach is based on the extension of the Palladio Component Model (PCM) and the Palladio Bench for QoS evaluation [BKR09]. In [DWS12], the authors have used model-driven engineering to map virtual machines configurations to feature models which they have transformed into constraint satisfaction problems (CSPs) that result in cloud configurations with optimized energy consumption. In [BMM12], the authors have proposed CloudML, a modeling language that models heterogeneous resources in an application that is provisioned through public cloud providers such as Amazon EC2 and Rackspace. The application is mapped to the resources metamodel. Models@run.time [BBF09b] paradigm is an evolution of MDE that permits to reason about the system at design time and extends the reasoning to the system while it is in a running state. It takes into consideration design-time information at execution time for continuous revaluation, to re-evaluate requirements satisfaction while the system is evolving. Cloud infrastructure is a concrete example that takes advantage of models@run.time techniques. As stated in [SV13], cloud parameters lead to complex state representation related to the heterogeneity of the different cloud elements manipulated by cloud infrastructure such as virtual machines, and applications that are hosted on top of them. We recur to models@run.time techniques since they offer an abstraction layer to reason about cloud adaptations and to resolve dynamically the different trade-offs that have to be envisioned at runtime. Models@run.time enable to build a synergistic relationship between the software model and the real system. Optimization can thus be performed at the level of the model for a given system snapshot before a concrete reflection of an optimal configuration at the level of the real system. We use Kevoree[22] as a models@run.time to provide a cloud ab-

[21]http://kevoree.org/kmf/
[22]http://kevoree.org/

straction that we describe in chapter 4. Kevoree provides different configuration, adaptation, and synchronization mechanisms for the distributed reconfigurable software development.

3.4.2 Architectural optimization

Some research initiatives have proposed optimization layers on the top of architectural modeling platforms. In [YLR+03], the authors have explored the effectiveness of model driven optimization by comparing it to empirical optimization. Their comparison was carried out on ATLAS (a system for generating a dense numerical linear algebra library called the BLAS). Based on optimization parameters, the performance of the code generated by empirical optimization was similar to the one generated through model driven optimization. In [ABGM09], the authors have designed an Eclipse-based tool called ArcheOpterix for the architectural optimization of embedded systems that they build on top of AADL (Architecture Analysis and Design Language) supporting platforms. Given conflicting quality objectives, the tool has been able to find deployment architectures with improved quality. In [MKBR10], the authors have used multi-objective evolutionary algorithms on top of architectural models built with Palladio Component Model. Our model-driven optimization approach described in chapter 4 joins these approaches to propose an optimization layer on top of runtime models to optimize cloud-based software deployment.

3.4.3 Summary

In this section, we have explored models@run.time concepts and architectural optimization. The overall chapter digs into the main concepts and definitions that will be presented throughout the thesis. Next chapters dig into the details of our contributions.

Chapter 4

Cloud-based software management: A multi-objective optimization problem

This chapter proposes a generic approach to build a cloud-based software optimization layer by combining modeling and search based paradigms. The proposed approach is two-fold: 1) To reason about a cloud environment, we use a models@run.time approach to have an abstraction layer of a cloud configuration that supports monitoring capabilities and represents cloud intrinsic parameters like cost, performance, etc. 2) We use a search-based algorithm to navigate through cloud candidate configuration solutions in order to solve the *C*loud *M*ulti-objective *O*ptimization *P*roblem *(CMOP)*. The approach has been validated on a case study defined with a cloud provider in Luxembourg, EBRC[1] as representative of a dynamic management problem of heterogeneous distributed cloud nodes. We implement a prototype of our PaaS supervision framework using Kevoree, a models@run.time platform. The prototype shows the efficiency of our approach in terms of finding possible cloud configurations in reasonable time. The prototype is flexible since it enables an easy reconfiguration of the cloud customer optimization objectives.

A major part of this chapter has been published in the following conference proceedings: ***Donia El Kateb****, François Fouquet, Grégory Nain, Jorge Augusto Meira, Michel Ackerman, and Yves Le Traon. Generic cloud platform multi-objective optimization leveraging models@run.time. In Proceedings of the 29th Annual ACM Symposium on Applied Computing (SAC), pp 343-350, 2014.*

[1] http://www.ebrc.com/

Contents

4.1 Introduction

Cloud computing promises scalable hosting by offering an elastic management of virtual machines, which run on top of hardware data centers. This elastic management as a cornerstone of PaaS (Platform As A Service) has to deal with trade-offs between conflicting requirements such as cost and quality of service. Solving such trade-offs is a challenging problem. Indeed, most of PaaS providers consider only one optimization axis or ad-hoc multi-objective resolution techniques using domain specific heuristics.

Current cloud platforms usually focus on one dimensional optimization. For instance, RedHat OpenShift [2] PaaS provides a rule-based engine to deal with horizontal computational power and disk space scalability by starting and stopping gears (computation nodes). In this engine, rules are uncorrelated and represent independent mono-objective optimization axis. In addition, such rule-based engine is not suitable to handle non deterministic choices such as selecting one action in case of several available gears types that can achieve horizontal scalability.

[2]https://www.openshift.com/wiki/architecture-overview#_3._Horizontal_scaling_Beta

This chapter tackles cloud multidimensional and orthogonal optimization. Our goal is to provide a decision making tool that can be reconfigured according to evolving customers optimization objectives. In the presence of conflicting objectives, our approach aims at calculating an optimal cloud configuration, given a set of resources that are dedicated to a cloud customer.

In a nutshell, our approach aims at generating an optimized PaaS layer. We model a cloud infrastructure using Kevoree, a models@run.time [MBJ+09] platform that simulates possible space of cloud reconfigurations and adaptations. We model the cloud management problem as a multi-objective search problem where customers requirements are captured through SLAs (Service Level Agreements) [PPSSA]. We use genetic algorithms applied on a given system configuration snapshot, to resolve the *Cloud* Infrastructure Management *Multi-objective Optimization Problem (CMOP)*, by dynamically choosing optimal configurations among the space of possible cloud configuration alternatives. We validate our approach through a use case defined with our partner EBRC.

The use case relies on a heterogeneous cloud infrastructure (different categories of virtual machines). In this use case, we address two research questions. 1) The first research question explores the feasibility of our approach by showing the ability to reconfigure a cloud infrastructure at runtime in the presence of conflicting objectives. In this research question, we have explored the feasibility of our optimization approach at the level of the architectural model that we have built using Kevoree models@run.time platform. 2) The second research question is related to the performance of our approach with regards to the customer infrastructure scalability.

This chapter is organized as follows. Section 4.2 describes the key concepts related to this chapter. Section 4.3 describes the approach that we have used to solve the multi-objective optimization problem. Section 4.4 details the experiments that we have run to validate our approach. Finally, section 4.5 concludes this chapter.

4.2 Problem definition and modeling

This section defines the *Cloud* Infrastructure Management *Multi-objective Optimization Problem (CMOP)* and motivates the use of models@run.time for cloud abstraction.

4.2.1 CMOP inputs

As introduced in the background section, cloud customers benefit from virtualization technology to host their data/applications according to Service Level Agreements (SLAs) [SSH12]. Cloud environments are dynamic environments in which customers and providers are expressing evolving requirements towards cost saving, QoS evolution, etc. PaaS providers mostly manage VMs allocation (i.e., by starting VMs or shutting them down) to regulate the quality of service of customer services.

In this work, we explore optimization from cloud customers perspective. A cloud customer has to adapt to evolving needs that might be conflicting. Given a cloud customer infrastructure, which consists in a dedicated set of allocated physical resources, we aim at developing a decision support that takes as input SLAs objectives and a snapshot of a cloud configuration and provides decision capabilities to place software components in VMs in the objective to maintain compliance with SLAs requirements and to adjust costs to specific customer needs.

VMs placement is an active research area [FRMMne] that illustrates the trade-offs that should be taken into consideration to manage a cloud infrastructure. As an illustrative example, to reduce costs, customers have to consolidate VMs that have to be started in physical machines. However at the same time, they have to host the loads of different end users on different VMs to achieve isolation.

VMs heterogeneity in terms of deployment costs or in terms of hosting geographical location adds more complexity to the VMs management problem and motivates the need to have well-defined reasoning techniques for VMs allocation.

We consider scenarios that present an actual interest to EBRC, a cloud provider in Luxembourg, which offers a complete range of tailored services in hosting and managing a cloud infrastructure. EBRC is interested in improving the supervision of its customer environments. We explore resolution strategies to deploy an optimized cloud environment for EBRC customers. Customer expectations are commonly captured through SLAs (Service Level Agreements), which consist in defined terms in the form of a contract between a service consumer and a service provider. As defined in the SLA Handbook [BB09]: "It is the Service Level Agreement that defines the availability, reliability and performance quality of delivered telecommunication services and networks". Most SLAs introduced in the literature have focused on the aspects related to performance and have not investigated the aspects related to security. In this work, we consider the isolation property in the context of a single customer infrastructure. Depending on the confidentiality of their data, we consider that customers are able to express some constraints related to the isolation of their workloads/data in the cloud environment. For example, a customer may require that his applications/data are hosted in dedicated VMs. In this work, we also consider cost reduction as an optimization objective. Next section shows how we model the different customer objectives through fitness functions.

4.2.2 Cloud infrastructure: a dynamic adaptive system

As shown in Figure 4.1, PaaS customers use cloud infrastructure resources to host applications and data that can be accessed by end users. (1) Cloud infrastructure optimization is triggered by some application increasing workload, a change in security requirements or the introduction of new objectives. (2) To achieve local optimization, the cloud provider recurs to our framework to trigger a VM reconfiguration at the level of the cloud infrastructure. The request is interpreted at the PaaS level so that VMs placement can be taken into account at the level of physical machines. Our optimization framework aims at reconfiguring a cloud infrastructure, at a given time, to find an optimal configuration given the presence of several conflicting constraints. The

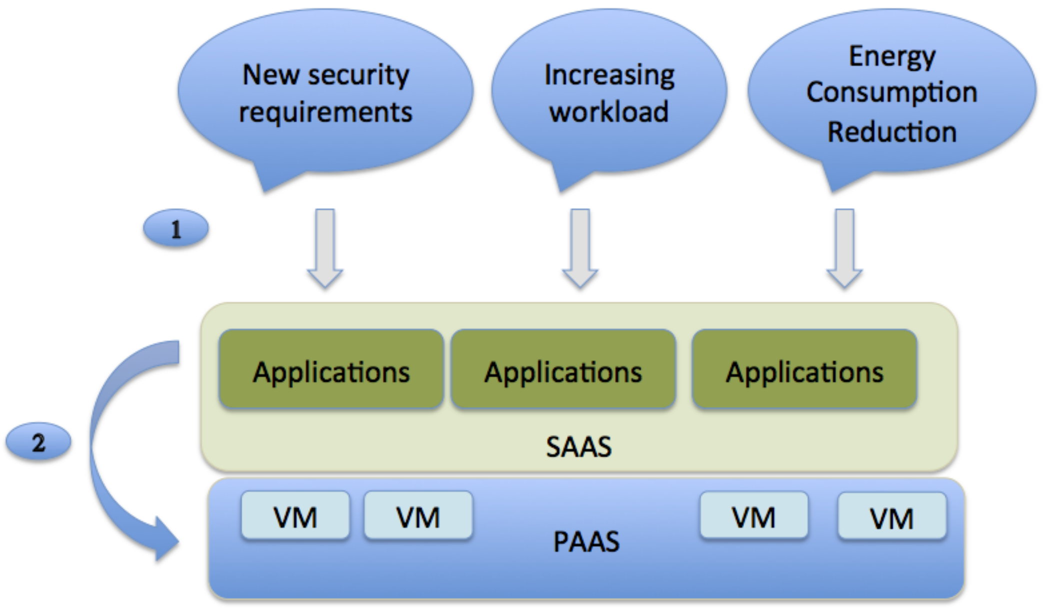


Figure 4.1: Positioning with regards to cloud service models

dynamic features of a cloud infrastructure are the key factors that have motivated us to consider the concepts related to dynamic adaptive systems (DAS) to reason about cloud adaptations. Clouds infrastructures are adaptive systems since they have to respond to changing conditions orthogonally to the business logic of the application [MBEB11]. The design of DAS systems involves many challenges including the representation of their states, their safe adaptation, the management of environmental conditions and the change in requirements [BCZ05]. To achieve application consistency and service continuity, DAS handle changes when the system is running, through dynamic reconfiguration. Reconfiguration capabilities at runtime enable DAS to adjust themselves to unpredictable conditions, in their operating environment. Dynamic reconfiguration techniques [HW04] ensure high availability and maintain Quality of Service parameters by reducing service interruption. In [KC03], Kephart *et al.*, have defined four main characteristics for an autonomous system, which are:

- Self-configuring: The system is able to define and adapt its architecture according to the variables of its execution platform.
- Self-healing: The system is able to diagnose its internal state.
- Self-optimizing: The system is able to adapt its internal resources to optimize its functioning.

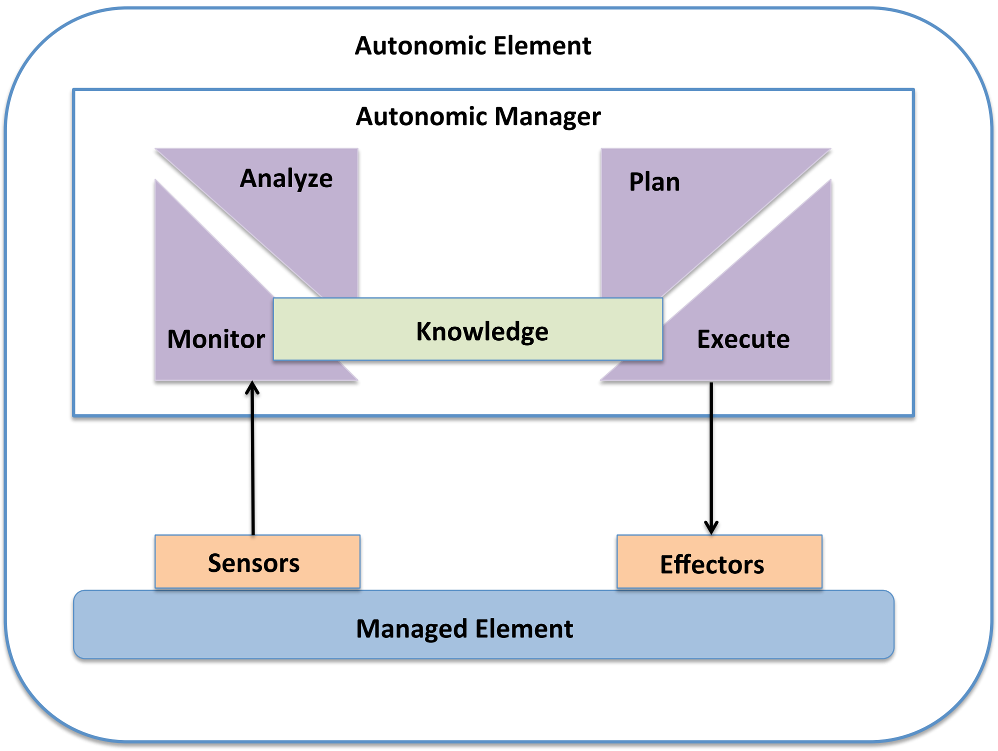


Figure 4.2: IBM's MAPE-K reference model for autonomic systems

- Self-protecting: The system is able to detect and protect itself against external threats.

These definitions introduce the MAPE loop that defines the standard process of an autonomous system as illustrated in Figure 4.2. Cloud infrastructures are adaptive systems that have to respond to changing conditions, orthogonally to the business logic of the application [MSST06]. In [MBEB11], the authors have extended the view of the MAPE on a cloud environment. DAS are empowered with reflection capabilities [DDKM08] that have been first introduced in [BGW93] to describe reconfiguration capabilities of programs. Reflection has two main features:

- Introspection: related to the capacity of dynamic adaptive systems to observe their own behaviour and to analyse their current state.

- Intercession: defines their ability to change their own behaviour.

Brice Morin [Mor10] has used reflection notions of the models@run.time [BBF09a] paradigm. He proposed an approach to maintain a causal link between an architectural model and a running system. The approach enables to perform reasoning for a given snapshot of the real system in an asynchronous manner. This reasoning can result in new configurations that can be tested and then embedded at the level of the real system. The separation between the model and the running system enables to get rid of the burden of performing verification at the level of the real system. Reflection capabilities will be used in order to reason about cloud states. The next section presents the platform that we use to handle cloud adaptations. The overall cloud dynamic reconfiguration approach is illustrated in Figure 4.3.

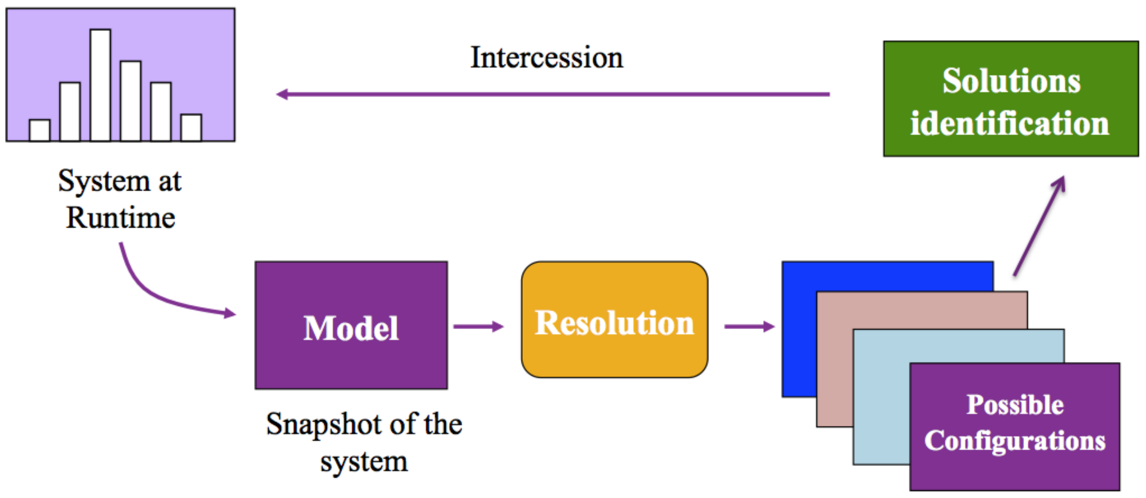


Figure 4.3: Overall approach

4.2.3 Models@run.time for cloud abstraction

We recur to models@run.time techniques since they offer an abstraction layer to reason about cloud adaptations and about the different trade-offs that have to be envisioned at runtime. The extension of Kevoree models@run.time platform to abstract layered cloud infrastructure has been proposed by [Dau13]. Daubert has defined the different characteristics that are required by a cloud infrastructure modeling layer and has shown how these characteristics are reflected in the proposed Kevoree cloud abstraction. The different characteristics and how they are supported in Kevoree platform are summarized in Table 4.1.

Kevoree provides a high level abstraction for clouds through a standard MOF-based metamodel [vDKV00]. Our approach bridges the gap between any MOF-based tool and real infrastructures. The framework is thus able to provide a supervision layer for a customer infrastructure by identifying optimal cloud local configurations in some evolving contexts like an increasing workload. We use Kevoree as a platform to run our implementation. A physical machine is abstracted as a Kevoree Infrastructure Node, a VM is abstracted as a Kevoree Child Node and a component is abstracted as a software service. The architectural model can be easily deployed in a real large-scale production environment [FMF+12] [FDP+12] like a cloud infrastructure.

Cloud feature	**Feature supported by Kevoree modeling layer through:**
Cloud-layered hierarchical model (representation of software components, platforms, physical nodes) and the representation of their interactions	Kevoree Infrastructure node, Kevoree child node, software service
Hosting information	Notion de container nodes
Distributed infrastructure	Nodes hierarchy and node container
Resources heterogeneity	Typing and associated treatments (DictionnaireType)
Extension capabilities	Extendable types
Asynchronous reflection capabilities	Model snapshot management capabilities

Table 4.1: Kevoree specific cloud features

4.3 Problem resolution

A cloud multi-objective optimization problem formulated by PaaS cloud providers can be stated as follows: “Given a dedicated set of virtual machines allocated by a PaaS provider, how to optimize software placement in VMs to reduce costs and to maintain compliance with SLAs requirements". We have opted for a search-based approach to study the effectiveness of the supervision framework.

```
public class AddNodeMutator implements <Cloud> {
    private Random rand = new Random();
    private DefaultCloudFactory cloudfactory = new DefaultCloudFactory();
    @Override
    public List<MutationVariable> enumerateVariables(Cloud cloud) {
    return Arrays.asList((MutationVariable) new QueryVar("target", "nodes[*]"));
    }
    @Override
        /**
         * This mutation operator defines a muation operation
         * applied on a metamodel that defines a cloud elements
         */
    public void mutate(Cloud parent, MutationParameters mutationParameters) {
        int i = rand.nextInt(1);
        if (i==0)
        {VirtualNode node = cloudfactory.createAmazon();
        node.setId("EC2_"+Math.abs(rand.nextInt()));
        node.setPricePerHour(10.0);
        parent.addNodes(node);          }
        else
        {
        VirtualNode node = cloudfactory.createRackspace();
        node.setId("Rack_"+Math.abs(rand.nextInt()));
        node.setPricePerHour(5.0);
        parent.addNodes(node);}     }}
```

Listing 4.1: AddNodeMutator Class

4.3.1 CMOP modeling as a search-based problem

As we have mentioned earlier, in this work, we leverage MOEAs. Formally, we define *CMOP* by the following triplet *(I,F,CO)* such that:

1. I denotes a cloud infrastructure model. A cloud infrastructure model I is an abstraction of a set of VMs. Each VM hosts n software components C [HC01].
2. F is a vector of objective functions $F(X) = (f_1(x), f_2(x), ..., f_m(X))$, that have to be minimized.
3. CO denotes a set of possible configurations in I that satisfy F. A configuration $co \in CO$ is obtained through a mapping function that maps the components C to VMs. A configuration co is represented like the following: co=($VM_1(c_1...c_k)$,...,$VM_m(c_1...c_l)$), for example $co = VM_1(c_1, c_2, c_3)$ denotes a configuration with a single virtual machine VM_1 hosting 3 components c_1, c_2, c_3.

We model the key elements in our genetic algorithm as follows:

- **Individual:** A solution vector $x \in X$ that corresponds to a cloud infrastructure model.
- **Genes:** A gene corresponds to a component in our context.
- **Population:** A population corresponds to a set of cloud infrastructure models.
- **Genetic operator:** A genetic operator enables to change the cloud configuration state.
- **Fitness Function:** The fitness function is required to assign a value to each individual that reflects its potential ability to improve a cloud configuration.

4.3.2 Fitness functions definition

The vector F(X) is composed of the following five objective functions:

1. $f_1(x)$= Cost(x) where Cost(x) is calculated based on the Amazon pricing model[3].
2. $f_2(x)$=Security(x) where x = $\{(C)\}$, security(x) defines the security level (classification or clearance) associated with a component [LB96]. If a component with a security level 4 shares a VM with a component with a security level 2, then a security violation of degree 2 is reported, if the component shares a VM with a component with a security level 1, then a security violation of degree 3 is reported.
3. $f_3(x)$=Completeness(x) where x=(C$_1$,....,C$_n$) denote the software components that have to placed in a cloud infrastructure. The output value of $f_3(x)$ is 100 if no component is placed and 0 if all C$_1$,....,C$_n$ are placed.

[3]http://aws.amazon.com/ec2/pricing/

4. $f_4(x)$=Overload(x) where x = {(C)}. Overload(C) defines the gap between the required and the observed % of virtual CPU for each software element.

5. $f_5(x)$=SLAPerformance(x) where x={(C)} defines the required CPU for every component as stated in the SLA.

The security fitness function aims at maximizing isolation. Isolation is a crucial requirement that has to be taken into consideration in a shared environment that relies on a multi-tenant [TJA10a] model for hosting services and data. In a cloud delivery model, a single instance of software can be shared between tenants who customize this instance according to their own specific requirements before delivering it to the end user. Software components run on the top of VMs. VMs management is commonly performed at the level of the hypervisor, which provides an abstraction layer between the physical network and the VMs. VMs are able to communicate without even going through the physical network, thus bypassing the checks that have to be performed at the network layer. This increases the risk of attack propagation to VMs residing in the same physical machine once a VM is compromised. Isolation enables to avoid data leakage that results from VMs interference attacks.

4.3.3 CMOP algorithm

CMOP is based on NSGA-II, an elitist non dominated sorting genetic algorithm [DPAM00]. At each iteration i, mutation and crossover operators enable to obtain an offspring population Q_i. Table 4.2 presents our set of defined operators O. The individuals of the populations P_i and Q_i are combined to compose a population R_i of size 2n. Two operations are performed in the population R_i:

- *Non-dominated sorting*: The elements of the population R_i are ranked into r ranks. Rank 1 is the non-dominated set and elements of a given rank r dominate the elements of the rank r+1.

- *Crowding distance ranking*: Crowding distance selection [DPAM00] is performed based on the ranking that results from the previous step. Crowding distance enables to prevent from premature convergence by preserving diversity within populations. It is calculated based the distance of an individual to its neighbours. Higher scores are assigned to individuals, which are less close to their neighbours. From each rank in R, the individual that has the best score in the crowding distance is selected as an element in the next generation P_{i+1}.

The two operations enable to maintain n individuals in the population P_{i+1}. Algorithm 4 describes the CMOP:

In the validation section, we compare two variants of *CMOP* to assess their effects on the algorithm convergence. The first algorithm is NSGA-II configured with ε-dominance [LTDZ02] whereas the second algorithm is a plain NSGA-II.

Algorithm 4 <CMOP Algorithm>

```
Input: (Population-Size int, Generation-Number int)
Generate an initial population P
while Stopping criterion is not reached do
    Apply randomly a set of mutation operators $O$ on P to get population Q
    Combine individuals of population P and population Q into a population R
    Perform non-dominated sorting of the population R
    Select elements in each front based on crowding distance to compose $P_{new}$
end while
Evaluate $P_{new}$ and select solutions $co \in CO$
```

Table 4.2: Genetic operators

Operators	**Description**
AddComponent(c,A)	Adds a component c in the the virtual machine A
RemoveComponent(c,A)	Removes a component c in the virtual machine A
MoveComponent(c,A,B)	Moves a component c from the virtual machine A to the virtual machine B
SwitchComponent(c,A), (c_1,B)	Switches the component c to the virtual machine B and the component c_1 to the virtual machine A

ε-dominance is a relaxed Pareto dominance where the user defines ε as a parameter. Given two solutions with ε difference in an objective k, the two solutions cannot be considered as dominating each other. ε-dominance allows to not discard less fit solutions.

4.3.4 Model-driven optimization

We have used Polymer[4] to run our optimization algorithms. Polymer is a model-driven optimization framework that builds an optimization layer on top of models. Polymer enables to absorb the complexity related to data representation and to the encoding of problem parameters. This enables non experts in search-based engineering to encode and solve complex optimization problems without prior knowledge about optimization algorithms.

Polymer takes advantage from model-driven engineering to build meta-models that represent the concepts that are relevant for a specific domain. Polymer uses the Kevoree Modeling Framework (KMF), which is an alternative to Eclipse Modeling Framework (EMF) [SBMP08]. KMF has been developed in François Fouquet's thesis [Fou13] with the objective to enhance the performance requirements of modeling frameworks. The different principles, used to build KMF in a way that reduces the memory footprint of the generated code, are explained in [FNM+14a]. KMF proposes an efficient modeling layer for runtime usage. Polymer aims at achieving optimiza-

[4]http://kevoree.org/polymer//

tion on any KMF compliant model that can be expressed with the usage of Meta-Object Facility (MOF) concepts [Poe06].

Polymer relies on object oriented programming design to apply operators and fitness functions on the model instead of the problem encoded parameters. Fitness functions are implemented by interfaces extending the modeling API that presents the application domain layer. Figure 4.4 defines a simplified meta-model of a cloud infrastructure. Listing 4.1 presents the code of the mutation operator that can be generated based on the defined meta-model.

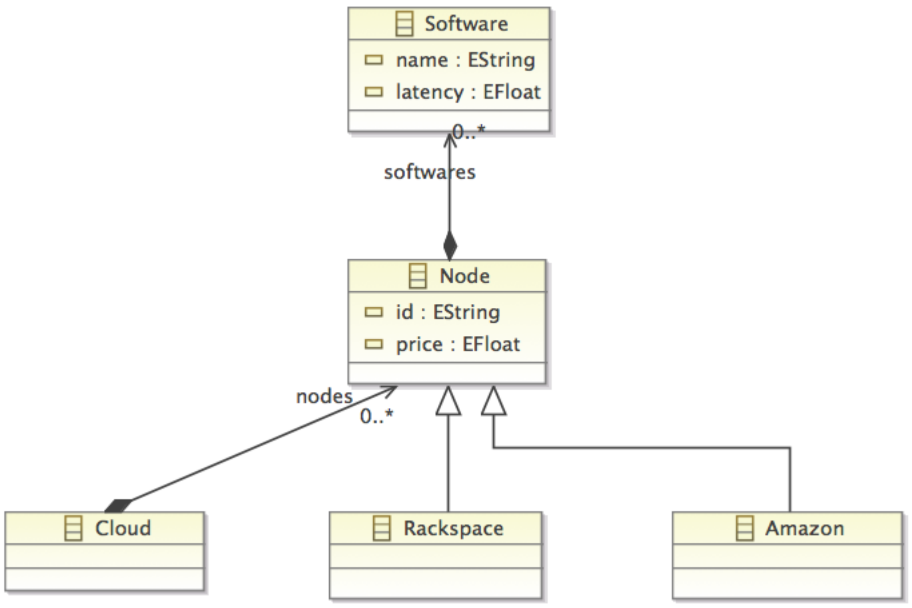


Figure 4.4: Simplified view of a cloud metamodel

4.4 Validation

To show the effectiveness of our approach, we have implemented CMOP (with and without ε-dominance) on Kevoree platform and we have compared it to random and mono-objective solutions. The validation of our approach is related to the scenarios of interest for EBRC provider and focuses on the following research questions:

- RQ_1**:** Comparison of mono-objective vs. multi-objective optimization. *What is the benefit of using multi-objective optimization compared to mono-objective one? How faster a good solution is found using a mono-objective algorithm compared to a multi-objective algorithm?*

- RQ_2**:** Comparison of multi-objective algorithms in terms of objectives satisfaction and scalability. Which algorithm, among CMOP with ε-dominance dominance, CMOP without ε-dominance and random, better satisfies the objectives while scaling to a more complex cloud infrastructure model?

In the validation, our goal is to show the effectiveness of search-based approaches in finding possible solutions at the level of the architectural model and to show that the approach scales. In our scenario, EBRC allocates to a customer X a cluster composed of five physical machines to host its on-line stock trading website that is composed of the following components: A database to store items, a load balancer, a database for payment, a database to manage users and a web front-end. Virtual CPU assignment depends on the physical machines, which are either Intel Xeon or ARM processor-based. In this scenario, we explore the optimization problem by considering optimization at the level of a single customer infrastructure.

4.4.1 Multi-objective optimization results

In this part, we will answer the research question: RQ_1*: What is the benefit of using multi-objective optimization compared to mono-objective one?* Nowadays, most cloud providers use mono-objective optimization for horizontal scaling. For instance RedHat OpenShift[5] only considers the performance of a customer front-end, by an adaptation algorithm defined in the load balancer. In the same manner as RedHat OpenShift, Amazon[6] defines trigger levels based on performance measures to start or stop virtual machines. Both approaches only consider one objective (i.e., the front-end performance) to achieve optimization at the level of cloud customer infrastructure and do not take into account other optimizing factors like isolation. The first experiment that we have conducted tries to evaluate the potential optimization quality by considering all fitness functions during the optimization.

4.4.1.1 Setup

Here is the description of the cloud infrastructure model corresponding to the five physical machines that we deploy in Kevoree:

- Three Low power consumption ARM based infrastructure nodes (1 virtual machine abstracted as child node where each node is 1GHz)
- Two High power consumption Xeon based infrastructure nodes (8 virtual machines abstracted as child node where each node is 1GHz)

Table 4.3 defines some parameters that are relevant for the estimation of the fitness function:

- The required CPU (GHz) for every component
- The required security level for each component
- The CPU load property, which defines the required virtual CPU percentage for each component to run

[5]https://www.openshift.com/wiki/architecture-overview#_3._Horizontal_scaling_Beta
[6]http://aws.amazon.com/autoscaling/

Component	**Required CPU (GHz)**	**Security Level**	**CPU Load (VCPU %)**
ItemDB	1.2	2	40
LoadBalancer	0.4	0	20
PaymentDB	0.6	4	60
UserDB	0.4	3	40
WebFrontend	1.2	1	40

Table 4.3: Services in the SLA model

Algorithm	**Completeness**	**Consumption**	**Overload**	**Security**	**SLAPerformance**
Mono-objective	0	100.0	24.00	70.0	**0.0**
Multi-objective	0.0	**43.47**	0.0	**0.0**	**0.0**

Table 4.4: Fitness vector values: Mono-objective vs multi-objective
(the values related to the multi-objective row are better). The row mono-objective only considers SLAPerformance fitness while the row multi-objective considers all fitness functions. It is worth to note that despite the SLAPerformance (in bold) presents a perfect score for both mono-objective and multi-objective algorithms, the multi-objective reaches better scores for security and global consumption.

Two algorithms for ten initial populations are thus compared on this basic configuration:

- A mono-objective algorithm that minimizes SLAPerformance fitness without taking into consideration other fitness functions.
- A multi-objective CMOP algorithm.

We have thus performed 20000 generations per run before selecting the best configuration and have used the techniques presented in Kevoree to increase the performance of our search based algorithm especially for the different operations defined in the previous section.

4.4.1.2 Results

Among the configurations identified as solutions of the search-based problem, the value of the fitness vector for (Completeness, Consumption, Overload, Security, SLAPerformance) is illustrated in Table 4.4 and the evolution of the fitness scores of all objective functions given by CMOP are given in Figure 4.5. As already demonstrated by Frey *et al* [FFH13c], the multi-objective algorithm maximizes the satisfaction of the fitness functions through reaching a mean that is better compared to a mono-objective algorithm. However, it is interesting to notice that the four fitness functions reach a perfect score with the multi-objective search while only the SLAPerformance reaches a perfect score for the mono-objective algorithm. The consumption fitness in both cases could not reach 0 because at least one machine must be started to host a software component, however we observe a significant consumption reduction while using the

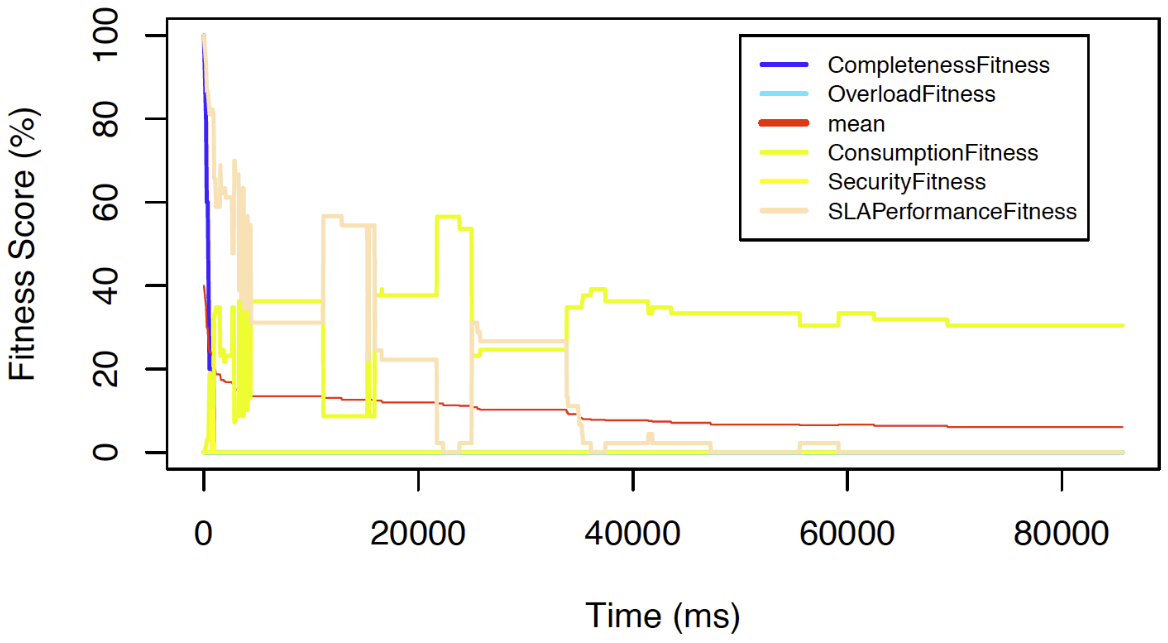


Figure 4.5: Evolution of the fitness scores of all objective functions and the evolution of the average function (mean)

It is important to notice that in order to reach a global mean (shown in red), the search algorithm satisfies better some objectives to the detriment of others objectives in order to globally reduce the global mean. For instance the consumption at the T=20000ms needs to be increased in order to reduce the mean and customer SLA satisfaction. The consumption increase is explained by the algorithm that tries to find available solutions to host all software components.

multi-objective algorithm. The mono-objective search (SLAPerformance objective in our example) achieves the best score in 600 ms, CMOP stabilizes a nearly optimal solution after 1700 ms and introduces then a time overhead compared to the mono-objective search. However the time overhead remains reasonable when running CMOP, and presents a time duration value that is acceptable for a cloud infrastructure reconfiguration.

