Integrating predictive analysis in self-adaptive pervasive systems

Ivan Dario Paez Anaya

ANNÉE 2015

THÈSE / UNIVERSITÉ DE RENNES 1
sous le sceau de l'Université Européenne de Bretagne

pour le grade de

DOCTEUR DE L'UNIVERSITÉ DE RENNES 1

Mention : Informatique

École doctorale Matisse

présentée par

Ivan Dario Paez Anaya

préparée à l'unité de recherche IRISA
Institut de Recherche en Informatique et Systèmes Aléatoires
Composante Universitaire : ISTIC

In English :

Integrating Predictive Analysis in Self-Adaptive Pervasive Systems

En français :

Intégration de l'analyse prédictive dans des systèmes auto-adaptatifs

Thèse soutenue à Rennes
le 22 Septembre 2015

devant le jury composé de :

Iraklis PARASKAKIS
Senior Lecturer in the Department of Computer Science at CITY College / *rapporteur*

Romain ROUVOY
Maître de conférences à l'Université de Lille 1 / *rapporteur*

Françoise BAUDE
Professeur à l'Université de Nice-Sophia Antipolis / examinateur

David BROMBERG
Professeur à l'Université de Rennes 1 / *examinateur*

Jean-Marc JÉZÉQUEL
Professeur à l'Université de Rennes 1 / *directeur de thèse*

Johann BOURCIER
Maître de conférences à l'Université de Rennes 1 / *co-directeur de thèse*

Acknowledgement

This thesis is the result of three years of work with in the DiverSE team at IRISA/Inria Rennes. First of all, I would like to thank my thesis supervisors Dr. Johann Bourcier and Dr. Noël Plouzeau, together with my thesis director Dr. Prof. Jean-Marc Jézéque, they give me the freedom to pursue my ideas, at the same time they gave me their advice, share their perspectives and constructive comments. I am very grateful for their constant supervision throughout the duration of the thesis.

I would also like to thank the members of the DiverSE group that were part of the RELATE[1] Project like Inti and David. I remember the travels we had to do to attend various seminars and project events. Thanks as well to other former colleagues of the DiverSE team, with whom I have closely come to know during these last three years for their constant collaboration and support. DiverSE offers a multi-cultural diversity and a great environment in terms of working experience, especially during the DiverSE coffee and the spring/autumn seminars.

I must also thank my former colleagues from the RELATE Project. Especially, I am grateful for the opportunity to undertake a research secondment during 3 months in FZI company in Karlsruhe, Germany, under the supervision of Dr. Klaus Krogman. I am grateful to former RELATE colleagues like Michal K., Viliam S. and Ilias G. for all the nice discussions and social activities we had in the context of RELATE.

In a personal level, I want to thank to my family, specially my parents, Ernesto and Raquel for their love and unconditional support. In particular, I want to express my deepest gratitude to my wife, *Bianca*, for her patient, encouragement and for always giving me the support and love I needed. I have been so blessed having you close to me!

Ivan Dario

[1] `http://www.relate-itn.eu/`

« Yesterday is history, tomorrow is a mystery, but today is a gift. That's why we call it the present.»
—Attributed to A. A. Milne

Abstract

Over the last years there is increasing interest in software systems that can cope with the dynamics of ever-changing environments. Currently, systems are required to dynamically adapt themselves to new situations in order to maximize performance and availability. Pervasive systems run in complex and heterogeneous environments using resource constrained devices where arising events may compromise the quality of the system. As a result, it is desirable to count on mechanisms to adapt the system according to problematic events occurring in the running context.

Recent literatures surveys have shown that dynamic adaptation is typically performed in a reactive way and therefore software systems are not able to anticipate recurrent problematic situations. In some situations, this could lead to resource waste and transient unavailability of the system. In contrast, a proactive approach does not simply act in response to the environment, but exhibit goal-directed behavior by taking the initiative in an attempt to improve the system performance or quality of service.

In this thesis we advocate for a proactive approach to dynamic adaptation. The benefits of combining predictive analysis with self-adaptive approach can be summarized as follows: 1) avoiding unnecessary adaptation and oscillatory behavior 2) managing allocation of exhaustible resources, and 3) proactivity in front of seasonal behavior. Focusing on the MAPE-K architecture, in this thesis we propose to enhance dynamic adaptation by integrating a Predict activity between the Analyze and Plan activities of the MAPE-K loop. We leverage ideas and techniques from the area of predictive analysis to operationalize the Predict activity.

We advocate for achieving proactive self-adaptation by integrating predictive analysis into two phases of the software process. At design time, we propose a predictive modeling process, which includes the following activities: define goals, collect data, select model structure, prepare data, build candidate predictive models, training, testing and cross-validation of the candidate models and selection of the "best" models based on a measure of model goodness. At runtime, we consume the predictions from the selected predictive models using the running system actual data. Depending on the input data and the time allowed for learning algorithms, we argue that the software system can foresee future possible input variables of the system and adapt proactively in order to accomplish middle and long term goals and requirements.

The proposal has been validated with a case study from the environmental monitoring domain. Validation through simulation has been done based on real data extracted from public environmental organizations. The answers to several research questions demonstrated the feasibility of our approach to guide the proactive adaptation of pervasive systems at runtime.

Résumé en Français

0.1 Introduction

Au cours des dernières années, il ya un intérêt croissant pour les systèmes logiciels capables de faire face à la dynamique des environnements en constante évolution. Actuellement, les systèmes auto-adaptatifs sont nécessaires pour l'adaptation dynamique à des situations nouvelles en maximisant performances et disponibilité [97]. Les systèmes ubiquitaires et pervasifs fonctionnent dans des environnements complexes et hétérogènes et utilisent des dispositifs à ressources limitées où des événements peuvent compromettre la qualité du système [123]. En conséquence, il est souhaitable de s'appuyer sur des mécanismes d'adaptation du système en fonction des événements se produisant dans le contexte d'exécution.

En particulier, la communauté du génie logiciel pour les systèmes auto-adaptatif (*Software Engineering for Self-Adaptive Systems - SEAMS*) [27, 35] s'efforce d'atteindre un ensemble de propriétés d'autogestion dans les systèmes informatiques. Ces propriétés d'autogestion comprennent les propriétés dites *self-configuring*, *self-healing*, *self-optimizing* et *self-protecting* [78]. Afin de parvenir à l'autogestion, le système logiciel met en œuvre un mécanisme de boucle de commande autonome nommé boucle MAPE-K [78]. La boucle MAPE-K est le paradigme de référence pour concevoir un logiciel auto-adaptatif dans le contexte de l'informatique autonome. Cet modèle se compose de capteurs et d'effecteurs ainsi que quatre activités clés : *Monitor*, *Analyze*, *Plan* et *Execute*, complétées d'une base de connaissance appelée *Knowledge*, qui permet le passage des informations entre les autres activités [78].

L'étude de la littérature récente sur le sujet [109, 71] montre que l'adaptation dynamique est généralement effectuée de manière réactive, et que dans ce cas les systèmes logiciels ne sont pas en mesure d'anticiper des situations problématiques récurrentes. Dans certaines situations, cela pourrait conduire à des surcoûts inutiles ou des indisponibilités temporaires de ressources du système [30]. En revanche, une approche proactive n'est pas simplement agir en réponse à des événements de l'environnement, mais a un comportement déterminé par un but en prenant par anticipation des initiatives pour améliorer la performance du système ou la qualité de service.

0.2 Thèse

Dans cette thèse, nous proposons une approche proactive pour l'adaptation dynamique. Pour améliorer l'adaptation dynamique nous proposons d'intégrer une activité *Predict* entre les activités *Analyze* et *Plan* de la boucle MAPE-K. Nous nous appuyons sur des idées et des techniques du domaine de l'analyse prédictive pour réaliser l'activité *Predict* en charge de prédire. Selon les données d'entrée et le temps accordé pour les algorithmes d'apprentissage, nous soutenons

qu'un système logiciel peut prévoir les futures valeurs d'entrée possibles du système et s'adapter de manière proactive afin d'atteindre les objectifs et les exigences du système à moyen et long terme.

Nous plaidons pour la réalisation de l'auto-adaptation proactive en intégrant l'analyse prédictive dans deux phases du processus logiciel. Au moment de la conception, nous proposons un processus de modélisation de prédiction, qui comprend les activités suivantes : définir des objectifs, recueillir des données, sélectionner la structure modèle, préparer les données, construire des modèles prédictifs candidats, configurer ces modèles par l'apprentissage, réaliser des essais et la validation croisée des modèles candidats, sélectionner le meilleur modèle en se fondant sur une mesure de de performances. À l'exécution, nous employons des modèles prédictifs sélectionnés en utilisant le système des données réelles de fonctionnement. Selon les données d'entrée et le temps imparti pour les algorithmes d'apprentissage, nous soutenons que le système logiciel peut prévoir les valeurs futures des variables d'entrée du système et permettre ainsi d'adapter de manière proactive afin d'atteindre les objectifs de moyen et long terme et exigences.

0.3 Scénario de Motivation

La proposition a été motivée et validée par une étude de cas portant sur la surveillance de l'environnement. Supposons qu'une forêt est équipée d'un réseau sans fil de capteurs et de nœuds de calcul (WSN), qui sont déployés géographiquement à des endroits stratégiques. Chaque nœud de calcul peut accueillir entre un et trois capteurs physiques (par exemple, les précipitations, l'humidité et température). Le but du système est de réguler de nouvelles reconfigurations (par exemple augmenter ou diminuer le débit de transmission de capteurs) selon les prévisions des futurs paramètres de valeur pour améliorer l'efficacité et d'étendre la durée de vie du système. La figure 1 illustre notre scénario concret. Les capteurs communiquent avec le système de gestion du réseau de capteurs (WSN) pour obtenir des configurations optimales en fonction des conditions actuelles.

0.4 Contributions

Les contributions principales de cette thèse sont les suivantes :

C1 : Extension du cadre d'architecture de référence *Monitor-Analyze-Plan-Execute* et *Knowledge* (MAPE-K) [78]. Nous proposons de renforcer l'adaptation dynamique en intégrant une phase *Predict* entre les phases *Analyze* et *Plan*.

C2 : Nous proposons une technique pour la mise en œuvre du processus de modélisation prédictive, selon les sept étapes suivantes : (1) définir des objectifs, (2) recueillir des données, (3) définir la structure du modèle, (4) traiter les données, (5) construire les modèles candidats, (6) réaliser l'apprentissage, tester et valider les modèles, et (7) mettre en œuvre les modèles.

C3 : Démonstration de notre approche avec un exemple concret tiré du domaine des *Cyber-Physical Systems (CPS)* (dans notre cas, surveillance de l'environnement).