4.4.2 Scalability discussion

In this part, we will answer the following research question: RQ_2: *What is the impact of scalability on objectives satisfaction and on algorithms convergence?* The scalability issues are explored as follows:

- 1) Scalability in width: How the quality of results is impacted by the size of the search domain? Does each solution continue to find qualitatively good results with the increase of the search domain?

- 2) Scalability in depth: How the size of the search domain impacts the convergence speed?

In all the experiments, we evaluate the results quality in terms of fitness scores. The execution time of our algorithms is only used to verify our compliance with the time constraints of our cloud case study. The current implementation can be greatly improved in terms of time of computation by parallelizing the steps of our algorithm.

4.4.2.1 Setup

To answer this research question, we use NSGA-II with and without ε-dominance and random generation:

- The random generation of 500 solutions using our mutation operators. We consider the best random solutions (over 500 configurations).
- The multi-objective genetic algorithm with 1000 generations on 10 populations (thus 10000 generations) per run, before returning the best solution.
- The same multi-objective genetic algorithm with and without ε-dominance.

In this experiment, we make the infrastructure scale as follows: (scale 1: 5 infrastructure hosts), 3 times bigger scale (15 hosts), and so on with scale 4, 5, 8 and scale 12 (60 hosts). The random uses 500 generations while the genetic uses 1000 since it is slower, we then adapt random generation to normalize the resolution time. This normalization is necessary since we compare qualitatively results after the same search time for each algorithm.

4.4.2.2 Results

The results shown in Figure 4.6 correspond to scale four runs and compare the evolution of mean scores for the CMOP with ε-dominance [HN08], CMOP without ε-dominance and random. They show the progress of the algorithms to satisfy the objectives. It appears that CMOP without ε-dominance performs better, which is not surprising since it performs evaluation without using a ε-dominance for solution comparison, which makes it faster. However, CMOP without ε-dominance algorithm often converges to a solution that minimizes one or two objectives perfectly, but degrades the other (premature convergence for some objectives). Thus the CMOP with ε-dominance appears to be the best solution to avoid having some privileged objectives and to ensure an uniform optimization distribution. For the first scale, the comparison between the three algorithms demonstrates that the best scores are obtained without ε-dominance, with the average of 5.21%, than with CMOP with ε-dominance algorithm with the average of 6.4%. Random search results in worst fitness scores, since we achieve an average value of 18%. These experiments with these specific configurations reveal the feasibility of a bounded-in-time reasoning on an abstract representation of a cloud infrastructure. In terms of algorithms comparison, multi-objective

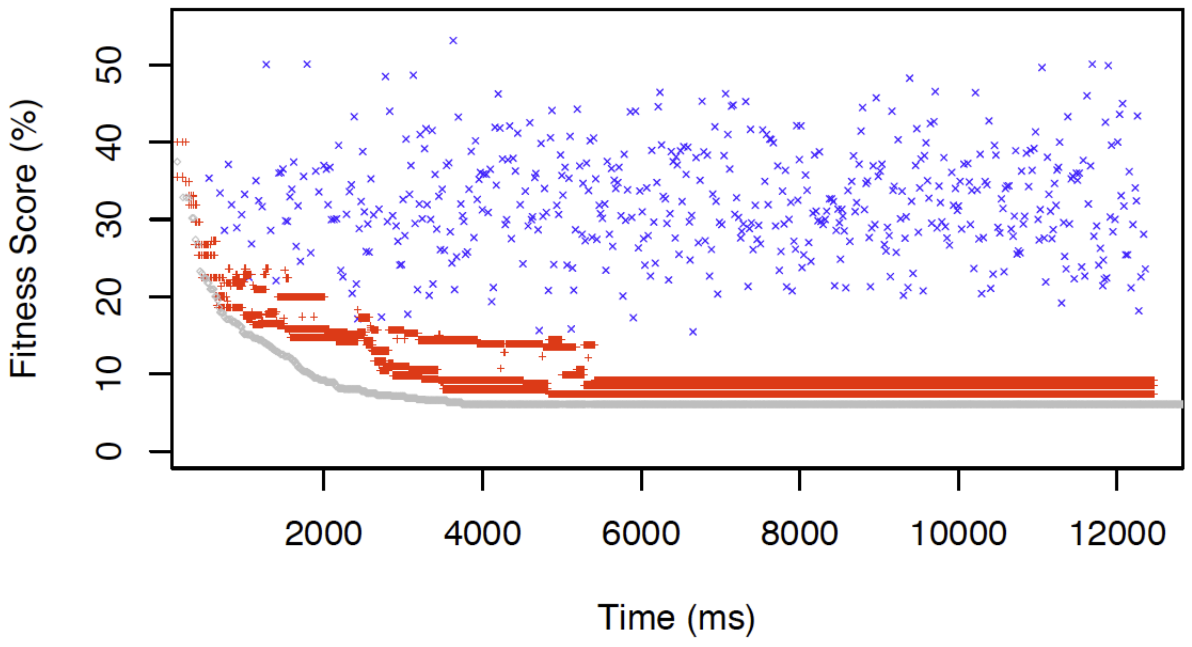


Figure 4.6: Comparison of multi-objective algorithms
(lower is better). In red, the five best mean values obtained with ε-dominance. In gray the mean value without ε-dominance and in blue the mean value of random solutions. It is important to notice that genetic search is always better than random and that ε-dominance converges slower than CMOP without ε-dominance.

Table 4.5: Comparison of multi-objective algorithms for the different scales

Scale	**1**	**3**	**4**	**5**	**8**	**12**
Random	**18**	**16**	**14.7**	**16**	**14.9**	**14**
Without ε-dominance	**6.4**	**5.9**	**7.9**	**7.1**	**6.69**	**8.3**
With ε-dominance	**5.21**	**6**	**8.9**	**8.8**	**9.4**	**11**

optimization outperforms random algorithms. Finally, CMOP with ε-dominance achieves the best trade-off in terms of computation time, and objectives satisfaction, while it is shown that it is not optimal in terms of mean objectives values. The lesson learned from this experiment is that the scalability can be greatly improved by using an hybrid approach combining 2 steps of CMOP, one with and one without ε-dominance to firstly generate good solutions and secondly to refine them with the multi-objective ε-dominance based-search. Table 4.5 shows the average of all fitness functions for the three scales and for the different algorithms. Since the configuration combinations are larger, all algorithms tend to find better trade-offs when the scale of the problem increases.

4.4.3 Threats to validity

In the present chapter, there are many factors that represent a threat to the validity of the obtained results. Internal validity concerns the possible courses of bias of the experiments that were conducted. More precisely, those threats are related to the:

- Number of generations and populations selected at the initial setup.
- Stopping criterion that has been chosen.
- The modeling of fitness functions that have been chosen.

To improve current experiments, we intend to conduct experiments on a varied range of populations size. In the current work, the stopping criterion has been based on a timing constraint, we plan to consider other stopping criteria that are based on generation comparison. This comparison will be based on similarity criteria that we plan to define on architectural models. We also plan to consider different other representations of fitness functions and to evaluate solutions quality through quality metrics [Bra11]. External validity are related to the pertinence of the optimization axis that we have chosen and to the resolution strategies that have been used in this chapter.

4.5 Summary

Model-driven engineering techniques have been advocated to build an abstraction layer for cloud infrastructures. This abstraction layer enables to reason about the problem of multi-objective optimization in the cloud. For instance, in [CLZ], Chatziprimou *et al.*, have built an automated model that helps the extraction of available resources in the cloud as a first step to select optimal configurations.

In this chapter, we go a step forward and we advocate the usage of models@run.time to propose a model-driven optimization approach to resolve cloud-based software trade-offs. We propose a tool-based approach that reconfigures dynamically a cloud infrastructure considering evolving providers and customers objectives. Our optimization approach, compared to optimization techniques surveyed in chapter 3, approaches the optimization problem from a software engineering perspective.

The findings of the work presented in this chapter show the effectiveness of search-based approaches to resolve a cloud multi-objective optimization problem. The optimization reasoning engine has been implemented using Polymer, a model driven optimization framework that operates on top of Kevoree, a models@run.time platform. Polymer eases the encoding of the multi-objective optimization parameters. The results have shown that the algorithm is able to find nearly-optimal solutions in a reasonable time, that can be considered acceptable in the context of cloud-based software dynamic reconfiguration. We believe that handling dynamically and in a fine-grained way customers deployment costs and adjusting them to the resources consumption enables to manage cloud elasticity. This is one of the main push factors that would influence moving customers workloads from local hosting to cloud hosting.

In the next chapter, we bring performance requirements of model-driven continuous optimization to light. We present full details of our optimization method that operates on the top of MOEAs to improve the performance optimization process defined in the current chapter.

CHAPTER 5

Optimizing Multi-Objective Evolutionary Algorithms to enable quality-aware software provisioning

In the last chapter, we have empirically shown that Multi-Objective Evolutionary Algorithms (MOEAs) are suitable candidates to resolve cloud-based deployment trade-offs. To save deployments costs, scaling down/up resources has to be performed in a reasonable time. Requirements related to cloud adaptation put some constraints on the efficiency of the optimization process used to find optimal cloud configurations. This fact drives us to explore the efficiency of the optimization process that we have defined in the previous chapter. MOEAs produce many dead-born solutions because of the Darwinian inspired natural selection, which results in a resources wastage. To tackle MOEAs efficiency issues, we propose in this chapter a process similar to modern biology. We choose specific artificial mutations by anticipating the optimization effect on the solutions instead of relying on the randomness of natural selection. We introduce the *Sputnik* algorithm, leverging the past history of mutations to enhance optimization processes such as cloud elasticity engines. We integrate *Sputnik* in a cloud elasticity engine, dealing with performance and quality criteria, and demonstrate significant performance improvement, meeting the runtime requirements of cloud optimization.

A major part of this chapter has been published in the following conference proceedings: ***Donia El Kateb****, François Fouquet, Johann Bourcier, and Yves Le Traon. Optimizing multi-objective evolutionary algorithms to enable quality-aware software provisioning. In 14th International Conference on Quality Software (QSIC), pp 85-94, 2014.*

Contents

5.1 Introduction

A major factor which threatens the adoption of MOEAs for self-adaptive systems such as a cloud infrastructure is the run-time resources consumption, which often requires ad-hoc empirical tuning of such algorithms to meet performance needs. Indeed, as in the Darwinian theory [Dar59, RR03], the evolution process of MOEAs relies on random mutations to ensure proper domain exploration. This randomness leads to sub-optimal performance due to the creation of many dead-born evolution branches. Coello *et al* [CL04] report that even if MOEAs usage simplifies the design of automatic configuration engines, empirical fine-tuning of evolutionary search parameters is still needed to save computational costs. These results motivate the need for software engineering techniques to avoid MOEAs ad-hoc tuning and to provide reusable techniques and frameworks for run-time usage.

Nowadays, modern genetics does not only rely on natural evolution process. Instead, based on the founding work of Muller *et al.* [PLA27], artificial mutation is now widely used to save time and generation cycles for instance to produce genetically modified organisms (GMO) [CdL10]. Instead of relying only on crossover and natural selection, Muller *et al.* [PLA27] studied artificial mutation using X-Ray to modify a fruit with an anticipated intent. These principles have led the genetic field to build instruments for such selective artificial mutations: the evolution process is accelerated by selecting some specific mutations that contribute to enhance a certain objective.

Going along the same line, in this chapter we study how such principles could be adapted to MOEAs to accelerate the convergence by guiding the evolutionary algorithms through dynamically selected mutation operators. Our intuition is that operators applied in a smart and artificial way would provide better results than operators applied randomly, and in particular would reduce the number of useless solutions. Thus, the new algorithm we propose is no longer inspired by Darwinian evolution, but by “artificial mutation", based on a smart and dynamic selection of the best mutation operator to apply at a given step. By applying such operators in priority, we aim at orienting the evolution process of a given population in the right direction for the problem to solve.

In this chapter, we present a hyper-heuristic [BHK+10], called *Sputnik*, inspired by artificial mutation. Our algorithm takes its name from a virus family, which evolves and mutates together with their host in order to perfectly fit their environment and to replicate more quickly. In the same manner, the *Sputnik* algorithm leverages a continuous ranking of operators according to their impact on fitness functions to smartly select dynamically the most relevant mutation operator as the search evolves.

We focus on performance as a key factor for run-time usage to reach faster acceptable trade-offs while saving computation time and generation cycles. For instance, the acceleration *Sputnik* provides is useful for adaptive systems when a solution/reaction has to be found in a short time. We evaluate our approach on a cloud reasoning engine that is able to continuously provision customers software while handling several conflicting objectives (i.e, isolation, cost). We have integrated *Sputnik* in the Polymer[1] framework and evaluated it using Kevoree[2]. We have conducted experiments to compare natural selection performance versus *Sputnik* performance. Our experiments highlight that *Sputnik* results in a faster convergence by reducing the number of generations while conserving the ability to achieve acceptable trade-offs in our use case. This chapter is organized as follows. Section 5.2 describes the key concepts related to this chapter. Section 5.3 presents *Sputnik* hyper-heuristic. Section 5.4 presents validation elements of our approach. Finally, sections 5.5 concludes this chapter.

5.2 MOEAs and run-time constraints

In this section, we highlight run-time usage control requirements and we give an overview about MOEAs performance issues. Finally, we introduce hyper-heuristics and we highlight their role in improving MOEAs algorithms efficiency.

5.2.1 Performance of MOEAs at run-time

MOEAs have been successfully applied in many domains such as finance, logistics, test cases optimization [BFJLT02] and recently in cloud engineering problems. In [CL04], the authors have provided a taxonomy of the different application domains of MOEAs. Some of these applications

[1]http://kevoree.org/polymer/
[2]http://kevoree.org/

have to be used in a run-time context where dynamic parameters adjustment is required such as running vehicles guidance.

The usage of MOEAs in run-time optimization problems [Jen03], for instance load balancing problems, which can be seen as a subset of scheduling problems, motivates the need to improve their performance. Several studies have explored the computational costs of MOEAs [KYD03, VVL00] by evaluating their performance on different problems using various MOEAs categories. According to [Jin05], MOEAs computational costs can be reduced by reducing their algorithmic complexity or the computational costs of the fitness function. Several studies have focused on computational costs of fitness evaluations and have proposed models to reduce the costs of such evaluation. In [BSK], the authors have tested complex fitness functions, and have proposed the concept of surrogate models to reduce their computational cost. Their approach is based on a gaussian optimization model that evaluates previous fitness functions to estimate future fitness functions scores instead of evaluating real fitness functions. In [TLK01], the authors highlight the impact of large populations on the computation time of the Pareto front [ISTN09, IND06]. In [KYD03], the authors highlight the efficiency loss and overhead introduced by a number of objectives above 3. All of these studies highlight MOEAs performance drawbacks and propose specific solutions to improve MOEAs. In this chapter, we focus on the notion of hyper-heuristic to propose a solution with an impact on the algorithm efficiency.

5.2.2 Hyper-heuristics: classification and objectives

In search based engineering, hyper-heuristics [BHK+10] define methods that act on adapting search parameters over the search process. Hyper-heuristics introduce modifications on the algorithm itself, in order to improve its computational efficiency or effectiveness to handle a specific purpose [SOTJ12]. Hyper-heuristics rely on machine learning mechanisms [BHK+10], to leverage knowledge assessed during each search iteration. For example, in [Hak10], the authors have proposed learning methods to store neighborhood information through a neural network to improve a genetic algorithm accuracy. Hyper-heuristics can be classified according to the following taxonomy [BHK+10] depending on the nature of the heuristic used (based on selection or generation methodologies). For each nature, we thus differentiate : (i) *Online learning* hyper-heuristics which learn while the search algorithm is running, (ii) *Offline learning* hyper-heuristics which learn from the system before the execution of the search process. The contribution of this chapter, falls into the category of hyper-heuristics that embed online learning mechanisms, which are based on both selection and generation methodologies to achieve performance improvement over a set of software engineering problems [GKÖ13, ÖBK08]. Indeed, our approach relies on mutation operator prioritization (selection nature) based on the evaluation of past execution efficiency (generation nature).

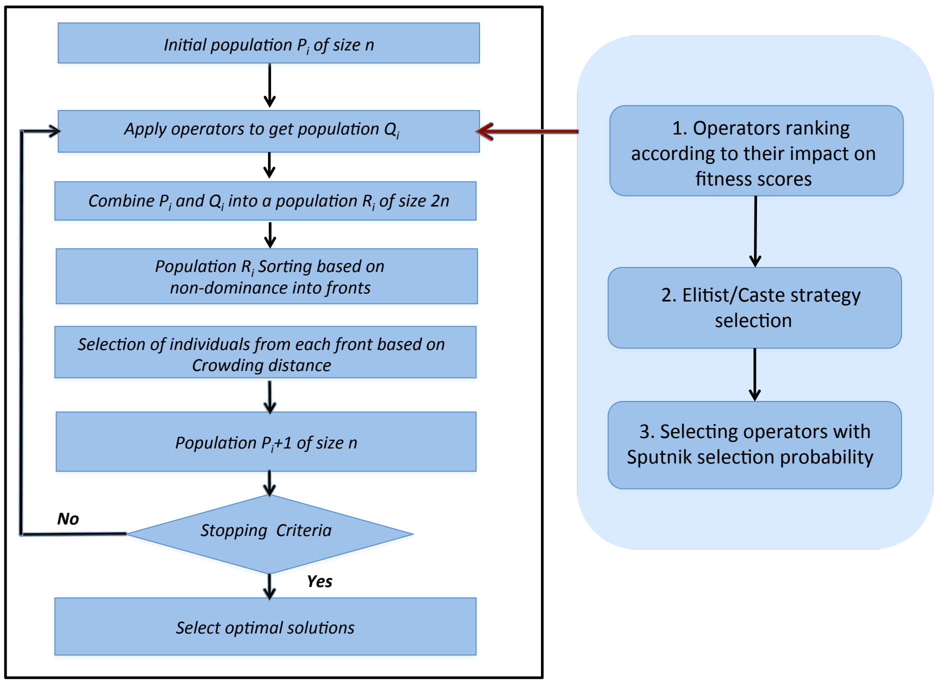

Initial population P_i of size n
Apply operators to get population Q_i
Combine P_i and Q_i into a population R_i of size 2n
Population R_i Sorting based on non-dominance into fronts
Selection of individuals from each front based on Crowding distance
Population P_i+1 of size n
Stopping Criteria
No
Yes
Select optimal solutions
1. Operators ranking according to their impact on fitness scores
2. Elitist/Caste strategy selection
3. Selecting operators with Sputnik selection probability

Figure 5.1: *Sputnik* workflow

5.3 *Sputnik*: a hyper-heuristic for an efficient self-adaptive systems optimization

In what follows, we present *Sputnik* hyper-heuristic and we compare it with existing approaches that aim at enhancing MOEAs efficiency.

5.3.1 *Sputnik* approach

The randomness introduced by natural selection of evolutionary approaches leads to suboptimal performance in terms of computational power and memory usage. Random selection of mutation operators produces useless candidate solutions that lead to computational resources wastage and therefore does not meet run-time optimization constraints. In modern biological studies, after identifying a gene impact on an individual phenotype trait, scientists like Muller *et al.* [PLA27] leverage artificial mutation to directly produce an individual combining the foreseen modification.

Our hypothesis is that the artificial mutation concept can be introduced in evolutionary algorithms to mimic modern biological genetics, thus reducing the number of required generations to reach acceptable solutions. The *a priori* scientific knowledge of a gene modification impact, could be replaced by a continuous ranking and learning approach leveraging execution history. Therefore, in this chapter we aim at optimizing MOEAs, by dynamically reducing the usage of mutation operators that are less effective in improving fitness functions scores. At the same time,

$$\begin{aligned} selection_{op} : operators \times generation \times objectives &\longrightarrow operator \times probability \\ (op_1, op_2, ..., op_m),\ g_l,\ (f_1, f_2, ..., f_n) &\longmapsto op_{max(\triangle_{impact} f_{g_l,op})} \times P_{selection} \end{aligned}$$

Figure 5.2: *Sputnik* selection function

we maintain the equity of natural selection, to ensure that the modified evolution algorithm is able to reach any solution. Thus, we replace the random mutation operator selection by a hyper-heuristic that detects for each individual the most pertinent operator to apply in order to achieve a faster trade-off.

After each application, mutation operators are classified according to the impact they introduce on each fitness function. Internally, *Sputnik* maintains an elitist group of mutation operators that are relevant to improve a fitness function score. To enhance operators selection, *Sputnik*, considers the current fitness scores reached by a solution, and selects the most relevant mutator in elite groups to improve the next generation[3].

As illustrated in Figure 5.1, *Sputnik* on top of NSGA-II [VVL00] takes as inputs an initial population and a generation number. Starting from an initial population of size n, an offspring population is generated using mutation and crossover operators. The original population and the offspring are merged. The resulting population of size 2n is ranked based on non-dominance. The ranking results in the creation of several fronts. The first front contains non-dominated solutions. The second front contains non-dominated solutions for all individuals in the population except those that exist in the first rank, etc. Based on this ranking, an individual is selected from each front based on the crowding distance until we reach a population of size n. The crowding distance is used as a selection criterion that enables to select individuals in the regions that are less crowded to maintain population diversity. The process is repeated until a stopping criterion is reached.

Sputnik introduces a favoritism operator approach in the mutation process described as follows: we consider a multi-objective evolutionary optimization of f with n objectives $(f_1, f_2, ..., f_n)$. The average fitness score for a generation g_l is defined by $\sum_{i=1}^{n} f_i/n$ for each individual in the generation. We define $\triangle_{impact} f_{g_l,op}$ as the fitness score variation between the average of fitness function evaluation for a generation g_{l-1} and a generation g_l that is achieved by the operator *op*: $\triangle_{impact} f_{g_l,op}$=$(\sum_{i=1}^{n} f_i/n)_{g_l,op}$-$(\sum_{i=1}^{n} f_i/n)_{g_{l-1},op}$.

Sputnik records the *selection occurrence* for the different mutation operators that have been involved in the search process. Once all mutation operators have been selected at least once, $\triangle_{impact} f_{g_l,op}$ is evaluated for all the operators and *Sputnik* is configured to select the operators that have $\triangle_{impact} f_{g_l,op} > 0$ in the generation g_{l+1} with the Elitist or the Caste strategies. More formally, *Sputnik* selection function $selection_{op}$ is specified in Figure 5.2:

[3] http://www.genetics.org/content/111/1/147.short

$P_{selection}$ depends on *Sputnik* strategy and is evaluated as follows:

- *Elitist Strategy*: The operator that has the highest $\triangle_{impact} f_{g_l,op}$ is selected in the generation g_{l+1} with a high $P_{selection}$. This configuration accords higher chance to the "winner operator" to be selected in the next generation. All others operators are selected with a probability 1-$P_{selection}$.

- *Caste Strategy*: A selection probability is partitioned between operators which have $\triangle_{impact} f_{g_l,op} \leq 0$ and is defined as follows:

  $P_{selection} = \triangle_{impact} f_{g_l,op} \big/ \sum\limits_{op \in operators} \triangle_{impact} f_{g_l,op}$.

  This configuration gives more equity in terms of selection probability for all operators which have a positive impact on a fitness score.

Sputnik-based mutation operators selection is described in Algorithm 5. In both settings, we set a selection probability of 10% for pure random selection of operators to not discriminate worst ranked operators. The random selection of operators mimics the natural evolution and aims at giving equitable chances to all solutions, and to any potential mutation operator. This random operators selection ensures a proper exploration of the domain and prevents the solutions to fall into a local minimum. *Sputnik* keeps 10% of mutation to give chance to less selected operators to be reintroduced in the elite group of a fitness function. Through this mechanism, we keep a minimal equity of species while conserving 90% of the mutation operators set for an efficient mutation.

Algorithm 5 *Sputnik operators selection*

```
Input: Population $P$, Generation Number $g$, Operators $Op_{set}$, List of operators operator-used=$\emptyset$,
boolean $sputnik\_active$=false
Output: Population $P$
  Apply randomly a mutation operator $Op_{current}$ on $P$ to get $P_{new}$
  for all j where j ranges from 1 to g do
    Evaluate $f_i(x)$ on $P$
    operator-used:=operator-used $\cup$ $Op_{current}$
    /* Sputnik is active only if all the mutation operators have been at least chosen once */
    if operator-used $\subseteq Op_{set}$ then
      $sputnik\_active$=true
      Evaluate $\triangle_{impact} f_{g_i,op}$ $\forall$ $op$ in $Op_{set}$
      Select $P_{selection} \in \{P_{elitist}, P_{cast}\}$
      Identify $Op_{best}$
      Apply $Op_{best}$ with $P_{selection}$, $Op$ with 1-$P_{selection}$ from $Op_{set}$ and get $P_{new}$
    else
      Select $Op$ randomly from $Op_{set}$ and get $P_{new}$
    end if
  end for
```

5.3.2 Comparison with other approaches

Several approaches have leveraged hyper-heuristics to improve MOEAs efficiency. In [SOTJ12], the authors embed learning techniques in classical MOEAs. They assume that objective functions are expensive to compute so they rank the Pareto front elements and they evaluate only the individuals that have higher ranks. In [LMS09], the authors have proposed a hyper-heuristic that relies on the hypervolume calculation to improve computational results. These approaches consider only the Pareto front set evaluation to improve MOEAs, whereas our approach evaluates operators contribution in improving fitness and thus injects mutation operators that are eligible to make MOEAs converge faster. In [YG04], the authors have shown that racing algorithms can be used to reduce the computational resources that result from using evolutionary algorithms in large scale experimental studies. The proposed approach automates solutions selection and discards solutions that do not introduce results improvement. Whereas racing techniques eliminate worst solutions candidates to speed up the search, in our approach we keep considering worst ranked candidates to maintain operators diversity. In [DÖB11], the authors have explored the advantages of using a controlled crossover on top of single-point search based hyper-heuristics. They maintain the best solutions obtained during the search and update crossover operator accordingly. They rely on a process focused on crossover as a biological selective breeding. This breeding assumes that fittest genes are already present in the initial population. In [SSL14], the authors have used hyper-heuristics and fuzzy logics to adapt parameters values during the search to control diversity. In [MGE13], the authors have proposed a selector that chooses between the following evolutionary algorithms: NSGA-II, SPEA2 and MOGA. At each step the algorithm that is chosen is the one that better contributes to the improvement of the system solution. In [ZHO+14], the authors have conducted an experimental study in which they have compared several hyper-heuristics and meta-heuristics on Hill Climbing, Simulated Annealing and NSGA-II. They come up to the conclusion that it is efficient to combine NSGA-II with other hyper-heuristics. They have also pointed out that the results obtained differ from the domain specific problem data. To the best of our knowledge, our approach is distinguishable by proposing an adaptive selector operator with two configurations (Elitist and Caste) experimentally validated in different MOEAs configurations as it will be shown in next section.

5.4 Validation

To evaluate *Sputnik*, we consider an experimental setting that is similar to the one defined in chapter 4. The scenarios consists in a problem of software components placement in virtual machines. As mentioned in 4, a cloud configuration is an architecture model which leverages virtual machines and components (i.e, provisioned software in our context) concepts. Based on our architectural model, an *Individual* represents a solution vector X that corresponds to a cloud infrastructure model. A *gene* corresponds to a component, a virtual machine or a physical machine

Table 5.1: Operators definition

Operators	**Description**
AddVMMutator(VM_i,PaaS)	Creates a Virtual node VM_i on the top of a PaaS
AddSoftwareMutator(S,VM_i,PaaS)	Creates a component S in the Virtual node VM_i
CloneNodeMutator(VM_i,PaaS)	Creates a clone of VM_i on the top of PaaS
RemoveNodeMutator(VM_i,PaaS)	Removes a Virtual node VM_i on the top of a PaaS
RemoveSoftwareMutator(S,PaaS)	Removes a software component S from the PaaS
AddSmartMutator(S,PaaS)	Adds a software component S from the Virtual Node that contains the least number of components
RemoveSmartMutator(S,PaaS)	Removes a component S from the Virtual Node that contains the largest number of components
SwitchOperator(S_1,VM_1,S_2,VM_2,PaaS)	Switches the component S_1 from the Virtual node VM_1 to VM_2 and switches S_2 from the VM_2 to VM_1

in our model. A *population* corresponds to a set of cloud infrastructure models. A *genetic mutation operator* corresponds to an elementary flip in the model that is introduced by an elementary operation. In the experimental set-up, we consider the following objectives:

- $f_1(x)$= Cost(x): Denotes virtual machines cost which is proportional to the number of active VMs on the top of a PaaS.
- $f_2(x)$= Isolation(x): This function is incremented by 1 whenever two components from different cloud customers share the same virtual machine. To achieve isolation, a cloud provider aims at hosting software components belonging to different customers workloads in different virtual machines.
- $f_3(x)$= Similarity(x): The similarity function quantifies the similarity between the components hosted in virtual machines to assess software diversity. Software diversity [BMM+] is an indicator of potential cascading failure to quantify cloud fault tolerance capabilities.
- $f_4(x)$= Redundancy(x): The redundancy function provides a score based on redundant software (*i.e.,* number of replicates of the same service).

All fitness values have been normalized to range in the interval [0,1]. Table 5.1 presents our set of operators O including seven mutation operators and one crossover (SwitchOperator) operator. As defined in chapter 4, software deployment in the cloud is a multi-objective optimization problem that aims at finding a cloud configuration $co \in CO$ such as $\min_{co} F(X)$. We have implemented an optimization prototype in the Polymer framework, which leverages a model based encoding to perform MOEAs optimization. Our validation aims at evaluating the performance improvement achieved by *Sputnik* hyper-heuristic, in terms of efficiency and effectiveness to solve the software deployment optimization problem.

5.4.1 Research questions

This validation section aims at exploring *Sputnik* efficiency to improve MOEAs convergence speed to achieve a certain level of trade-off between several objectives. Thus, we compare the efficiency achieved with and without *Sputnik* hyper-heuristic. We also explore the effectiveness of *Sputnik* once embedded in most popular MOEAs algorithms such as ϵ-MOEA and NSGA-II. The different features of these algorithms are summarized in Table 5.2.

As a metric to compare the solutions of the different algorithms under study, we choose the hypervolume [ZBT07] metric as Pareto-front quality indicator that defines the total size of space dominated by the solutions in the Pareto-front. Our validation steps are summarized in the following research questions:

- RQ_1 : ***Sputnik* efficiency:** 1) Considering that an acceptable trade-off is 90% of the best solutions, is the Darwin *Sputnik* operator selection strategy successful to reduce the number of generations to reach the defined acceptable trade-off compared to a classical random strategy? 2) What is the gain in terms of execution time of *Sputnik* compared to MOEAs that are configured without *Sputnik*? 3) How does *Sputnik* perform with different probability selection values? 4) What is *Sputnik* impact on the different objectives functions that have been chosen?

- RQ_2 : ***Sputnik* effectiveness:** Does *Sputnik* produce comparable results in terms of trade-offs achieved compared to classical random mutation selection even with modifying the equity of operators selection?

- RQ_3 : **Generalization:** What are the applicability limits of *Sputnik*? How does *Sputnik* behave with different objectives? How does it behave with the different variants of MOEAs?

5.4.2 Experimental results

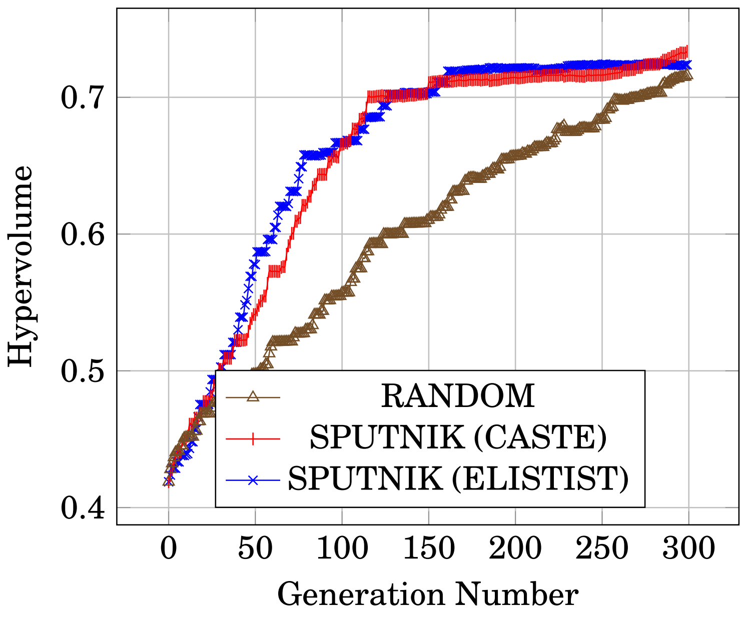
To answer RQ_1, we have embedded *Sputnik* in our cloud optimization engine that manages 100 virtual nodes and maintains a web front-end and a load-balancer software components that have to be dispatched in the different virtual machines. Our cloud reasoning engine is configured to leverage an ϵ-NSGA-II [D$^+$01]. We perform 30 runs of our experiments with five populations and 300 generations using the following configurations: *Sputnik* with Caste Strategy, *Sputnik* with Elitist Strategy and finally with random operators selection. The average hypervolume values of the results obtained in the 30 runs, are depicted in Figure 5.3 according to the generation number and in Figure 5.4.a according to the elapsed time.

We consider that a solution achieves an acceptable trade-off if it reaches 90% of the best obtained solution. In our case study, the best obtained solution achieves an hypervolume of 0.79 (acceptable hypervolume value is 0.71 in our case), it has been reached with a 250 generations previous run. A run with *Sputnik* (with both strategies) reaches this value after 176 generations

	NSGA-II [DPAM02]	ϵ-**NSGA-II** [KR06]	ϵ-**MOEA** [DMM03]	**SMS-EMOA** [BNE07]
Crowding distance	x	x	x	x
Non-dominance ranking	x	x	x	x
Adaptive population sizing		x	x	
Self-termination		x	x	
ϵ-dominance		x	x	
Archive-based		x	x	
Steady state			x	x
Description	The parent and the offspring populations are combined and the ranking is achieved based on non-dominance and crowding distance	The search evolves by maintaining an archive that is updated with non-dominated solutions. Solutions from the archive are used to double the population size at each run. The search stops if the stopping criterion is reached or if the population size does not improve non dominated solutions identified in previous generations.	An archive that contains Non-dominated solutions is maintained as the search evolves. At each generation, two offspring are created from the individuals that are in the archive and in the population. The two offspring are used to update the parent population and the archive. The population is updated based on concept of non-dominance whereas the archive is updated based on the concept of ϵ dominance.	Differs from NSGA-II by the fact that one individual is created and one is discarded at each iteration. The individual that is discarded at each step is the one that contributes less to the hyper-volume indicator.