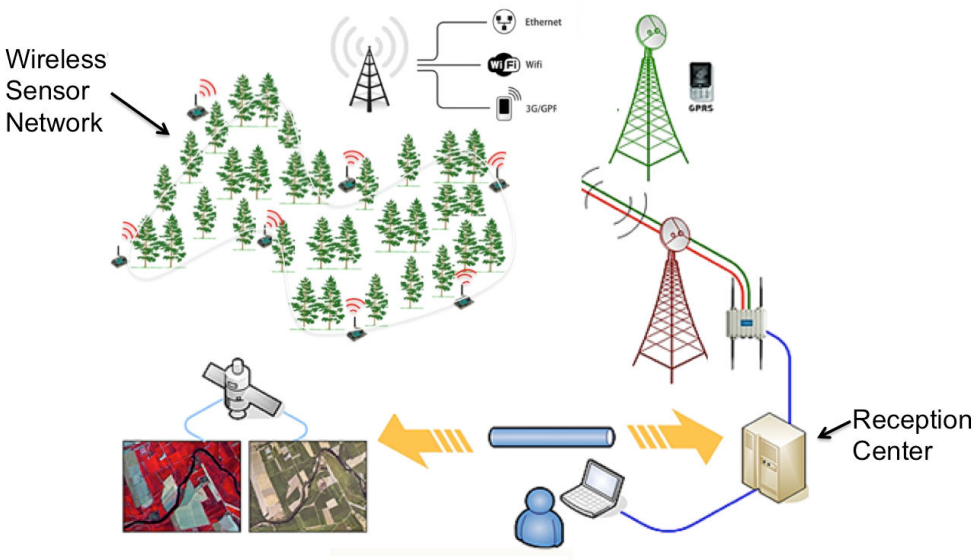


FIGURE 1 – L'application de surveillance de l'environnement réseau de capteurs sans fil [98])

0.5 Mise en Œuvre et Validation

Les différentes contributions de cette thèse ont été concrétisées et intégrées dans une architecture de référence de type MAPE-K [78] (voir la figure 2). Les composants concrétisant notre cadre prédictif peuvent être divisés en deux phases : la phase de conception et la phase d'exécution.

La phase de conception inclut la configuration hors ligne de l'approche. Dans cette phase, nous effectuons des activités liées au processus de modélisation prédictive. Cela comprend des activités telles que la collecte de données, le prétraitement des données, de construire les modèles candidats, réaliser l'apprentissage, tester et évaluer des modèles prédictifs fondés sur des observations de données passées.

La phase d'exécution implique les activités mentionnées dans la boucle MAPE-K, en commençant par la surveillance et l'analyse de l'état actuel du système en ligne. Pendant la phase en ligne, nous évaluons les modèles prédictifs définis au moment de la conception. Cette évaluation est effectuée de deux manières : comparaison des stratégies réactives contre des stratégies d'adaptation proactives. Ensuite, en prenant en considération les résultats de l'évaluation nous procédons à la prise de décision de reconfiguration. La phase en ligne porte alors sur les nouvelles ré-configurations.

Nous avons évalué notre approche avec le système de surveillance de l'environnement présenté comme scénario de motivation. À cette fin, nous avons employé un logiciel d'analyse prédictive *open source* (KNIME), des standard de modélisation prédictive (p. ex. Predictive Modeling Markup Language (PMML)) et des API et outils de*Machine learning* APIs (e.g., R, octave, Weka).

Tout d'abord, nous avons modélisé une *wireless sensor network* (WSN) pour la détection précoce des départs de feu, où nous combinons des données de température fournies par

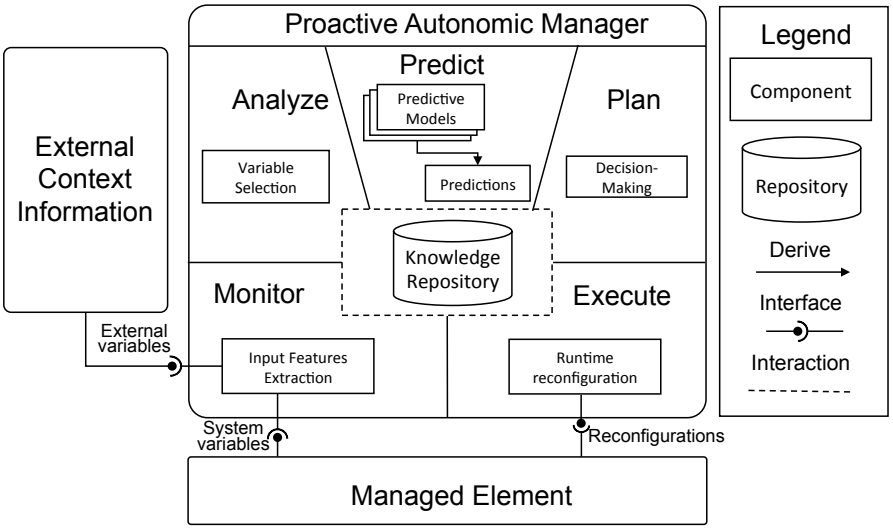


FIGURE 2 – Extension de le boucle MAPE-K

le *National Climatic Data Center* (NCDC) et des rapports d'incendie de *Moderate Resolution spectroradiomètre imageur* (MODIS). Nous avons évalué plusieurs modèles de classification et prédit la probabilité d'incendie pour les prochaine *N* heures. Les résultats confirment que notre approche proactive surpasse un système réactif typique dans les scénarios avec comportement saisonnier.

0.6 Organisation de la thèse

La thèse comprend six chapitres, organisés comme suit :

- Chapitre 1 : Introduit le sujet de cette thèse.
- Chapitre 2 : Décrit le contexte et les fondations de l'analyse prédictive et des systèmes logiciels autonomes, pour comprendre les sujets liés à cette thèse.
- Chapitre 3 : Présente l'état de l'art de l'auto-adaptation proactive dans trois domaine différents : génie logiciel, intelligence artificielle et théorie et ingénierie du contrôle.
- Chapitre 4 : Présente en détail l'approche que nous proposons et son architecture MAP^{2}E-K. En outre, nous détaillons la phase de conception et la phase d'exécution de notre approche.
- Chapitre 5 : Décrit les détails de mise en œuvre et l'évaluation du notre approche en comparant les stratégies réactives textit vs. proactives.
- Chapitre 6 : Nous résumons notre travail et ses perspectives.

Contents

Chapter 1

Introduction

In recent years, the expansion of computing infrastructure has caused software systems to be ubiquitous. Currently, computing environments blend into the background of our lives, basically everywhere and anywhere [123]. This pervasive characteristic has drastically increased the dynamicity and complexity of software systems. Usually the burden of managing such heterogeneous computing environments (e.g., devices, applications and resources) falls on engineers, which must manually redesign applications and adjust their settings according to available resources. Today, systems are required to cope with variable resources, system errors and failures, and changing users priorities, while maintaining the goals and properties envisioned by the engineers and expected by the final users [97].

Since its early beginnings, the software engineering community has envisioned the development of flexible software systems in which modules can be changed on the fly [44]. More recently, this vision has been extended to consider software systems that are able to modify their own behavior in response to changes in their operating conditions and execution environment in which they are deployed [70]. This concept is widely known as *self-adaptation* and it has been a topic of study in various domain areas, including autonomic computing, robotics, control systems, programming languages, software architecture, fault-tolerant computing, biological computing and artificial intelligence [27, 35, 71].

In particular, the *Software Engineering for Self-Adaptive Systems* (SEAMS)[1] community [27, 35] has been working towards achieving a set of self-management properties in software systems. These self-management properties include *self-configuring*, *self-healing*, *self-optimizing* and *self-protecting* [78]. In order to achieve self-management, the software system implements an autonomic control loop mechanism known as the MAPE-K loop [78]. The MAPE-K loop is the *de facto* paradigm to design self-adaptive software in the context of autonomic computing. This reference model consists of sensors and effectors as well as four key activities: *Monitor*, *Analyze*, *Plan* and *Execute* functions, with the addition of a shared *Knowledge* base that enables the passing of information between the other components [78].

Traditionally, the MAPE-K loop is a runtime model where the *Monitor* activity includes reading from sensors or probes that collect data from the managed system and environment. This information reflects the system's current state and its context environment. The *Analyze* ac-

[1] `http://www.self-adaptive.org`

tivity involves organizing the data, which is then cleaned, filtered, pruned to portray an accurate model of the current state of the system. Subsequently, the *Plan* activity involves building the execution plans to produce a series of adaptations to be effected on the managed system. Finally, the *Execute* activity involves carrying out the changes to the managed elements through the effectors. The propagated adaptations can be coarse-grained, for example enabling or removing functionality, or fine-grained for instance changing configuration parameters.

In 2001, P. Horn presented in [70] IBM's perspective on the state of information technology and he coined the term *Autonomic Computing*. The author explained in an illustrative way the analogy of the autonomic nervous system of the human body compared to an autonomous software system. Adopting the same analogy, we can consider the human body has two main "reasoning" capabilities: "reflex" and "thinking". Reactive techniques can be seen as a kind of reflex, since they involve a short term response to a particular stimulus. When the system is in a critical context, these adaptations can quickly reconfigure the system into an acceptable configuration. On the other hand, proactive techniques based on predictions resembles more the definition of "thinking", because they involve learning mechanisms based on historical data that can analyze middle and long term goals.

In this thesis we propose to enhance dynamic adaptation by integrating a *Predict* activity between the *Analyze* and *Plan* activities of the MAPE-K loop. We leverage ideas and techniques from the area of *predictive analysis* [126] to operationalize the *Predict* activity. Depending on the input data and the time allowed to the learning algorithms, we argue that a software system can foresee future possible input variables of the system and adapt proactively in order to accomplish middle or long term goals and requirements.

1.1 Motivation

During the last couple of years there is a growing interest into self-adaptive systems [27, 35]. Recent literature surveys [37, 71, 109] present that most of the existing self-adaptive systems have a reactive nature. A reactive adaptation strategy is triggered when a problem occurs and then the system addresses it. This capability of reactiveness is not generally a disadvantage. However, for some specific domains (e.g., environmental monitoring, safety critical systems), it is required to have proactiveness in order to decrease the aftereffects of changes and to block change propagation. Accordingly, S. W. Cheng et al. [30] demonstrate that a reactive strategy of adaptation might optimize instantaneous utility. However, it may often be suboptimal over a long period of time when compared with a predictive strategy. This means there is a great demand for proactive decision-making strategies.

A common scenario where the proactive strategy has the potential to outperform the reactive strategy regards applications that experiment cyclical behaviors. For instance, a personal laptop running on battery has several scheduled background tasks (e.g., backup, virus scan, updates) that can run periodically (e.g., daily, weekly). Often the execution of these administrative tasks cannot be interrupted or suspended once they have started, but the time and length of the execution is known beforehand. The resource utilization imposed by such tasks has a known behavior and it can certainly be predictable. Failure to recognize this behavior and not estimate its resource consumption in advance may lead to the system's halt in the middle of a critical

update.

Although certainly not a new technology, *predictive analysis* (a.k.a. predictive analytics) is a statistical or data analysis solution consisting of algorithms and techniques that can be used on both structured and unstructured data to determine possible outcomes [126]. Predictive analytics in software engineering has been widely used and has many application fields, thereby becoming a mainstream technology. The application fields include medical diagnostics, fraud detection, recommendation systems, and social networks. It is also being used for bank companies to perform credit risk assessment [112].