Table 5.2: MOEAs algorithms main features

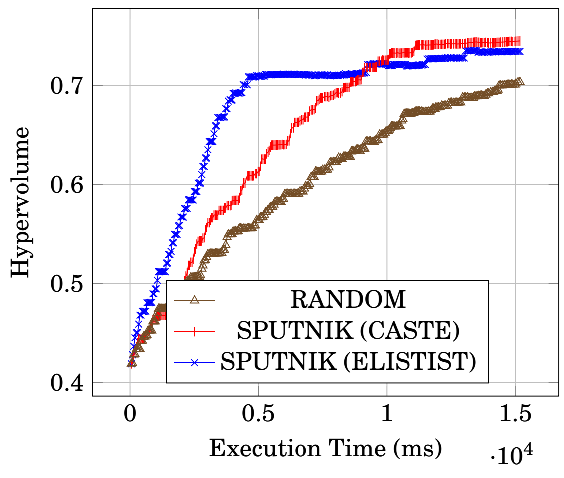
Figure 5.3: Hypervolume: *Sputnik* (Elitist & Caste) versus random selection



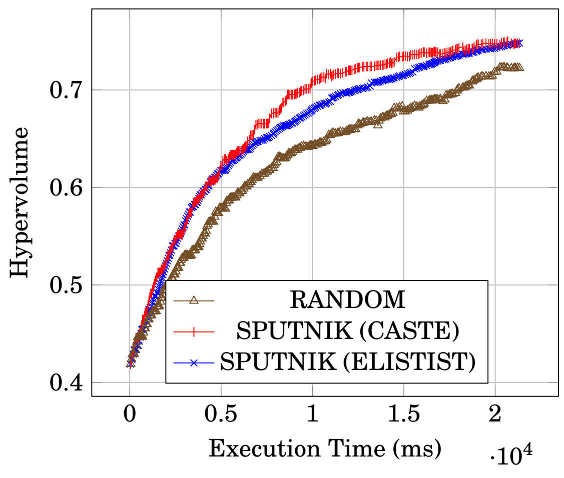
whereas a run with random operators selector reaches this value after 279 generations, respectively 8s for *Sputnik* and 16s for the random selection. *Sputnik* strategies are very similar in terms of hypervolume achievements, they both reach a value around 0.76. We notice that elitist strategy converges slightly faster however, the caste strategy can reach better hypervolume scores. These results can be justified by mutators diversity introduced by the caste strategy, which favors at a certain extent operators equity. In average, *Sputnik* (both with caste and elitist configurations) outperforms random selection by reducing around 37% the number of necessary generations, and around 50% the time to reach acceptable solutions. This confirms our first hypothesis, which states that a smart mutation selection strategy is successful to improve efficiency to reach acceptable trade-offs compared to a classical random selection strategy.

To explore the impact of the selection probability of the elitist strategy on *Sputnik* efficiency, we run the same previous experiment with two different probability values of $P_{selection}$. The results of a selection probability of 90% and 50% are shown in Figure 5.4.a and Figure 5.4.b. Unsurprisingly, we observe that the more *Sputnik* uses its learning strategy, the best is the convergence speed comparing to a random selection.

We also have evaluated the values reached by the different objectives of our case study (Cost, Redundancy, Similarity, Isolation). The results are illustrated in Figure 5.5. The chart presents the cost per hour and SLA satisfaction percentage reached for the Redundancy, Similarity, Isolation objectives. Note that the values obtained for the mono-objective optimization correspond to distinct runs in which we aimed at optimizing one objective at once on the detriment of other objectives. For our minimization multi-objective optimization problem, *Sputnik* achieves three better objectives values in 400 generations compared to a standard NSGA-II: For the different



(a) Probability of selection equal to 90%



(b) Probability of selection equal to 50%

Figure 5.4: NSGA-II hypervolume with different probabilities of *Sputnik* selector
The figures present the hypervolume obtained with two different probabilities: The figure in the left side is configured with a probability of selection equal to 90% and the second with a probability of *Sputnik* selector equal to 50%

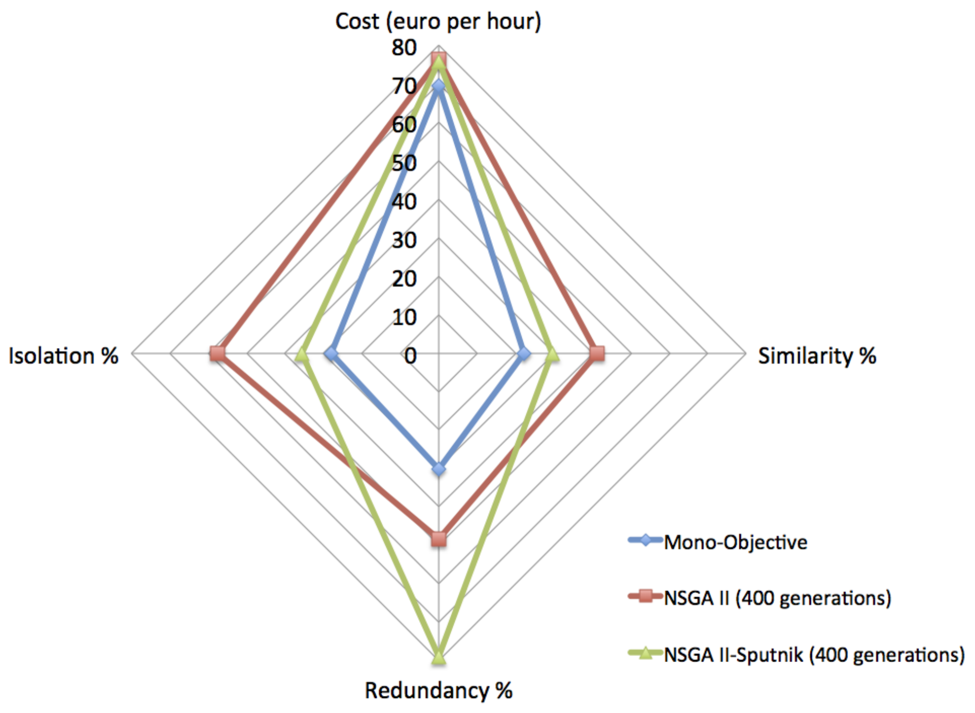
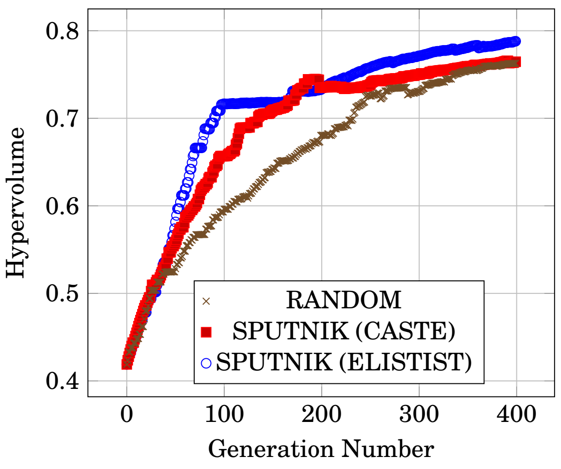
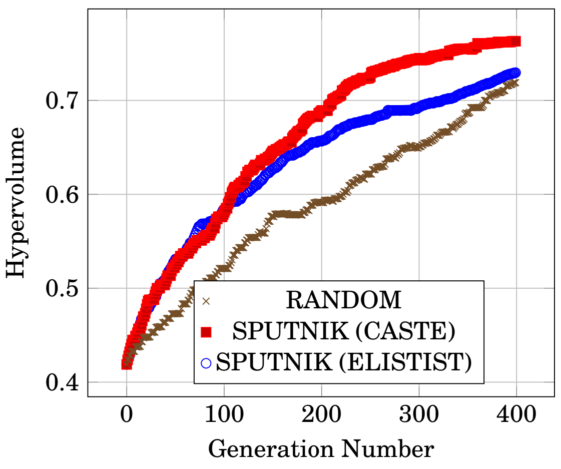


Figure 5.5: Objectives chart radar

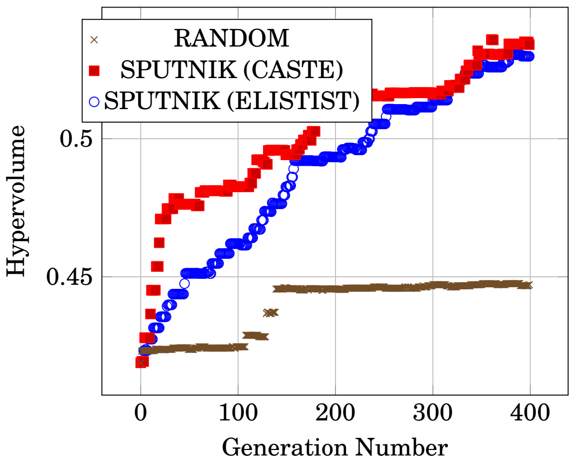
objectives (Cost, Similarity, Redundancy, Isolation), a possible solution presents the following values (75.96, 29.41, 78.94, 35.57) compared to (76.41, 41.26, 48.38, 57.54) for standard NSGA-II. We conclude that for the same number of generations, we obtain results that exhibit better



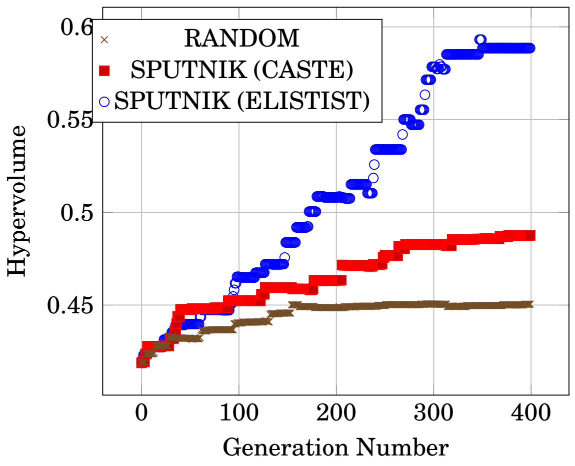
(a) NSGA-II



(b) ϵ-NSGAII



(c) ϵ-MOEA



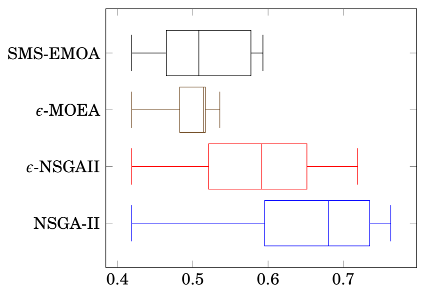
(d) SMS-EMOA

Figure 5.6: Hypervolume over 400 generations

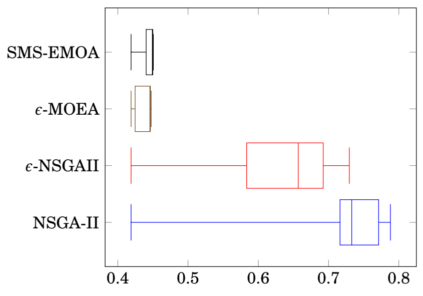
trade-offs with *Sputnik* activated on top of NSGA-II.

To answer RQ_2, we run a similar experiment with both random and *Sputnik* selector until we reach an unchanged value of hypervolume over 50 generations. Final values for *Sputnik* (elitist: 0.77, caste: 0.81 and random 0.78 allow us to conclude that our hyper-heuristic does not decrease the quality of the results in terms of degree of trade-off achieved. Moreover the caste strategy improves the hypervolume score.

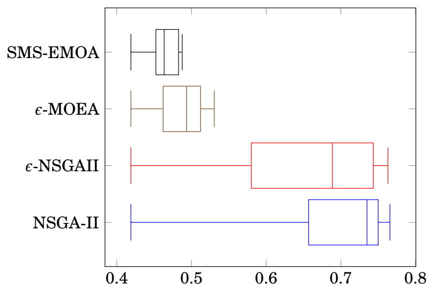
To answer RQ_3, we have evaluated *Sputnik* with different MOEAs algorithms and various number of objectives. Given that several factors (i.e., implementation aspects, existence of other processes running in the machine, etc.) may influence execution time of our approach, we have compared the hypervolume reached over 400 generations. The results are shown in Figure 5.6 with hypervolume distribution in Figure 5.7. For NSGA-II algorithm, an hypervolume value of 0.7 is reached after 90 generations with Elitist strategy, 120 generations with Caste strategy, and reached after 240 generations with random strategy. A *Sputnik* on top of ϵ-NSGA-II for



(a) Random



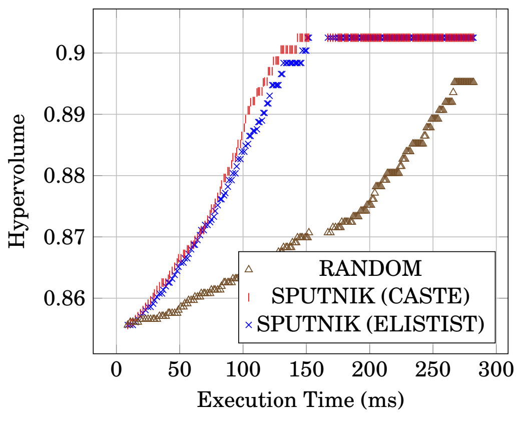
(b) SPUTNIK (ELITIST)



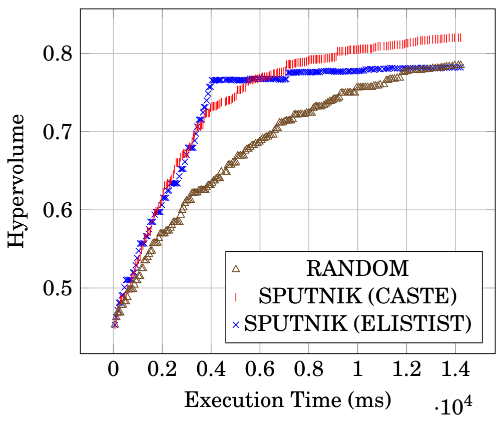
(c) SPUTNIK (CASTE)

Figure 5.7: NSGA-II, ϵ-NSGAII, ϵ-MOEA, SMS-EMOA box plots
Statistical distribution for the hypervolume over 400 generations shown above: Max values reached with *Sputnik* with its two settings outperform max values reached with plain NSGA-II and ϵ-NSGA-II. ϵ-MOEA, SMS-EMOA have weaker results due to their selection strategy (one mutation per generation) that slows down *Sputnik*.

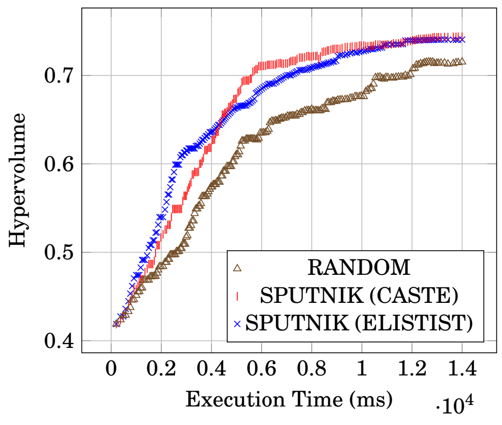
almost both the Caste and the Elitist versions achieves an hypervolume of 0.6 in 100 generations. Similarly to NSGA-II-based approaches, we observe a similar speed up for SMS-MOEA and ϵ-MOEA algorithms. Above 300 generations for NSGA-II and ϵ-NSGA-II, we also observe that the Elitist strategy could lead to little decreased effectiveness. This result could be explained by the impact introduced by Elitist strategy on diversity. We conclude that the Caste strategy is less intrusive hyper-heuristic which maintains better the algorithm effectiveness. From these runs, we notice that the *Sputnik* hyper-heuristic can be generalized on several MOEAs. Secondly, we have explored the generalization of *Sputnik* with variable objectives by analyzing the hypervolume while varying the objectives from one to four. The results, generated with NSGA-II for 200 generations are presented in Figure 5.8 in terms of optimization time. We observe that *Sputnik* with the Caste and Elitist strategies provides faster hypervolume convergence compared to random with smaller number of objectives. Indeed for one and three objectives, the *Sputnik* strategy selects efficient operators faster, however above four objectives, *Sputnik* effect tends to be similar to random selection. We explain such effect by the fact that *Sputnik* does not keep the history of previous selected operator selection, thus above four objectives, the selection tends to be random. In future work, we plan to add operators selection history to maintain *Sputnik* efficiency independently of the objectives number.



(a) 1 objective



(b) 3 objectives



(c) 4 objectives

Figure 5.8: NSGA-II hypervolume over execution time with variable objectives

5.4.3 Threats to validity

This section discusses the threats of validity of our approach.

Internal validity is related to the parameters setting of our experiments (i.e., generation number, objectives number, population size, operators, etc.), such as values chosen for the value of ϵ in the ϵ-dominance which might impact our validation results. More specifically, *Sputnik* incrementally builds a mapping of each operator impact on a particular fitness function. Thus, the coupling effect between the eight used operators and a particular fitness function could introduce a bias on our results.

Construct validity arises from the bias introduced in the way we build our experiments. In each experiment presented in the chapter, we have compared one run of *Sputnik* against a random mutator selector. As for any random based techniques, a set of repeated experiments should be run to draw statistically significant results. This bias is mitigated by the number of different experiments that we run on *Sputnik* which all demonstrate the effectiveness of the approach.

External validity is related to the generalization of observed results with other MOEAs algorithms.

5.5 Conclusion

In this chapter, we have introduced *Sputnik*, a hyper-heuristic breaking the random natural selection of classical MOEAs to leverage an elitist artificial mutation inspired by biological studies [PLA27].

Sputnik relies on a mutation operator selection based on a continuous learning of past effect on fitness functions, instead of random mutation operators selection. The overall goal of *Sputnik* is to enhance the optimization algorithm itself, and to guide the search towards faster trade-offs achievements to finally save generation cycles and time. Experimentally, we provide evidence of the effectiveness of artificial mutation to reduce significantly the number of necessary generations to find acceptable trade-offs. Unlike cited approaches in section 8.2, *Sputnik* focuses on artificial mutation selection, therefore mimicking the process used to produce genetically modified organisms. As far as we know, there is no other hyper-heuristic that proposes artificial mutation at mutation operators level.

Sputnik has experimentally demonstrated that hyper-heuristics are efficient candidate for improving run-time optimization of cloud-based software deployment optimization. More experiments are needed to explore the overall optimization life cycle of the cloud adaptive infrastructures. We need basically to perform intercession of the optimized configurations at the level of the real cloud system. We also need to compare te current real system snapshot with the system snapshot that has been used in the optimization. This enables to verify the validity of the calculated optimized snapshots for the evolving real cloud system. Next chapters focus on the design of guest applications to explore cloud-based software trade-offs.

Chapter 6

Towards a full support of obligations in XACML

In the two previous chapters, we have proposed an approach to deploy efficiently cloud-based software. Our described approach considers deployed software as black boxes. In what follows, we focus on the design of guest applications that can be deployed in a cloud infrastructure to improve cloud-based software trade-offs. As mentioned in chapter 1, we consider guest applications that are built upon XACML-based systems design. XACML enables to build applications that meet security and flexibility requirements. However, while rights are well captured by authorizations, duties, also called obligations, are not well managed by XACML architecture. The current version of XACML lacks (1) well-defined syntax to express obligations and (2) an unified model to handle decision making w.r.t. obligation states and the history of obligations fulfilment/violation. A fine-grained obligations support is required to handle authorizations scenarios in which actions have to be carried out before/during/after access to resources. In this chapter, we propose an extension of the XACML reference model that integrates obligation states in the decision making process. We extended the XACML language and architecture for a better obligation support and have shown how obligations are managed in our proposed extended XACML architecture: *OB-XACML*.

A major part of this chapter has been published in the following conference proceedings: ***Donia El Kateb****, Yehia ElRakaiby, Tejeddine Mouelhi, Iram Rubab and Yves Le Traon. Towards a Full Support of Obligations In XACML. In Proceedings of the 9th International Conference on Risks and Security of Internet and Systems (CRiSIS), pp 213-221, 2014.*

Contents

6.1 Introduction

Cloud customers are still suffering from the lack of standard APIs in terms of interfaces and applications connectors exposing customers to vendor lock-in risks. To ensure high interoperability and portability between heterogeneous cloud platforms, standardization initiatives have to be developed. XACML proposes a standardized approach to handle access control resources. This makes it suitable to handle permissions management for heterogeneous cloud-based applications. Obligations extend binary authorizations to express fine-grained actions. These actions are required to handle usage control requirements.

To meet the challenges of reinforcing XACML standard to handle obligations, we enhance the support of obligations in XACML policy language and its underlying architecture while maintaining the highly-desirable separation of concerns between the system and its policy. Going along the same line than Bertino for XACML 2.0 [LCB12], who pioneered the extension of XACML for usage control, we propose 1) Well-defined XML constructs that are compliant with XACML 3.0 to specify obligations. 2) *OB-XACML*: An underlying architecture that extends the current XACML architecture and that is able to take into consideration the history of obligations fulfillment/violation and obligation states at the level of the decision making process. *OB-XACML* introduces an interaction schema between the different key entities in XACML architecture to keep track of obligations fulfillment/violation related to the users in the system.

To the best of our knowledge, our work is a first initiative that considers obligation states as a key element that must be considered at the access control evaluation time. This chapter is organized as follows. Section 6.2 introduces obligations in XACML and gives an overview about existing approaches that tackled obligations support in XACML. Section 6.3 describes our extended architecture *OB-XACML*. Sections 6.4 describes the constructs of XACML syntax to support obligations. Section 6.5 presents our validating prototype and our supported usage control scenarios. Finally, section 6.6 concludes this chapter.

6.2 Introducing obligations in XACML

This section introduces obligations in XACML and surveys existing approaches that have been proposed in the literature to enhance obligations support in XACML standard.

6.2.1 Obligations handling in XACML

XACML defines obligations as actions that have to be returned to the PEP with the PDP response. The decision-making is not anymore about a deny/access decision only but includes obligations fulfillment/violation information as well. XACML defines three PEP categories based on PDP decision and the ability of the PEP to handle obligations.

- *Base PEP* setting: The PEP decision is permit if the PDP decision is permit and the PEP is notified of the fulfillment of all the obligations returned by the PDP. The PEP decision is also permit if the PDP decision is deny and the PEP is not notified of the fulfillment of all the obligations returned by the PDP. In all other cases, the PEP decision is deny.
- *Deny-biased PEP* setting: The PEP decision is permit if the PDP decision is permit and the PEP is notified of the fulfillment of all the obligations returned by the PDP. In all other cases, the PEP decision is deny.
- *Permit-biased PEP*: The PEP decision is deny if the the PDP decision is deny and the PEP is notified of the fulfillment of all the obligations returned by the PDP. In all other cases, the PEP decision is permit.

```
<Obligation ObligationId=‘‘send−email"FulfillOn=‘‘Deny">
<AttributeAssignment AttributeId=‘‘email">donia.kateb@uni.lu</AttributeAssignment>
</Obligation>
```

Listing 6.1: Obligation example

In [LCB12], Bertino *et al.*, have proposed a synthesized classification of PEPs categories in XACML presented in Table 6.1. The reader may refer to [LCB12] for more details about PEPs classification in XACML. In the reminder of this chapter, we will only consider the *Deny-biased PEP*.

XACML 2.0[1] defines obligations as simple attributes assignment that are attached to the policy set or to the policy. XACML 3.0[2] considers that obligations can also be added to the rules besides the policies and policy sets. An obligation element contains two required elements, which

[1]http://docs.oasis-open.org/xacml/2.0/access control-xacml-2.0-core-spec-os.pdf
[2]http://www.oasis-open.org/committees/xacml/

Table 6.1: PEP categories in XACML [LCB12]

PDP Decision	**Permit, obligations fulfilled**	**Permit, obligations not fulfilled**	**Deny, obligations fulfilled**	**Deny, obligations not fulfilled**
Base PEP	Permit	Deny	Deny	Permit
Deny-Biased PEP	Permit	Deny	Deny	Deny
Permit-Biased PEP	Permit	Permit	Deny	Permit

are the *obligation identifier* and the *FulfillOn* element which specifies the effect on which the obligation should be fulfilled by the PEP. For example, *FulfillOn* “Permit” specifies that the obligation should be enforced if the PEP decides to permit the request. The XML snippet in Listing 6.1 shows an example that illustrates that if the PDP decision is deny, the subject has to send an email to the address “donia.elkateb@uni.lu”.

6.2.2 Existing work on obligations in XACML

In the last few years, several research initiatives have motivated the support of obligations in XACML at the level of the XACML language and architecture. In [MSB+12], the authors have proposed a framework and a supporting language extending XACML to take into consideration UCON features. They have thus added some identifiers to the XACML reference language to support mutability of attributes and the continuity of access feature in UCON model. The work presented in [MCM10] [LMM12] follows the same direction and aims at enriching the XACML model to take into consideration UCON features. The authors have added the identifier in the condition element to distinguish between pre-obligations, ongoing and post-obligations. The element *AttrUpdates* is added to reason about attributes update and an XML retrieval policy has been introduced to specify where the attributes have to be retrieved for update. In [Lis10], the authors have defined a negotiation schema for obligations between the PEP and the PDP and a language to handle obligations: obligation markup language (XOML). They have also presented a classification of the different types of relationships between obligations. In [LCB12], the authors have proposed a language and an underlying architecture to handle obligations. Obligations are commonly defined as application-specific and thus their handling is left to the platform that manages them. The authors have presented a framework to handle obligations in which they have specified and designed an application-dependent module to deal with obligation management. Their proposed obligation schema includes a list of event families that categorize event types interacting with an obligation. It also includes rules that are mapped to obligation status which have to be evaluated according to a given obligation state. In [TN11], the authors have defined

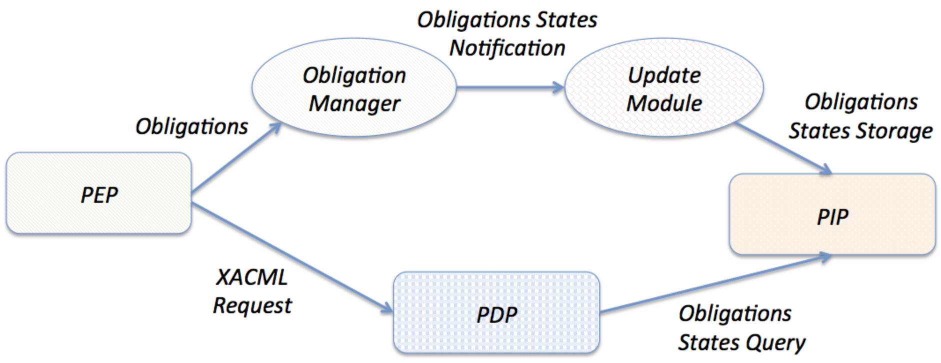


Figure 6.1: OB-XACML workflow

obligations in XACML as an extension of the PPL language which is used to allow the user to specify restrictions on his personal data. These restrictions specify how data controllers can verify data across several domains. To the best of our knowledge, our work is a first initiative that proposes to handle decision making by considering the history of obligation states.

6.3 *OB-XACML* architecture

In an XACML-based architecture, the policy decision point (PDP) is a stateless component. Access control decisions are thus taken without taking into consideration obligations fulfillment or violation in previous accesses. We propose to introduce obligation information violations/fulfillment at the decision making time. In our proposed model, access control decisions are taken based on information related to obligation states or related to previous users obligations fulfillment/violation. Here are some motivating scenarios:

- In a medical health system in which a nurse has to send a report to the patient's treating doctor after each access to the patient's medical data. The report should describe some specific indicators about patient's health status. If the post-obligation of sending a report after the access is violated then the nurse should be prohibited from accessing patient's data in another access and some penalties measures have to be taken against her as a reaction to this non professional behavior.

- A user has the pre-obligation to sign a form before he accesses a web application. If the system keeps track of his fulfilled obligations then the user does not have to sign the form in every session. The system can thus record the fulfillment of the obligation in a first login and then the user can access the system in future sessions without the need to fulfill his pre-obligations.

To support such scenarios, we extend the current XACML architecture so that information inherent from subjects fulfillment/violation of their obligations is taken into consideration at the decision making time. Keeping track of obligations fulfillment/violation states, gives an insight about how users are using the system and enables to prevent future users' accesses when those are violating their obligations. In what follows, we present the components and the exchange of messages that allow access control decisions to be taken according to obligation fulfillment/violation information. To provide a decision making process that takes into consideration obligation states, we propose to store obligation states in the Policy Information Point (PIP) additionally to attributes values such as resource, subject, environment. Such information is retrieved dynamically at the decision making time and used for request evaluation. The PDP sends to the PEP the access decision and the obligations that have to be monitored by the PEP. Each obligation is handled by the obligation manager, which tracks obligation states evolution by monitoring their execution in the system. An obligation life cycle can been modeled as a state machine as it has been presented by Cuppens *et al.* [CCBE13]. An obligation state can be 1) Inactive when the fulfillment of the obligation is not needed, 2) Active when the obligation fulfillment is required, 3) Fulfilled when the obligation is satisfied, 4) Violated when the obligation is violated (In the context of this work, we consider that an obligation becomes violated when it is not fulfilled after a fixed deadline). 5) Fulfilled/violated when the obligation is violated and later it has been fulfilled 6) An obligation is inactive when it ends. The transition between the different states is driven by contexts defined in Cuppens *et al.* [EMT12]. Contexts ($\mathscr{C}$) represent conditions on subjects, actions and resources. An activation context Ctx$_a$ specifies the different conditions under which the obligation has to be fulfill. The violation context Ctx$_v$ specifies the different conditions under which an obligation is violated. Contexts specify conditions on the actions that activate or deactivate them. They are specified using the following propositions:

$$Hold_e(S,A,O,start/end(Ctx)) \text{ after do(S,A,O) if } p_1,...,p_n$$

This means that the context (Ctx) starts or stops to hold after the action A is performed by the subject S on object O, if the conditions specified by the fluents $p_1,...,p_n$ are met, for example the following rule:

$$Hold_e(user,-,-,start(Loggin-session) \text{ after do(user,perform authentication,Login\_Form)}$$

specifies that the logging-session context starts when the user performs authentication on the login form. Thus, an obligation rule in our language is defined like the following:

Obligation(N,SR,A,O,Ctx$_a$,Ctx$_v$) where N is a unique security rule identifier, SR is a subject or a role, A is an action, O is an object, Ctx$_a$ is an activation context and Ctx$_v$ is the violation context. The following obligation specifies that the doctor has to examine the patient at the start of working hours: Obligation(O_1, doctor, examine, patient, start(working_hours),-). Figure 6.2 illustrates the different states transitions between the different states. The obligation to send a report after an access to an administration system is handled as follows by the obligation

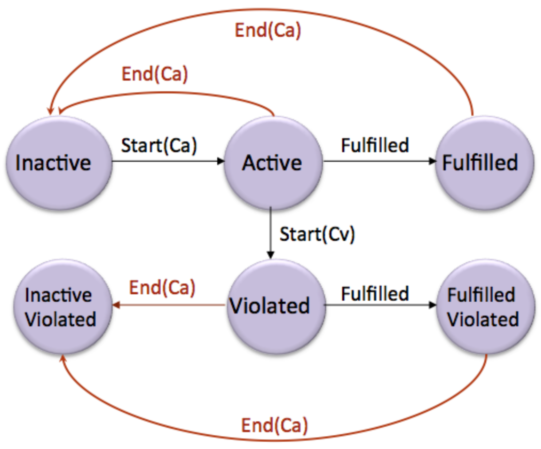


Figure 6.2: Obligations state machine [EMT12]

manager: 1) The user has an access to the platform: The obligation is in an *active* state. 2) The user sends the report to the platform: The obligation is in a *fulfilled* state. 3) The user does not send the report within a given time: The obligation is in a *violated* state.

Our XACML extension is illustrated in Figure 6.1: The user behavior at the system level is monitored through aspects which capture the different users actions [KLM+97]. The update module receives information related to the changes in obligations states and updates the PIP with obligation state attributes that are provided by the obligation manager. We enrich XACML conditions to express access control rules that are conditioned by obligations fulfillment/violation. These conditions will be introduced in the next section. Figure 6.3 illustrates the interactions between the system that interacts with a security policy, the obligation manager, the PEP, and the PIP in *OB-XACML*:

Algorithm 6 - *PEP obligation management*

```
Input: PDP Decision, a set of obligations $O$
Output: PEP Decision
/*PDP decision is provided after all preobligations become in an inactive state*/
for all Preobligations $o_{pre} \in O$ do
    $o_{pre}$.state=active
end for
while $o_{pre}$.state $\neq$ inactive $\forall\ o \in O$ do
    /*Since the PEP is deny-based, all preobligations need to be fulfilled to permit the access if
    PDP decision is permit */
    if $o_{pre}$.state=violated then
        return Deny
    end if
    PIP update with ($o_{pre}$.id, $o_{pre}$.state, $o_{pre}$.subject, $o_{pre}$.update_time)
end while
if PDP Decision $\neq$ Deny then
    return Permit
else
    return Deny
end if
/*PDP decision is revoked if some ongoing obligations are violated*/
for all Ongoing obligations $o_{ongoing} \in O$ do
    $o_{ongoing}$.state=active
end for
while $o_{ongoing}$.state $\neq$ inactive $\forall\ o_{ongoing} \in O$ do
    if $o_{ongoing}$.state=violated then
        PDP Decision = Deny
    end if
    PIP update with ($o_{ongoing}$.id, $o_{ongoing}$.state, $o_{ongoing}$.subject, $o_{ongoing}$.update_time)
end while
/*Postobligations do not impact PEP decision*/
for all Postobligations $o_{post} \in O$ do
    $o_{post}$.state=active
end for
while $o_{post}$.state $\neq$ inactive $\forall\ o_{post} \in O$ do
    PIP update with ($o_{post}$.id, $o_{post}$.state, $o_{post}$.subject, $o_{post}$.update_time)
end while
```

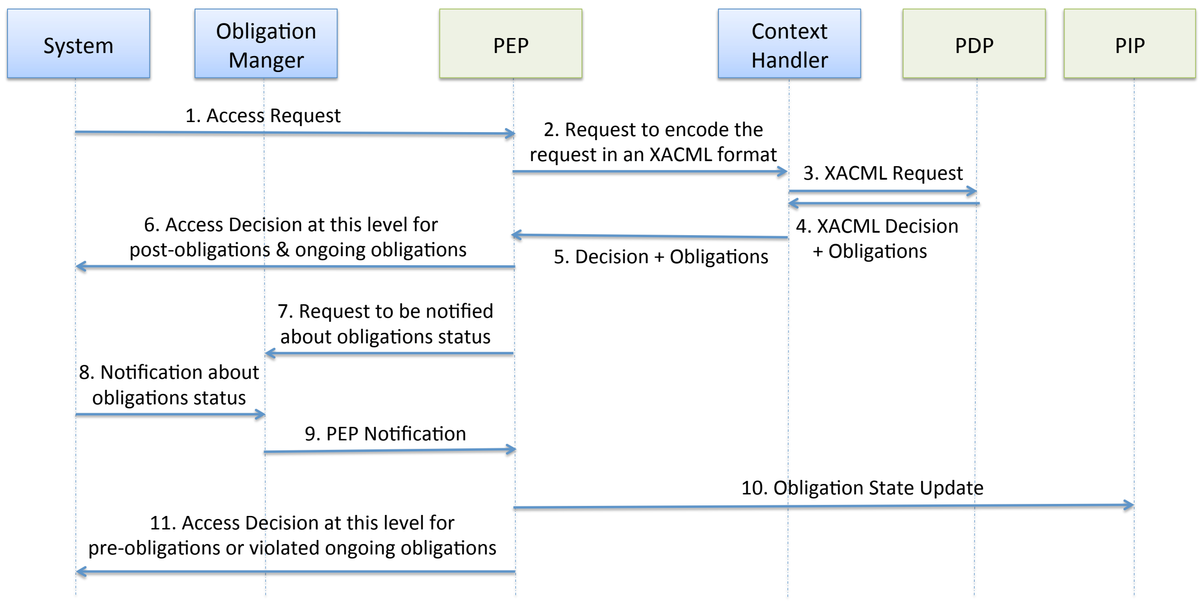

System
Obligation Manger
PEP
Context Handler
PDP
PIP
1. Access Request
2. Request to encode the request in an XACML format
3. XACML Request
4. XACML Decision + Obligations
5. Decision + Obligations
6. Access Decision at this level for post-obligations & ongoing obligations
7. Request to be notified about obligations status
8. Notification about obligations status
9. PEP Notification
10. Obligation State Update
11. Access Decision at this level for pre-obligations or violated ongoing obligations

Figure 6.3: Message exchange in *OB-XACML*

- The PEP receives an access control request from the system that is regulated by an XACML policy and sends a request to the context handler to encode the request in an XACML format.
- The context handler sends a request to the PDP in an XACML format.
- The PDP sends to the context handler an XACML decision and a list of obligations.
- The context handler sends to the PEP the decision and the list of obligations.
- If obligations do not include pre-obligations then the final decision is sent to the system at this level of message exchange.
- The PEP requests the obligation manager for notifications about obligations status.
- The obligation manager is notified about obligation states.
- The PEP is notified whenever there is an obligation state update.
- The PEP updates the PIP with obligations state records.
- An access decision can be provided at this level for pre-obligations. An access decision can also be revoked at this level if ongoing obligations are violated.