In general, predictive analysis can be used to solve many kinds of problem. In this thesis we argue that it can be applied to solve *self-optimizing*, *self-protecting* and *self-configuring* problems. For instance, a software system self-optimizes its use of resources when it may decide to initiate a change to the system proactively, as opposed to adapting reactively, in an attempt to improve performance or quality of service [71]. In contrast to that, from our own perspective, the implementation of the *self-healing* property can be seen as a reactive adaptation strategy due to the fact that by its definition it aims at solving a problem that has previously occured.

1.2 Research Questions

The overall goal of this thesis can be formulated as follows:

> *How should predictive analysis be performed when the main purpose of prediction is to support proactive self-adaptation in pervasive systems?*

Two important points here are *purpose* and *context*. The *purpose* of our study is predictive modeling as means to support proactive self-adaptation. The *context* is self-adaptive systems that implement the autonomic control loop. More in particular, we deal with pervasive systems deployed in heterogeneous and dynamic environments. In order to narrow the aspects of integrating predictive modeling with self-adaptive systems that are investigated, three partial research questions are formulated:

RQ1: How do several measurements observed in the monitoring component correlate over a certain period of time?

RQ2: How to build a machine-learning predictive model based on the temporal behavior of many measurements correlated to externally known facts?

RQ3: What measurements might indicate the cause of some event, for example, do similar patterns of measurements preceded events that lead to a critical situation such as a failure, and how to diagnosis this causal relationship?

1.3 Research Objectives

The major goal of this thesis is to **propose a predictive analysis based approach using machine learning techniques and integrate it into the autonomic control loop in order to en-**

able proactive adaptation in self-adaptive systems. Therefore, this thesis targets the following objectives:

RO1: Explore proactive adaptation approaches that leverages on existing context and environment streams of data in order to anticipate erroneous behavior of the system by predicting conflicting situations.

RO2: Study the activeness (e.g., reactive *vs.* proactive) of self-adaptive systems, analyze the potentials and limitations of system's parameters predictability (e.g., prediction horizon, prediction granularity), and investigate solutions to deal with parameter's future value uncertainty (e.g., prediction confidence level).

RO3: Develop a methodological framework of predictive modeling using machine learning techniques to enhance the effectiveness of proactive adaptation in self-adaptive software systems.

1.4 Contributions

The study and analysis of the aforementioned research objectives has generated the following contributions:

C1: Extension of the *Monitor-Analyze-Plan-Execute* and *Knowledge* (MAPE-K) reference architecture [78]. We propose to enhance dynamic adaptation by integrating a *Predict* phase between the *Analyze* and *Plan* phases.

C2: We propose a stepwise technique for the operationalization of the predictive modeling process divided in seven steps, namely: (1) define goals, (2) collect data, (3) define model structure, (4) data processing, (5) build candidate models, (6) train, test and validate models, and (7) models implementation.

C3: The demonstration of our approach with a concrete example taken from the *Cyber-Physical Systems* (CPS) domain (e.g., environmental monitoring). This implementation is described in more detail in Chapter 5.

1.5 Evaluation

We evaluated our approach in the environmental monitoring system presented on the motivation scenario. For this purpose, we implemented the proposed approach using an open source predictive analysis framework (*i.e.* KNIME), predictive modeling standards (e.g., Predictive Modeling Markup Language (PMML)) and machine learning libraries/tools (e.g., R, Octave, Weka).

Firstly, we modeled a *wireless sensor network* (WSN) for early detection of fire conditions, where we combined hourly temperature readings provided by *National Climatic Data Center* (NCDC) with fire reports from *Moderate Resolution Imaging Spectroradiometer* (MODIS) and simulated the behavior of multiple systems. We evaluated several classification models and predicted the fire potential value for the next N hours. The results confirmed that our proactive approach outperforms a typical reactive system in scenarios with seasonal behavior [98].

1.6 Organization

The rest of this thesis is organized as follows:

- Chapter 2 describes the background and the foundations in predictive analytics and autonomic computing to understand the subjects related to this dissertation.
- Chapter 3 overviews the state of the art in proactive self-adaptation approaches coming from three different domain areas: software engineering, artificial intelligence and control theory/engineering.
- Chapter 4 describes our proposed MAP^{2}E-K loop in detail. Additionally, we elaborate the design phase and runtime phase of our approach. These phases include the following activities: data collection, prepare data, build predictive models, training, testings and cross-validation of models, implementation of predictive models and implementation of the models.
- Chapter 5 describes the implementation details and the evaluation of the proposed proactive framework comparing the reactive *vs.* proactive strategies.
- Chapter 6 presents the conclusions and describes the future research lines of this thesis.

Chapter 2

Background

This chapter presents the foundations to understand the subjects related to this thesis. Firstly, in Section 2.1 we present a motivation scenario extracted from the environmental monitoring domain to illustrate the research problem we are tackling on. Secondly, in Section 2.2 we present *Predictive Analysis* [126], which is one of the building blocks for proactive self-adaptation. Thirdly, in Section 2.3 we present in details the area of *Autonomic Computing* [70], which is another domain area that enables our research approach. Finally, in Section 2.4 we discuss some similarities and potential conflicting issues in the previously described disciplines and give some conclusions.

2.1 Motivation Scenario

In this section we describe our motivation scenario that is at the heart of the problem we are focusing on. Let us assume that a forest is equipped with a *wireless sensor network* (WSN), where *sensor-nodes* are deployed geographically at strategic locations. Each *sensor-node* can host between one and three physical sensors (e.g., precipitation, humidity and temperature).

The goal of the system is to regulate new reconfigurations (e.g., increase or decrease sensor's transmission rate) according to predictions of future value parameters to improve the effectiveness and extend the life span of the system. Figure 2.1 illustrates our motivation scenario. Sensors communicate with the environment *wireless sensor network* (WSN) management system to obtain optimal configurations according to current conditions.

Each sensor node component is functioning at a standard sampling rate. However, the transmission of data can be controlled using three different levels of transmission rate: LOW (e.g., 1 transmission every 24 hours), MEDIUM (e.g., 1 transmission every 8 hours) and HIGH (e.g., 1 transmission every hour). Sensor nodes are equipped with 6lowPan radio for inter-node communication. *Data-collector* nodes are in charge of receiving raw data from the *sensor-nodes* and perform as gateway of the system.

The environmental WSN monitoring system has a global entity, which has access to all available information from sensors and maintains a global model of the current state of the system. In this context, the environmental manager has access to weather forecast and historical information of actual fires in similar locations. Combining both data sources, we can relate individual

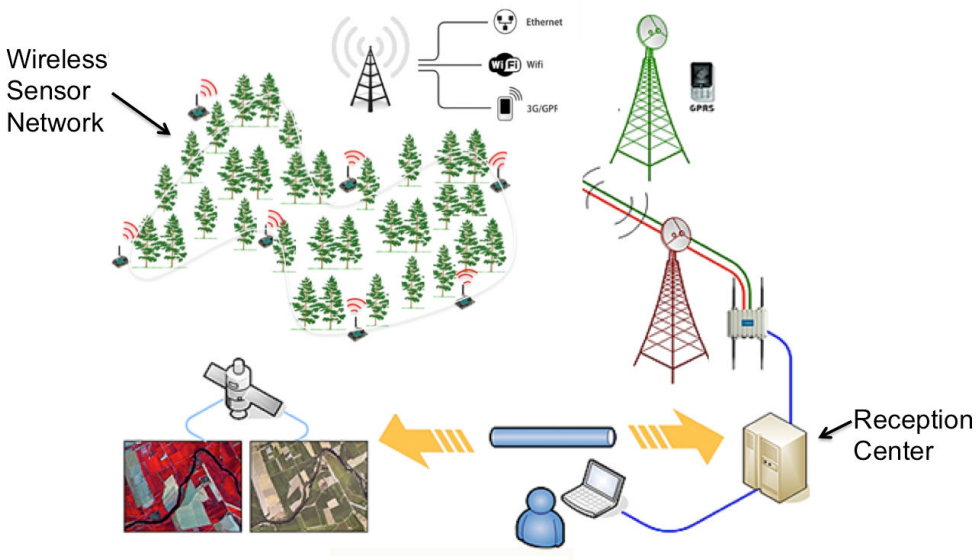


Figure 2.1 – Environmental monitoring wireless sensor network [98])

fires to weather conditions in space and time. The environmental WSN can be considered as a cloud-based dedicated service that decides the reconfiguration of each sensor-node, taking into account the following relevant parameters: (i) position, (ii) battery life and (iii) current environmental condition expressed as a cost function of precipitation, temperature and humidity.

A stochastic model predicts the forest behavior based on historical data. The model's accuracy is a function of the transmission rate, number and distribution of *sensor-nodes* contributing raw data through the data-collectors. Therefore, there is an implicit need to evolve the initial configuration over time because of system constrains or environmental changes (e.g., batteries are running low, or because a seasonal drought requires a more frequent and accurate transmission of current conditions).

To sum up, the main requirements of the environment monitoring motivation scenario are:

1. The system should provide feedback on potential fire risks to its users (e.g., environmental guards, fire department) allowing them to act proactively and anticipate and avoid possible critical situations.

2. The system should support the coordinated reconfiguration of sensor nodes, in order to avoid service failure due to battery exhaustion, with the goal of extending the life time of the system as much as possible.

The approach we propose assumes that the following information technologies are available:

1. The forest is equipped with a wireless sensor network, where each node has several sensing capabilities (e.g., temperature, humidity, smoke). Sensor nodes can communicate among themselves (i.e. intra-node communication) and with external data collector devices (i.e., extra node communication) to transmit the raw data.

2. Nodes can be reconfigured via wireless. Reconfigurations represent changes by increasing or decreasing the transmitting frequency (e.g., every 1 hr, 4 hrs, 8 hr). Other kind of adaptation may involves enabling or disabling sensor capabilities (e.g., enabling temperature) or changing the sleeping and wake up cycle in the sensor nodes.

2.1.1 Benefits of Proactive Adaptation

Proactive adaptation offers the benefit of anticipating events in order to optimize system behavior with respect to its changing environment. By analyzing the limitations of the reactive strategy, we have identified three potential benefits under which the proactive approach is likely to be better than the reactive strategy:

1. *Avoiding unnecessary adaptation:* This happens when the conditions that trigger an adaptation may be more short-lived than the duration for propagating the adaptation changes.

2. *Managing allocation of exhaustible resources:* by managing allocation of perishable resources (e.g., battery) with proactive adaptation enable us to make provision for future time when the resource is scarce.

3. *Proactivity in front of seasonal behavior:* proactive adaptation in front of seasonal behavior requires detecting seasonal patterns, which can be provided by predictors like time-series analysis.

2.1.2 Engineering and Adaptation Challenges

Engineering self-adaptive system deployed in dynamic and ever-changing environments poses both engineering and adaptation challenges. These challenges can be briefly described in terms of the following quality of services (QoS) properties:

Proactivity: Proactivity captures the anticipatory aspect in self-adaptive system. It can be broadly defined as the ability of the system to anticipate and predict when a change or problem is going to occur, and to take action about it [109]. On the contrary, in a reactive mode the system responds when a change or problem has already happened. When dealing with dynamically adaptive systems, as in our previously described motivation scenario, reactiveness has some limitations: (i) information used for decision making does not extend into the future, and (ii) the planning horizon of the strategy is short-lived and does not consider the effect of current decisions on future utility [30].

Predictability: Predictability can be described in two ways: predictability associated with the environment, which is concerned with whether the source of change can be predicted ahead of time, and predictability associated with the running system, which deals with whether the consequences of self-adaptation can be predictable both in value and time [4]. In the context of this thesis we are concerned with predictability associated to the environment and the different techniques need it depending on the degree of anticipation.

Reliability: Reliability can be broadly defined as the probability of successfully accomplishing an assigned task when it is required [46]. In particular, the meaning of *success* is domain dependent, for instance, that the execution of the task satisfies convenient properties (e.g., it has been completed without exceptions, within an acceptable timeout, occupying less than a certain amount of memory, etc).

Attaining the previously described QoS properties poses a great challenge and illustrates the opportunity of improvement in the existing adaptation approaches. Currently, there is an over-abundance of public data (e.g., sensors, mobiles, pervasive systems). Thus, there is great demand for solutions to make sense of this data, particularly using predictive mechanisms regarding the specific domain and its operational environment. In the following section we explore in detail the enablers of *proactivity*, *predictability* and *reliability* in the context of self-adaptive systems.

2.2 Predictive Analysis

This section presents predictive analysis as an enabler of our proactive self-adaptation approach. Predictive analysis (PA) encompasses a variety of statistical techniques ranging from modeling, machine learning, and data mining techniques that analyze current and historical facts to make predictions about future, or otherwise unknown, events [95].

Currently, there is an explosion of data being collected everywhere around ever increasing and ubiquitous monitoring processes. Data continues to grow and hardware is struggling to keep up, both in processing power and large volume data repositories [126]. Data can be broadly categorized as either structured or unstructured. Structured data has well-defined and delimited fields, and this is the usual type of data used in statistical modeling. Unstructured data includes time series, free text, speech, sound, pictures and video. Statistical modeling can also be divided into two broad categories: search and prediction. For search problems we try to identify categories of data and then match search requests with the appropriate data records. In prediction problems we try to estimate a functional relationship, so we can provide an output to a set of inputs. These prediction statistical models are in general the types of modeling problems that are considered in this thesis.

E. Siegel, in his book *Predictive Analytics: The power to predict who will click, buy, lie or die* [112] presented 147 examples of different application domains for predictive analytics. Applications can be found in a wide range of different domains including: stock prices, risk, accidents, sales, donations, clicks, health problems, hospital admissions, fraud detection, tax evasion, crime, malfunctions, oil flow, electricity outages, opinions, lies, grades, dropouts, romance, pregnancy, divorce, jobs, quitting, wins, votes, and much more.

The predictive analysis process is generally divided in two parts: off-line and on-line. The off-line phase involves collecting the data from the environment and applying the machine-learning techniques to the datasets to build predictive models. The on-line phase include the activities of characterization of an individual, next the individual is evaluated with predictive models in order to generate the results that will drive the decision-making process. Figure 2.2 illustrates the general predictive analysis process and the relationships between its main components.

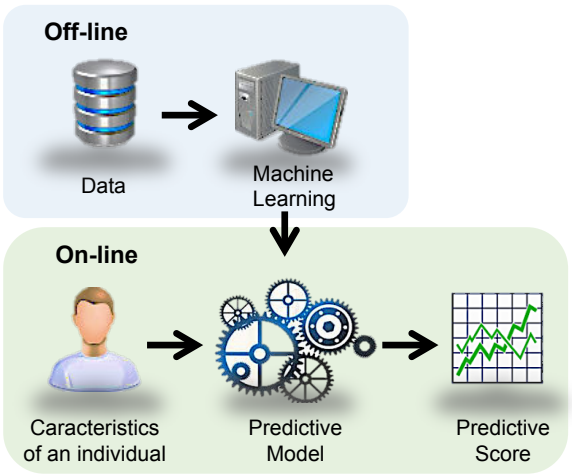


Figure 2.2 – General overview of the predictive analysis process (based on [112])

Namely, the basic elements that make feasible the predictive analysis process are: data, predictions, machine-learning, ensemble of models and the action of decision-making [112, 126]. The following subsections explain in more detail each element and their relationships in the predictive analysis process.

2.2.1 Data is the New Gold

Following the agricultural and industrial revolutions from previous centuries, in the last decades we have experienced the information revolution [103]. Forbes, the American business magazine stated that we are entering a new era, the era of data where **data is the new gold**[1]. Several other authors have coined data as "the new oil" or "data is the new currency of the digital world" [103].

Day by day, each on-line and off-line bank transaction is recorded, websites visited, movies watched, links clicked, friends called, opinions posted, dental procedures endured, sports games won, traffic cameras passed and flights taken. Countless sensors are deployed daily. Mobile devices, drones, robots, and shipping containers record movement, interactions, inventory counts, and radiation levels, just to mention a few devices that generate and consume data. Now imagine all those deployed devices interconnected at an unprecedented scale and pace, this is what is known as the Internet of Things (IoT), a term first coined by Kevin Ashton in 1999 in the context of supply chain management [7]. The next revolution will be the interconnection between objects to create a smart environment. Only in 2011 did the number of interconnected devices on the planet overtake the actual number of people. Currently there are 9 billions interconnected devices and it is expected to reach 24 billions devices by 2020 [62].

Further more, free public data is overflowing and waiting at our fingertips. Following the open data movement, often embracing a not-for-profit organization, many data sets are available on-line from different fields like biodiversity, business, cartography, chemistry, genomics, and

[1] http://www.forbes.com/sites/bradpeters/2012/06/21/the-big-data-gold-rush/

medicine. For example KDnuggets[2] is one of the top resource since 1997 for data mining and analytics. Another example is the United States official website Data.gov, whose goal is "to increase public access to high value, machine readable datasets generated by the Government of USA." Data.gov[3] contains over 390,000 data sets, including data about marine casualties, pollution, active mines, earthquakes, and commercial flights [103].

Big data is defined as data too large and complex to capture, process and analyze using current computer infrastructure. It is now popularly characterized by five V's (initially it was described as having three, but two have since then been added to emphasize the need for data authenticity and business value) [63]:

- *Volume:* data measurements in tera (10^{12}) are now the norm, or even peta (10^{15}), and is rapidly heading towards exa (10^{18});
- *Velocity:* data production occurs at very high rates, and because of this sheer volume some applications require near real-time data processing to determine whether to store a piece of data;
- *Variety:* data is heterogeneous and can be highly structured, semi-structured, or totally unstructured;
- *Veracity:* due to intermediary processing, diversity among data sources and in data evolution raises concerns about security, privacy, trust, and accountability, creating a need to verify secure data provenance; and
- *Value:* through predictive models that answer *what-if* queries, analysis of this data can yield counterintuitive in sights and actionable intelligence.

With such an overflow of free public data, there is a need for new mechanisms to analyze, process, and take advantage of its great potential. Thus, the goal is to apply predictive analysis techniques to process this abundance of public data. This involves a variety of statistical techniques such as modeling, machine learning, and data mining that analyze current and historical facts in order to make predictions about future or unknown events [95]. New data repository structures have evolved, for example the MapReduce/Hadoop[4] paradigm. Cloud data storage and computing is growing, particularly in problems that can be parallelized using distributed processing [63].

2.2.2 Prediction Offers Real Value

J. A. Paulos, professor of mathematics at Temple University in Philadelphia defined "Prediction is a very difficult task and uncertainty is the only certainty there is. Knowing how to live with insecurity is the only security". Thus, uncertainty is the reason why accurate prediction is generally not possible. In other words, predicting is better than pure guessing, even if is not

[2] http://www.kdnuggets.com/datasets/index.html
[3] http://www.data.gov/
[4] http://hadoop.apache.org/

100 percent accurate, **predictions deliver real value**. A vague view of what is coming outperforms complete darkness [112]. Therefore analyzing errors is critical. Learning how to replicate past successes by examining only the positive cases does not work and induces over-fitting to the training data [89]. Thus, negative examples are our friends and should be taken into account.

From Figure 2.2 that illustrates the predictive analysis process we see that before using a predictive model we have to build it using machine learning techniques. By definition a *predictive model* is a mechanisms that predicts a behavior of a system. It takes characteristics of the system as input (e.g. internal, external), and provides a predictive score as output. The higher the score, the more likely it is that the system will exhibit the predicted behavior [112].

From the predictive analysis process we can clearly identify two phases. In the off-line phase data must be collected and processed. This means identifying the *predictor variables*, which are the parameters that better describe the behavior of the system. This part also involves cleaning the data, which can be done by re-dimensional analysis and removing out-layers from the training dataset. Alternatively, *surrogate variables* can be used to handle missing values of predictor variables. Next, the machine learning algorithms crunches the data to build the predictive model.

In general, the statistical modeling approaches can be categorized in many ways [15], for instance: (1) *classification*, i.e. predicting the outcome from a set of finite possible values, (2) *regression*, *i.e.* predicting a numerical value, (3) *clustering* or segmentation, *i.e.* summarizing data and identifying groups of similar data points, (4) *association analysis*, *i.e.* finding relationships between attributes, and (5) *deviation analysis*, *i.e.* finding exceptions in major trends or structures. Other authors, such as [126], categorize the prediction approaches from a different perspective: (1) linear modeling and regression, (2) non-linear modeling, and (3) time series analysis.

In our case, we focused on *classification* and *regression* methods in the scope of time series analysis. *Classification*, because we focused on finding a model that, when applied on a certain time series (e.g. hourly temperature readings), is able to classify current weather conditions as having significant fire potential or not. *Regression* models are applied in the forecasting scope to make precise predictions for the next N hours in the time series.

Linear Modeling and Regression

The most common data modeling methods are regressions, both linear and logistic. According to [126], it is likely that 90% or more of real world applications of data mining end up with a relatively simple regression as the final model, typically after very careful data preparation, encoding, and creation of variables.

In statistics, linear regression is an approach for modeling the relationship between a scalar dependent variable y and one or more explanatory (or independent) variable denoted X. The case of one explanatory variable is called *simple linear regression*, whereas for more than one explanatory variable the process is called *multiple linear regression*.

There are several reasons why regressions are so commonly used. First, they are generally straightforward both to understand and compute. The *mean square error (MSE)* objective function has a closed-form linear solution obtained by differentiating the MSE with respect to the unknown parameter and setting the derivatives to zero.

Non-linear Modeling

Previously, we discussed many of the important and popular linear modeling techniques. Moving one step towards complexity we can find the nonlinear models. These classes of models can be understood as fitting linear models into local segmented regions of the input space. Additionally, fully nonlinear models beyond local linear ones include: clustering, support vector machine (SVM), fuzzy systems, neural networks, and others. Formally defined, the goal of nonlinear modeling is to find the best-fit hyper plane in the space of the input variables that gives the closest fit to the distribution of the output variable [126].