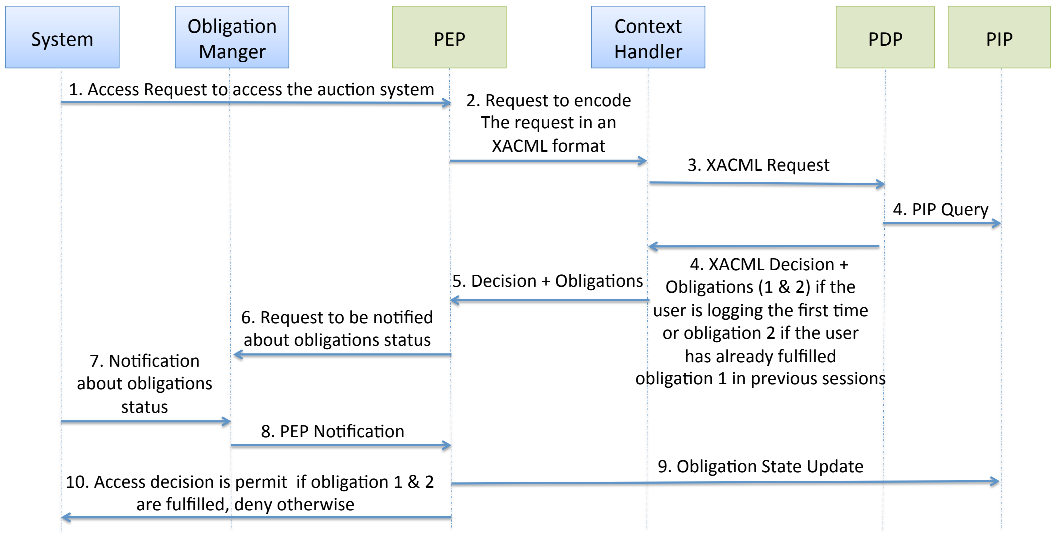

System
Obligation Manger
PEP
Context Handler
PDP
PIP
1. Access Request to access the auction system
2. Request to encode The request in an XACML format
3. XACML Request
4. PIP Query
4. XACML Decision + Obligations (1 & 2) if the user is logging the first time or obligation 2 if the user has already fulfilled obligation 1 in previous sessions
5. Decision + Obligations
6. Request to be notified about obligations status
7. Notification about obligations status
8. PEP Notification
9. Obligation State Update
10. Access decision is permit if obligation 1 & 2 are fulfilled, deny otherwise

Figure 6.4: Pre-obligations sequence diagram

To explain the processing of obligations in *OB-XACML*, we took some illustrative examples from an Auction Management System (AMS). AMS allows users to perform some bidding operations online if they have enough money in their account before starting bidding operations. We consider the following pre-obligations in the AMS system:

1. The user has to accept the usage terms of the auction before joining the auction system.

2. The user has to validate his payment for the session.

Pre-obligation 1 has just to be fulfilled in the first login whereas the pre-obligation 2 has to be fulfilled in every session. Figure 6.4 illustrates message exchange to process the two pre-obligations in AMS. For a given access request, the PEP decision is taken under the assumption that our PEP is a deny-based PEP which means that once the PEP will receive an access decision with obligations, an access is permitted only if the PDP decision is permit and the PEP is able to receive an information about the fulfillment of all the obligations, otherwise the decision is deny. Obligations O include pre-obligations o_{pre}, postobligations o_{post} and/or ongoing obligations $o_{ongoing}$. The algorithm 6 specifies how access control decision are handled by the PEP when the PEP receives an access decision with obligations. Next section describes the XML syntax that we propose to support *OB-XACML*.

6.4 Proposed syntax: obligations in XACML 3.0

This section introduces the obligation syntax that supports *OB-XACML*. We aim through this design to: 1) Present obligations elements (Subject, Object, Action, Context, etc.) in the form of

Table 6.2: Obligation syntax in XACML

Obligation element	Abstract language	XACML syntax
Identifier	N	ObligationId
Subject	SR	subject-category:obligation-subject
Action	AA	action-category:obligation-action
Resource	AA	resource-category:obligation-object
Activation context	Ctx_a	obligation-activation-context
Violation context	Ctx_v	obligation-violation-context

standardized XML. 2) Categorize obligations by distinguishing between pre-obligations, ongoing obligations and post-obligations at the level of the policy language. In XACML 3.0, users are able to extend XACML syntax and to define their own categories, we added new identifiers to define obligation elements in XACML 3.0. To refer to the entity that is responsible of enforcing an obligation, we introduce a new attribute identifier: the obligation-subject encoded as shown in Listing 6.2:

```
<Category=urn: oasis: names: tc: xacml:
    1.0:subject-category:obligation-subject>
```

Listing 6.2: Obligation subject

In the same manner as shown above, action and resource identifiers are added using *action-category:obligation-action* and *resource-category:obligation-object*. We specify obligation activation, deactivation, fulfillment, violation contexts using environment categories as shown in Listing 6.3:

```
<AttributeDesignator Category=urn: oasis: names: tc:xacml:
1.0: environment-category: obligation-activation-context
Type=urn: oasis: names: tc: xacml: 3.0: environment-type: access platform>
```

Listing 6.3: Obligation context

Table 6.2 presents the obligation elements introduced in [ECCB12a] and the identifiers that we have defined in XACML 3.0. We define a new type to distinguish between post-obligations, ongoing-obligations and pre-obligations: “TimingType", which takes the following values: pre, post and ongoing. We also distinguish between the obligations that have to be performed in each session by a given subject and those that have just to be performed by the first login using the identifier “each session" to specify obligations that have to be fulfilled in every access and “first login" to specify obligations that have just to be fulfilled in the first access.

6.5 *OB-XACML* validating scenarios

In this section, we give some examples of usage control scenarios supported by *OB-XACML* and describe the different interactions between its different components.

6.5.1 Description of our architecture

Figure 6.5 illustrates the interactions between the different components in *OB-XACML*, described as follows:

a) AMS application: We consider an Auction Management System (AMS) which is a Java policy-based application. AMS policy is encoded in XACML format and enables to externalize authorization rules from the application business logic. AMS contains 122 classes and 797 methods. The application encapsulates Policy Enforcement Points (PEPs) which represent Java methods that perform permission checking whenever a service regulated by an access control policy is requested.

b) Obligation manager: The obligation manager receives obligations from the PEP and maintains their states. It includes two mapping modules:

- A Mapper from abstract obligations to concrete obligations: This modules translates obligation parameters included in XACML obligations to parameters that are interpreted at the application level. For instance a required action in an obligation is translated to a method call that triggers some functionality at the business level logic, a role is mapped to a user, etc. The mapping between high level elements at the policy level and low level elements at the implementation level is a a key concept that has been detailed in several policy languages [DDLS01b].
- An obligation states monitor: For obligations state monitoring, we define abstract rules that describe the impact of application parameters on obligation states. For example, the obligation to put a starting bid before joining an auction session evolves from an active state to a fulfilled one when the user validates the payment. This requirement is described by the rule $\Re_i$ that describes the operations needed for the obligation "joining an auction" to transition from an active state to a fulfilled state:

  $\Re_i$: State(Obl_1:joining an auction, active) $\mapsto$ State(Obl_1:joining an auction, fulfilled) ***If*** call method(Validate Bid.amount()) && Bid.amount(subject s) returns amount && amount > allowed_minimum_seuil)

  To monitor the different parameters related to obligations state changes, which are defined in our mapping rules, we use aspect oriented programming [KLM+97]. Aspects enable to track that some operations are triggered at the program level and subsequently trigger changes in obligations states. The obligation manager is a Java module that monitors a set of events.

Each obligation is a Java class that extends an abstract class event. Listing 6.4 presents the obligation class of the obligation to send a report to the auction moderator. The report describes the goods that will be involved in the auction.

```
public class SendReportEvent extends Event{
    public SendReportEvent(){
    this.command_name = MeetingSecurityModel.SEND_REPORT_METHOD;}
    @Override
    /**
     * @param args[0] the user who sent the report
     * @param args[1] the auction identifier
     */
    public void notifyCommandExecution(Object args[]) {
        User user = (User) args[0];
        Auction auction = (Auction) args[1];
        //Update the obligations manager when the
        //user sends the report after accessing the auction
        System.out.println("The user " + user.getName() + " has sent a
        report of the auction " + auction.getName());
    }
}
```

Listing 6.4: Obligation Class

c) PIP attributes database: The PIP is a MySQL database that is updated with records describing obligation parameters whenever a change in an obligation state is reported using following form: (Obligation_ID, Obligation_Subject, Obligation_Object, Obligation_Action, Time, Obligation State). This database is queried by the PDP during access requests to fetch information related to the obligations status or related to obligations violation/fulfillment.

d) Update module component: The update module is triggered by aspects in each obligation state change and it updates the PIP with obligation state attributes.

e) Extended PDP: The extension of Sun's XACML implementation with a PDP that supports the new types and the new attributes that we have defined in this work can be done as explained in XACML factory extension[3].

f) Timer: We use a Java timer to define a timer that starts when the activation context starts, the violation context starts when the timer expires and the obligation is not fulfilled.

6.5.2 *OB-XACML* usage control scenarios and performance evaluation

In what follows, we present some motivating examples of some usage control scenarios that can be supported by *OB-XACML* architecture.

- (1) Pre-obligation: A user has to accept the usage terms of the auction before joining the auction session. This obligation is of type "each session".

- (2) Pre-obligation: A user has to fill a form including his personal information. This obligation is of type "first login".

[3]http://sunxacml.sourceforge.net/guide.html

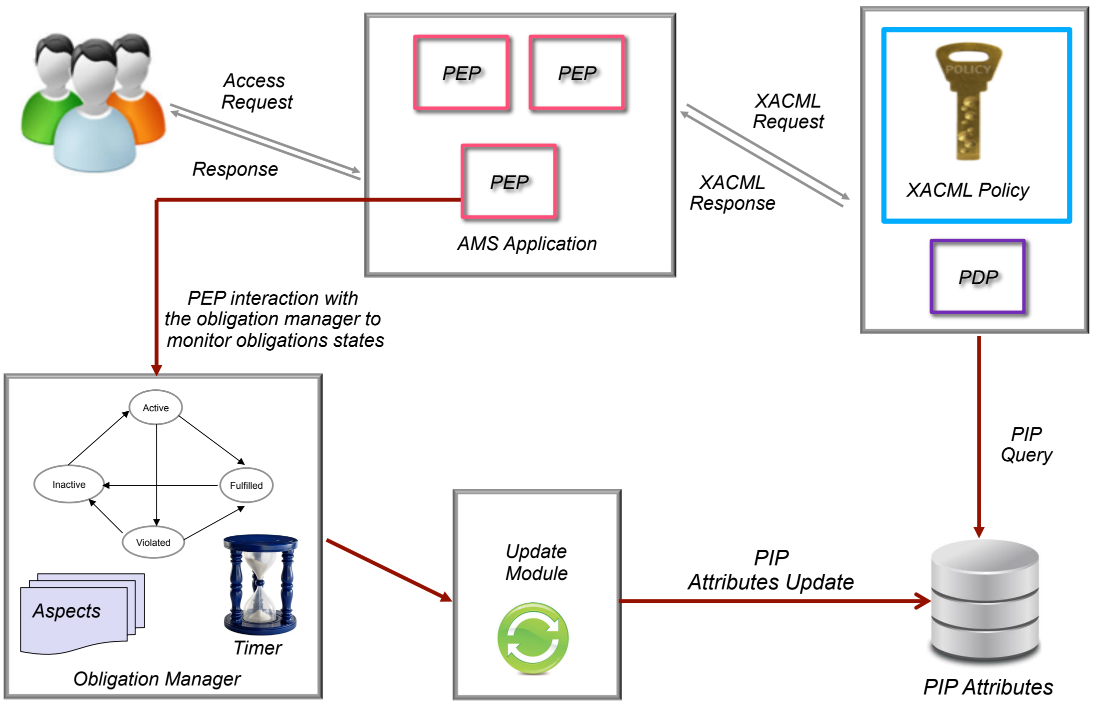


Figure 6.5: *OB-XACML* workflow

- (3) Ongoing-obligation: For each auction, the user has to maintain the window that describes the bids involved in the auction, open, otherwise his access is revoked.
- (4) Post-obligation: A bidding analysis report has to be sent by the moderator accessing the auction to the administrator including all the items that have been used in the auction. If this obligation is violated then the moderator can not access the auction system in future usage.

In section 6.4, we have proposed syntax constructs where the obligation type is specified in the obligation syntax. The PEP behavior will be dependent on the obligation type that is exchanged between the PEP and the PDP at the decision making time. For *pre-obligation 1*, the user joins the auction session only if the obligation to accept the usage terms of the auction is fulfilled. Since this obligation is specified with an obligation type “each session" then the user has to fulfill it in future access requests to join the auction. *Pre-obligation 2* needs to be just fulfilled in the first login. For such type of obligations, the PDP queries the PIP to verify if there is a record that states that this obligation has been fulfilled in previous sessions. If the user has fulfilled this obligation then this obligation will not be returned with the decision to the PEP. Otherwise it will be sent to the PEP and its fulfillment will be mandatory for access. For *Pre-obligation 3*, if the violation context is activated then the PEP revokes the access. For *Pre-obligation 4*, the PIP is updated with attributes related to obligations violation/fulfillment. The policy has a condition that states that access is conditioned by “the obligation to send a report to the administrator has been fulfilled in previous accesses". The PDP checks PIP attributes in future access to check if the obligation has been fulfilled in previous accesses, if the user has tried to log in to the system.

6.6 Conclusion

In this chapter, we proposed a syntax to support obligation polices in XACML and an extension of the standard XACML architecture to take into consideration obligations states and information related to their violation/fulfillment in the decision making process. The changes that we have introduced at the level of XACML architecture do not require to perform many modifications at the level of XACML reference model. This eases the adoption of *OB-XACML* on top of existing XACML-based systems. XACML model implements a flexible access control security model through its component-based architecture. The adoption of XACML as a standardised approach to build applications is strongly related to its capacity to handle a wide range of authorizations scenarios. We strongly believe that introducing obligations states at the level of the decision making time can be used to track users reputation and to update it based on their interactions with the system. Beyond expressiveness, in this thesis we focus on performance concerns that can be an effective obstacle against the adoption of XACML-based systems. Next chapter will address the trade-offs between security and performance in XACML-based systems.

CHAPTER 7

IMPROVING PERFORMANCE IN XACML-BASED SYSTEMS

Access control policies regulating access in cloud-based applications can grow in terms of rules size. This stems from the multitude of cloud stakeholders, their intricate workflows and the amount of sensitive resources that have to be protected. Request evaluation time is an important indicator to evaluate the efficiency of a policy-based system. Huge access control policies result in long request evaluation time and engenders performance bottlenecks. In this chapter, we propose a mono-objective optimization strategy that operates at the level of policy structure to reduce the request evaluation time of policy based systems. For centralized systems that are regulated by access control policies, we propose to refactor access control policies by splitting a policy (handled by a single PDP) into its corresponding multiple policies with a smaller number of rules (handled by multiple PDPs). We define attribute-set-based splitting criteria to facilitate splitting a policy. We have conducted an evaluation on three subjects of real-life Java systems, each of which interacts with access control policies. Our evaluation results show that (1) our approach preserves the initial architectural model in terms of interaction between the business logic and its corresponding rules in a policy, and (2) our approach enables to substantially reduce request evaluation time for most splitting criteria.

A major part of this chapter has been published in the following conference proceedings: ***Donia El Kateb****, Tejeddine Mouelhi, Yves Le Traon, JeeHyun Hwang, and Tao Xie. Refactoring access control policies for performance improvement. In Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering (ICPE), pp 323-334, 2012.*

Contents

7.1 Introduction

XACML architecture facilitates managing access rights in a fine-grained way by decoupling the business logic from the access control decision logic, which can be standardized and separately managed. As mentioned earlier, this architecture presents an ideal design for cloud-based software. However, this architecture may cause performance degradation especially when policy authors maintain a single policy with a large number of rules to regulate the whole system's resources. Consider that the policy is centralized with *only* one single PDP. The PDP evaluates requests (issued by PEPs) against the large number of rules in the policy in real-time. Such centralization can be a major factor for degrading performance. This performance bottleneck issue may impact service availability as well, especially when dealing with a huge number of requests within a short time.

In order to address this performance bottleneck issue, we propose an approach to refactor policies automatically to significantly reduce request evaluation time. As manual refactoring is tedious and error-prone, an important benefit of our automated approach is to reduce significant human efforts as well as improving performance. Our approach includes two techniques: (1)

refactoring a policy (handled by single PDP) to its corresponding multiple policies each with a smaller number of rules (handled by multiple PDPs), and (2) preserving the architectural property stating that a single PDP is triggered by a given PEP at a time.

In the first technique, our approach takes a splitting criterion and an original global policy (i.e., a policy governing all of access rights in the system) as an input, and returns a set of corresponding sub-policies, each of which consists of a smaller number of rules. This refactoring involves grouping rules in the global policy into several subsets based on the splitting criterion. More specifically, we propose a set of splitting criteria to refactor the global policy to smaller policies. A splitting criterion selects and groups the rules handled by the overall PDP into specific PDPs. Each criterion-specific PDP encapsulates a sub-policy that represents a set of rules that share the same combination of attribute elements (Subject, Action, and/or Resource). In the second technique, our approach aims at preserving the architectural property that only a single PDP is triggered by a given PEP at a time. More specifically, given a request, each PEP should be mapped to a PDP loaded with a policy, which includes a set of rules to be applicable for the request. Therefore, our refactoring maintains the architectural property of centralized architectures in policy-based systems.

We collect three subjects of real-life Java systems. Each system interacts with access control policies, whose corresponding request evaluation faces performance degradation. These policies are specified in XACML. We conduct an evaluation to show performance improvement achieved by our approach in terms of request evaluation time. We leverage two types of PDPs to measure request evaluation time. The first one is the Sun PDP implementation[1], which is a popular open source PDP, and the second one is XEngine [LCHX08], which transforms an original policy into its corresponding policy in a tree format by mapping attribute values with numerical values. Our evaluation results show that our approach enables reducing the request evaluation time substantially. This chapter makes the following three main contributions:

- We propose an automated approach that refactors a single global policy to policies each with a smaller number of rules. This refactoring helps improving performance of request evaluation time.
- We propose a set of splitting criteria to help refactor a policy in a systematic way. Our proposed splitting criteria do not alter policy behaviors of the centralized architectures.
- We conduct an evaluation on three Java systems interacting with XACML policies. We measure performance in terms of request evaluation time. Our evaluation results show that our approach achieves substantially faster than that of the centralized architectures in terms of request evaluation time.

The remainder of this chapter is organized as follows. Section 7.2 introduces concepts related to our research problem addressed in this chapter and surveys existing approaches that have

[1]http://sunxacml.sourceforge.net/

focused on the improvement of the performance of policy-based systems. Section 7.3 presents the overall approach. Section 7.4 presents evaluation results and discusses the effectiveness of our approach. Section 7.5 concludes this chapter.

7.2 Centralization: a threat for performance

This section further details a centralized architecture, its two desirable features such as synergy and reconfigurability, and its induced penalty (performance bottlenecks). Managing access control policies is one of the most challenging issues faced by an organization due to frequent changes in a policy. For example, a policy-based system has to handle some specific requirements such as role swapping when employees are given temporary assignments, as well as changes in the policies and procedures, new assets, users and job positions in the organization.

7.2.1 Centralization of architectures

To facilitate policy management, an access control policy is commonly modeled, analyzed, and implemented as a separate component encapsulated in a PDP. This separation leads to the centralized architecture, in which one single PDP is responsible for granting/denying the accesses that are requested. This centralized architecture is a simple solution to easily handle changes in policy-based systems by enabling the policy author to directly change policies on the single PDP. When a huge number of access requests are sent by the PEP to the PDP, two bottlenecks cause performance degradation:

- all the access requests have to be managed through the same input channel of the PDP.
- the centralized PDP computes an access request by searching which rule is applicable among all the rules that the encapsulated policy contains.

A request evaluation time is thus strongly related to:

- the number of rules in the policy that the PDP contains [MSSS09].
- the workload (i.e., the number of requests) that have to be evaluated by the system.

The request evaluation time depends on the size (number of rules) of the policy that the PDP encapsulates. For a given policy size, the evaluation time to evaluate requests increases linearly with the workload (i.e., the number of requests). Our *Hypothesis 1* is that the more rules a policy contains, the higher the slope of the evaluation time with an increasing workload. *Hypothesis 1* validity is discussed in Section 8.4.1. As a consequence, one possibility to improve performance consists in splitting the centralized PDP into PDPs with smaller policy sizes. We consider keeping the same input channel in the decentralized architecture. Therefore, we do not change the PEP code. Note that if a specific input channel is required for each PEP, developers are required to change the PEP code to map each PEP with its corresponding PDP.

7.2.2 Centralization: PEPs and PDP synergy

Centralization offers a desirable feature by simplifying the routing of requests to the right PDP. Figure 7.1 illustrates the model of the access control architecture. In this model, a set of business processes, which comply to users' needs, is illustrated by the business logic, which is enforced by multiple PEPs. Conceptually, the decision is decoupled from the enforcement and involves a decision making process in which each PEP interacts with the same single PDP. The key point concerns the cardinality linking PEPs to the PDP. While a PDP is potentially linked to many PEPs, any PEP is strictly linked to exactly one PDP (which is unique in the centralized model). Since there is only one PDP, the requests are all routed to this unique PDP. No particular treatment is required to map a given PEP in the business logic to the corresponding PDP, embedding the requested rules. Another advantage of this many-to-one association is the clear traceability between what has been specified by the policy and the PEPs enforcing this policy at the business logic level. In such setting, when access control policies are updated or removed, the related PEPs can be easily located and updated or removed. Thus the application is updated synchronously with the policy changes. We call this desirable property *synergy* of the access control architecture: an access control architecture is said to be *synergic* if any PEP always sends its requests to the same PDP. As a consequence, splitting the centralized PDP into PDPs of smaller policy sizes may break this synergy since calls issued by PEPs can be handled by several PDPs. In this work, we consider various splitting criteria to transform a centralized PDP into PDPs with smaller policy size. Our *Hypothesis 2* is with comparable PDP policy sizes, the evaluation time will be reduced when the architecture is synergic. This hypothesis is investigated in Section 8.4.1.

7.2.3 Trade-off for refactoring

The following facts are taken into account in our approach:

- Access control architectures are centralized with a unique PDP.
- Centralization eases reconfiguration of an access control policy.
- Centralization threatens performance.
- Direct mapping from any PEP to only one PDP makes the access control architectures synergic.
- A synergic system facilitates PEP request routing and eases policy maintenance.

The goal of our work is to improve performance by refactoring the centralized model into its corresponding decentralized model with multiple PDPs. The resulting architecture must have an equivalent behavior and should not impact the desirable properties of the centralized model, namely reconfigurability and synergy. Automating the transformation from a centralized to a decentralized architecture is required to preserve reconfigurability. With automation, we can still reconfigure the centralized policy, and then automatically refactor the architecture. We propose automatic refactoring of a centralized model into its corresponding decentralized model while preserving high reconfigurability. However, refactoring the architecture by splitting the centralized PDP into smaller ones may break the initial synergy. This phenomenon is studied in the empirical study of Section 8.4.1 together with *Hypothesis 2*.

7.2.4 XACML policies and performance issues

XACML policies become more complex when handling increasing complexity of organizations in terms of structure, relationships, activities, and access control requirements. In such a situation, a policy often consists of a large number of rules to specify policy behaviors for various resources, users, and actions in the organizations. In policy-based systems, policy authors manage a centralized and a single PDP loaded with a single policy to govern all system resources. However, due to a large number of rules for evaluation, this centralization raises performance concerns related to request evaluation time for access control policies and may degrade the system efficiency and slow down the overall business processes. We present the following three main factors that may cause to degrade XACML request evaluation performance:

- An XACML policy may contain various attribute elements including `target` elements. Retrieval of attribute values in the `target` elements for request evaluation may increase the evaluation time.

- A `policy set` consists of a set of policies. Given a request, a PDP determines the final authorization decision (i.e., effect) of the whole `policy set` after combining all the applicable rules' decisions for the request. Computing and combining applicable rules' decisions contribute to increasing the evaluation time.

- `Condition` elements in rules can be complex because these elements are built from an arbitrary nesting of boolean functions and attributes. In such a situation, evaluating `condition` elements may slow down request evaluation time.

7.2.5 Existing work on improving XACML policy-based systems performance

There are several previous approaches about performance issues in security mechanisms. Ammons *et al.*[AdCGS04], have presented techniques to reduce the overhead engendered from implementing a security model in IBM's WebSphere Application Server (WAS). Their approach identifies bottlenecks through code instrumentation and focuses on two aspects: the temporal redundancy (when security checks are made frequently) and the spatial redundancy (using the same security techniques on the same code execution paths). For the first aspect, they use caching mechanisms to store checks results, so that the decision is retrieved from the cache. For the

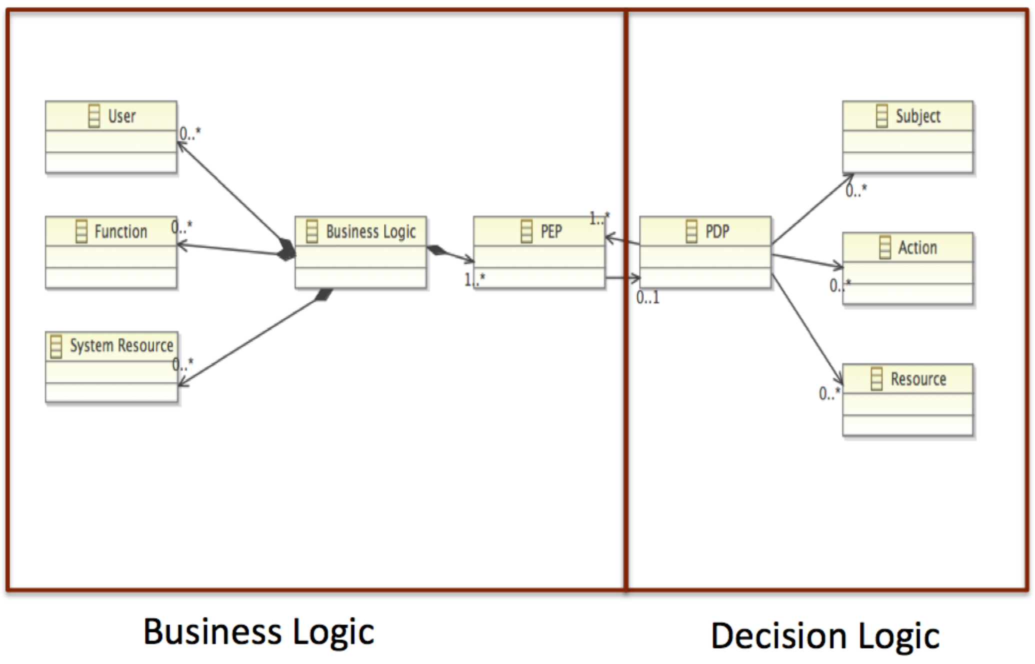


Figure 7.1: Access control model

second aspect, they used a technique based on specialization, which consists in replacing an expensive check with a cheaper one for frequent paths. While this previous approach focuses on bottlenecks in program code, in this chapter, we propose a new approach to refactor access control policies by reducing the number of rules in each split policy.

Caching mechanisms to improve policy-based systems have also been used in the work proposed by Trabesli *et al.* [TEADC]. The authors have proposed a caching-based approach to optimize access control systems to meet performance requirements of distributed cloud-based systems. Caching enables to store authorizations rules in a tree structure that is loaded in memory in the form of traditional database or in the form of a hash table.

Various approaches [JGHO11, MSSS09, LRB+08] have been proposed to address performance issues in systems interacting with access control policies. Jahid *et al.* [JGHO11] focus on XACML policy verification for database access control. They presented a model that converts attribute-based policies into access control lists. They implemented their approach called MyABDAC. While they measured performance of MyABDAC in terms of request evaluation, they did not show how much MyABDAC gains improvement over an existing PDP.

Marouf *et al.* [MSSS09] have proposed an approach for policy evaluation based on a clustering algorithm that reorders rules and policies within the policy set so that the access to applicable policies is faster. Their categorization is based on the subject target element. Their approach requires identifying the rules that are frequently used. Our approach follows a different strategy and does not require knowing which rules are used frequently. In addition, the rule reordering is tightly related to specific systems. If the PDP is shared between several systems, their approach could not be applicable since the most "used" rules may vary between systems.

Lin *et al.* [LRB+08] have decomposed a global XACML policy into local policies related to collaborating parties, and the local policies are sent to corresponding PDPs. The request evaluation is based on local policies by considering the relationships among local policies. In their approach, the optimization is based on storing the effect of each rule and each local policy for a given request. Caching decision results is then used to optimize evaluation time for an incoming request. However, there were no experimental results for measuring the efficiency of their approach when compared to the traditional architecture. While the previous approaches have focused on the PDP component to optimize the request evaluation, Miseldine *et al.* [Mis08] addressed this problem by analyzing rule location on XACML policies and requests at the design level so that the relevant rules for the request are accessed faster on evaluation time.

Our contribution in this chapter brings new dimensions over our previous work on access control [LCHX08, MTB09, MFBT08]. We have used XEngine [LCHX08], which focuses particularly on performance issues addressed with XACML policy evaluation. XEngine proposes an alternative solution to brute force searching based on an XACML policy conversion to a tree structure to minimize the request evaluation time. It involves a refactoring process that transforms the global policy to a decision diagram that is then converted to forwarding tables. In our contribution described in this chapter, we introduce a new refactoring process that involves splitting the policy into smaller sub-policies. XEngine combined with our refactoring process enables to decrease the evaluation time.

7.3 Policy refactoring

This section describes our approach of refactoring access control policies to improve performance by reducing the number of policy rules potentially applicable to a request. For refactoring policies in a systematic way, we propose seven policy splitting criteria based on attribute sets. Moreover, we explain how to select a splitting criterion that preserves the synergy in the access control architecture.

7.3.1 Policy splitting criteria

During the evaluation process, the attribute values in a given request are compared with the attribute values in the target of a rule. If there is a match between the request's attribute values and target's attribute values, the rule is then applicable to the request. In the decision making process, applicable rules contribute to determining the final authorization decision whereas non-applicable rules are not relevant in this process. For request evaluation, not all the rules are applicable to the request. In other words, only part of the rules (i.e, relevant rules) are applicable to the request and can contribute to determining the final decision.

We propose an approach to evaluate a request against only the relevant rules for the given request by refactoring the access control policies. Our approach aims at splitting a single global policy into multiple smaller policies based on attribute combination. For a given policy-based system, we transform its policy P into policies P_{SC_w} containing a smaller number of rules and conforming to a Splitting Criterion SC_w. An SC_w defines the set of attributes that are considered to classify all the rules into subsets each with the same attribute values and w denotes the number of attributes that have to be considered conjointly for aggregating rules based on specific attribute elements. Table 7.1 shows our proposed splitting criteria categorized according to attribute element combinations.

Table 7.1: Splitting criteria

Categories	**Splitting criteria**
SC_1	$\langle Subject \rangle, \langle Resource \rangle, \langle Action \rangle$
SC_2	$\langle Subject, Action \rangle, \langle Subject, Resource \rangle$ $\langle Resource, Action \rangle$
SC_3	$\langle Subject, Resource, Action \rangle$

To illustrate our approach, we present examples that take into consideration the XACML language features. In Figure 7.2, our approach refactors an XACML policy P according to the splitting criterion $SC_1 = \langle Subject \rangle$. Our refactoring results in two sub-policies Pa and Pb. Each sub-policy consists of relevant rules with regards to the same subject (Alice or Bob in this case).

Technically, to split a given policy P according to $SC_1 = \langle Subject \rangle$, we start by parsing the global policy P and by collecting the overall subject attribute values in the policy. For each collected subject attribute value Sa, we consider the global policy and we delete the rules that do not contain Sa as a subject attribute value in the target element attributes. After all the successive deletions, the global policy is refactored to a policy that contains only the rules with Sa in their subject attribute values. Algorithm 7 describes the splitting process for $SC_1 = \langle Subject \rangle$.

```xml
<Subjects>
<Subject>
<SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
<AttributeValue>Administrator</AttributeValue>
<SubjectAttributeDesignator AttributeId="role"/>
</SubjectMatch>
<SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
<AttributeValue >true</AttributeValue>
<SubjectAttributeDesignator AttributeId="isEq-subjUserId-resUserId"/>
</SubjectMatch>
</Subject>
</Subjects
```

Listing 7.1: Multi-attribute values in `target` element

Our algorithm is safe in the sense that it does not change the authorization behavior of the

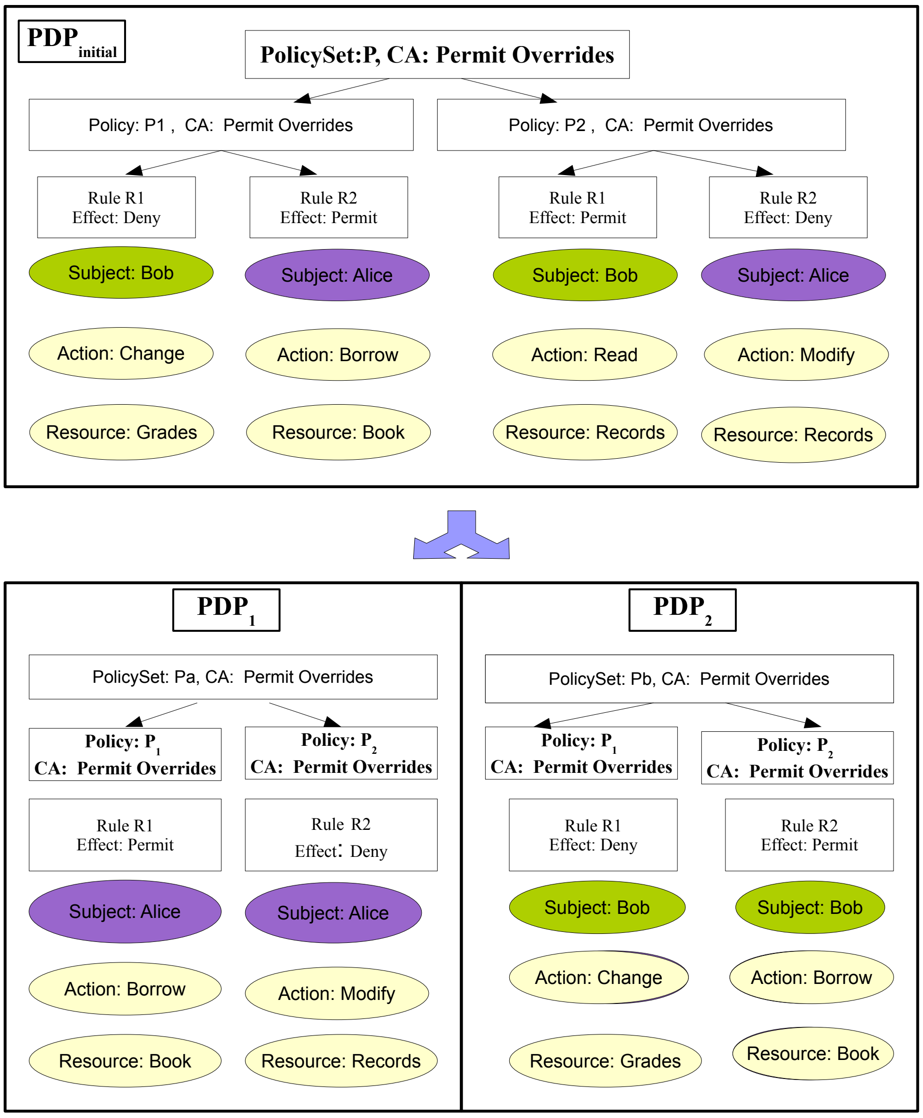


Figure 7.2: Refactoring a policy according to $SC_1 = \langle Subject \rangle$

PDP. There are two important issues to be considered when reasoning about the safety of the algorithm:

- Can the splitting impact authorization results when a policy set includes multiple policies with different combining algorithms?
- When AnySubject, AnyAction or AnyResource are used as target element values, does the splitting change the behavior of the PDP?

The first issue is addressed by the way the algorithm operates. The first step of the algorithm goes through all the rules and extracts the set of target element values (the set of subjects, the

Algorithm 7 Policy splitting algorithm for $SC_1 = \langle Subject \rangle$

```
Input: XACML Policy $P$, Splitting Criterion $SC_1$=$\langle Subject \rangle$
Output: Sub-policies Set: S
SplitPolicy()
S=Ø
/* Collect all subjects in all the rules /*
for each Rule $R_i$ in Policy $P$ do
   /* Fetch all the targets to extract attribute collection based on $SC$ */
   for each Target.Subject in $R_i$ do
      SubjectCollection.add(SubjectElement.attribute)
   end for
end for
/* Build sub-policies based on subjects collected in SubjectCollection */
for int $i = 0$; $i <$ SubjectCollection.size(); $i++$ do
   /* Remove all the rules that do not contain SubjectCollection.at(i) in their Target */
   for each Rule $R_i$ in Policy $P$ do
      if $R_i.Target.SubjectElement$ != AnySubject then
         if (Target.SubjectElement.attribute in $R_i$) != SubjectCollection.at(i) then
            Remove $R_i$
         end if
      end if
   end for
   /* $P_{(SubjectCollection.at(i))}$ is a sub-policy with only rules where the subjectAttribute is equal to
   SubjectCollection.at(i) */
   /* Add the sub-policy to the set of sub-policies */
   $S = S \cup P_{(SubjectCollection.at(i))}$
end for
```

set of actions, and/or the set of resources) based on the splitting criterion. Then, based on the extracted result, the splitting is performed by removing the rules with different splitting criterion values (such as a subject different from the splitting criterion subject). The rules that are kept are therefore not modified and their behavior is not altered. When there are several policies with different combining algorithms, the rules that are kept do not impact the evaluation behavior because they remain attached to the same combining algorithm. Moreover their order and their content are not modified.

The second issue is addressed by keeping all the rules that involve AnySubject, AnyAction, or AnyResource in all sub-policies because by definition during evaluation, these values are taken into consideration for evaluating all possible values of subjects, actions, and resources. It is worth mentioning this following consideration related to the refactoring process: XACML supports multi-valued attributes in policies and requests. In XACML policies, `target` elements define a set of attribute values, which match with the context element in an access control request. In Listing 7.1, the subject attribute includes two attributes (one is "role" and the other is "isEq-subjUserId-resUserId"). In order to match the subject with multi-valued attributes, a request should include at least `pc-member` and `true` for “role” and “isEq-subjUserId-resUserId”, respectively. Our approach considers such a whole subject element as a single entity, which is not split by the policy splitter component.

After the splitting is performed, our approach creates one or more PDPs that comply with the splitting criterion. We use the Sun PDP[2] to evaluate a request against policies specified in XACML. During request evaluation, the Sun PDP checks the request against the policy and determines whether the decision is permit or deny. Given a request, our approach runs the Sun PDP loaded with the request’s relevant policy, which is used during the decision making process. The PDP then retrieves the rules that are applicable to the request. Figure 7.3 presents our approach to handling request evaluation with multiple policies. During the evaluation process, given a request, our approach verifies the matching between the request’s attribute value and the policy target elements attributes. Our approach then selects only the relevant policy among all the policies for a given request. After the selection of the relevant policy, all of its relevant rules for the decision making are evaluated. Figure 7.4 shows an overview of our approach. In our approach, the policy splitter component plays a role to refactor access control policies. Given a single PDP loaded with the initial global policy, the policy splitter component conducts automated refactoring by creating multiple PDPs loaded with XACML policies, which are split from the initial global policy based on the user-specified splitting criterion. If the initial global policy is changed, the policy splitter component is required to refactor the policy again to create PDPs with the most recent relevant policies. Our refactoring approach is safe in the sense that the approach does not impact existing security aspects in a given system.

7.3.2 Architecture model preservation: PEP-PDP synergy

We propose to preserve the synergy property in the access control architecture by mapping a PEP and a PDP loaded with the relevant policy for a request dynamically at runtime. As shown in Section 7.3.1, given multiple PDPs after the policy refactoring, we consider (1) how PEPs are organized at the application level, and (2) how PEPs are linked to their corresponding PDPs. In the worst case, splitting the initial PDP into multiple PDPs may lead to a non-synergic system: a PEP may send its requests to several PDPs. The PDP that handles a given request is only known at runtime. Such a resulting architecture breaks the PEP-PDP synergy and the conceptual simplicity of the initial architecture model. In the best case, the refactoring preserves the simplicity of the initial architecture by keeping a many-to-one association between PEPs to PDPs. Given a request, our approach maps a PEP to a PDP with relevant rules for the request. Therefore, different requests issued from a PEP should be handled by the same PDP.

[2] http://sunxacml.sourceforge.net/

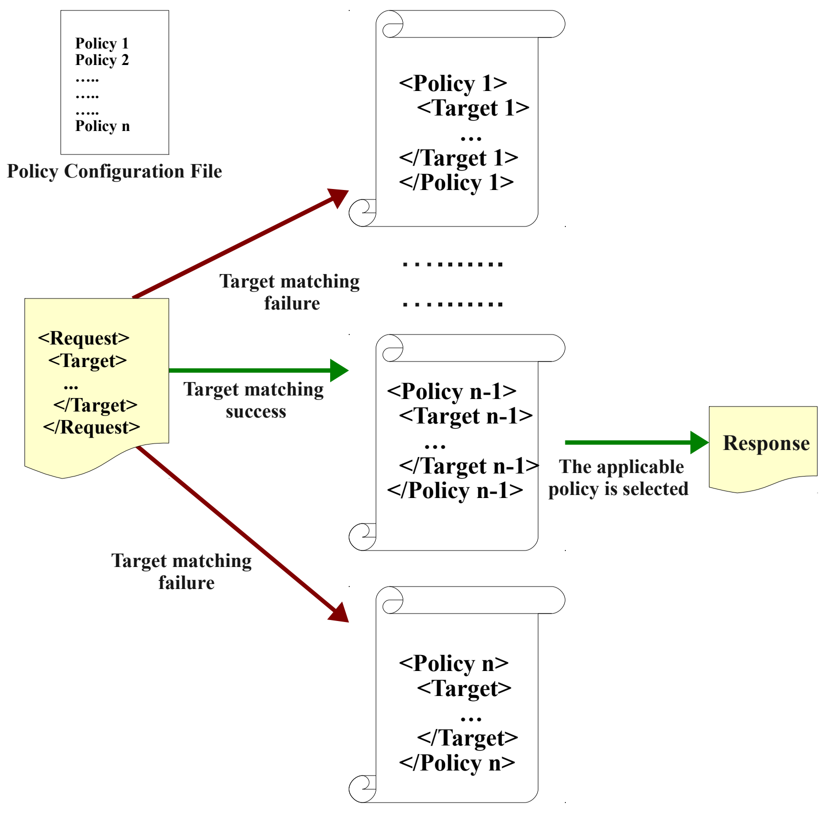

Policy 1
Policy 2
.....
.....
.....
Policy n
Policy Configuration File
<Policy 1> <Target 1> ... </Target 1> </Policy 1>
....
....
Target matching failure
<Request> <Target> ... </Target> </Request>
Target matching success
<Policy n-1> <Target n-1> ... </Target n-1> </Policy n-1>
The applicable policy is selected
Response
Target matching failure
<Policy n> <Target> ... </Target> </Policy n>

Figure 7.3: Applicable-policy selection

Operationally, the request evaluation involves one policy. In this case, our refactoring does not impact the conceptual architecture of the system.

Figure 7.5 presents a PDP encapsulating a global policy that has been refactored. The system that is presented on the left results from a desirable refactoring whereas the one on the right results from an undesirable refactoring. At the application level, a PEP is represented by a method call that triggers a decision making process. Listing 7.2 presents a sample PEP code snippet [LTMPB08]. This code snippet shows an example of a PEP represented by the method `checkSecurity`, which calls a method of the class `SecurityPolicyService`, which formulates a request to invoke the PDP component. The PEP represented by the method `ServiceUtils.checkSecurity` may issue requests that have subject “user” along fixed action and resource (“LibrarySecurityModel.BORROWBOOK_METHOD”), (“LibrarySecurityModel.BOOK_VIEW”). Consider that we refactor a policy using $SC_2 = \langle Resource, Action\rangle$, $SC_1 = \langle Action\rangle$, or $SC_1 = \langle Resource\rangle$. Given a request issued from the PEP, our approach runs a PDP loaded with a policy containing rules sharing the same action and resource attribute values. Thus the splitting process that preserves the mapping between the PEPs and the PDP is the one that considers the

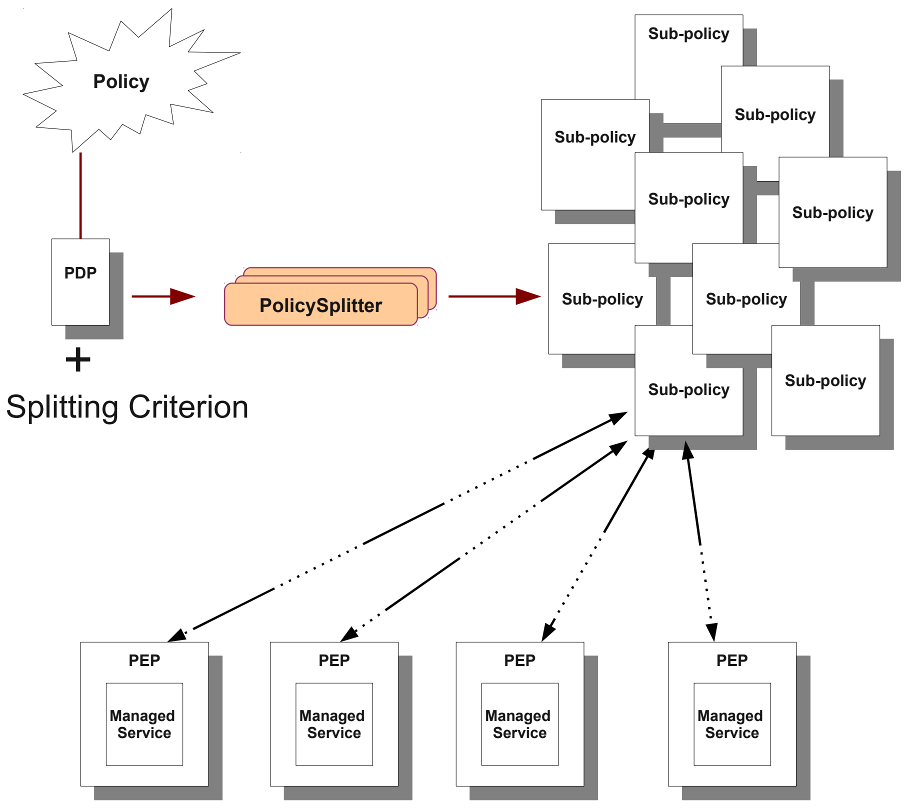


Figure 7.4: Overview of the refactoring process

following splitting criteria: $SC_2 = \langle Resource, Action \rangle$, $SC_1 = \langle Action \rangle$, and $SC_1 = \langle Resource \rangle$ in this case. In the evaluation section, we investigate the impact of the synergy property on performance.

```
public void borrowBook(User user, Book book) throws
SecuritPolicyViolationException {
// call to the security service
ServiceUtils.checkSecurity(user,
LibrarySecurityModel.BORROWBOOK\_METHOD,
LibrarySecurityModel.BOOK\_VIEW),
ContextManager.getTemporalContext());}
// call to business objects
// borrow the book for the user
book.execute(Book.BORROW, user);
// call the dao class to update the database
bookDAO.insertBorrow(userDTO, bookDTO);}
```

Listing 7.2: PEP deployment example

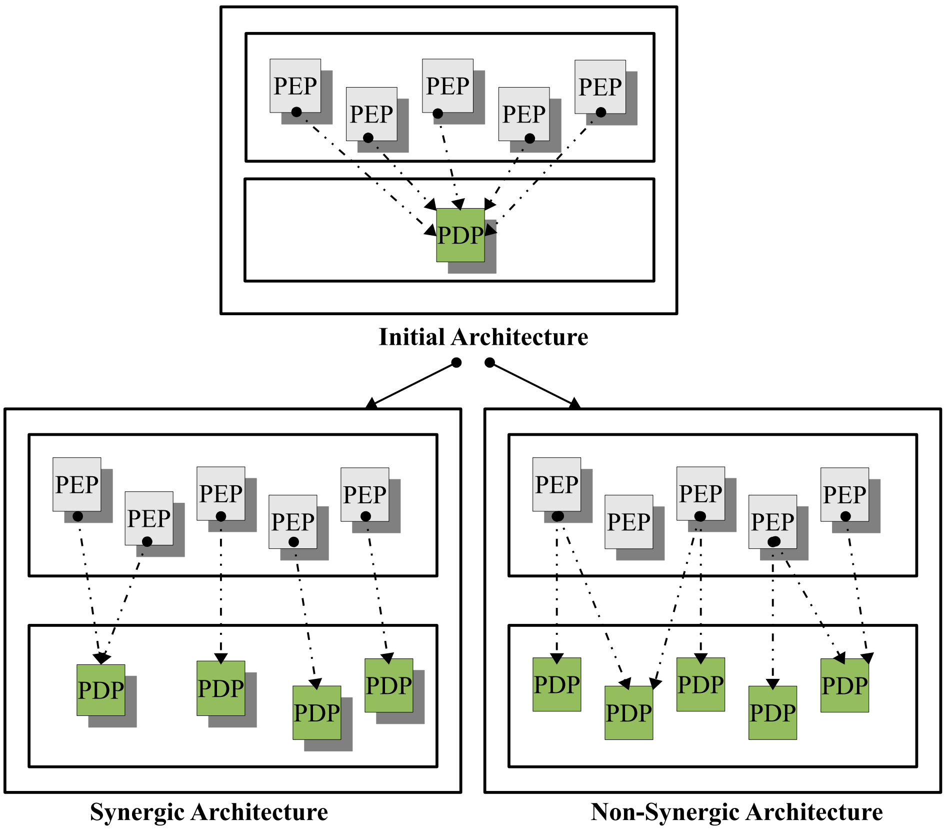


Figure 7.5: Synergic vs non-synergic system

7.4 Evaluation

We carried out our evaluation on a desktop PC running Ubuntu 10.04 with a Core i5, 2530 Mhz processor, and 4 GB of RAM. We have implemented a tool, called `PolicySplitter` to split policies according to a given splitting criterion automatically. The tool is implemented in Java and is available for download[3].

7.4.1 Objectives and metrics

Our evaluation intends to answer the following research questions:

1. **RQ1.** How faster can request evaluation time of multiple Sun PDPs with policies split by our approach achieve compared to that of an existing single Sun PDP? This question helps show that our approach can improve performance in terms of request evaluation time. Moreover, we compare request evaluation time for different splitting criteria.

[3]https://code.google.com/p/policysplitter/

2. **RQ2.** With comparable PDP policy sizes, is request evaluation time of a system faster when its architecture is synergic? This research question investigates *Hypothesis 2* presented in Section 7.2.

3. **RQ3.** How faster can request evaluation time of multiple XEngines with policies split by our approach achieve compared to that of an existing single XEngine? This question helps show that our approach can improve performance in terms of request evaluation time for other advanced policy evaluation engines such as XEngine.

4. **RQ4.** How faster does request processing time of multiple XEngines with policies split by our approach achieve compared to that of the Sun PDP with policies split by our approach? This question aims at checking whether XEngine in combination with our approach performs better than the Sun PDP combined with our approach as well.

5. **RQ5.** For larger PDP policy size, do we observe higher slope of the evaluation time with an increasing workload? This research question investigates *Hypothesis 1* (presented in Section 7.2) on the impact of the number of rules in a given PDP on the evaluation time.

To address these research questions, we go through the following evaluation setup based on two different empirical studies:

- First, we evaluate the performance improvement regarding the decision making process by taking into consideration the whole system (PEPs and PDPs). We compared request evaluation time with a single global policy (handled by a single PDP) against request evaluation time with split policies. All the splitting criteria have been considered in our evaluation. `IA` denotes an "Initial Architecture", which uses the single global policy for request evaluation. This step allows studying the behavior of splitting criteria that preserve the synergy property in the access control architecture.

- Second, we apply our approach on the Sun PDP and XEngine [LCHX08], respectively, to investigate the effectiveness of our approach on various decision engines. We aim at showing that our approach is complementary to an existing decision engine, even an optimized one such as XEngine.

7.4.2 Subjects

The subjects include three real-life Java systems each of which interacts with access control policies. Full details on our subjects are available elsewhere [MTB09, LTMPB08, MFBT08]. We next describe our three subjects.

- The Library Management System (LMS) provides web services to manage books in a public library.

- The Virtual Meeting System (VMS) provides web conference services. VMS allows users to organize online meetings in a distributed platform.

- The Auction Sale Management System (ASMS) allows users to buy or sell items online. A seller initiates an auction by submitting a description of an item that she wants to sell with its expected minimum price. Users then participate in the bidding process by bidding the item. To bid on the item, user must have enough money in his/her account before bidding.

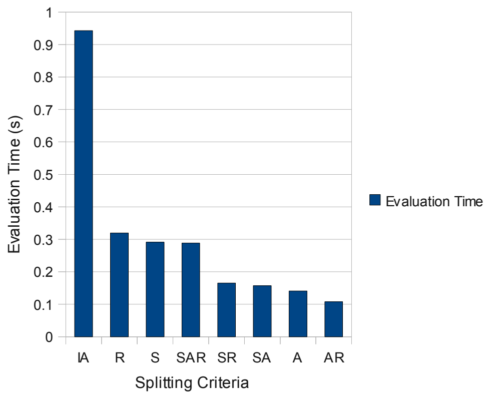
Our subjects are initially built upon the Sun PDP[4] as a decision engine, which is a popularly used PDP to evaluate requests. We started by a processing step, in which we have augmented the rules in the three original policies for these studies, as it would be difficult to observe performance improvement results with systems including few rules. In our evaluations, LMS policy contains 720 rules, VMS has 945 rules, and ASMS has 1760 rules. The rules that we added do not modify the system behavior as they are conform to the specifications. Moreover, to assess performance improvement over an existing advanced PDP, we adopt XEngine (instead of the Sun PDP) in our subjects to evaluate requests. XEngine is an advanced policy evaluation engine, which transforms the hierarchical tree structure of the XACML policy to a flat structure to reduce request evaluation time. XEngine also handles various combining algorithms supported by XACML.

7.4.3 Performance improvement: Sun PDP

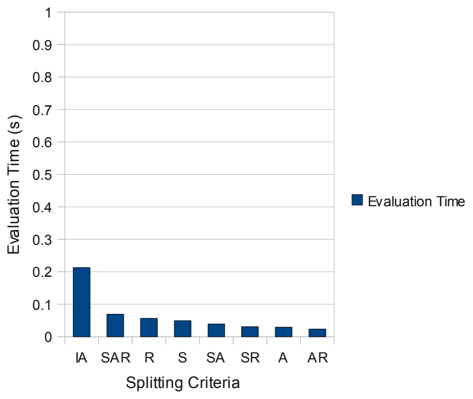
In order to answer **RQ1**, we generated the resulting sub-policies for all the splitting criteria defined in Section 7.3.1. For each splitting criterion, we have executed system tests to generate requests that trigger all the PEPs in the evaluation. The test generation step leads to the execution of all combinations of possible requests described in our previous work [MTB09]. The process of test generation is repeated ten times to alleviate the impact of randomness. We applied this process for each splitting criterion and calculated evaluation time on average of a system under test. Figure 7.6 presents evaluation time for policies split based on each splitting criterion and the global policy of the subjects. We can make two observations:

- Compared to the evaluation time of `IA`, our approach improves performance for all of splitting criteria in terms of evaluation time. This observation is consistent with our expected results; the evaluation time against policies with a smaller number of rules (compared with the number of rules in `IA`) is faster than that against policies in `IA`.

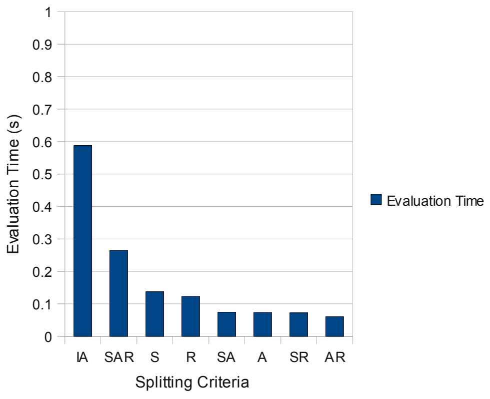
- The splitting criterion $SC = \langle Action, Resource \rangle$ enables to show the fastest evaluation time. Such observation is due to the fact that the PEPs in our three subjects are organized based on $SC_2 = \langle Action, Resource \rangle$. This observation pleads in favor of applying a splitting criterion that takes into account the PEP-PDP synergy.



[LMS]



[VMS]



[ASMS]

Figure 7.6: Request evaluation time for the three subjects

To identify the splitting criterion that generates the smallest number of PDPs, we have studied the number of policies generated by the splitting. Figure 7.7 shows the results. We observed the number of policies based on our proposed three categories: (1) the SC_1 category leads to the smallest number N_1 of PDPs, (2) the SC_2 category leads to a medium number N_2 (N_1<N_2<N_3) of PDPs, and (3) SC_3 leads to the largest number N_3 of PDPs. While the SC_1 category leads to the smallest number of PDPs, each PDP encapsulates a relatively large number of rules in a policy (compared with that of SC_2 and SC_3, which leads to performance degradation). We have classified splitting criteria according to their preservation of the synergy property considering our subjects. The classification is shown in Table 7.2 where S denotes Subject, R denotes Resource, A denotes Action, and IA denotes Initial Architecture. For example, AR denotes SC=$< Action, Resource >$. AR, A, and R are synergic splitting criteria since all the PEPs in our considered three systems are organized as shown in Figure 7.2.

To answer **RQ2**, we have evaluated PDPs in the three systems and for the different splitting criteria. The results presented in Figure 7.8 show the average number of rules in each PDP, for each splitting criterion in the three systems. We can observe that the AR criterion produces comparable size of PDPs with the SR criterion; however, as shown in Figure 7.6, AR is the

[4]http://sunxacml.sourceforge.net/

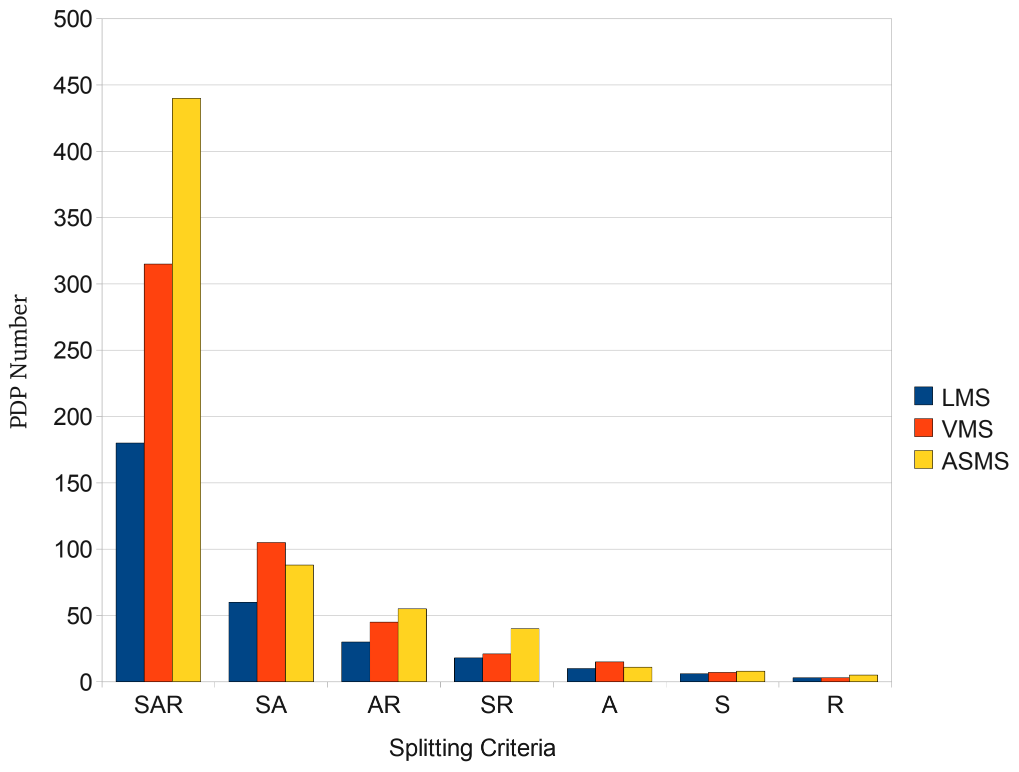


Figure 7.7: PDP number produced with splitting criteria

Table 7.2: Splitting criteria classification

	S	A	R	SA	SR	AR	SAR	IA
Synergic		x	x			x		x
Not-Synergic	x			x	x		x	

best splitting criterion in terms of evaluation time performance. Moreover, the number of PDPs produced with the splitting critera S and A is comparable; the criterion A, which is synergic, has evaluation time less than the one produced by the splitting criterion S, which is not synergic. This result supports our *Hypothesis 2*, which states that with comparable PDP sizes, the evaluation time would be reduced when the architecture is synergic.

7.4.4 Performance improvement: XEngine

In order to answer **RQ3** and **RQ4**, we measure request evaluation time of multiple XEngines with policies split by our approach compared with that of an existing single XEngine and that of multiple Sun PDPs with policies split by our approach, respectively. The goal of this empirical study is to show the impact of combining XEngine with our splitting process. XEngine itself

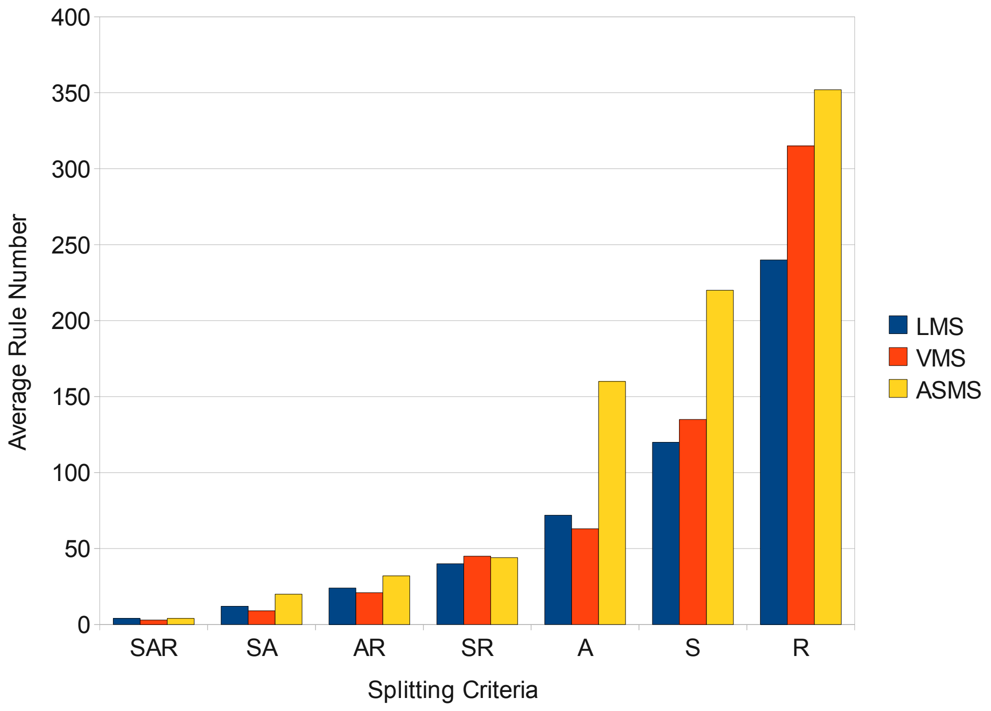


Figure 7.8: Average of rule numbers per PDP in the three systems

improves dramatically the performance of the Sun PDP mainly for three reasons:

- It uses a refactoring process that transforms the hierarchical structure of the XACML policy to a flat structure.
- It converts multiple combining algorithms to a single one.
- It relies on a tree structure that minimizes the request evaluation time.

We propose to use XEngine conjointly with the refactoring process presented in this work. We have evaluated our approach in two settings:

- Considering evaluation with decision engines based on XEngine with split policies and with the initial policy.
- Considering evaluation with decision engines based on the Sun PDP with split policies and with the initial policy.

In this step, we do not reason about the synergy, since we do not consider the application level for the three systems. We measure request evaluation time by evaluating a randomly generated set of 10,000 requests as proposed in our previous work [MXY06a]. The request evaluation time is evaluated for the three systems. The results are presented in Tables 7.3, 7.4, and 7.5.

In the three tables, the percentage of performance improvement denoted as "% PI" shows the reduction of request evaluation time (achieved by our approach) over the request evaluation with the initial architecture (IA).

Multiple XEngines with split policies, in most cases, enable to reduce the evaluation time compared to XEngine with a single policy. This result is shown in Table 7.5 for ASMS where the evaluation time is reduced about 33 times from 1639 ms in the initial architecture (IA) to 49 ms with the splittig criterion SAR. This empirical observation shows that our refactoring conjointly with XEngine enables to improve the performance of the evaluation process for most of the splitting criteria and thus answers **RQ3**.

As shown in the three tables, there are some splitting criteria that lead to decrease of performance such as the splitting criterion R in VMS system, which leads to degrade the evaluation time to -44%. These results need to be investigated with further studies.

Through the three tables, we observe that, when our subjects are equipped with XEngine, our proposed approach substantially improves performance (compared to the results with the Sun PDP) for most of the splitting criteria. For the splitting criterion SC=$\langle Action \rangle$ abbreviated as A, in the LMS system, the evaluation time is reduced about 22 times: from 2703 ms to 120 ms with XEngine, this observation enables to answer **RQ4**.

7.4.5 Impact of increasing workload

To investigate **RQ5**, we have calculated request evaluation time according to the number of requests incoming to a system. For each policy in the three systems (ASMS, LMS, and VMS), we generated 5000, 10000, .., 50000 random requests to measure the evaluation time (ms). The results are shown in Figure 7.9. For the three systems, we observe that the evaluation time increases when the number of requests increases in a system. With an increasing system load, the request evaluation time is considerably improved when using the splitting process compared to the initial architecture. The results shown in Figure 7.9 are interpreted by the average of PDP sizes presented in Figure 7.8. The results are consistent with *Hypothesis 1* (presented in Section 7.2), which states that the slope of evaluation time increases with PDP size in a system with an increasing workload.

To deploy our approach, we need to fetch the relevant PDP for a given request at runtime. Therefore, request processing time includes both fetching time and request evaluation time. Figure 7.10 shows percentage of fetching time over the global evaluation time for request evaluation in LMS. The fetching time increases according to the PDP size. The fetching time is relatively small in comparison with the total evaluation time and thus does not impact significantly the evaluation time.

Table 7.3: Evaluation Time (ms) in LMS

	SAR	AR	SA	SR	R	S	A	IA
Sun PDP	485	922	1453	1875	2578	2703	2703	2625
% PI Sun PDP	81.5	64.9	44.6	28.6	1.8	-3	-3	0
XEngine	26	47	67	95	190	164	120	613
% PI XEngine	95.7	92.3	89.0	84.5	69	73.2	80.4	0

Table 7.4: Evaluation Time (ms) in VMS

	SAR	AR	SA	SR	R	S	A	IA
Sun PDP	1281	2640	3422	3734	6078	5921	6781	5766
% PI Sun PDP	77.8	54.2	40.6	35.2	-5.4	-2.7	-17.6	0
XEngine	34	67	96	145	384	274	149	265
% PI XEngine	87.2	74.7	63.8	45.3	-44.9	-3.4	43.8	0

Table 7.5: Evaluation Time (ms) in ASMS

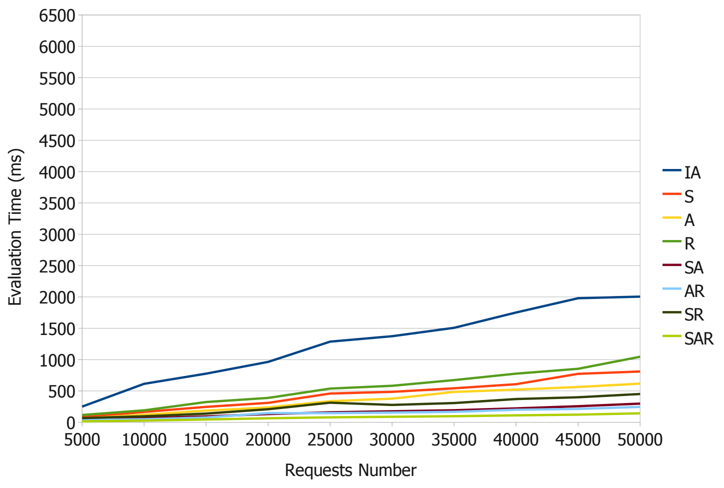
	SAR	AR	SA	SR	R	S	A	IA
Sun PDP	2280	2734	3625	8297	7750	8188	6859	7156
% PI Sun PDP	68.1	61.8	49.3	-15.9	-8.3	-14.4	4.1	0
XEngine	49	60	104	196	310	566	262	1639
% PI XEngine	97	96.3	93.65	88	81	65.5	84	0

7.4.6 Evaluation summary

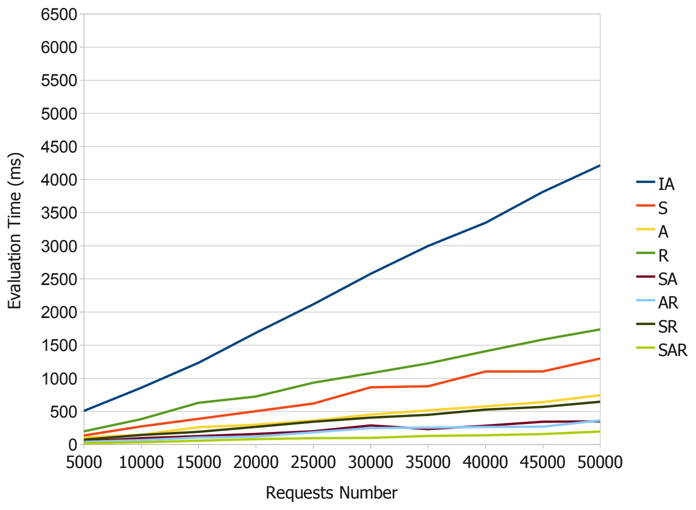
Our approach generates random test requests, which may induce bias or randomness in our results. To prevent such biases, we conduct our evaluation for 10 times and measure an average value of evaluation results. We have experimentally shown the effectiveness of the splitting in reducing the evaluation time. Our refactoring process improves both a typical PDP (the Sun PDP) and an advanced PDP (XEngine). When the sizes of PDPs are comparable, the splitting criteria that are synergic enable to have the best results in terms of evaluation time. The evaluation of the synergy property on improving performance has to be strengthened by conducting other experiments on other evaluation subjects and by considering different organizations of PEPs at the application level. Our approach is based on only seven proposed splitting criteria. We could develop additional splitting criteria to split policies and measure efficiency in terms of request evaluation time.