In particular, in Section 5.2.4 we choose from existing classification models and evaluate 8 non-linear models in the implementation of the motivation scenario. These models are: (1) Multi Layer Perceptron [107], (2) Fuzzy Rules [14], (3) Probabilistic Neural Network [16], (4) Logistic Regression [126], (5) Support Vector Machine [104], (6) Naive Bayes [126], (7) Random Forest [23], and (8) Functional Trees [55]. In general, the more nonlinear the modeling paradigm, the more powerful the model. However, at the same time, the easier it is to overfit.

Time Series Analysis

One major distinction in modeling problems as time series analysis is that the next value of the series is highly related to the most recent values, with a time-decaying importance in this relationship to previous values. Before looking more closely at the particular statistical methods, it is appropriate to mention that the concept of time series is not quite new. In fact its beginnings date back to mid-19th century, when ship's captains and officers had long been in the habit of keeping detailed logbooks during their voyages (e.g., knots, latitude, longitude, wildlife, weather, etc.). Then they carried out analysis in the collected data that would enable them to recommend optimal shipping routes based on prevailing winds and currents [39].

Moreover, the time series forecasting problem has been studied during the last 25 years [34], developing a wide theoretical background. Nevertheless, looking back 10 years, the amount of data that was once collected in 10 minutes for some very active systems is now generated every second [39]. Thus, the current overabundance of data poses new challenges that need different tools and the development of new approaches.

The goal of modeling a problem in terms of time series analysis is to simplify it as much as possible regarding the time and frequency of generated data. R. H. Shumway and D. S. Stoffer categorized the time series analysis from two different approaches: time domain approach and frequency domain approach [111], see Figure 2.3.

The *time domain approach* is generally motivated by the presumption that correlation between adjacent points in time is best explained in terms of a dependence of the current value on past values. Conversely, the *frequency domain approach* assumes the primary characteristics of interest in time series analyses relate to periodic or systematic sinusoidal variations found naturally in most data. The best way to analyzing many data sets is to use the two approaches in a complementary way.

- **Time domain approach**. The time domain approach focuses on modeling some future value of a time series as a parametric function of the current and past values. Formally, a

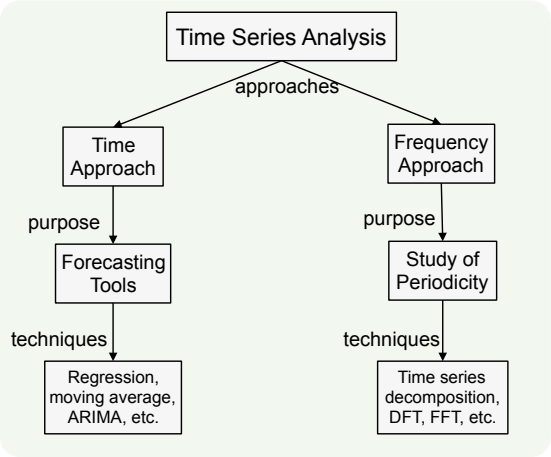


Figure 2.3 – Time series approaches (based on [111])

time series X is a discrete function that represents real-valued measurements over time as represented in Formula 2.1.

$$X = \{x_1, x_2 ... x_n\} : X = \{x_t : t \in T\} : T = \{t_1, t_2 ... t_n\} \tag{2.1}$$

The n time points are equidistant, as in [90]. The elapsed time between two points in the time series is defined by a value and a time unit. For example, we may consider a time series as a sequence of random variables, x_1; x_2; x_3; ... , where the random variable x_1 denotes the value taken by the series at the first time point t_1, the variable x_2 denotes the value for the second time period t_2, x_3 denotes the value for the third time period t_3, and so on. In general, a collection of random variables, x_t, indexed by t is referred to as a stochastic process. In this analysis, t will typically be discrete and vary over the integers t = 0; $\pm$ 1; $\pm$ 2; or some subset of the integers [111]. This time series generated from uncorrelated variables is used as a model for noise in engineering applications and it is called *white noise* [111].

If the stochastic behavior of all time series could be explained in terms of the white noise model, classical statistical methods would suffice. Two ways of introducing serial correlation and more smoothness into time series models are *moving averages* and *autoregressions*.

To smooth a time series we might substitute the white noise with a moving average by replacing every value by the average of its current value and its immediate neighbors in the past and future. For instance, the following Formula 2.2 represents a 3-point moving average.

$$v_t = \frac{1}{3}(w_{t-1} + w_t + w_{t+1}) \tag{2.2}$$

Classical regression is often insufficient for explaining all of the interesting dynamics of a time series. Instead, the introduction of correlation as a phenomenon that may be generated through lagged linear relations leads to proposing the autoregressive (AR) and *autoregressive moving average (ARMA)* models. The popular Box and Jenkins [21] *Autoregressive integrated moving average (ARIMA)* create models to handle time-correlated modeling and forecasting. The approach includes a provision for treating more than one input series through multivariate ARIMA or through transfer function modeling [22, 20].

- **Frequency domain analysis**. Seasonality can be identified and removed as follows. First we need to identify the natural periodicity of data. This can be done in a variety of ways, such as (a) through expert understanding of the dynamics, (b) through statistical analysis using different window lengths, or (c) through frequency analysis looking for the fundamental frequencies. Formally defined, a time series X_t with values $(x_{t-k}, x_{t-k-1}, ..., x_{t-2}, x_{t-1})$ (where t is the current time) can be represented as the addition of four different time series:

$$X_t = T_t + S_t + R_t \tag{2.3}$$

  where T_t, S_t and R_t are the trend, seasonality and random components of the time series. The trend T_t describes the long term movement of the time series. The seasonality component S_t describes cyclic behaviors with a constant level in the long term, it consists of patterns with fixed length influenced by seasonal factors (e.g., monthly, weekly, daily). Finally, the random component R_t is an irregular component to be described in terms of random noise. Therefore, modeling X_t can be described as the addition of its components. Figure 2.4 illustrates a time series decomposition analysis of a temperature variable over 30 days.

  - The ***trend*** component can be described as a monotonically increasing or decreasing function, in most cases a linear function, that can be approximated using common regression techniques. It is possible to estimate the likelihood of a change in the trend component by analyzing the duration of historic trends.
  - The ***seasonal*** component captures recurring patterns that are composed of at least on or more frequencies (e.g daily, weekly, monthly) patterns. These frequencies can be identified by using a Discrete Fourier Transformation (DFT) or by auto-correlation technique [111].
  - The ***random*** component is an unpredictable overlay or various frequencies with different amplitudes changing quickly due to random influences on the time series.

2.2.3 Machine-Learning

Machine Learning aims at finding patterns that appear not only in the data at hand, but in general, so that what is learned will hold true in new situations never yet encountered. By definition, “a computer program is said to learn from experience E with respect to some class of tasks T

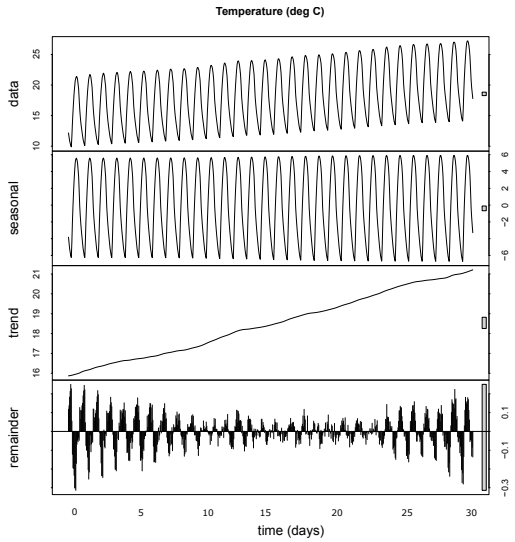


Figure 2.4 – A time series decomposition analysis example

and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E", which has become the formal definition of Machine Learning [88]. Training data sets are use by machine learning algorithms to generate a predictive model. Testing data sets are used to evaluate the accuracy of the predictive models to the real data.

Table 2.1 shows a classification of datasets and a selection of candidate predictive models that can be applied to such datasets. Data sets can be organized by three characteristics: default task, data type and attribute type. This classification is based in the Machine Learning Repository (MLR)[5] from the University of California, School of Information and Computer Science [9].

For each considered model there are myriads of variations proposed in the literature, and it would be a hopeless task to consider all existing varieties. Our strategy is therefore to consider the basic version of each model and categorize them by the purpose of their task, data type and attribute type. The rationale is that most users will more likely prefer to consider the basic form at least in their first attempt to understand its functionality.

2.2.4 Ensemble of Predictive Models

An ensemble of predictive models consists of a set of individually trained classifiers (e.g. neural networks, decision trees) whose results are combined to improve the prediction accuracy of a machine learning algorithm [126]. Previous research has shown that a collection of statistical classifiers, or ensembles, is often more accurate than a single classifier [95].

A random forest model is the typical example for predictive models ensemble used for classification and regression problems. It operates by constructing a multitude of decision trees at training time and by outputting the class that is the mode of the classes output by individual trees [95]. The general abstraction is collecting models and having them vote. When joined as

[5] http://archive.ics.uci.edu/ml/datasets.html

Property	Types	Definition
Default task	Classification	Is a supervised problem where inputs are divided into two or more classes known beforehand (e.g., Bayes, function, fuzzy, meta-classifiers, rule-based, trees, Support vector machines)
	Regression	Also a supervised problem, the outputs are continuous rather than discrete (e.g. linear regression, logistic regression, SVM)
	Clustering	A set of inputs is to be divided into groups. Unlike in classification, the groups are not known beforehand, making this typically an unsupervised task (e.g. centroid-based clustering, K-nearest, distribution-based clustering)
	Recommendation	Is an information filtering system that seek to predict the "rating" or "preference" that users would give to an item (e.g., collaborative filtering, content-based filtering, hybrid recommender).
	Forecast	Estimates a future event or trend as a result of study and analysis of available pertinent data (e.g., Times series analysis, autoregressive moving average (ARMA), autoregressive integrated moving average (ARIMA), Holt-Winters (HW))
Data type	Univariate	Considers only one factor, or *predictor variable*, about the system under study
	Multivariate	Considers multiple factors or *predictor variables* at a time of the system under study
	Sequential	Considers sequence of data values, usually ordinal or categorical data (e.g. months of the year, days of the week)
	Time Series	Considers a set of values of a quantity obtained at successive times, often with equal intervals between them (e.g. hourly temperature readings)
	Tex	Considers text collections that belongs to a vocabulary or specific language (e.g. used for spam filtering)
Attribute type	Categorical	Considers qualitative attributes
	Numerical	Considers quantitative attributes
	Mixed	Support both qualitative and quantitative attributes

Table 2.1 – Classification of datasets, based on [9]

ensembles, predictive models compensate for their limitations, so the ensemble as a whole is more likely to have higher accuracy rather than its individual predictive models.