7.5 Conclusion

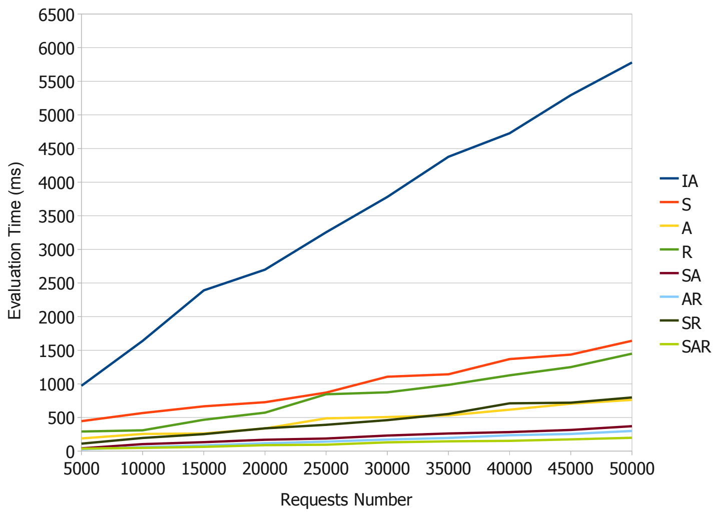
This chapter aims at exploring the trade-offs between security and performance at the level of guest applications. We have particularly analysed these trade-offs at the level of XACML policy-based applications. To improve the performance of XACML-policy based systems, we have proposed an automated refactoring process that enables to reduce request evaluation time substantially. To support and automate the refactoring process, we have designed and implemented the PolicySplitter tool, which transforms a given policy into small ones, according to a chosen splitting criterion. Most obtained results have shown a significant gain in evaluation time. The best gain in performance is reached by the criterion that respects the synergy property. This result pleads in favor of a refactoring process that takes into account the way PEPs are scattered inside the system business logic. Our contribution aims at building secure cloud-based software while reducing detrimental effects on performance, which is a key metric for cloud-based software. Next chapter points out another aspect of cloud-based software: costs, security trade-offs.



[LMS]



[VMS]



[ASMS]

Figure 7.9: Evaluation time for our subjects, LMS, VMS, and ASMS depending on the request number

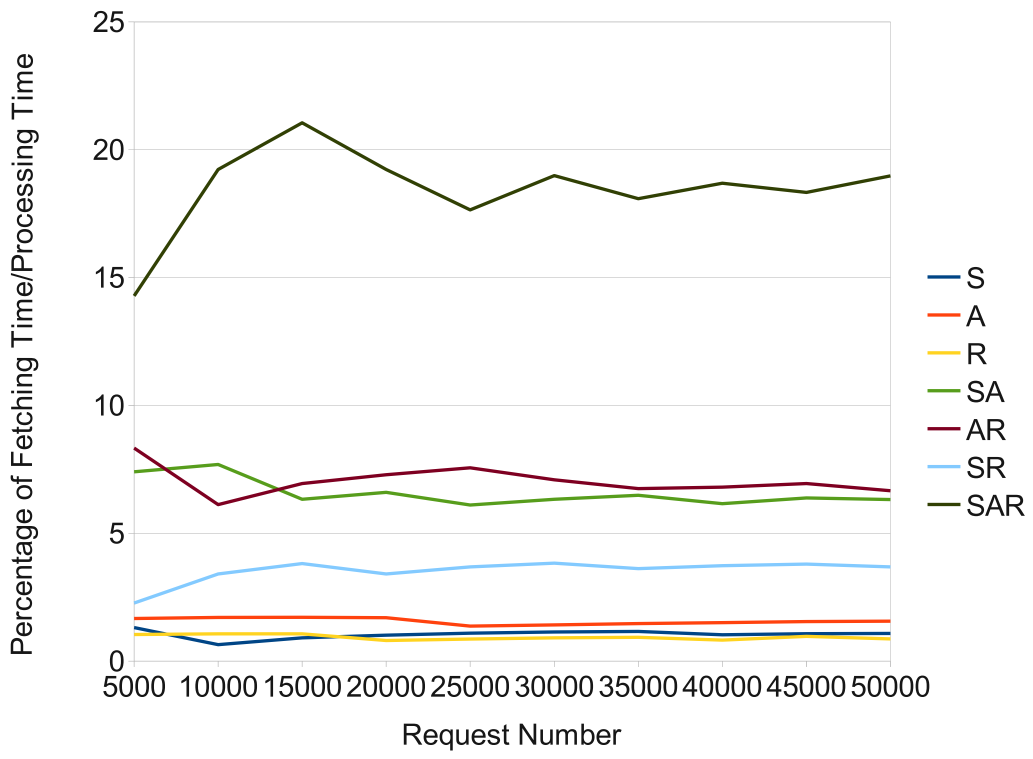


Figure 7.10: Percentage of fetching time

Chapter 8

Selection of regression system tests for security policy evolution

As security requirements of cloud-based software often change, developers may modify security policies such as access control policies to cope with evolving requirements. To increase confidence that the modification of policies is correct, developers conduct regression testing. Regression testing is an expensive activity that has to be performed whenever policies, regulating cloud-based applications evolve. The approach that we propose in this chapter aims at improving trade-offs between testing costs and secure policy evolution for policy-based systems. To improve the quality of cloud-based software, trade-offs improvement has also to be achieved in the overall life cycle of software development including software testing phases. To address this issue, we develop a regression-test-selection approach, which selects every system test case that may reveal regression faults caused by policy changes. Our evaluation results show that our test selection techniques reduce a significant number of system test cases efficiently.

A major part of this chapter has been published in the following conference proceedings: JeeHyun Hwang, Tao Xie, ***Donia El Kateb****, Tejeddine Mouelhi, and Yves Le Traon. Selection of regression system tests for security policy evolution. In Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering (ASE), pp 266-269, 2012.*

Contents

8.1 Introduction

Security requirements of cloud-based software often change during software development and maintenance, developers may modify policies to comply with requirements. In consequence, the multi-faceted nature of policies can affect system's behavior in various ways such as changes of access control privileges of a group or users. In order to increase confidence that the modification of policies is correct and does not introduce unexpected behavior, developers periodically conduct regression testing. For example, new security requirements include new security concerns to be added to a policy. Developers may change policies without changing program code related to actual system functionality. In such a situation, validating and verifying program code and policies after policy changes increases confidence of correct behaviors of the given system.

In this chapter, we focus on the regression testing problem in the context of policy evolution. For policy evolution, regression testing is a crucial task because policy behavior changes may result in unexpected behaviors of program code, these behaviors can be undesirable. The typical regression testing for program code interacting with a policy is as follows. Given a program and a policy P, the developers prepare initial system test cases, where each test case maps to rules (in the policy) exercised by the test case. Given P and its modified policy P', the developers compare impacts of P and P' to reveal different policy behaviors, which are "dangerous" portions to be validated with test cases. For validating these "dangerous" portions, the developers often select the test cases (from test cases for P) that exercise the dangerous portions of P'.

For regression testing, instead of writing new system test cases, developers reuse the initial test cases in practice. A naive regression testing strategy is to rerun all system test cases. However, this strategy is costly and time-consuming, especially for large-scale systems. Moreover, if the number of the initial system test cases is large, this strategy may require significant time for developers to conduct testing. In order to reduce the cost of regression testing, developers often adopt regression test selection, which selects and executes only test cases to expose different behaviors across different versions of the system. This approach sometimes may require substantial costs to select and execute such system test cases from the initial test cases that could reveal faults introduced by the changes. If the cost of regression test selection is smaller than re-running all of initial system test cases, test selection helps reduce overall cost in validating whether the modification is correct.

Besides costs reduction, safeness is important as well. Safe regression-test-selection approach selects every test case that may reveal a fault in modified program [RH96]. In contrast, unsafe regression-test-selection approach may omit test cases that may reveal a fault in the modified program. Similarly, we use safeness in this chapter in case of policy evolution by referring modified program as modified policies.

To address these issues, we propose safe regression-test-selection approach. Given an original policy P and its modified policy P', our approach selects superset SU_t of the set of test cases (i.e., fault-revealing test cases) that reveal faults for P' by analyzing different behaviors in program code due to policy changes. In the policy context, different policy behaviors refer to that, given a request, its evaluated decisions (for P and P', respectively) are different. These different policy behaviors are dangerous portions to be validated.

Our test-selection approach includes three techniques. The first one uses mutation analysis. This technique first analyzes correlation between test cases and rules impacted by policy changes. For correlation setup, this technique creates a mutant $P'(r_i)$ by changing decision of rule r_i in P. On executing test cases on program code interacting with P and $P'(r_i)$, respectively, this technique collects test cases that relate to r_i by analyzing different policy behaviors induced by policy changes. This technique is easy to implement however it is costly in terms of test case execution. Given the number n of rules in P, this technique requires at least n execution of test cases against mutants to find all correlation between test cases and rules. This technique next conducts change-impact-analysis statically that analyzes changes of rules between P and P' to select regression-test-cases.

To reduce cost in terms of test case execution, the second one uses coverage analysis for correlation setup. While executing tests cases against P, this technique finds correlation between test cases and rules by analyzing which rules are covered (i.e., evaluated) with each test case. This technique requires test case execution at once. However, these first two techniques use change-impact-analysis which is often costly in terms of time by comparing all policy behaviors between P and P' statically.

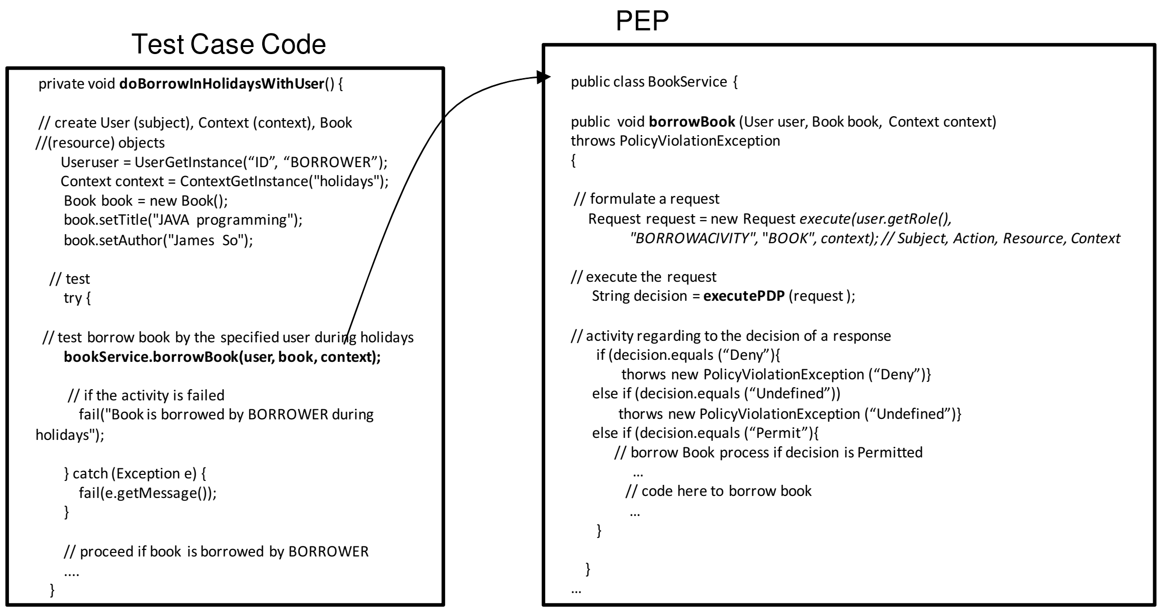
To reduce such change-impact-analysis cost, the third technique captures requests triggered from test cases at runtime. This technique next evaluates these requests against P and P' and selects only requests (that reveal different policy behaviors) and corresponding test cases. Our main objective, through test-selection, is to detect faults efficiently in the modified policy.

To sum up, this chapter proposes three techniques for regression test selection: the first one is based on mutation analysis, the second one is based on coverage analysis, and the third one is based on evaluated decisions of requests issued from test cases. We use the same applications that have been presented in the previous chapter to validate our approach. Our evaluation results show that our test selection techniques achieve up to 51%~97% of test reduction for a modified version with given 5~25 policy changes. Among our three techniques, our results show that the third technique is the most efficient compared with the first and the second techniques in terms of elapsed time. The third technique is 43 and 8 times faster than the first and the second techniques, respectively.

The rest of the chapter is organized as follows. Section 8.2 presents background information about policy-based software systems, policy context, and regression testing. Section 8.3 presents our approach. Section 8.4 presents and discusses our evaluation results. Section 8.5 concludes the chapter.

8.2 Regression testing for policy-based systems

Software testing [Mye79] refers to the activity of generating tests cases to verify the conformity of output results provided by a program to the expected output that meets its functional and non functional requirements. Testing activity aims at establishing a trade-off between cost, time and quality. Figure 8.1 shows simplified test case code and PEP code examples extracted from *LMS* subject [MLTB09]. `doBorrowInHolidaysWithUser` method in the test case code illustrates a test case that `BOWRROWER` borrows books during holidays. This method first creates subject (i.e., `Borrower` user), context (i.e., `Holidays`), and resource (i.e., Book). This method invokes `borrowBook` method in `bookService` class where a concrete request is formulated with regards to "BorrowBook Activity" for given subject role and context by adding necessary action. In the `borrowBook`, `executePDP` method calls a PDP to evaluate the request. The caller next proceeds based on decision (Deny, Undefined, or Permit) of a response from the PDP. In this example, the formulated request is that `Borrower` is permitted to do `borrowbook` activity during `Holidays`. Software is subject to changes that occur at the design stage or in later stages at the deployment or maintenance phases. These changes are usually supposed to meet changes in the requirements or to overcome errors that can be detected in later stages of software life cycle. Regression testing refers to the research field that aims at retesting the system to verify that the new changes have not altered the initial system behavior. As highlighted by Rothermel *et al.* in [RH96], regression testing is defined like the following: "Given a program P, a modified version P', and a set T



```
 private void doBorrowInHolidaysWithUser() {

 // create User (subject), Context (context), Book
//(resource) objects
     Useruser = UserGetInstance("ID", "BORROWER");
     Context context = ContextGetInstance("holidays");
      Book book = new Book();
      book.setTitle("JAVA  programming");
      book.setAuthor("James  So");

   // test
      try {

  // test borrow book by the specified user during holidays
      bookService.borrowBook(user, book, context);

        // if the activity is failed
          fail("Book is borrowed by BORROWER during
holidays");

      } catch (Exception e) {
          fail(e.getMessage());
      }

      // proceed if book is borrowed by BORROWER
      ....
    }
```

```
public class BookService {

public  void borrowBook (User user, Book book,  Context context)
throws PolicyViolationException
{

 // formulate a request
   Request request = new Request execute(user.getRole(),
           "BORROWACIVITY", "BOOK", context); // Subject, Action, Resource, Context

// execute the request
    String decision = executePDP (request );

// activity regarding to the decision of a response
     if (decision.equals ("Deny"){
          thorws new PolicyViolationException ("Deny")}
    else if (decision.equals ("Undefined"))
         thorws new PolicyViolationException ("Undefined")}
    else if (decision.equals ("Permit"){
         // borrow Book process if decision is Permitted
             ...
            // code here to borrow book
            ...
     }

   }
...
```

Figure 8.1: Test case code and PEP code examples

of test cases used previously to test P, regression analysis and testing techniques attempt to make use of a subset of T to gain sufficient confidence in the correctness of P' with respect to behaviors from P retained in P'". The main objectives of regression testing is to reduce the cost of rerunning initial test cases and to maximize the capability of selected test cases to detect potential faults induced by changes. To the best of our knowledge, there is no previous research work on regression testing that considers policy changes in policy-based software systems. The global scenario that illustrates regression testing process for such systems is presented in Figure 8.2. In the scenario, our test-selection technique identifies the changed policy behaviors (i.e., rules impacted by policy changes) between a policy P and its modified version P' and selects only the portion of test cases T' to reveal different behaviors impacted by policy changes. Moreover, a test selection algorithm is `safe` if, under certain well-defined conditions, the algorithm includes the set of every fault-revealing test case $F \subseteq T'$ that would reveal faults in the modified version. In order words, in `safe` regression algorithm, fault detection capabilities of executing selected test cases is equivalent to that of executing all of test cases.

In this chapter, we classify system test cases into two types illustrated in Figure 8.3. The first one is a set of functional system test cases, which are produced based on functional requirements. However, these functional system test cases are not involved in testing security requirements (e.g., triggering application code to generate and evaluate requests against a policy under test). The second one is a set of security system test cases. Different from the functional system test cases, these security system test cases trigger application code to generate and evaluate requests against a policy under test. Moreover, security system test cases may include test oracles to determine whether program behaviors interacting with a policy are correct.

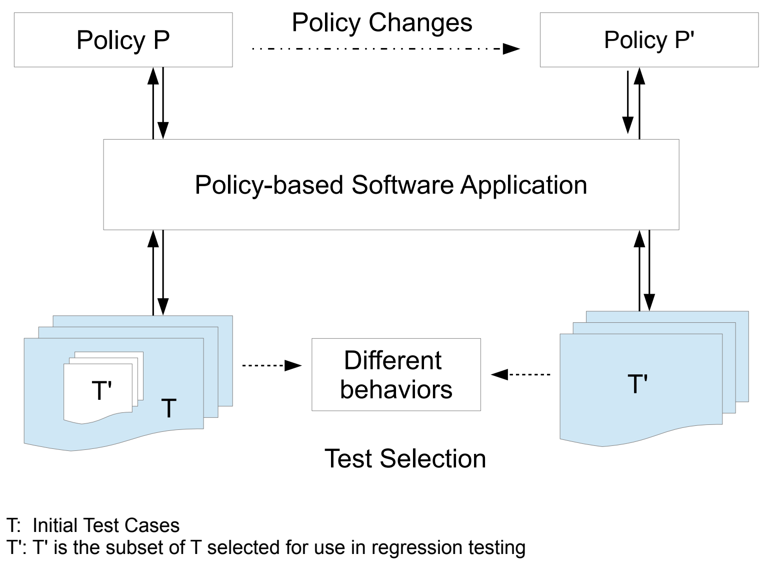

Policy P
Policy Changes
Policy P'
Policy-based Software Application
T'
T
Different behaviors
T'
Test Selection
T: Initial Test Cases
T': T' is the subset of T selected for use in regression testing

Figure 8.2: Regression testing process

To the best of our knowledge, our work is the first initiative for automatic test-selection approach in context of policy evolution. Prior work that is closest to ours is Mouelhi *et al.*'s work [MLTB09]. They have proposed a technique to transform functional test cases into security test cases. They defined various mutation operators at the policy level. Given a policy and its mutated policy, the technique selects only impacted test cases to be transformed as security test cases (with additional manual task to add code for checking security). While this work focuses on security test case selection from initial test cases, our work focuses on test selection problem impacted by policy changes to reveal faults induced by policy changes.

To facilitate verifying and validating the correctness of policies, prior research work has been done in the area of policy testing, which generates and executes test requests against a policy. Martin *et al.* [MXY06b] proposed a framework to detect policies faults by analyzing policy test suites that involve requests-responses pairs. The framework is based on mutation operators [MX07b] that enable measuring fault detection capability. It uses a tool [MX07a] that aims at minimizing the test cases to be generated by analyzing the structural coverage of the policies. Hu *et al.* [HMHX07] proposed a generic model-based conformance checking technique for access control policies written in XACML. These approaches focus on test request generation at a policy level (e.g., policies written in XACML). Instead of generating new requests, our work focuses on regression testing at the implementation level (the system level) by reusing existing hard-coded test cases.

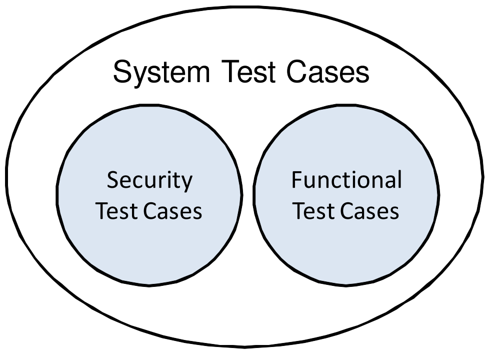


Figure 8.3: System test cases

8.3 Our proposed regression techniques

As manual selection of test cases for regression testing is tedious and error-prone, we have developed three techniques to automate selection of test cases for regression testing in policy evolution. Our approach takes two versions of a program that interact with v_1 (original) and v_2 (new) access control policy, respectively. The existing test cases are taken as an input; these tests invoke methods in program. We analyze given program and policies to select *only* test cases for regression testing in case of policy evolution. Among given test cases, our selected test cases invoke methods to reveal changed policy behaviors between v_1 and v_2.

Formally, C denotes a program, which interacts with an access control policy P. P_m is the modified version of P. T denotes an initial test suite for C. Our first step involves the regression test selection. We select $T' \subseteq T$ where T' is a set of test cases. T' execute on C and reveal changed policy behavior between P and P_m. In the second step, we measure rule coverage of changed policy behavior P_m with T'. We next describe our proposed three test selection techniques.

8.3.1 Test selection based on mutation analysis

Our first proposed test selection technique uses mutation analysis to select test cases as follows. The technique needs a preliminary step which is necessary to establish a rule-test correlation. We next describe rule-test correlation and test selection steps.

Rule-test correlation setup. Given a policy P, we create its rule-decision-change (RDC) mutant policy Pr_i by changing decision (e.g., Permit to Deny) of each rule r_i in P in turn. An example of a mutated policy is shown in Figure 8.4. In this policy, original Rule 1's decision Permit is changed to Deny. The technique finds affected test cases for this rule decision change. We execute test cases T on program for P and Pr_i. To detect changed policy behaviors, the technique monitors responses of evaluated requests formulated from test cases T. The test cases, which evaluate different policy decisions against P and P', enable to map rule r_i to test cases $t \in T$. The preliminary step ends by establishing a correlation between each rule in P and corresponding tests $t \in T$ that trigger this rule.

Algorithm 8 Test selection based on mutation analysis algorithm

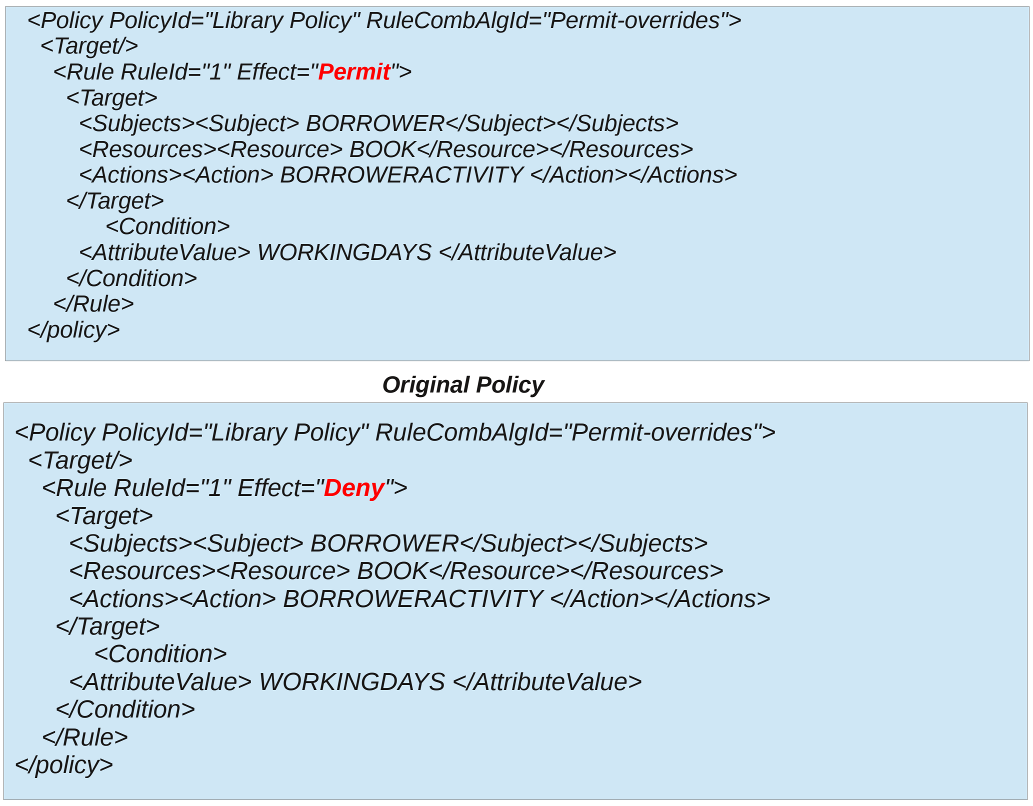
```
*TestSelection1*($P$, $P_m$, $T$): $T'$
**Input:** XACML Policy $P$, modified policy $P_m$, Initial Test Cases $T$
**Output:** $T' \subseteq T$ where $T'$ is the subset of $T$ selected for use in regression testing $P_m$
$T'$=∅
/*Rule-test set-up phase*/
**for** each rule $r_i$ in Policy $P$ **do**
   $T_{r_i}$=∅ where $T_{r_i} \subseteq T$ are the tests correlated to $r_i$
   /*We mutate the policy $P$ by creating a rule-decision change (RDC) on $r_i$ to get $P_{r_i}$*/
   $P_{r_i} \xleftarrow[RDC(r_i)]{} P$
   Execute $T$ with $P_{r_i}$
   **for** each $t$ in $T$ **do**
      Let $E(t)$ be the test execution result, $E(t) = Success, Failure$
      **if** $E(t) \leftarrow$ Failure **then**
         $T_{r_i} \leftarrow T_{r_i} \cup t$
      **end if**
   **end for**
   Map($r_i$,$T_{r_i}$)
**end for**
/*Test selection phase*/
$\{r_m\}_{i=1..m} \leftarrow diff(P, P_m)$
**for** Each rule $r_i$ in $\{r_m\}_{i=1..m}$ **do**
   $T' \leftarrow T' \cup T_{r_i}$
**end for**
return $T'$
```

Test selection. The selection of test cases for regression testing on P and its modified policy P_m starts by conducting change impact analysis of P and P_m to find which rules' decision are changed. Once these rules are identified, we use the mapping established in the preliminary step to select the subset of test cases which are correlated with changed rules.

Algorithm 8 describes our algorithm used for the technique. While the technique can quickly select test cases, the technique requires rule-test correlation setup (in the preliminary step), which could be costly in terms of execution time. Given n rules in P, we execute T for 2×n times. As the preliminary step is applied for only existing rules R in P, our technique requires addition of rule-test correlation for newly added rules R_n where $R_n \notin R$ in P_m. In addition, if a new system test is introduced, we execute this test for 2×n times.

8.3.2 Test selection based on coverage analysis

Our preceding technique finds correlation of all of existing rules N in a given policy with the test cases. To reduce such correlation setup efforts, we develop a technique to correlate only rules, which can be evaluated (i.e., covered) by test cases. Our intuition is that test cases may interact only with a small number of rules in a policy instead of all the rules in the policy. We



```
<Policy PolicyId="Library Policy" RuleCombAlgId="Permit-overrides">
 <Target/>
  <Rule RuleId="1" Effect="Permit">
   <Target>
    <Subjects><Subject> BORROWER</Subject></Subjects>
    <Resources><Resource> BOOK</Resource></Resources>
    <Actions><Action> BORROWERACTIVITY </Action></Actions>
   </Target>
       <Condition>
    <AttributeValue> WORKINGDAYS </AttributeValue>
   </Condition>
  </Rule>
</policy>
```

Original Policy

```
<Policy PolicyId="Library Policy" RuleCombAlgId="Permit-overrides">
 <Target/>
  <Rule RuleId="1" Effect="Deny">
   <Target>
    <Subjects><Subject> BORROWER</Subject></Subjects>
    <Resources><Resource> BOOK</Resource></Resources>
    <Actions><Action> BORROWERACTIVITY </Action></Actions>
   </Target>
       <Condition>
    <AttributeValue> WORKINGDAYS </AttributeValue>
   </Condition>
  </Rule>
</policy>
```

Mutated Policy

Figure 8.4: An original policy and its rule-decision-change (RDC) mutated policy

next describe rule-test correlation step.

Rule-Test correlation Setup. Given a policy P, we execute test cases T on program that interacts with P. Our technique monitors which rules in a policy are evaluated with requests formulated from test cases T. Then, we establish correlation between test cases and evaluated (i.e., covered) rules in P.

Test selection. Once the mapping rule-test is established, we proceed with the test selection step described in the first approach with the results of change impact analysis of P and P_m. The results include information to show which rules' decision have changed. This test selection maps test cases with those rules to constitute the subset of existing test cases.

Algorithm 9 describes our algorithm used for the technique. An important benefit of this technique is to reduce cost in terms of mutation analysis and execution time. This technique does not require generating mutants by changing rule's decision in turn. Moreover, the technique can significantly reduce execution time. While the technique can quickly select system tests in the second step, the technique requires rule-test correlation setup (in the preliminary step), which could be costly in terms of execution time. Considering that the requests Rs are formulated from test cases and interact with only n_1 rules ($n_1 \leq n$) in a policy, we execute T only once. Our

Algorithm 9 Test selection based on coverage analysis algorithm

```
*TestSelection2*($P$, $P_m$, $T$): $T'$
**Input:** XACML Policy $P$, modified policy $P_m$, Initial Test Cases $T$
**Output:** $T' \subseteq T$ where $T'$ is the subset of $T$ selected for use in regression testing $P_m$
$T'=\emptyset$
/*Rule-test set-up phase*/
**for** Each test case $t$ in $T$ **do**
  Execute $t$ with $P$
  $MAP$=Map($t$,$\{r_p\}_{i=1..p}$) where $\{r_p\}_{i=1..p}$ are the rules in $P$ that are evaluated (i.e., covered) by $t$
**end for**
/*Test selection phase*/
$\{r_m\}_{i=1..m} \leftarrow diff(P,P_m)$
**for** Each rule $r_i$ in $\{r_m\}_{i=1..m}$ **do**
  $T' \leftarrow T' \cup T_{r_i}$
**end for**
return $T'$
```

Algorithm 10 Test selection based on recorded request evaluation

```
*TestSelection3*($P$, $P_m$, $T$): $T'$
**Input:** XACML Policy $P$, modified policy $P_m$, Initial Test Cases $T$
**Output:** $T' \subseteq T$ where $T'$ is the subset of $T$ selected for use in regression testing $P_m$
$T'=\emptyset$
$R_{T'}=\emptyset$ where $R_{T'}$ are the requests corresponding to $T'$
Execute system requests $R$
**for** each request $Re$ in $R$ **do**
  **if** $decision(Re/_P) \neq decision(Re/_{P_m})$ **then**
    $R_{T'} \leftarrow R_{T'} \cup Re$
  **end if**
**end for**
$T' \leftarrow R_{T'}$
return $T'$
```

technique requires addition of rule-test correlation for newly added rules R_n where $R_n \notin R$ in P_m as the same with the preceding technique.

8.3.3 Test selection based on recorded request evaluation

To reduce such correlation setup efforts in the preceding techniques, we develop a technique, which does not require rule-test correlation setup. Instead of correlation, our technique records test cases and their issued requests as a preliminary step. More specifically, our technique executes test cases T on a program for P and records all requests issued to policy decision point (PDP) for each test case. For test selection, our technique evaluates all issued requests against P and P_m and selects the test subset of requests (with corresponding system test cases) that engender different decisions for two different policy versions.

Algorithm 10 describes our algorithm used for the technique. The current approach requires the execution of system test cases T only once. Moreover, while the two preceding techniques are white-box testing since access control policies are available, the present technique does not require the availability of access control policies. This can present a considerable advantage when developers don't want to reveal their access control policies.

8.4 Validation

Our implementation includes two components: regression simulator and test selection. To simulate regression in a policy, we used three types of policy change types: RMR (Rule Removal), RA (Rule Addition), and RDC (Rule Decision Change). For RMR, given a randomly selected rule in a policy P, the regression simulator removes the rule. For RDC, given a randomly selected rule in a policy P, the regression simulator changes the decision of the rule. For RA, the regression simulator adds a randomly generated rule with random attributes collected from P in a random place. The simulator injects (i.e., mutates) more than one type of changes to P for simulating regression on P. Given n number of requests, the regression simulator analyzes a policy and injects n changes into the policy. Among three policy change types, RA is the most flexible because rules can be composed of any combination of randomly selected attributes.

We next describe rule-test correlation and request recording step in the test selection component. For test selection technique based on mutation analysis technique, our test-selection component first mutates a rule with RDC in turn. Then, the component executes all of test cases for each mutated policy, and finds rule-test correlation by monitoring whether test cases expose different behaviors of an original policy and its mutated policy. For test selection technique based on coverage analysis technique, the component finds rule-test correlation by executing all of test cases at once and monitoring which rules are evaluated for a given test case. For the test selection technique based on recorded request evaluation, the component logs all requests issued from given test cases.

For change impact analysis, the component leverages an existing policy verification tool called Margrave [FKMT05]. Given two versions of policies, P and P', the component uses the generic APIs of Margrave to print out all the changed policy behaviors. Our first and second test-selection techniques analyze the results of Margrave and find the rules R_i impacted by policy changes. Our third test-selection technique does not require change impact analysis. The component evaluates requests against P and P' to reveal different evaluated decisions.

8.4.1 Experiments

This section presents the experiments that we conducted to evaluate our proposed selection techniques of regression tests for security policy evolution. We carried out our evaluation on a PC, running Windows 7 with Intel Core i5, 2410 Mhz processor, and 4 GB of RAM. Given a program

Table 8.1: Subjects used in our evaluation

Subject	LOC	# Test Cases	# Security Test Cases
LMS	3749	46	29
VMS	3734	52	10
ASMS	7836	93	91
Average	5106	64	43

Table 8.2: Policy statistics used in our subjects

Subject	Attributes				# Rules	Policy Coverage		
	# Sub	# Act	# Res	# Cond		# Cov	# Not-Cov	% Cov
LMS	6	10	3	4	42	42	0	100
VMS	7	15	3	3	106	13	106	12
ASMS	8	11	5	4	129	109	21	83
Average	7	12	4	4	93	55	42	65

code interacting with a policy P, the regression simulator implements its modified policy P' by changing/adding/removing random rules in P. Our test-selection techniques select test cases that may reveal different behaviors in program code impacted by policy changes between P and P'. We measure effectiveness and efficiency of our test selection techniques in terms of the number of selected test cases and elapsed time, respectively. In this section, we first describe the objectives, measures, and instrumentation. We next present and discuss the evaluation results.

8.4.2 Subjects

Table 8.1 summarizes the basic characteristics of our subjects [MLTB09] that were used in chapter 7. The first column shows the subject names. Columns 2-4 show the lines of code, and the numbers of test cases and security test cases. Security test cases refer to test cases that issue at least one request to the PDP. Table 8.1 shows that our subjects include about 64 test cases and 43 security test cases on average. In particular, `VMS` is equipped with only 10 security tests (out of 52 test cases), which is smaller than those of other subjects.

Table 8.2 summarizes the basic statistics of policies used in our subjects. The first column shows the subject names. Columns 2-4 show the numbers of subjects, actions, resources, and conditions within attribute column group, respectively. Column 5 shows the number of rules for each policy. Columns 6-7 show the numbers of covered rules with test cases, not-covered rules with test cases, and the percentage of coverage within policy coverage column group, respectively. The largest policy is related to `ASMS`, which consists of 129 rules. For coverage, we measure which rules are evaluated (i.e., covered) with existing test cases under test. We observe that `LMS`, `VMS`, and `ASMS` achieve 100%, 12%, and 83% policy coverage with given test cases, respectively. `LMS`, `VMS`, and `ASMS` may represent subjects with high, low, and medium policy coverage, respectively.

Moreover, while VMS consists of 106 rules, VMS is equipped with only 10 security test cases that result in low policy coverage.

8.4.3 Objectives and measures

In the evaluation, we intend to address the following research questions:

- RQ1: How high percentage of test cases (from an existing test suite) is reduced by our test selection techniques? This question helps to show that our techniques can reduce the cost of regression testing. We also show how many changed policy behaviors are covered with our selected test cases.
- RQ2: How much time do our techniques take to conduct test selection by given subjects? This question helps to compare performance of our techniques by measuring their efficiency.

To help answer these questions, we collect few test metrics to show the effectiveness and the efficiency of our test selection techniques. The following metrics are measured for each subject under test interacting with each modified policy and each technique.

- *Test reduction percentage.* Given a policy and its modified policy, the test reduction percentage is the number of selected test cases for regression testing divided by the number of security test cases.
- *The number of rules impacted by policy changes (#CT).* Given a policy and its modified policy, this metric shows the number of rules impacted by policy changes.
- *The coverage of rules impacted by policy changes (#COV).* Given a policy and its modified policy, this metric shows the number of impacted rules covered with existing test cases.
- *The percentage of coverage of rules impacted by policy changes (%COV).* Given a policy and its modified policy, the percentage of coverage of rules impacted by policy changes is $\frac{\#COV}{\#CT}$.
- *Elapsed time.* The elapsed time is time (measured in milliseconds) elapsed for each step during the test selection process. To assess effectiveness of test reduction (RQ1), we use a test reduction percentage metric with additional $\#CT$, $\#COV$, $\%COV$ metrics. To assess efficiency of test selection (RQ2), we use elapsed time.