Another example of model ensembles is boosting algorithms, which is an approach to machine learning based on the idea of creating a highly accurate predictor by combining many weak and inaccurate learners. Some popular algorithms in this category are AdaBoost, LPBoost, TotalBoost, BrownBoost, MadaBoost, among others [110].

2.2.5 Supporting Decision-Making

Once the predictive model or the ensembles of predictive models is trained and tuned, the next step is to deploy it on-line. The key part is to integrate it with a decision-making mechanism. In a previous paper we integrated our prediction based proactive approach with a ruled-based mechanism to take decisions whether to adapt or not the software system [98].

Prediction does not offer a real value by itself unless it feeds a reasoning engine to support the decision-making process. Therefore, **prediction implies action**. Particularly, in the implementation of our approach we observed that predictions offer three kinds of improvement when compared to existing reactive adaptation approaches:

1. *Prediction prevents unnecessary self-adaptation:* At times the conditions that triggers an adaptation may be more short-lived that the duration for propagating the adaptation changes, resulting in unnecessary adaptation that incur potential resource cost and service disruption [30].

2. *Prediction to manage the allocation of exhaustible resources:* In [93], the author proposed a taxonomy that classifies each computing resources into one of three categories: time-shared, space-shared and exhaustible. For instance, CPU and bandwidth are time-shared resources, while battery is an exhaustible resource. A proactive strategy offers the advantage to make provision of exhaustible resources (e.g., battery) for future time when the resource is scarce [98].

3. *Proactivity in front of seasonal behavior:* If a similar shift in system conditions occurs seasonally, this means once every period of time such as every day at 10 AM, the same pattern of adaptations would repeat every period. One workaround is to learn the seasonal pattern from historical data and predict adaptations on time [30].

2.3 Autonomic Computing

Autonomic Computing is the other enabler of our approach. Autonomic computing is an IBMś initiative presented P. Horn [70] in 2001. It describes computing systems that are said to be self-managing. The term “autonomic” comes from a biology background and is inspired by the human body’s autonomic nervous system. Similarly, a self-adaptive system (SAS) modifies its own behavior in response to changes in its operating environment, which is anything observable by the software system, such as end-user inputs, external hardware devices and sensors, or program instrumentation [97]. The concepts of autonomic computing and self-adaptive systems are

strongly related and share similar goals, thus within the scope of this thesis both terms are used interchangeably.

Autonomic computing original goal is:

> "To help to address complexity by using technology to manage technology. The term autonomic is derived from human biology. The autonomic nervous system monitors your heartbeat, checks your blood sugar levels and keeps your body temperature closer to 98.6°F without any conscious effort on your part. In much the same way, self-managing autonomic capabilities anticipate and resolve problems with minimal human intervention. However, there is an important distinction between autonomic activity in the human body and the autonomic activities in IT systems. Many of the decisions made by autonomic capabilities in the body are involuntary. In contrast, self-managing autonomic capabilities in software systems perform tasks that IT professionals choose to delegate to the technology according to policies [70]."

As mentioned earlier, *Autonomic Computing* (AC) and *Self-adaptive System* (SAS) share the same goal, which is to improve computing systems with a decreasing human involvement in the adaptation process. Many researchers use the terms self-adaptive, autonomic computing, and self-managing interchangeably [32, 37, 71]. Salehie and Tahvildari [109] describe a slightly different point of view, where "the self-adaptive software domain is more limited, while autonomic computing has emerged in a broader context." According to the authors [109] "self-adaptive software has less coverage and falls under the umbrella of autonomic computing."

2.3.1 The Self-* Properties

Upon launching the AC initiative, IBM defined four general adaptivity properties that a system should have to be considered self-managing. These properties are also knowns as the *self-* properties* and include: *self-configuring*, *self-healing*, *self-optimizing* and *self-protecting*. These properties are described as follows.

Self-configuring is the capability of reconfiguring automatically and dynamically in response to changes by installing, updating, integrating, and composing/decomposing software entities [109].

Self-optimizing is the capability to monitor and tune resources automatically to meet end-users requirements or business needs [70]. An autonomic computing system optimizes its use of resources. It may decide to initiate a change to the system *proactively*, as opposed to a reactive behavior, in an attempt to improve performance or quality of service [71].

Self-healing is the capability of discovering, diagnosing and reacting to disruptions [78]. The kinds of problems that are detected can be interpreted broadly: they can be as low-level as bit errors in a memory chip (hardware failure) or as high-level as an erroneous entry in a directory service. Fault tolerance is an important aspect of self-healing [71].

Self-protecting is the capability to anticipate, detect, identify and protect against malicious attacks but also from end users who inadvertently make software changes, for example by deleting an important file. The system autonomously tunes itself to achieve security, privacy and data protection [78].

According to Salehie and Tahvildari [109] the self-* properties can be divided into a three levels hierarchy, as illustrated in Figure 2.5. The aforementioned core self-properties are categorized into the *Major Level*. The *[General Level* contains global properties of SAS. A subset of these properties consists of self-managing, self-governing, self-maintenance, self-control and self-evaluating. Another subset at this level is self-organizing, which emphasizes decentralization and emergent functionalities.

The *Primitive Level* contains the underlying primitive properties. Self-awareness means that the system is aware of its self states and behaviors. This property is based on self-monitoring, which reflects what is monitored. Context-awareness means that the system is aware of its own context, which is its operational environment. This classification serves as the *de facto* standard in this domain.

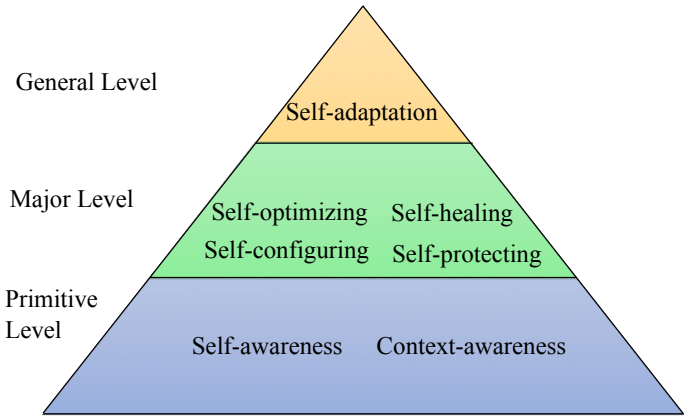


Figure 2.5 – Hierarchy of the self-* properties [109]

2.3.2 The Autonomic Control Loop

According to Salehie and Tahvildari [109], adaptation approaches can be divided into the following two categories with respect to the separation of the adaptation mechanism and application logic:

- The *internal* approach interweaves application and adaptation specifications. It is based on the programming languages features, such as conditional expressions, parametrization and exceptions [97, 48]. Since application and adaptation specifications are mixed, this approach can lead to poor maintainability and scalability.
- The *external* approach considers an external subsystem (or engine) containing adaptation process to control the adaptation of the software system. Therefore it separates the adaptation concerns from other functional concerns of the system. The external subsystem

implements the adaptation logic, mostly with the aid of a middleware [48] or a policy-engine [17].

Due to the limitations of the internal approach, we base our solution on the external autonomic control loop proposed by IBM, which is the *de facto* standard reference model. IBM's vision was presented by J. Kephart and D. Chess [78]. In 2003 they introduced the autonomic control loop, also known as the MAPE-K loop, which involves the activities of Monitoring, Analyze, Planing and Executing (see Figure 2.6). The Knowledge component binds those activities together and allows exchange of information between the mentioned activities. The loop is completed by connecting to the adaptable Managed System through Sensors and Actuators. We detail each process hereafter.

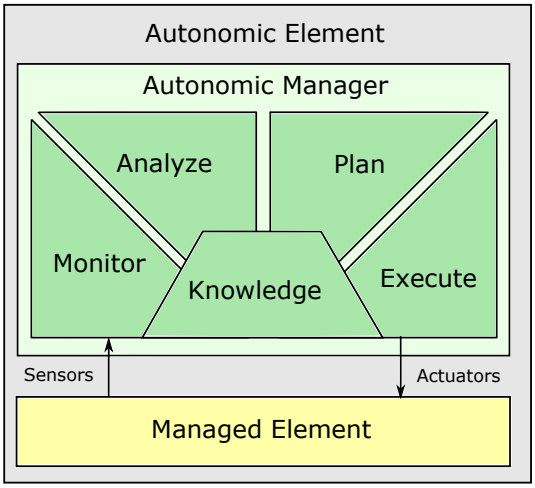


Figure 2.6 – Autonomic computing MAPE-K reference model [78]

- The ***Monitor*** process collects and correlates data and converts them to behavioral patterns and symptoms. The software or hardware components used to perform monitoring are called Sensors. In [109] the authors describe examples of different kinds of sensors ranging from different techniques: logging, monitoring, and event information models, management protocols, profiling and signal monitoring.
- The ***Analyze*** process detects the symptoms provided by the Monitor process and the system's history to detect when a change needs to be applied. It also helps to identify where the source of a transition to a new state takes place.
- The ***Plan*** process determines what needs to be changed and how to change it to achieve the best outcome.
- The ***Execute*** process applies the adaption actions on the Managed System using Actuators. In [109] the authors described a different set of actuators that use the following techniques: design, architectural and autonomic patterns, middleware-based effectors, dynamic aspect weaving and function pointers.

2.4 Discussion

In this section we discuss some potential conflicting issues in the areas of Predictive Analytics and Autonomic Computing.

Regarding some limitations of predictive analysis. In [112] the author uses predictive analytics for the purpose of predicting the future behavior of individuals in front of future situations. However, there is an open debate at this point [3], because individuals and people in general are influenced by their environment in innumerable ways. If one puts the exact person in the same situation tomorrow, he/she may take a completely different decision.

For that reason, we strongly believe in the potential of predictive analytics for predicting the behavior of software systems. We claim that there is a set of variable parameters than can properly describe the behavior of a software system (e.g. internal and external parameters). If we can find the right metrics then it is feasible to quantify the response of the system in front of different situations. Another key element is that software systems are not emotionally vulnerable to weather or personal relationships. For instance, the Network Weather Service (NWS) [125] is an attempt to provide accurate forecast of dynamically changing performance in set of distributed computing resources.