For each policy change, the regression simulator first randomly chooses one of regression types from RMR, RA, and RDC and a rule, and changes the rule with a selected type. We configured the regression simulator to inject 5, 10, 15, 20, and 25 changes in a policy, respectively. Our evaluation is repeated 12 times in order to avoid the impact of randomness of changes.

For our test selection approach, we use test selection based on mutation analysis (Mut-$Selection$), test selection based on coverage analysis (Cov-$Selection$), and test selection based

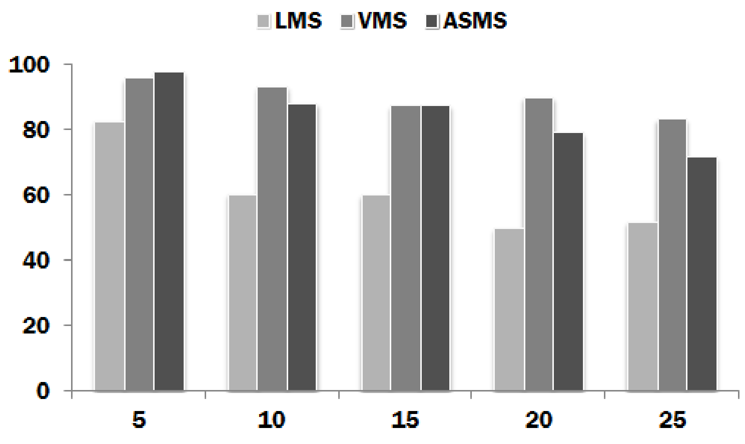


Figure 8.5: Regression test selection results (test reduction) for our subjects and each modified version

Y axis denotes the percentage of test reduction. X axis denotes the number of policy changes in the modified policy of our subjects.

on recorded request evaluation (*Req-Selection*) described in Section 8.3. To measure the effectiveness of our three techniques, we measure how many test cases are selected for regression testing.

We next compare the efficiency of our three techniques. The objective of this evaluation is to investigate how our three test selection techniques impact performance. We measure elapsed time to conduct rule-test correlation, change impact analysis, and test selection for each technique. For *Mut-Selection* and *Cov-Selection*, note that we require rule-test correlation analysis and change impact analysis, which should be done before actual test selection process.

8.4.4 Results

To answer $RQ1$, we measure test reduction percentage. The goal of this research question is to show the reduction in the number of test cases that are selected using our techniques. Figure 8.5 shows the results of test reduction percentage for the three subjects and modified policies. We observe that our techniques achieve 51%~97% of test reduction for a modified policy with 5~25 changed policy behaviors. Such test reduction may reduce a significant cost in terms of test execution time for regression testing. We also observe that all of our test techniques show the same set of selected test cases. This observation shows that all of our techniques are effective to select every test case impacted by policy changes.

To answer $RQ2$, we measure elapsed time. The goal of this research question is to compare efficiency of our three test-selection techniques. Table 8.3 shows the evaluation results for the three subjects and each technique. For *Mut-Selection* and *Cov-Selection*, the table shows the elapsed time of rule-test correlation ("Rule-Test"), change-impact-analysis ("CIA"), and test selection ("Test Selection"), respectively. For *Req-Selection*, the table shows the elapsed time of

Table 8.3: Elapsed time for each step of test selection technique, and each policy

Subject	Mut-Selection			Cov-Selection			Req-Selection	
	Rule-Test	CIA	TSel	Rule-Test	CIA	TSel	Req Collection	TSel
LMS	70496	2083	4	5214	2083	4	2096	2
VMS	19771	3333	1	7506	3333	1	1873	2
ASMS	118248	4000	11	22423	4000	11	1064	21
Average	69505	3139	5	11714	3139	5	1678	8

Table 8.4: Coverage results of test cases selected for changed policy behaviors for each policy

Subject	Regression - 5			Regression - 15			Regression - 20			Regression - 25		
	# CT	# Cov	% Cov	# CT	# Cov	% Cov	# CT	# Cov	% Cov	# CT	# Cov	% Cov
LMS	5.0	2.4	48.3	13.0	6.3	48.1	17.4	7.8	44.5	20.7	8.8	42.7
VMS	4.9	0.4	8.5	13.8	1.3	9.6	18.9	1.1	5.7	23.8	1.8	7.4
ASMS	5.0	1.9	38.3	14.4	7.3	50.9	18.7	9.4	50.4	23.3	12.1	51.8
Average	4.97	1.58	31.71	13.75	4.97	36.19	18.33	6.08	33.56	22.58	7.56	33.97

request recording (“Req-Collection”), and test selection (“Test Sel”). Note that “Rule-Test” and “Req-Collection” are preliminary steps, which can be done before test selection. Note that, as both Mut-$Selection$ and Cov-$Selection$ have the same change impact analysis step, “CIA” column shows the same elapsed time.

We observe that rule-test correlation TR_1 of Mut-$Selection$ takes significantly more time than TR_2 of Cov-$Selection$. TR_2 and TR_1 take 11714 milliseconds and 69505 milliseconds on average, respectively. This result is expected as TR_2 executes existing test cases only once but TR_1 executes existing test cases for $2 \times n$ times where n is the number of rules in a policy under test. We calculate total elapsed time for each technique. We observe that the total elapsed time of Req-$Selection$ is 43 and 8 times faster than those of Mut-$Selection$ and Cov-$Selection$, respectively. As a result, Req-$Selection$ is the most efficient in terms of elapsed time.

Table 8.4 shows the coverage results of selected test cases for the three subjects and the number of changes injected into the policy. Given a policy P, “Regression - N” denotes a group of modified policies where N denotes the number of changes injected into P. For each subject and “Regression - N”, the table shows changed policy behavior count (“# CT”), changed policy behavior coverage count (“# Cov”), and changed policy coverage percentage (“% Cov”). “# CT” is equal or less than N because a modified policy may not reflect some of injected policy changes. The reason is because injected policy changes may not change policy behaviors semantically. One example is that policy behaviors do not change when RDC are injected to the same rule twice. We observe that the changed policy behavior coverage percentages across all subjects are quite similar. Our techniques achieve 31.71%~36.19% of changed policy behavior coverage percentage. Unfortunately the changed policy behavior coverage percentage is still low when considering revealing faults caused by policy changes.

8.4.5 Discussion

We need to investigate the instrumentation effects that can bias our results such as faults in Sun's PDP, faults in Margrave, and faults in our implementation. In particular, our proposed regression model based on `RMR`, `RA`, and `RDC` modifies policy elements such as subject, resource, action, and condition attributes. While additional regression model to modify various policy attributes could simulate various policy modifications, our regression model still can simulate any kinds of policy changes using three policy modification types together. `RMR` and `RDC` may not inject specific changes of attribute elements in a rule since these modifications apply to rules (instead of attribute elements in rules). However, `RA` could be more flexible to simulate adding/changing any attributes in rules that the developers would like to modify in practice. For example, to change specific attribute elements in rules rs, we may replace rs by removing rs and add new rules rs' reflected by the specific changes at the same location.

8.5 Conclusion

In this chapter, we have proposed three test selection techniques for access control policy evolution. We have conducted evaluation with few test metrics to measure the effectiveness of our approach and the efficiency of our three test selection techniques. The evaluation results demonstrated that our approach is effective to select test cases for test reduction. Among the proposed three test-selection techniques, the evaluation results demonstrated that the technique based on recorded request evaluation is the most efficient compared with the first two techniques. To the best of our knowledge, this work pioneers regression test-selection in the context of policy evolution. Our approach aims at achieving a trade-off between security and regression testing costs. We believe that trade-offs have to be enacted at all the level of software development including testing phases. For cloud-based software, handling these trade-offs is more crucial to reduce time and costs in each validation of a new version of cloud-based software.

Chapter 9

Conclusion

Contents

This chapter starts by summarizing the objectives of this thesis in section 9.1. Section 9.2 delves into the details of the contributions presented throughout the thesis. Section 9.3 goes through the limitations of this work and finally section 9.4 points out future research directions that have emerged from our current work.

9.1 Summary

During the last years, cloud computing phenomenon has drastically impacted the business model of IT companies. Cloud computing reflects an extension of distributed software development models. The outsourcing features of cloud computing enable to implement a cost effective model in which companies resort to external hosting to manage their running services, avoiding the burden of software maintenance. This research focuses on quality attributes trade-offs in cloud-based software. We have conducted literature review in the optimization of cloud-based software and we have identified a lack in the literature in handling cloud self-adaptations and run-time reasoning.

To build a decision support method that addresses trade-offs in the context of cloud-based software, we have built a *runtime reasoning layer* in cloud infrastructure to continuously optimize software deployment from both the cloud customers and provider perspectives. To promote efficient services delivery in cloud paradigm, we have proposed an algorithm that operates on the top of MOEAs to improve the efficiency of our optimization process. Additionally, we have pointed out the importance of handling design *trade-offs at the level of guest applications*. Besides our approach to optimally deploy applications in the cloud infrastructure, we also advocate that trade-offs have to be addressed at the level of guest applications and not only at deployment phase.

To improve trade-offs at the level of guest applications, we have focused on the security quality attribute and more particularly on *access control concerns*. We have advocated XACML policy-based systems for the design of guest applications as they provide a certain trade-off between security and flexibility in terms of authorizations management. We have *extended current XACML policy-based systems* to support a wide range of *usage control scenarios*. Refactoring techniques on top of XAML policies, have been used to achieve a trade-off between security and performance at the level of XACML policy-based systems. Finally, we pointed out that trade-offs analysis have to be carried out during all the stages of software development and particularly in the *testing phase*. In the context of policy-based systems evolution, we have proposed test selection techniques to achieve a trade-off between regression testing costs and security policy evolution.

9.2 Revisited contributions

To take full advantage of cloud computing paradigm, this thesis pursues the objective of investigating the trade-offs issues at the level of cloud-based software in order to help both users and service providers to maximize software quality attributes and to minimize costs. The aim of the research reported in this dissertation is to design and to continuously optimize cloud hosted software. This approach combines advanced bio-inspired evolutionary approaches, models@run.time paradigm and policy-based systems. The different contributions that have been proposed throughout this thesis are summarized as follows:

A) *We built a decision module to optimize cloud-based software.* This module offers reasoning capabilities and takes into consideration design time and runtime information to re-evaluate the system at runtime. We evaluate the approach based on a case study defined with EBRC, a cloud provider in Luxembourg. We have modeled trade-offs in the deployment of cloud-based software as a multi-objective optimization problem and we have used evolutionary algorithms to find near optimal cloud configurations solutions. This contribution investigates the properties of dynamic adaptive systems to propose an abstraction of a cloud infrastructure and a model driven optimization method that achieves optimization at the level of cloud-based software deployment.

B) *We have proposed a hyper-heuristic that operates at the top of the optimization algorithms that have been used in the previous contribution to enhance the effectiveness of the optimization approach.* Elastic intrinsic features of cloud computing impose severe requirements on the optimization process that is used to resolve conflicting objectives at the level of cloud-based software deployment. We have validated the effectiveness of the proposed hyper-heuristic using standard multi-objective evolutionary algorithms.

Our contributions encompass trade-offs handling at the level of the hosting phase to include methods that improve trade-offs design at the level of guest applications. For the trade-offs management at the level of guest applications, we have focused particularly on security quality attribute and more specifically on XACML policy-based systems. In what follows, we summarize our contributions to handle trade-offs at the level of guest applications:

C) *Extension of XACML model for a better obligations support.* For the systems that respect XACML architectural design, we have identified some limitations in the support of XACML to handle obligations. To widen the range of usage control scenarios that can be supported by XACML policy, we have proposed an extension at the level of XACML architectural model and at the level of the policy language that improves XACML obligations support. *OB-XACML*, our proposed model, can be easily adopted since it does not require several changes at the level of XACML reference model.

D) *Refactoring technique for performance versus access control trade-offs*. Performance is a crucial requirement that has to be considered in the management of XACML policy-based systems. We have proposed a refactoring technique at the level of XACML policies to achieve a trade-off between security and performance. Our approach has been evaluated on three Java policy-based systems and has experimentally demonstrated the effectiveness of the proposed refactoring techniques in enhancing access control request processing.

E) *Promoting multi-layers optimization.* Trade-offs analysis has to be achieved in all the stages of software development. For evolving policy-based systems, software engineers have to re-run security tests whenever the security policy is updated. We have proposed a regression-test selection approach that ensures security policy evolution while reducing regression test selection costs.

9.3 Limitations

This section gets insight into the main limitations that we have identified in our work.

Cloud-based platforms

Up to this stage, we have limited our work to an experimentation that considers only Kevoree as a models@run.time platform for cloud nodes management. To gain more confidence in the efficiency of our reasoning layer, we believe that experiments have to be conducted in a real production environment. More precisely, besides using Kevoree nodes, another pertinent step to test our approach would be to test our optimization approach with common commercial cloud nodes such as Amazon and Rackspace nodes. Another step to concretize our work is to integrate our reasoning layer the top of open source hypervisors such as Xen open source virtualization platforms[1]. The reasoning engine can be tested in a cloud bursting tool that helps cloud customers clients to choose a cloud provider between a set of potential cloud providers. The criteria to choose between best cloud providers can be based on providers best alternatives offers in terms of cost and services quality attributes.

Comparison with well known optimization benchmarks

Sputnik hyper-heuristic that we have proposed in chapter 5 operates on the top of MOEAs algorithms to improve the process of continuous optimization at the level of the cloud infrastructure. This contribution can be categorized as a search-based contribution. To generalize obtained results with *Sputnik* on other optimization problems, we need to test this hyper-heuristic in common optimization benchmarks that include common optimization problems described in the literature such as DTLZ problems and ZDT problems[2].

Obligations in the selected policy-based systems

Our extension to improve the support of obligations in XACML language and architecture includes only obligations with deadline. To have a comprehensive support of obligations in XACML, we need to enhance the support of obligations in XACML by considering the different obligations types. These categories have been surveyed in [HBP05a, ECCB12b].

9.4 Future Work

Based on the previous presented contributions and the aforementioned limitations, this section introduces relevant research topics that will be explored in the near future.

[1]http://www.xenproject.org/
[2]http://www.tik.ee.ethz.ch/sop/download/supplementary/testproblems/

Usage of trade-offs analysis methods

This thesis addresses trade-offs in cloud based software and particularly security trade-offs. However, we recognize the lack of a validating methodology that analyzes and quantifies the different trade-offs achieved. A potential improvement of the current work is the usage of trade-offs analysis methods such as the Architecture Trade-off Analysis Method (ATAM) that evaluates a software architecture quality attributes based on predefined goals. ATAM can be used in different stages of software development life-cycle and is based on an iterative process that assesses to which extent quality attributes are able to achieve customers objectives.

Access control architecture for the cloud

We have chosen XACML as an architectural model for access control to regulate accesses in cloud-based software. XACML is based on an Attribute Based Access Control Model (ABAC) that captures several access control scenarios. However, an access control model that captures all the subtleties of cloud computing is still needed. Such model has to capture cloud providers reputation, identify risk measures that could be used before outsourcing data control to the cloud providers. New measures of trust between stakeholders that guide decision making process have also to be taken into consideration in an access control model dedicated for cloud-based applications.

Optimization of XACML-policy based system in the cloud

In this thesis, all the optimization results related to the policy-based systems have been explored in local settings. A follow-up work related to policy-based systems would be to propose an optimized distributed design of policy-based systems. By hosting an XACML policy-based system in a cloud environment, several questions can arise which are related to the optimal placement of PDPs, PEPs in a given cloud infrastructure. An interesting dimension that we would like to explore in the future is the placement of the different XACML components given that tenants are geographically distributed in several geographical zones. An optimization of an access control system based on XACML-based architecture would be to analyze the header of each tenant access control request, to identify its source location and to forward it to the closest PEP/PDP hosting point. These placement mechanisms have to be adaptive given the fact that localisation of tenants and of PEPs/PDPs in an XACML policy-based system changes over time.

Elastic load balancer

In the light of our reasoning approach, we plan to design and implement an auto-scaling system that defines scaling up/out rules to adjust resources provisioning to the workload or to the costs variations. This elastic load balancer has to explore the link between the PaaS layer and the IaaS layer to adjust cloud infrastructure resources based on the near optimal cloud configurations calculated by our reasoning engine. Based on our previous optimization findings, it would also be interesting to define prediction mechanisms so that the elastic load balancer can leverage the history of resources scaling events to adjust provisioning operations to the workload. We plan also to guide the behaviour of our elastic load balancer by keeping in mind the uncertainty related to costs and to the workload while defining the prediction mechanisms.

Exploring diversity as an optimization axis

Software diversity has already been used for aero-spacial software engineering to build high fault-tolerant systems. Some previous work has investigated the impact of diverse software to build resilient software systems. In [BDG+09], the authors have defined a set of twenty configuration rules for diversifying the implementation and/or the runtime environment of operating systems and applications before deployment phases. In [A+85], the authors have presented the impact of N-version approach on fault tolerance to study the impact of using different software versions on software faults. In the future, we are planning to explore the impact of diversity aspects on a cloud infrastructure. We will leverage the research outcomes of the DIVERSIFY[3] project which is a pioneering project that explores the impact of bio-diversity on the resilience of self-adaptive systems.

Stopping criteria in MOEAs

In the experiments that we have conducted in chapters 4 and 5, we have only considered time constraints or generations number as stopping criteria. Additional stopping criteria, based on defined acceptable values assigned to our fitness functions can also be explored. We plan also to define some stopping criteria that are based on the absence of improvements in our optimization process.

[3]http://diversify-project.eu/

BIBLIOGRAPHY

[A^{+}85] Algirdas Avizienis et al. The n-version approach to fault-tolerant software. *IEEE Trans. Software Eng.*, 11(12):1491–1501, 1985.

[ABGM09] Aldeida Aleti, Stefan Bjornander, Lars Grunske, and Indika Meedeniya. Archeopterix: An extendable tool for architecture optimization of aadl models. In *Model-Based Methodologies for Pervasive and Embedded Software, 2009. MOMPES'09. ICSE Workshop on*, pages 61–71. IEEE, 2009.

[AdCGS04] Glenn Ammons, Jong deok Choi, Manish Gupta, and Nikhil Swamy. Finding and removing performance bottlenecks in large systems. In *Proceedings of European Conference on Object-Oriented Programming*, pages 170–194, 2004.

[AF09] Abbas Afshar and Habib Fathi. Fuzzy multi-objective optimization of finance-based scheduling for construction projects with uncertainties in cost. *Engineering Optimization*, 41(11):1063–1080, 2009.

[AHK^{+}03] Paul Ashley, Satoshi Hada, Günter Karjoth, Calvin Powers, and Matthias Schunter. Enterprise privacy authorization language (epal 1.2). *Submission to W3C*, 2003.

[ASG07] Ines Alaya, Christine Solnon, and Khaled Ghedira. Ant colony optimization for multi-objective optimization problems. In *Proceedings of the 19th IEEE International Conference on Tools with Artificial Intelligence - Volume 01*, ICTAI '07, pages 450–457, 2007.

[AYKM14] Younis A Younis, Kashif Kifayat, and Madjid Merabti. An access control model for cloud computing. *Journal of Information Security and Applications*, 2014.

[BB09] Diana Berberova and Boyan Bontchev. Design of service level agreements for software services. In *Proceedings of the International Conference on Computer Systems and Technologies and Workshop for PhD Students in Computing*, pages 26:1–26:6, 2009.

[BBF09a] Gordon Blair, Nelly Bencomo, and Robert B France. Models@ run. time. *Computer*, pages 22–27, 2009.

[BBF09b] Gordon Blair, Nelly Bencomo, and Robert B France. Models@run.time. *Computer*, 42(10):22–27, 2009.

[BBG10] Rajkumar Buyya, James Broberg, and Andrzej M Goscinski. *Cloud computing: Principles and paradigms*, volume 87. John Wiley & Sons, 2010.

[BC95] Shumeet Baluja and Rich Caruana. Removing the genetics from the standard genetic algorithm. In *ICML*, pages 38–46, 1995.

[BCZ05] Daniel M Berry, Betty HC Cheng, and Jia Zhang. The four levels of requirements engineering for and in dynamic adaptive systems. In *11th International Workshop on Requirements Engineering Foundation for Software Quality (REFSQ)*, page 5, 2005.

[BDE$^+$05] Patrik Berander, Lars-Ola Damm, Jeanette Eriksson, Tony Gorschek, Kennet Henningsson, Per Jönsson, Simon Kågström, Drazen Milicic, Frans Mårtensson, Kari Rönkkö, et al. Software quality attributes and trade-offs. *Blekinge Institute of Technology*, 2005.

[BDG$^+$09] Alysson Bessani, Alessandro Daidone, Ilir Gashi, Rafael Obelheiro, Paulo Sousa, and Vladimir Stankovic. Enhancing fault/intrusion tolerance through design and configuration diversity. In *Proceedings of the 3rd Workshop on Recent Advances on Intrusion-Tolerant Systems–WRAITS*, volume 9, 2009.

[Béz04] Jean Bézivin. In search of a basic principle for model driven engineering. *Novatica Journal, Special Issue*, 5(2):21–24, 2004.

[BFJLT02] Benoit Baudry, Franck Fleurey, J Jezequel, and Yves Le Traon. Genes and bacteria for automatic test cases optimization in the. net environment. In *Software Reliability Engineering, 2002. ISSRE 2003. Proceedings. 13th International Symposium on*, pages 195–206. IEEE, 2002.

[BFJLT05] Benoit Baudry, Franck Fleurey, Jean-Marc Jézéquel, and Yves Le Traon. Automatic test case optimization: a bacteriologic algorithm. *ieee Software*, 22(2):76–82, 2005.

[BGW93] Daniel G Bobrow, Richard P Gabriel, and Jon L White. Clos in context-the shape of the design space. *Object Oriented Programming: The CLOS Perspective*, pages 29–61, 1993.

[BHK$^+$10] Edmund K Burke, Matthew Hyde, Graham Kendall, Gabriela Ochoa, Ender Özcan, and John R Woodward. A classification of hyper-heuristic approaches. In *Handbook of Metaheuristics*, pages 449–468. Springer, 2010.

[BKLW95] Mario Barbacci, Mark H Klein, Thomas A Longstaff, and Charles B Weinstock. Quality attributes. Technical report, DTIC Document, 1995.

[BKR09] Steffen Becker, Heiko Koziolek, and Ralf Reussner. The palladio component model for model-driven performance prediction. *Journal of Systems and Software*, 82(1):3–22, 2009.

[BLP76] E. D. Bell and J. L. La Padula. Secure computer system: Unified exposition and multics interpretation. MITRE Corporation, 1976.

[BMB93] David Beasley, RR Martin, and DR Bull. An overview of genetic algorithms: Part 1. fundamentals. *University computing*, 15:58–58, 1993.

[BMM$^+$] Benoit Baudry, Martin Monperrus, Cendrine Mony, Franck Chauvel, Franck Fleurey, Siobhán Clarke, et al. Diversify-ecology-inspired software evolution for diversity emergence. In *CSMR*.

[BMM12] Eirik Brandtzæg, Sébastien Mosser, and Parastoo Mohagheghi. Towards cloudml, a model-based approach to provision resources in the clouds. In *8th European Conference on Modelling Foundations and Applications (ECMFA)*, pages 18–27, 2012.

[BNE07] Nicola Beume, Boris Naujoks, and Michael Emmerich. Sms-emoa: Multiobjective selection based on dominated hypervolume. *European Journal of Operational Research*, 181(3):1653–1669, 2007.

[Bra11] Lucas Bradstreet. *The Hypervolume Indicator for Multi-objective Optimisation: Calculation and Use*. University of Western Australia, 2011.

[Bre12] Paul C Brebner. Is your cloud elastic enough?: performance modelling the elasticity of infrastructure as a service (iaas) cloud applications. In *Proceedings of the third joint WOSP/SIPEW international conference on Performance Engineering*, pages 263–266, 2012.

[BSK] Dirk Buche, Nicol N Schraudolph, and Petros Koumoutsakos. Accelerating evolutionary algorithms with gaussian process fitness function models. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 35(2):183–194.

[BT80] Aharon Ben-Tal. Characterization of pareto and lexicographic optimal solutions. In *Multiple Criteria Decision Making Theory and Application*, pages 1–11. Springer, 1980.

[BVB08] James Broberg, Srikumar Venugopal, and Rajkumar Buyya. Market-oriented griashley2003enterprise and utility computing: The state-of-the-art and future directions. *Journal of Grid Computing*, 6(3):255–276, 2008.

[BYV08] Rajkumar Buyya, Chee Shin Yeo, and Srikumar Venugopal. Market-oriented cloud computing: Vision, hype, and reality for delivering it services as computing utilities. In *High Performance Computing and Communications, 2008. HPCC'08. 10th IEEE International Conference on*, pages 5–13. IEEE, 2008.

[BZP$^+$10] C-P Bezemer, Andy Zaidman, Bart Platzbeecker, Toine Hurkmans, and A t Hart. Enabling multi-tenancy: An industrial experience report. In *Software Maintenance (ICSM), 2010 IEEE International Conference on*, pages 1–8. IEEE, 2010.

[CCBE13] Frédéric Cuppens, Nora Cuppens-Boulahia, and Yehia Elrakaiby. Formal specification and management of security policies with collective group obligations. *Journal of Computer Security*, 21(1):149–190, 2013.

[CdL10] Ildefonso Cases and Víctor de Lorenzo. Genetically modified organisms for the environment: stories of success and failure and what we have learned from them. *International microbiology*, 8(3):213–222, 2010.

[CdPL09] Lawrence Chung and Julio Cesar Sampaio do Prado Leite. On non-functional requirements in software engineering. In *Conceptual modeling: Foundations and applications*, pages 363–379. Springer, 2009.

[CF12] David W Chadwick and Kaniz Fatema. A privacy preserving authorisation system for the cloud. *Journal of Computer and System Sciences*, 78(5):1359–1373, 2012.

[CK08] Jason Crampton and Hemanth Khambhammettu. Delegation in role-based access control. *International Journal of Information Security*, 7(2):123–136, 2008.

[CL04] Carlos A Coello Coello and Gary B Lamont. *Applications of multi-objective evolutionary algorithms*, volume 1. 2004.

[CLN12] Sivadon Chaisiri, Bu-Sung Lee, and Dusit Niyato. Optimization of resource provisioning cost in cloud computing. *Services Computing, IEEE Transactions on*, 5(2):164–177, 2012.

[CLZ] Kleopatra Chatziprimou, Kevin Lano, and Steffen Zschaler. Towards a meta-model of the cloud computing resource landscape. In *1st International Conference Model-Driven Engineering and Software Development, Barcelona, Spain, Feb*, pages 19–21.

[CLZ13] Kleopatra Chatziprimou, Kevin Lano, and Steffen Zschaler. Towards a meta-model of the cloud computing resource landscape. In *MODELSWARD*, pages 111–116, 2013.

[CSK12] ByungRae Cha, JaeHyun Seo, and JongWon Kim. Design of attribute-based access control in cloud computing environment. In *Proceedings of the International Conference on IT Convergence and Security 2011*, pages 41–50. Springer, 2012.

[D⁺01] Kalyanmoy Deb et al. *Multi-objective optimization using evolutionary algorithms*, volume 2012. John Wiley & Sons Chichester, 2001.

[Dar59] Charles Darwin. *On the origins of species by means of natural selection*. 1859.

[Dau13] Erwan Daubert. *Adaptation et cloud computing: un besoin d'abstraction pour une gestion transverse*. PhD thesis, INSA de Rennes, 2013.

[DB10] Marco Dorigo and Mauro Birattari. Ant colony optimization. In *Encyclopedia of Machine Learning*, pages 36–39. Springer, 2010.

[DDKM08] Dylan Dawson, Ron Desmarais, Holger M Kienle, and Hausi A Müller. Monitoring in adaptive systems using reflection. In *Proceedings of the 2008 international workshop on Software engineering for adaptive and self-managing systems*, pages 81–88. ACM, 2008.

[DDLS01a] Nicodemos Damianou, Naranker Dulay, Emil Lupu, and Morris Sloman. The ponder policy specification language. In *Policies for Distributed Systems and Networks*, pages 18–38. 2001.

[DDLS01b] Nicodemos Damianou, Naranker Dulay, Emil Lupu, and Morris Sloman. The ponder policy specification language. In *Proceedings of the International Workshop on Policies for Distributed Systems and Networks*, POLICY '01, pages 18–38, 2001.

[Deb14] Kalyanmoy Deb. Multi-objective optimization. In *Search methodologies*, pages 403–449. Springer, 2014.

[DMM03] Kalyanmoy Deb, Manikanth Mohan, and Sikhar Mishra. A fast multi-objective evolutionary algorithm for finding well-spread pareto-optimal solutions. *KanGAL report*, 2003002, 2003.

[DNdL⁺11] Yuri Demchenko, Canh Ngo, Cees de Laat, Tomasz Wiktor Wlodarczyk, Chunming Rong, and Wolfgang Ziegler. Security infrastructure for on-demand provisioned cloud infrastructure services. In *Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on*, pages 255–263. IEEE, 2011.

[DÖB11] John H Drake, Ender Özcan, and Edmund K Burke. Controlling crossover in a selection hyper-heuristic framework. *School of Computer Science, University of Nottingham, Tech. Rep. No. NOTTCS-TR-SUB-1104181638-4244*, 2011.

[DPAM00] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and T. Meyarivan. A fast elitist multi-objective genetic algorithm: Nsga-ii. pages 182–197, 2000.

[DPAM02] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *Trans. Evol. Comp*, 6(2), April 2002.

[DWD12] Tien Tuan Anh Dinh, Wang Wenqiang, and Anwitaman Datta. City on the sky: Extending xacml for flexible, secure data sharing on the cloud. *Journal of Grid Computing*, 10(1):151–172, 2012.

[DWS12] Brian Dougherty, Jules White, and Douglas C Schmidt. Model-driven auto-scaling of green cloud computing infrastructure. *Future Generation Computer Systems*, 28(2):371–378, 2012.

[DXX09] Chen Danwei, Huang Xiuli, and Ren Xunyi. Access control of cloud service based on ucon. In *Cloud Computing*, pages 559–564. Springer, 2009.

[ECCB12a] Yehia Elrakaiby, Frédéric Cuppens, and Nora Cuppens-Boulahia. Formal enforcement and management of obligation policies. *Data Knowl. Eng.*, 2012.

[ECCB12b] Yehia Elrakaiby, Frédéric Cuppens, and Nora Cuppens-Boulahia. Formal enforcement and management of obligation policies. *Data & Knowledge Engineering*, 71(1):127–147, 2012.

[EMT12] Yehia Elrakaiby, Tejeddine Mouelhi, and Yves Le Traon. Testing obligation policy enforcement using mutation analysis. In *ICST*, pages 673–680, 2012.

[ER97] AE Eiben and Zs Ruttkay. Constraint satisfaction problems. 1997.

[Erl08] Thomas Erl. *Soa: principles of service design*, volume 1. Prentice Hall Upper Saddle River, 2008.

[ES01] Russell C Eberhart and Yuhui Shi. Particle swarm optimization: developments, applications and resources. In *Evolutionary Computation, 2001. Proceedings of the 2001 Congress on*, volume 1, pages 81–86. IEEE, 2001.

[EY07] Golnaz Elahi and Eric Yu. A goal oriented approach for modeling and analyzing security trade-offs. In *Conceptual Modeling-ER 2007*, pages 375–390. Springer, 2007.

[FACDN13] Davide Franceschelli, Danilo Ardagna, Michele Ciavotta, and Elisabetta Di Nitto. Space4cloud: A tool for system performance and costevaluation of cloud systems. In *Proceedings of the 2013 international workshop on Multi-cloud applications and federated clouds*, pages 27–34. ACM, 2013.

[FBEK11] Funmilade Faniyi, Rami Bahsoon, Andy Evans, and Rick Kazman. Evaluating security properties of architectures in unpredictable environments: A case for cloud. In *Software Architecture (WICSA), 2011 9th Working IEEE/IFIP Conference on*, pages 127–136. IEEE, 2011.

[FCK95] David Ferraiolo, Janet Cugini, and D Richard Kuhn. Role-based access control (rbac): Features and motivations. In *Proceedings of 11th annual computer security application conference*, pages 241–48, 1995.

[FDP$^+$12] François Fouquet, Erwan Daubert, Noël Plouzeau, Olivier Barais, Johann Bourcier, and Jean-Marc Jézéquel. Dissemination of reconfiguration policies on mesh networks. In *DAIS*, pages 16–30, 2012.

[FFH13a] Sören Frey, Florian Fittkau, and Wilhelm Hasselbring. Search-based genetic optimization for deployment and reconfiguration of software in the cloud. In *Proceedings of the 2013 International Conference on Software Engineering*, pages 512–521, 2013.

[FFH13b] Sören Frey, Florian Fittkau, and Wilhelm Hasselbring. Search-based genetic optimization for deployment and reconfiguration of software in the cloud. In *Proceedings of the 2013 International Conference on Software Engineering*, pages 512–521, 2013.

[FFH13c] Sören Frey, Florian Fittkau, and Wilhelm Hasselbring. Search-based genetic optimization for deployment and reconfiguration of software in the cloud. In *Proceedings of the 2013 International Conference on Software Engineering*, pages 512–521, 2013.

[FKMT05] Kathi Fisler, Shriram Krishnamurthi, Leo A. Meyerovich, and Michael Carl Tschantz. Verification and change-impact analysis of access-control policies. In *Proc. 27th International Conference on Software Engineering (ICSE)*, pages 196–205, 2005.

[FKTZ05] Carlos M Fonseca, Joshua D Knowles, Lothar Thiele, and Eckart Zitzler. A tutorial on the performance assessment of stochastic multiobjective optimizers. In *Third International Conference on Evolutionary Multi-Criterion Optimization (EMO 2005)*, volume 216, page 240, 2005.

[FMF$^+$12] François Fouquet, Brice Morin, Franck Fleurey, Olivier Barais, Noël Plouzeau, and Jean-Marc Jézéquel. A dynamic component model for cyber physical systems. In *CBSE*, pages 135–144, 2012.

[FNM$^+$14a] François Fouquet, Grégory Nain, Brice Morin, Erwan Daubert, Olivier Barais, Noël Plouzeau, and Jean-Marc Jézéquel. Kevoree modeling framework (KMF): efficient modeling techniques for runtime use. *CoRR*, abs/1405.6817, 2014.

[FNM$^+$14b] Fouquet Francois, Grégory Nain, Brice Morin, Erwan Daubert, Olivier Barais, Noël Plouzeau, and Jean-Marc Jézéquel. Kevoree modeling framework (kmf): Efficient modeling techniques for runtime use. *arXiv preprint arXiv:1405.6817*, 2014.

[Fou13] François Fouquet. *Kevoree: Model@ Runtime pour le développement continu de systèmes adaptatifs distribués hétérogènes*. PhD thesis, Université Rennes 1, 2013.

[FRMMne] E. Feller, C. Rohr, D. Margery, and C. Morin. Energy management in iaas clouds: A holistic approach. In *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, pages 204–212, June.

[FSG$^+$01] David F. Ferraiolo, Ravi S. Sandhu, Serban I. Gavrila, D. Richard Kuhn, and Ramaswamy Chandramouli. Proposed NIST standard for role-based access control. *ACM Transactions on Information and System Security*, 4(3):224–274, 2001.

[FZRL08] Ian Foster, Yong Zhao, Ioan Raicu, and Shiyong Lu. Cloud computing and grid computing 360-degree compared. In *Grid Computing Environments Workshop, 2008. GCE'08*, pages 1–10. Ieee, 2008.

[Gav02] Marco Gavanelli. An algorithm for multi-criteria optimization in csps. In *ECAI*, volume 2, pages 136–140, 2002.

[GC00] Mitsuo Gen and Runwei Cheng. *Genetic algorithms and engineering optimization*, volume 7. John Wiley & Sons, 2000.

[GD91] David E Goldberg and Kalyanmoy Deb. A comparative analysis of selection schemes used in genetic algorithms. *Urbana*, 51:61801–2996, 1991.

[GKÖ13] IA Güney, Gürhan Küçük, and Ender Özcan. Hyper-heuristics for performance optimization of simultaneous multithreaded processors. In *Information Sciences and Systems 2013*, pages 97–106, 2013.