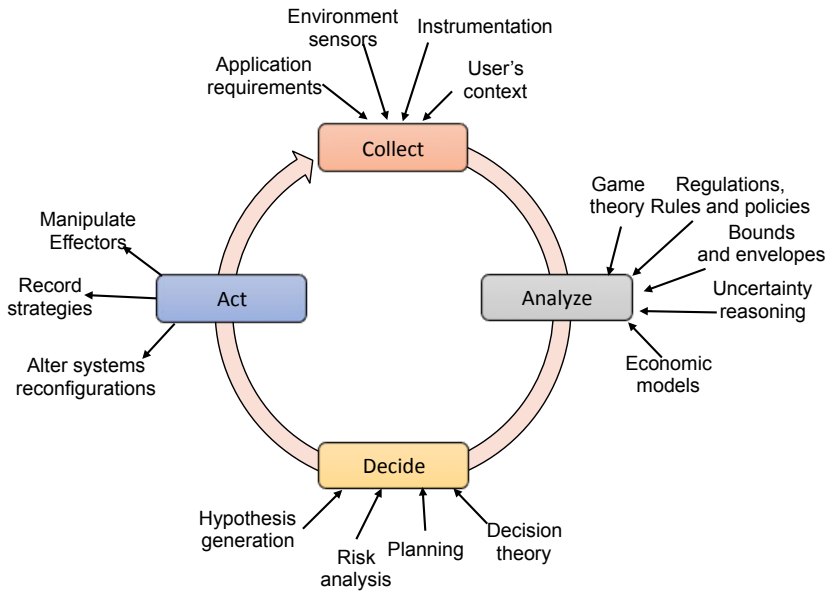


Figure 2.7 – The autonomic control loop (based on [37])

Regarding potential conflicting issues in Autonomic Computing. Dobson et al. [37] propose a twin vision of an autonomic control loop, illustrated in Figure 2.7. According to [37], the autonomic control loop is more related to the autonomic communication domain and includes the *Collect*, *Analyze*, *Decide* and *Act* processes (CADA). These activities perform similar tasks to the ones described in the MAPE-K loop earlier.

The feedback cycle starts with the *collection* of relevant data from environmental sensors and other sources that reflect the current state of the system. Next, the systems *analyzes* the

collected data. There are many approaches to structuring and reasoning about the raw data (e.g. using models, theories, rules). Next, a *decision* must be made about how to adapt the system in order to reach a desirable state. Finally, to implement the decision, the system must *act* via available actuators or effectors. It is important to highlight that the flow of control among these components is unidirectional. Moreover, while the figure shows a single control loop, multiple separate loops are typically involved in a practical system [24].

Despite their evident similarities with the MAPE-K loop, [37] describes that the autonomic communication is more oriented towards distributed systems, while autonomic computing is more directly oriented towards application software and management of computing resources [109]. Accordingly, the twin visions of autonomic communication and computing are aligned in identifying the need for decentralized algorithms and control, context-awareness, novel programming paradigms, end-to-end privacy management, and comprehensive evaluation in order to deriver the desired self-* properties [37].

Chapter 3

State of the Art

In this chapter we present the state-of-the-art of proactive self-adaptive systems. Firstly, in section 3.1 we present a taxonomy that allow us to categorize the different proactive approaches. Secondly, in section 3.2, we describe in detail each of the selected approaches coming from three different domain areas. These related research areas are: Software Engineering, Artificial Intelligence and Control Theory/Engineering. Other existing areas have greatly contributed to the development of this research topic, including: cyber-physical systems, robotics or biological inspired computing. Finally, in section 3.3 we present a summary and discuss about the gap in the state of the art and position our work with respect to it.

3.1 Taxonomy of Self-Adaptive Systems

In this section we propose a taxonomy to facilitate the analysis of the related work in the selected research areas. This taxonomy has a set of dimensions that describe several expected facets of a proactive self-adaptation approach. Some previous research works have addressed the challenge of characterizing the broad area of self-adaptive systems. Salehie and Tahvildari [109] proposed a taxonomy and several representative projects were surveyed in terms of a set of adaptation concerns: *how*, *what*, *when* and *where* of software self-adaptation.

Similarly, Buckley et al. [25] provide a taxonomy based on the objects of change (*where*), system properties (*what*), temporal properties (*when*) and change support (*how*) similar to the taxonomy presented for Salehie and Tahvildari [109] presented in Figure 3.1. Likewise, Andersson et al. [4] describe an extended and general classification of self-adaptive systems including the *goals*, *change*, *mechanisms* and *effects* dimensions. The *goal* dimension includes the objectives the system under consideration should achieve. The *change* dimension deals with the cause of adaptation. The *mechanisms* involve what is the reaction of the system towards change. Finally, the *effects* represents what is the impact of the adaptation upon the system.

In Table 3.1 we reuse four analysis dimensions: *change*, *temporal*, *control* and *realization* dimensions. These four dimensions are subdivided into characteristics allowing us to classify the related work. As mentioned earlier, some dimensions were adopted from previous taxonomies [4, 25, 109] because of their suitability to be applied in our classification. The following subsections further elaborate on the taxonomy dimensions and their characteristics.

Characteristic	Definition	Degrees
Change dimension - What is the cause of adaptation?		
Source	Where is the source of change?	- External context - Internal configuration
Type	What is the nature of change?	- Functional - Non-functional
Uncertainty	What is the kind of uncertainty?	- Noise - Context - Parameters over time - Lack of knowledge
Temporal dimension - When is the adaptation taking place?		
Activeness	Whether changes can be done reactively or proactively?	- Reactive - Proactive - Hybrid
Anticipation	Whether the approach considers predictions?	- Use predictions - Do not use predictions
Frequency	What is the frequency of change?	- Continuously - Periodically - At arbitrary intervals
Control dimension - What is the control structure of adaptation?		
Classification	Main research area that supports the system?	- Architecture-based adaptation - Policy-based adaptation - Reinforcement learning adaptation - Hybrid
Control structure	What is the control structure for adaptation?	- Fixed control - Predictive control - Adaptive control - Reconfiguring control
Realization dimension - How is the adaptation synthesize?		
Decision-making	Is decision process hard-coded or is it dynamic?	- Static - Dynamic
Making-vs- achieving	Is engineering self-adaptation done from scratch or through learning?	- Making - Achieving

Table 3.1 – Selected dimensions to classify proactive self-adaptation approaches

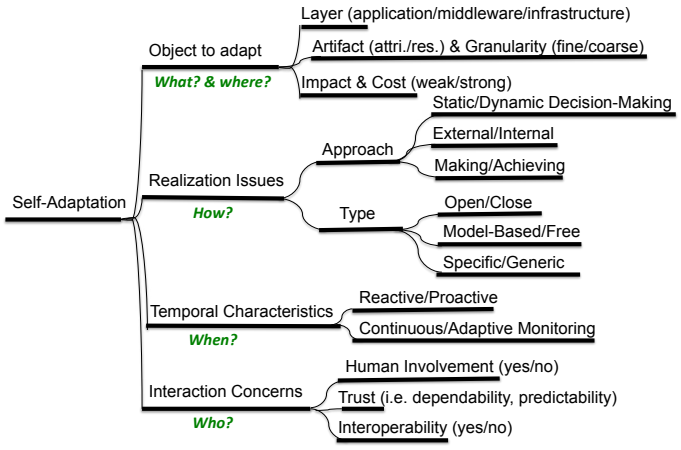


Figure 3.1 – An overview of the taxonomy of self-adaptive systems [109]

3.1.1 Change Dimension

This subsection explores what are the sources of change, thus the initial cause of self-adaptation. The characteristics associated with the change dimension are as follows:

Source: This characteristic identifies the origin of the change, which can be external to the managed element, for instance in its context (e.g. a dramatic drop in the temperature sensor readings). The source can be also internal to the managed element, like a change in the operating environment (e.g. sensors are running low battery) [109].

Type: This characteristic refers to the nature of the change that triggers the adaptations. It can be functional, which involves predictions done over functional parameters (e.g response time) or where predictions are done over non-functional parameters of the software system (e.g. battery life).

Uncertainty: This characteristic identifies the sources of uncertainty. N. Esfahani and S. Malek [43] declare that the root cause of uncertainty is the loose coupling between the autonomic manager and the other elements of a self-adaptive software (i.e. user, managed element and environment), see Figure 3.2. Moreover, they classified uncertainties into two categories: uncertainties due to *variability* and uncertainties due to a *lack of knowledge*.

Uncertainties due to *variability* are those rooted in the fact that the systems' behavior may change due to the ever-changing environment and after adaptation decisions are made. Uncertainty due to variability can be subdivided into the following categories:

- *Uncertainty due to noise*: This source of uncertainty correspond to *"Is monitored"* interfaces and is due to variation in a phenomenon, such as a monitored system parameter, which rarely corresponds to a single value, but rather to a set of values obtained over the observation period. This kind of uncertainty can be either internal and external [43].

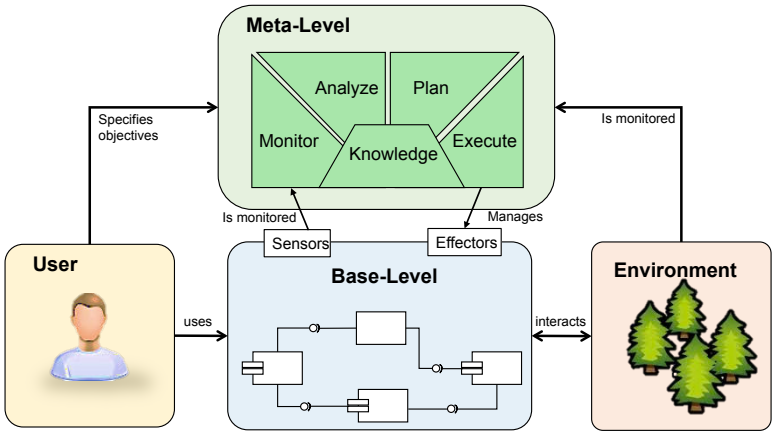


Figure 3.2 – Overview of sources of uncertainty in self-adaptive systems (based on [43])

- *Uncertainty of parameters in future operations*: This source of uncertainty is also related to *"Is monitored"* interfaces in Figure 3.2; it is due to the actual fluctuation in the monitored phenomenon. Without considering the behavior of the system in its future operation, a self-adaptive software may not be able to achieve reliability [43].

- *Uncertainty in the context*: Many self-adaptive software systems are intended to be used in different execution contexts. To that end, the meta-level subsystem is expected to detect the change in the context and adapt the base level to behave accordingly [43].

The authors [43] further elaborate in the category of uncertainty that is due to a *lack of knowledge*.

- *Uncertainty due to a lack-of-knowledge*: it involves uncertainties related to simplifying assumptions, model drift, human in the loop, unclear objectives and decentralization, because of the complexity of the models, loose coupling, ambiguity or distribution. This kind of uncertainty is mainly derived from the interactions between the user and the managed system.

In the scope of this thesis we focus on the uncertainties due to *variability* and we do not focus on the uncertainties due to a *lack of knowledge*.

3.1.2 Temporal Dimension

This subsection explores the time-related aspects of self-adaptation. The characteristics involved with the time dimension are as follows:

Activeness: This characteristic identifies whether a self-adaptive approach can be reactive (changes are driven externally) or proactive (the system autonomously drives changes to itself). Typically, for a system to be proactive, it must contain some monitors that record external and

internal state. It must also contain some logic that allows self-change based on the information received from those monitors [97].

Anticipation: This characteristic considers whether the proactive self-adaptation approach uses predictions to drive the decision-making mechanism or it is based on instantaneous parameter values instead.

Frequency: This characteristic involves the frequency of adaptations on the running system. The categories that can be found in this dimension are: continuously, periodically, or at arbitrary intervals.

3.1.3 Control Mechanisms

This subsection explores the categories regarding the control mechanisms of self-adaptation. The characteristics associated with the control structure dimension are as follows:

Classification: This characteristic identifies the main mechanisms that enable the analysis in self-adaptive systems, we can find architecture-based adaptation, model-based adaptation, reinforcement learning adaptation, policy-based adaptation, probabilistic software analysis.