[Glo89] Fred Glover. Tabu search-part i. *ORSA Journal on computing*, 1(3):190–206, 1989.

[Gol89] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1989.

[Gol90] David E Goldberg. E.(1989). genetic algorithms in search, optimization and machine learning. *Reading: Addison-Wesley*, 1990.

[GPDB14] Mateusz Guzek, Johnatan E Pecero, Bernabé Dorronsoro, and Pascal Bouvry. Multi-objective evolutionary algorithms for energy-aware scheduling on distributed computing systems. *Applied Soft Computing*, 24:432–446, 2014.

[GSH$^+$07] Chang Jie Guo, Wei Sun, Ying Huang, Zhi Hu Wang, and Bo Gao. A framework for native multi-tenancy application development and management. In *E-Commerce Technology and the 4th IEEE International Conference on Enterprise Computing, E-Commerce, and E-Services, 2007. CEC/EEE 2007. The 9th IEEE International Conference on*, pages 551–558. IEEE, 2007.

[Hak10] Deriving operating policies for multi-objective reservoir systems: Application of self-learning genetic algorithm. *Applied Soft Computing*, 10(4):1151 – 1163, 2010.

[Har07] Mark Harman. The current state and future of search based software engineering. In *2007 Future of Software Engineering*, pages 342–357. IEEE Computer Society, 2007.

[HBP05a] Manuel Hilty, David Basin, and Alexander Pretschner. On obligations. In *Computer Security–ESORICS 2005*, pages 98–117. Springer, 2005.

[HBP05b] Manuel Hilty, David A. Basin, and Alexander Pretschner. On obligations. In *ESORICS*, pages 98–117, 2005.

[HC01] George T. Heineman and William T. Councill, editors. *Component-based Software Engineering: Putting the Pieces Together*. 2001.

[Hel09] Matt Helsley. Lxc: Linux container tools. *IBM devloperWorks Technical Library*, 2009.

[HHK$^+$10] Michael Hauck, Matthias Huber, Markus Klems, Samuel Kounev, Jörn Müller-Quade, Alexander Pretschner, Ralf Reussner, and Stefan Tai. Challenges and opportunities of cloud computing. *Karlsruhe Reports in Informatics*, 19, 2010.

[HK03] Limin Han and Graham Kendall. An investigation of a tabu assisted hyper-heuristic genetic algorithm. In *Evolutionary Computation, 2003. CEC'03. The 2003 Congress on*, volume 3, pages 2230–2237. IEEE, 2003.

[HKK07] Julia Handl, Douglas B Kell, and Joshua Knowles. Multiobjective optimization in bioinformatics and computational biology. *IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB)*, 4(2):279–292, 2007.

[HLS$^+$12] M Harman, Kiran Lakhotiaa, Jeremy Singerb, David R Whiteb, and Shin Yooa. Cloud engineering is search based optimization too'. *Journal of Systems and Software*, 2012.

[HLS$^+$13] Mark Harman, Kiran Lakhotia, Jeremy Singer, David R White, and Shin Yoo. Cloud engineering is search based software engineering too. *Journal of Systems and Software*, 86(9):2225–2241, 2013.

[HMHX07] Vincent C. Hu, Evan Martin, JeeHyun Hwang, and Tao Xie. Conformance checking of access control policies specified in XACML. In *Proc. 1st IEEE International Workshop on Security in Software Engineering (IWSSE)*, 2007.

[HN08] Christian Horoba and Frank Neumann. Benefits and drawbacks for the use of epsilon-dominance in evolutionary multi-objective optimization. In *Proceedings of the 10th annual conference on Genetic and evolutionary computation*, pages 641–648, 2008.

[HRFMF13] Keiko Hashizume, David G Rosado, Eduardo Fernández-Medina, and Eduardo B Fernandez. An analysis of security issues for cloud computing. *Journal of Internet Services and Applications*, 4(1):1–13, 2013.

[HW04] Jamie Hillman and Ian Warren. An open framework for dynamic reconfiguration. In *Proceedings of the 26th International Conference on Software Engineering*, pages 594–603. IEEE Computer Society, 2004.

[IBM] *IBM. 2003. An architectural blueprint for autonomic computing. Tech. rep., IBM.*

[ILFL12] Sadeka Islam, Kevin Lee, Alan Fekete, and Anna Liu. How a consumer can measure elasticity for cloud platforms. In *Proceedings of the third joint WOSP/SIPEW international conference on Performance Engineering*, pages 85–96. ACM, 2012.

[IND06] H. Ishibuchi, Yusuke Nojima, and T. Doi. Comparison between single-objective and multi-objective genetic algorithms: Performance comparison and performance measures. In *Evolutionary Computation, 2006. CEC 2006. IEEE Congress on*, 2006.

[ISTN09] H. Ishibuchi, Y. Sakane, N. Tsukamoto, and Yusuke Nojima. Evolutionary many-objective optimization by nsga-ii and moea/d with large populations. In *Systems, Man and Cybernetics, 2009. SMC 2009. IEEE International Conference on*, pages 1758–1763, Oct 2009.

[IYW06] Keith Irwin, Ting Yu, and William H. Winsborough. On the modeling and analysis of obligations. In *Proceedings of the 13th ACM conference on Computer and communications security*, CCS'06, pages 134–143, 2006.

[JCH$^+$05] Wan-Rong Jih, Shao-you Cheng, Jane YJ Hsu, Tse-Ming Tsai, et al. Context-aware access control in pervasive healthcare. *Computer Science and Information Engineering, National Taiwan University, Taiwan. jih@ agents. csie. ntu. edu. tw.,{r93070, yjhsu}@ csie. ntu. edu. tw*, 2005.

[Jen03] Mikkel T Jensen. Reducing the run-time complexity of multiobjective eas: The nsga-ii and other algorithms. *Evolutionary Computation, IEEE Transactions on*, 7(5):503–515, 2003.

[JGHO11] Sonia Jahid, Carl A. Gunter, Imranul Hoque, and Hamed Okhravi. MyABDAC: Compiling XACML Policies for Attribute-Based Database Access Control. In *Proceedings* *of the first ACM Conference on Data and Application Security and Privacy*, pages 97–108, 2011.

[JHJ$^+$10] Gueyoung Jung, Matti A Hiltunen, Kaustubh R Joshi, Richard D Schlichting, and Calton Pu. Mistral: Dynamically managing power, performance, and adaptation cost in cloud infrastructures. In *Distributed Computing Systems (ICDCS), 2010 IEEE 30th International Conference on*, pages 62–73, 2010.

[Jin05] Yaochu Jin. A comprehensive survey of fitness approximation in evolutionary computation. *Soft computing*, pages 3–12, 2005.

[Jos04] James BD Joshi. Access-control language for multidomain environments. *Internet Computing, IEEE*, 8(6):40–50, 2004.

[KBK$^+$99] Rick Kazman, Mario Barbacci, Mark Klein, S Jeromy Carriere, and Steven G Woods. Experience with performing architecture tradeoff analysis. In *Software Engineering, 1999. Proceedings of the 1999 International Conference on*, pages 54–63. IEEE, 1999.

[KBM$^+$03] Anas Abou El Kalam, Salem Benferhat, Alexandre Miège, Rania El Baida, Frédéric Cuppens, Claire Saurel, Philippe Balbiani, Yves Deswarte, and Gilles Trouessin. Organization based access control. In *Proceedings of 10th IEEE International Conference on Policies for Distributed Systems and Networks*, pages 120–131, 2003.

[KC03] Jeffrey O Kephart and David M Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, 2003.

[Ken02] Stuart Kent. Model driven engineering. In *Integrated Formal Methods*, pages 286–298. Springer, 2002.

[Kir84] Scott Kirkpatrick. Optimization by simulated annealing: Quantitative studies. *Journal of statistical physics*, 34(5-6):975–986, 1984.

[KKB$^+$98] Rick Kazman, Mark Klein, Mario Barbacci, Tom Longstaff, Howard Lipson, and Jeromy Carriere. The architecture tradeoff analysis method. In *Engineering of Complex Computer Systems, 1998. ICECCS'98. Proceedings. Fourth IEEE International Conference on*, pages 68–78. IEEE, 1998.

[KLM$^+$97] Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Videira Lopes, Jean-Marc Loingtier, and John Irwin. Aspect-oriented programming. In *ECOOP*, pages 220–242, 1997.

[KMK12] Rouven Krebs, Christof Momm, and Samuel Kounev. Architectural concerns in multi-tenant saas applications. In *CLOSER*, pages 426–431, 2012.

[KR06] Joshua B Kollat and Patrick M Reed. Comparing state-of-the-art evolutionary multi-objective algorithms for long-term groundwater monitoring design. *Advances in Water Resources*, 29(6):792–807, 2006.

[KSTI11] Herald Kllapi, Eva Sitaridi, Manolis M Tsangaris, and Yannis Ioannidis. Schedule optimization for data processing flows on the cloud. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, pages 289–300. ACM, 2011.

[KV10] Ronald L. Krutz and Russell Dean Vines. *Cloud Security: A Comprehensive Guide to Secure Cloud Computing*. Wiley Publishing, 2010.

[KYD03] V. Khare, X. Yao, and K. Deb. Performance scaling of multi-objective evolutionary algorithms. In *Proceedings of the 2Nd International Conference on Evolutionary Multi-criterion Optimization*, EMO'03, pages 376–390, 2003.

[Lam71] B. Lampson. Protection. In *Proceedings of the 5th Princeton Conference on Information Sciences and Systems*, 1971.

[Lan10] Ulrich Lang. Openpmf scaas: Authorization as a service for cloud & soa applications. In *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*, pages 634–643. IEEE, 2010.

[LB96] Leonard J LaPadula and D Elliott Bell. Secure computer systems: A mathematical model. *Journal of Computer Security*, pages 239–263, 1996.

[LBB13] Romain Laborde, François Barrère, and Abdelmalek Benzekri. Toward authorization as a service: a study of the xacml standard. In *Proceedings of the 16th Communications & Networking Symposium*, page 9. Society for Computer Simulation International, 2013.

[LBW] Jay Ligatti, Lujo Bauer, and David Walker. Edit automata: Enforcement mechanisms for run-time security policies. *International Journal of Information Security*, (1-2):2–16.

[LCB12] Ninghui Li, Haining Chen, and Elisa Bertino. On practical specification and enforcement of obligations. In *CODASPY*, pages 71–82, 2012.

[LCHX08] Alex X. Liu, Fei Chen, JeeHyun Hwang, and Tao Xie. XEngine: A fast and scalable XACML policy evaluation engine. In *Proceedings of International Conference on Measurement and Modeling of Computer Systems*, pages 265–276, 2008.

[LCWL09] Jim Zhanwen Li, John Chinneck, Murray Woodside, and Marin Litoiu. Fast scalable optimization to configure service systems having cost and quality of service constraints. In *Proceedings of the 6th international conference on Autonomic computing*, pages 159–168. ACM, 2009.

[LHM07] Kiran Lakhotia, Mark Harman, and Phil McMinn. A multi-objective approach to search-based test data generation. In *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 1098–1105. ACM, 2007.

[Lis10] Mario Lischka. Dynamic obligation specification and negotiation. In *NOMS*, pages 155–162, 2010.

[LMM10] Aliaksandr Lazouski, Fabio Martinelli, and Paolo Mori. Usage control in computer security: A survey. *Computer Science Review*, 4(2):81–99, 2010.

[LMM12] Aliaksandr Lazouski, Fabio Martinelli, and Paolo Mori. A prototype for enforcing usage control policies based on xacml. In *TrustBus*, pages 79–92, 2012.

[LMMM12] Aliaksandr Lazouski, Gaetano Mancini, Fabio Martinelli, and Paolo Mori. Usage control in cloud systems. In *Internet Technology And Secured Transactions, 2012 International Conference for*, pages 202–207. IEEE, 2012.

[LMS09] Coromoto León, Gara Miranda, and Carlos Segura. Hyperheuristics for a dynamic-mapped multi-objective island-based model. In *Distributed Computing, Artificial Intelligence, Bioinformatics, Soft Computing, and Ambient Assisted Living*, pages 41–49. 2009.

[LPL$^+$03] Markus Lorch, Seth Proctor, Rebekah Lepro, Dennis Kafura, and Sumit Shah. First experiences using xacml for access control in distributed systems. In *Proceedings of the 2003 ACM workshop on XML security*, pages 25–37. ACM, 2003.

[LRB$^+$08] Dan Lin, Prathima Rao, Elisa Bertino, Ninghui Li, and Jorge Lobo. Policy decomposition for collaborative access control. In *Proceedings of the 13th ACM Symposium on Access Control Models and Technologies*, pages 103–112, 2008.

[LSGM10] Xiao-Yong Li, Yong Shi, Yu Guo, and Wei Ma. Multi-tenancy based access control in cloud. In *Computational Intelligence and Software Engineering (CiSE), 2010 International Conference on*, pages 1–4. IEEE, 2010.

[LTDZ02] Marco Laumanns, Lothar Thiele, Kalyanmoy Deb, and Eckart Zitzler. Combining convergence and diversity in evolutionary multiobjective optimization. volume 10, pages 263–282, 2002.

[LTMPB08] Yves Le Traon, Tejeddine Mouelhi, Alexander Pretschner, and Benoit Baudry. Test-driven assessment of access control in legacy applications. In *Proceedings of the 2008 International Conference on Software Testing, Verification, and Validation*, pages 238–247, 2008.

[LZX13] Jian Lin, Li Zha, and Zhiwei Xu. Consolidated cluster systems for data centers in the cloud age: a survey and analysis. *Frontiers of Computer Science*, 7(1):1–19, 2013.

[MA04] R Timothy Marler and Jasbir S Arora. Survey of multi-objective optimization methods for engineering. *Structural and multidisciplinary optimization*, 26(6):369–395, 2004.

[MAB09] Indika Meedeniya, Aldeida Aleti, and Barbora Buhnova. Redundancy allocation in automotive systems using multi-objective optimisation. In *Symposium of Avionics/Automotive Systems Engineering (SAASE,Äô09), San Diego, CA*, 2009.

[Mah95] Samir W Mahfoud. Niching methods for genetic algorithms. *Urbana*, 51(95001), 1995.

[MBEB11] Michael Maurer, Ivan Breskovic, Vincent C Emeakaroha, and Ivona Brandic. Revealing the mape loop for the autonomic management of cloud infrastructures. In *Computers and Communications (ISCC), 2011 IEEE Symposium on*, pages 147–152. IEEE, 2011.

[MBJ+09] Brice Morin, Olivier Barais, J-M Jézéquel, Franck Fleurey, and Arnor Solberg. Models@ run. time to support dynamic adaptation. *Computer*, pages 44–51, 2009.

[MCM10] Fabio Martinelli Maurizio Colombo, Aliaksandr Lazouski and Paolo Mori. A proposal on enhancing xacml with continuous usage control features. In *Grids, P2P and Services Computing*, 2010.

[Meu02] Hervé Meunier. *Algorithmes évolutionnaires parallèles pour l'optimisation multi-objectif de réseaux de télécommunications mobiles*. PhD thesis, Lille 1, 2002.

[MFBT08] Tejeddine Mouelhi, Franck Fleurey, Benoit Baudry, and Yves Traon. A model-based framework for security policy specification, deployment and testing. In *Proceedings of 11th International Conference on Model Driven Engineering Languages and Systems*, pages 537–552, 2008.

[MG11] Peter Mell and Timothy Grance. The nist definition of cloud computing (draft). *NIST special publication*, 800(145):7, 2011.

[MGE13] Mashael Maashi, Kendall Graham, and Ozcan Ender. A choice function based hyper-heuristic for multi-objective optimization. 2013.

[Mis08] Philip L. Miseldine. Automated XACML policy reconfiguration for evaluation optimisation. In *Proceedings of 4th International Workshop on Software Engineering for Secure Systems*, pages 1–8, 2008.

[MKBR10] Anne Martens, Heiko Koziolek, Steffen Becker, and Ralf Reussner. Automatically improve software architecture models for performance, reliability, and cost using evolutionary algorithms. In *Proceedings of the first joint WOSP/SIPEW international conference on Performance engineering*, pages 105–116. ACM, 2010.

[ML85] Naftaly H. Minsky and Abe D. Lockman. Ensuring integrity by adding obligations to privileges. In *Proceedings of the 8th international conference on Software engineering*, ICSE '85, pages 92–102, 1985.

[MLH10] Ming Mao, Jie Li, and Marty Humphrey. Cloud auto-scaling with deadline and budget constraints. In *Grid Computing (GRID), 2010 11th IEEE/ACM International Conference on*, pages 41–48. IEEE, 2010.

[MLTB09] Tejeddine Mouelhi, Yves Le Traon, and Benoit Baudry. Transforming and selecting functional test cases for security policy testing. In *Proc. 2nd International Conference on Software Testing, Verification, and Validation (ICST)*, pages 171–180, 2009.

[Mor10] Brice Morin. *Leveraging models from design-time to runtime to support dynamic variability*. PhD thesis, Rennes 1, 2010.

[MSB$^+$12] Rahat Masood, Muhammad Awais Shibli, Muhammad Bilal, et al. Usage control model specification in xacml policy language. In *Computer Information Systems and Industrial Management*, pages 68–79. Springer, 2012.

[MSSS09] Said Marouf, Mohamed Shehab, Anna Squicciarini, and Smitha Sundareswaran. Statistics & clustering based framework for efficient XACML policy evaluation. In *Proceedings of 10th IEEE International Conference on Policies for Distributed Systems and Networks*, pages 118–125, 2009.

[MSST06] Philip K McKinley, Farshad A Samimi, Jonathan K Shapiro, and Chiping Tang. Service clouds: a distributed infrastructure for constructing autonomic communication services. In *Dependable, Autonomic and Secure Computing, 2nd IEEE International Symposium on*, pages 341–348. IEEE, 2006.

[MTB09] Tejeddine Mouelhi, Yves Le Traon, and Benoit Baudry. Transforming and selecting functional test cases for security policy testing. In *Proceedings of 2009 International Conference on Software Testing Verification and Validation*, pages 171–180, 2009.

[MX07a] Evan Martin and Tao Xie. Automated test generation for access control policies via change-impact analysis. In *Proc. 3rd International Workshop on Software Engineering for Secure Systems (SESS)*, pages 5–11, 2007.

[MX07b] Evan Martin and Tao Xie. A fault model and mutation testing of access control policies. In *Proc. 16th International Conference on World Wide Web (WWW)*, pages 667–676, 2007.

[MXY06a] Evan Martin, Tao Xie, and Ting Yu. Defining and measuring policy coverage in testing access control policies. In *Proceedings of 8th International Conference on Information and Communications Security*, pages 139–158, 2006.

[MXY06b] Evan Martin, Tao Xie, and Ting Yu. Defining and measuring policy coverage in testing access control policies. In *Proc. 8th International Conference on Information and Communications Security (ICICS)*, pages 139–158, 2006.

[Mye79] Glenford J. Myers. *Art of Software Testing*. John Wiley & Sons, Inc., New York, NY, USA, 1979.

[NBY09] Vivek Nallur, Rami Bahsoon, and Xin Yao. Self-optimizing architecture for ensuring quality attributes in the cloud. In *Software Architecture, 2009 & European Conference on Software Architecture. WICSA/ECSA 2009. Joint Working IEEE/IFIP Conference on*, pages 281–284. IEEE, 2009.

[New11] IDC New. It cloud services survey: top benefits and challenges. *Retrieved April*, 8, 2011.

[Nid96] Sarma R Nidumolu. Standardization, requirements uncertainty and software project performance. *Information & Management*, 31(3):135–150, 1996.

[NLA$^+$08] Antonio J Nebro, Francisco Luna, Enrique Alba, Bernabé Dorronsoro, Juan José Durillo, and Andreas Beham. Abyss: Adapting scatter search to multiobjective optimization. *Evolutionary Computation, IEEE Transactions on*, 12(4):439–457, 2008.

[NMDdL12] Canh Ngo, Peter Membrey, Yuri Demchenko, and Cees de Laat. Policy and context management in dynamically provisioned access control service for virtualized cloud infrastructures. In *Availability, Reliability and Security (ARES), 2012 Seventh International Conference on*, pages 343–349. IEEE, 2012.

[NPD$^+$10] Srijith K Nair, Sakshi Porwal, Theo Dimitrakos, Ana Juan Ferrer, Johan Tordsson, Tabassum Sharif, Craig Sheridan, Muttukrishnan Rajarajan, and Afnan Ullah Khan. Towards secure cloud bursting, brokerage and aggregation. In *Web Services (ECOWS), 2010 IEEE 8th European Conference on*, pages 189–196. IEEE, 2010.

[ÖBK08] Ender Özcan, Burak Bilgin, and Emin Erkan Korkmaz. A comprehensive analysis of hyper-heuristics. *Intelligent Data Analysis*, 12(1):3–23, 2008.

[OK96] Ibrahim H Osman and James P Kelly. Meta-heuristics: an overview. In *Meta-Heuristics*, pages 1–21. Springer, 1996.

[PHB06] Alexander Pretschner, Manuel Hilty, and David A. Basin. Distributed usage control. *Commun. ACM*, 49:39–44, 2006.

[PLA27] HH PLASKETT. Artificial transmutationof the gene. *tic*, 66(1699), 1927.

[Poe06] Iman Poernomo. The meta-object facility typed. In *Proceedings of the 2006 ACM symposium on Applied computing*, pages 1845–1849. ACM, 2006.

[Poo01] John D Poole. Model-driven architecture: Vision, standards and emerging technologies. In *Workshop on Metamodeling and Adaptive Object Models, ECOOP*, volume 2001, 2001.

[PPSSA] A. Ranabahu P. Patel and A. Sheth. Service level agreement in cloud computing. Technical report, Conference on Object Oriented Programming Systems Languages and Applications, Orlando, Florida, 2009, USA.

[PRS09] Pankesh Patel, Ajith Ranabahu, and Amit Sheth. Service level agreement in cloud computing. *Cloud Workshops at OOPSLA*, 2009.

[PS] Jaehong Park and Ravi Sandhu. The uconabc usage control model. *ACM Trans. Inf. Syst. Secur.*, pages 128–174.

[PS02a] Jaehong Park and Ravi Sandhu. Towards usage control models: beyond traditional access control. In *Proceedings of the seventh ACM symposium on Access control models and technologies*, SACMAT '02, pages 57–64, 2002.

[PS02b] Jaehong Park and Ravi Sandhu. Towards usage control models: beyond traditional access control. In *Proceedings of the seventh ACM symposium on Access control models and technologies*, pages 57–64. ACM, 2002.

[PS04] Jaehong Park and Ravi Sandhu. The ucon abc usage control model. *ACM Transactions on Information and System Security (TISSEC)*, 7(1):128–174, 2004.

[PWGB10] Suraj Pandey, Linlin Wu, Siddeswara Mayura Guru, and Rajkumar Buyya. A particle swarm optimization-based heuristic for scheduling workflow applications in cloud computing environments. In *Advanced Information Networking and Applications (AINA), 2010 24th IEEE International Conference on*, pages 400–407. IEEE, 2010.

[PYK$^+$10] Lucian Popa, Minlan Yu, Steven Y Ko, Sylvia Ratnasamy, and Ion Stoica. Cloudpolice: taking access control out of the network. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, page 7. ACM, 2010.

[RH96] Gregg Rothermel and Mary Jean Harrold. Analyzing regression test selection techniques. *IEEE Trans. Softw. Eng.*, 22:529–551, 1996.

[RNSE09] Himanshu Raj, Ripal Nathuji, Abhishek Singh, and Paul England. Resource management for isolation enhanced cloud services. In *Proceedings of the 2009 ACM workshop on Cloud computing security*, pages 77–84, 2009.

[RR03] Colin R Reeves and Jonathan E Rowe. *Genetic algorithms: principles and perspectives: a guide to GA theory*, volume 20. 2003.

[RUCH01] Gregg Rothermel, Roland H. Untch, Chengyun Chu, and Mary Jean Harrold. Prioritizing test cases for regression testing. *Software Engineering, IEEE Transactions on*, 27(10):929–948, 2001.

[S$^+$00] Richard Soley et al. Model driven architecture. *OMG white paper*, 308:308, 2000.

[San03] Ravi Sandhu. Good-enough security: Toward a pragmatic business-driven discipline. *IEEE Internet Computing*, 7(1):66–68, 2003.

[SBMP08] Dave Steinberg, Frank Budinsky, Ed Merks, and Marcelo Paternostro. *EMF: eclipse modeling framework*. Pearson Education, 2008.

[SCFY96a] Ravi S Sandhu, Edward J Coyne, Hal L Feinstein, and Charles E Youman. Role-based access control models. *Computer*, pages 38–47, 1996.

[SCFY96b] Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman. Role-based access control models. pages 38–47, 1996.

[SCZ03] François Siewe, Antonio Cau, and Hussein Zedan. A compositional framework for access control policies enforcement. In *Proceedings of the 2003 ACM workshop on Formal methods in security engineering*, FMSE '03, pages 32–42, 2003.

[SD08] SN Sivanandam and SN Deepa. *Genetic Algorithm Optimization Problems*. Springer, 2008.

[SdCdV01] Pierangela Samarati and Sabrina de Capitani di Vimercati. Access control: Policies, models, and mechanisms. In *FOUNDATIONS OF SECURITY ANALYSIS AND DESIGN (TUTORIAL LECTURES*, pages 137–196, 2001.

[SH06] Nigamanth Sridhar and Jason O Hallstrom. A behavioral model for software containers. In *Fundamental Approaches to Software Engineering*, pages 139–154. Springer, 2006.

[SOTJ12] Chun-Wei Seah, Yew-Soon Ong, Ivor W Tsang, and Siwei Jiang. Pareto rank learning in multi-objective evolutionary algorithms. In *Evolutionary Computation (CEC), 2012 IEEE Congress on*, pages 1–8, 2012.

[SP12] Balwinder Sodhi and TV Prabhakar. Cloud platforms: Impact on guest application quality attributes. In *Services Computing Conference (APSCC), 2012 IEEE Asia-Pacific*, pages 329–334. IEEE, 2012.

[SS94] Ravi S Sandhu and Pierangela Samarati. Access control: principle and practice. *Communications Magazine, IEEE*, 32(9):40–48, 1994.

[SSH12] Le Sun, Jaipal Singh, and Omar Khadeer Hussain. Service level agreement (sla) assurance for cloud services: a survey from a transactional risk perspective. In *Proceedings of the 10th International Conference on Advances in Mobile Computing & Multimedia*, pages 263–266, 2012.

[SSL14] Eduardo Segredo, Carlos Segura, and Coromoto Leon. Control of numeric and symbolic parameters with a hybrid scheme based on fuzzy logic and hyper-heuristics. In *Evolutionary Computation (CEC), 2014 IEEE Congress on*, pages 1890–1897. IEEE, 2014.

[SSSS10] Upendra Sharma, Prashant Shenoy, Sambit Sahu, and Anees Shaikh. Kingfisher: A system for elastic cost-aware provisioning in the cloud. *Dept. of CS, UMASS, Tech. Rep. UM-CS-2010-005*, 2010.

[SSSS11a] Upendra Sharma, Prashant Shenoy, Sambit Sahu, and Anees Shaikh. A cost-aware elasticity provisioning system for the cloud. In *Distributed Computing Systems (ICDCS), 2011 31st International Conference on*, pages 559–570. IEEE, 2011.

[SSSS11b] Upendra Sharma, Prashant Shenoy, Sambit Sahu, and Anees Shaikh. Kingfisher: Cost-aware elasticity in the cloud. In *INFOCOM, 2011 Proceedings IEEE*, pages 206–210. IEEE, 2011.

[SV13] Basem Suleiman and Srikumar Venugopal. Modeling performance of elasticity rules for cloud-based applications. In *Enterprise Distributed Object Computing Conference (EDOC), 2013 17th IEEE International*, pages 201–206. IEEE, 2013.

[TB03] Andrea Toffolo and Ernesto Benini. Genetic diversity as an objective in multi-objective evolutionary algorithms. *Evolutionary Computation*, 11(2):151–167, 2003.

[TC07] Ma Guadalupe Castillo Tapia and Carlos A Coello Coello. Applications of multi-objective evolutionary algorithms in economics and finance: A survey. In *IEEE congress on evolutionary computation*, volume 7, pages 532–539, 2007.

[TEADC] Slim Trabelsi, Adrien Ecuyer, Paul Cervera Y Alvarez, and Francesco Di Cerbo. Optimizing access control performance for the cloud.

[TJ12] Hassan Takabi and James BD Joshi. Policy management as a service: an approach to manage policy heterogeneity in cloud computing environment. In *System Science (HICSS), 2012 45th Hawaii International Conference on*, pages 5500–5508. IEEE, 2012.

[TJA10a] Hassan Takabi, James B. D. Joshi, and Gail-Joon Ahn. Security and privacy challenges in cloud computing environments. *IEEE Security and Privacy*, pages 24–31, 2010.

[TJA10b] Hassan Takabi, James BD Joshi, and Gail-Joon Ahn. Securecloud: Towards a comprehensive security framework for cloud computing environments. In *Computer Software and Applications Conference Workshops (COMPSACW), 2010 IEEE 34th Annual*, pages 393–398. IEEE, 2010.

[TLK01] Kay Chen Tan, Tong Heng Lee, and Eik Fun Khor. Evolutionary algorithms with dynamic population size and local exploration for multiobjective optimization. *Evolutionary Computation, IEEE Transactions on*, 5(6):565–588, 2001.

[TLS13] Bo Tang, Qi Li, and Ravi Sandhu. A multi-tenant rbac model for collaborative cloud services. In *Privacy, Security and Trust (PST), 2013 Eleventh Annual International Conference on*, pages 229–238. IEEE, 2013.

[TMMVL12] Johan Tordsson, Rubén S Montero, Rafael Moreno-Vozmediano, and Ignacio M Llorente. Cloud brokering mechanisms for optimized placement of virtual machines across multiple providers. *Future Generation Computer Systems*, 28(2):358–367, 2012.

[TN11] Slim Trabelsi and Akram Njeh. Policy implementation in xacml. In *Privacy and Identity Management for Life*, pages 355–374. Springer, 2011.

[UNR$^+$05] Rich Uhlig, Gil Neiger, Dion Rodgers, Amy L Santoni, Fernando CM Martins, Andrew V Anderson, Steven M Bennett, Alain Kagi, Felix H Leung, and Larry Smith. Intel virtualization technology. *Computer*, 38(5):48–56, 2005.

[vDKV00] Arie van Deursen, Paul Klint, and Joost Visser. Domain-specific languages: An annotated bibliography. *SIGPLAN Not.*, pages 26–36, 2000.

[VTM09] Hien Nguyen Van, Frederic Dang Tran, and J-M Menaud. Sla-aware virtual resource management for cloud infrastructures. In *Computer and Information Technology, 2009. CIT'09. Ninth IEEE International Conference on*, volume 1, pages 357–362. IEEE, 2009.

[VV99] David A Van Veldhuizen. *Multiobjective evolutionary algorithms: classifications, analyses, and new innovations*. PhD thesis, 1999.

[VVL98] David A Van Veldhuizen and Gary B Lamont. Multiobjective evolutionary algorithm research: A history and analysis. Technical report, Citeseer, 1998.

[VVL00] David A Van Veldhuizen and Gary B Lamont. Multiobjective evolutionary algorithms: Analyzing the state-of-the-art. *Evolutionary computation*, 8(2):125–147, 2000.

[WBN07] Tobias Wagner, Nicola Beume, and Boris Naujoks. Pareto-, aggregation-, and indicator-based methods in many-objective optimization. In *Evolutionary multi-criterion optimization*, pages 742–756, 2007.

[WRWR05] Barbara Weber, Manfred Reichert, Werner Wild, and Stefanie Rinderle. Balancing flexibility and security in adaptive process management systems. In *On the Move to Meaningful Internet Systems 2005: CoopIS, DOA, and ODBASE*, pages 59–76. Springer, 2005.

[WSYO11] Hiroshi Wada, Junichi Suzuki, Yuji Yamano, and Katsuya Oba. Evolutionary deployment optimization for service-oriented clouds. *Software: Practice and Experience*, 41(5):469–493, 2011.

[WVLY$^+$10] Lizhe Wang, Gregor Von Laszewski, Andrew Younge, Xi He, Marcel Kunze, Jie Tao, and Cheng Fu. Cloud computing: a perspective study. *New Generation Computing*, 28(2):137–146, 2010.

[XL08] Mingdi Xin and Natalia Levina. Software-as-a service model: Elaborating client-side adoption factors. 2008.

[YA11] Stephen S Yau and Ho G An. Software engineering meets services and cloud computing. *Computer*, 44(10):47–53, 2011.

[Yee04] Ka-Ping Yee. Aligning security and usability. *IEEE Security & Privacy*, 2(5):48–55, 2004.

[YG04] Bo Yuan and Marcus Gallagher. Statistical racing techniques for improved empirical evaluation of evolutionary algorithms. In *Parallel Problem Solving from Nature-PPSN VIII*, pages 172–181, 2004.

[YH07] Shin Yoo and Mark Harman. Pareto efficient multi-objective test case selection. In *Proceedings of the 2007 international symposium on Software testing and analysis*, pages 140–150. ACM, 2007.

[YLL13] Shin-Jer Yang, Pei-Ci Lai, and Jyhjong Lin. Design role-based multi-tenancy access control scheme for cloud services. In *Biometrics and Security Technologies (ISBAST), 2013 International Symposium on*, pages 273–279. IEEE, 2013.

[YLR$^+$03] Kamen Yotov, Xiaoming Li, Gang Ren, Michael Cibulskis, Gerald DeJong, Maria Garzaran, David Padua, Keshav Pingali, Paul Stodghill, and Peng Wu. A comparison of empirical and model-driven optimization. In *ACM SIGPLAN Notices*, volume 38, pages 63–76. ACM, 2003.

[YT05] Eric Yuan and Jin Tong. Attributed based access control (abac) for web services. In *Web Services, 2005. ICWS 2005. Proceedings. 2005 IEEE International Conference on*. IEEE, 2005.

[YT12] Zeratul Izzah Mohd Yusoh and Maolin Tang. Composite saas placement and resource optimization in cloud computing using evolutionary algorithms. In *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, pages 590–597. IEEE, 2012.

[ZBT07] Eckart Zitzler, Dimo Brockhoff, and Lothar Thiele. The hypervolume indicator revisited: On the design of pareto-compliant indicators via weighted integration. In *Evolutionary Multi-Criterion Optimization*, pages 862–876, 2007.

[Zha06] Xinwen Zhang. *Formal model and analysis of usage control*. PhD thesis, 2006.

[ZHO$^+$14] Yuanyuan Zhang, Mark Harman, Gabriela Ochoa, Guenther Ruhe, and Sjaak Brinkkemper. An empirical study of meta-and hyper-heuristic search for multi-objective release planning. *RN*, 14:07, 2014.

[ZQL$^+$11] Aimin Zhou, Bo-Yang Qu, Hui Li, Shi-Zheng Zhao, Ponnuthurai Nagaratnam Suganthan, and Qingfu Zhang. Multiobjective evolutionary algorithms: A survey of the state of the art. *Swarm and Evolutionary Computation*, 1(1):32–49, 2011.

[ZT99] Eckart Zitzler and Lothar Thiele. Multiobjective evolutionary algorithms: a comparative case study and the strength pareto approach. *Evolutionary Computation, IEEE Transactions on*, 3(4):257–271, 1999.

[ZTL$^+$03] Eckart Zitzler, Lothar Thiele, Marco Laumanns, Carlos M Fonseca, and Viviane Grunert Da Fonseca. Performance assessment of multiobjective optimizers: An analysis and review. *Evolutionary Computation, IEEE Transactions on*, 7(2):117–132, 2003.

[ZWH$^+$13] Zhenji Zhou, Lifa Wu, Zheng Hong, Zhao Liang, Lu Jun, Xu Sheng-Jun, Chen Dengfeng, Xiao Peng, Qu Peixin, Qu Xilong, et al. Context-aware access control model for cloud computing. *International Journal of Grid and Distributed Computing*, 6(6):1–12, 2013.

[ZZX$^+$10] Minqi Zhou, Rong Zhang, Wei Xie, Weining Qian, and Aoying Zhou. Security and privacy in cloud computing: A survey. In *Semantics Knowledge and Grid (SKG), 2010 Sixth International Conference on*, pages 105–112. IEEE, 2010.