- ***Architecture-based adaptation*** involves a clearer encapsulation of software-adaptation concerns separating the adaptation logic from the domain logic [97]. Garlan et al. [58, 60] present Rainbow, an architecture-based framework to support self-adaptation of software systems. It allows designers to define adaptation policies that are triggered when the associated invariant is not respected.

  J. Kramer and J. Magee in [82] propose a 3-layer architecture model for self-management, including: the component control layer, the change management layer and the goal management layer. S.W. Cheng et al. [29] propose another 3-layer view approach for architecture-based adaptation, including: the model layer, the task layer and the runtime layer. These proposals clearly identify the importance of monitoring the observed runtime information and filtering upwards those architecture-relevant observations in order to render a high-level view of the system.

- ***Extensive model-based adaptation***: it involves the use of analytical models to reason about the current state of the system, for instance the models@runtime initiative [18] relies in a reflection model of the managed system. Then this model is used to generate new possible reconfigurations that can be validated and evaluated at runtime before redeploying it at runtime. This reconfigurations can be deployed at runtime using tools to build, adapt and synchronize distributed systems (e.g., Kevoree[1] [33, 52]).

[1] http://kevoree.org/

Dynamic software product lines (DSPL) uses abstraction models to reason about the variability and adaptability of the system [26, 92]. DSPL represent the commonalities and variation point of the system using feature models coming from the feature oriented domain analysis [76]. This allows to conceptualize each component-based configuration as a product or variant of the dynamically adaptive system [13].

- In ***Policy-based adaptation*** the choice of the actual implementations of the component is realized via goal policies, expressed as utility functions. Each component implementation is associated with some reward, which precisely specifies the impact of a particular implementation on (QoS) properties. Next, a global utility function, which can aggregate intermediate utility functions, computes the overall utility of the application. In this way, the system can evaluate different configurations and choose the most useful one, using brute force (i.e. by exploring the space of possible configurations). Some approaches that falls on this category are MADAM [2] and its follow up MUSIC Project [108]. The policy-based approach has also been applied to the robotics domain [61].
- ***Reinforcement learning adaptation*** according to Kaelbling et al. [75] is the problem faced by a software system that must learn behavior through trial-and-error interactions with a dynamic environment. There are two main strategies for solving reinforcement-learning problems. The first is to use statistical techniques and dynamic programming methods to estimate the utility of taking actions in states of the world. The second is to search in the space of behaviors in order to find one that performs well in the environment [75]. This approach has been taken by work on genetic algorithms and genetic programming, and it is out of the scope of this thesis.
- ***Hybrid*** We can also have *hybrid* mechanism of adaptation than combine more than one adaptation mechanisms.

Controller structure: This characteristic identifies the options for the controller structure. According to Patikirikorala et al. [102], in a broad sense there are four possible alternatives to control schemes that can be implemented in self-adaptive systems, which are: fixed control, predictive control, adaptive control and reconfiguring control.

- ***Fixed control***: the fixed controllers are a basic type of controllers, where the tuning parameters are set off-line. Thus, after selecting the tuning parameters they remain fixed during the operation time (i.e. they cannot be changed at runtime). Consequently, fixed controllers may be useful for applications whose operations do not have highly divergent conditions [99]. Fixed controllers are reactive in terms that the future behavior of the system is not considered in the current time instance when making adaptation decisions.
- ***Predictive control***: In contrast to fixed controllers described above, predictive controllers uses dynamic models and predict future behavior of the system to come up with the decision to optimize such future behavior. In addition, predictive controllers are considered attractive for multi-objective scenarios in order to derive close to optimal decisions in the presence of complex policies and constraints [121].

- ***Adaptive control:*** Adaptive control address some of the limitations of fixed controllers by adjusting the tunning parameters online. It has the online estimation techniques that construct the dynamic model of the system in each sampling instance. Then given the high-level objectives of the user and using estimated model, the adaptive controller tunes its parameters online [99].

- ***Reconfiguring control***: The reconfiguring controllers are a form of adaptive controllers, but in contrast to adaptive controllers described above, the controller algorithms can be changed at runtime depending on the different conditions of the environment. Multi-Model-Switching and Tuning (MMST) [94] adaptive control is one of the reconfiguring control techniques proposed in the literature with formal stability proofs [102].

3.1.4 Realization Issues

This subsection explores the engineering part and the realization issues regarding self-adaptation. The characteristics associated with the realization dimension are as follows:

Decision-making: This characteristic captures whether the decision making mechanism is *static* or *dynamic*. In other words, whether the decision process is hard coded or can be changed during runtime [109].

Making-vs-achieving: This characteristic captures whether self-adaptivity can be introduced into software systems using a making or achieving approach [70]. The first strategy is to engineer self-adaptivity into the system at development time. which implies engineering the adaptivity from scratch into the software systems.

The second strategy is to *achieve* self-adaptivity through adaptive learning. *Achieving* implies artificial intelligence and reinforcement learning to achieve adaptive behavior. Both approaches do not necessarily contradict each other in the sense that their combination can be utilized as well [109].

3.2 Related Work

In this section we present an overview of the related work in proactive self-adaptation approaches. It is important to highlight that self-adaptive software is inherently a multidisciplinary research area. Its success depends on the combination and appropriate synergy between those disciplines for building a specific self-adaptive software system [109].

In the following subsections we further elaborate on the related work coming from three main disciplines: Software Engineering, Artificial Intelligence and Control Theory/Engineering. However, several other disciplines have greatly contributed and could be added to this list, including: network, distributed computing or optimization theory. But in the scope of this thesis we will focus on the three previously mentioned ones.

3.2.1 Software Engineering

Software engineering for self-adaptive systems has recently received considerable attention with a proliferation of journals, conferences and workshops. For instance, among relevant journals we can find the ACM *Transactions on Autonomous and Adaptive Systems* (TASS) and the *International Journal of Autonomic Computing* (IJAC). Relevant conferences include the *International Conference on Autonomic Computing* (ICAC), the *Self-Adaptive and Self-Organizing Systems* (SASO) and *Self-Organizing Architectures* (SOAR).

In particular the *Software Engineering for Adaptive and Self-Managing Systems (SEAMS*[2]*)* [27, 35] community consolidated a number of workshops and it has a leading role in dealing with the increasing complexity, distribution, and dynamism of many software-intensive systems, such as cloud-based, cyber-physical and pervasive systems. Solutions to complement software with self-adaptive capabilities have been proposed by researchers from different areas including software architecture [97, 59], fault-tolerance [82], operating systems, networking, distributed systems, embedded systems, and even biologically-inspired computing.

Software Architecture-based Adaptation

Poladian et al. [105, 106] propose an anticipatory approach to self-adaptation that combines the benefits of resource prediction research into an existing framework for dynamic reconfiguration. Their approach is based on four concepts: utility function, penalty, time horizon and application profiles. We categorize this approach according to our previously selected dimensions as follows:

1. **Change dimension**: the cause of adaptation is the malfunctioning of resource availability. For this reason their goal is to anticipate such issues and improve utility of the resource usage over the duration of the task. They consider internal and external configuration parameters (e.g. bandwidth, CPU).

2. **Temporal dimension**: this approach considers predictions to drive the decision-making process. For this purpose they implement an auto-regressive models (AR), moving average (MA) and auto-regressive moving average (ARMA) models in order to handle the uncertainty of resources availability (e.g. bandwidth, CPU).

3. **Control dimension**: The authors based the control scheme of their approach on three types of predictors: 1) linear recent history that predicts the next value in the series of resource availability, 2) relative move that predicts step-up or step-down changes in resource availability, and 3) bounding predictor that specifies the maximum and minimum possible levels of resource availability for a union of time intervals. This approach has a fixed control structure because the authors assume that application profiles are static (i.e. they are computed off-line using off-line profiling and do not change over time and are sufficient accurate) [105].

4. **Realization dimension**: Poladian et al. [105] implement their approach using an anticipatory algorithm and then compare a proactive strategy with a reactive one based on certain metrics (e.g. number of resources, penalty, duration of the task).

[2] www.self-adaptive.org

S. W. Cheng et al. [30] built on top of the architecture-based Rainbow [58] framework by turning it from a reactive nature into a proactive approach reusing the work done by Poladian et al. [105]. In [30] the authors describe four potential integration points for resource predictions, namely: monitoring, detection, strategy and effectors. Recently, it was admitted by co-author D. Garlan in a 10-year perspective on self-adaptive systems, that "in many cases it may be better to do things before the problem occurs and the need to balance the cost and benefits of a proactive approach" [57].

1. **Change dimension**: The source of change in the Rainbow framework is the violation of architectural constraints previously defined (e.g. request/response latency). In this approach when the evaluation component determines whether the system is not operating within the accepted range, it triggers the adaptation manager to initiate the adaptation process and choose the adequate adaptation strategy.

2. **Temporal dimension**: Initially the Rainbow framework was conceptually designed to target the self-healing property of self-adaptive systems. This design decision makes it of a reactive nature. However with the integration of the resource prediction component done by Poladian et al. [105] it can support forward looking decisions and can handle uncertainty in the future parameters values. Therefore, we classify this approach as having a hybrid temporal dimension (i.e. reactive and proactive components).

3. **Control dimension**: For the control dimension Rainbow uses Event-Condition-Action (ECA) rules as decision-making mechanism. However, this approach has some limitations, such as a fixed set of reconfiguration strategies. The Rainbow framework reasons about instantaneous parameter values, however at times, the conditions that trigger an adaptation may be more short-lived than the duration for propagating the adaptation change, resulting in an unnecessary adaptation. Second, reactive adaptation lags behind current system conditions, and the degree of that lag depends on the sensitivity of the system sensors to present, versus historical values of a system condition (e.g. CPU load, bandwidth). If the system conditions undergo a dramatic and rapid shift, it may take numerous adaptation cycles for sensors to catch up, resulting in more than one incremental adaptation, while with prediction-based adaptation a single adaptation might suffice [30].

4. **Realization dimension**: the implementation of the adaptation is done by predefined adaptation strategies. Strategy selection is done at run time based on their expected utility. However, a shortcoming is the limited set of adaptation strategies [57], because system's conditions can vary during runtime and some strategies can become non applicable.

Extensive Model-based Adaptation

Over the last couple of years several large-scale joint projects have tackle the challenges of dynamic adaptive systems. The MUSIC[3] European project focused on providing techniques and tools to reduce the time and effort to develop self-adaptive mobile applications. They rely on

[3] http://ist-music.eu/

the notion of component-based framework to describe their applications [108]. A component framework is an assembly of component types (i.e. a template of architecture where component types will be substituted by actual implementations). Quite similarly to the Rainbow approach, the choice of the actual implementations of the component types is realized via goal policies, expressed as utility functions. Each component implementation is associated with some property predictors, which precisely specify the impact of a particular implementation on (QoS) properties. For example, the utility function of a given component implementation could be defined as follows: *utility = w_acc*norm(acc) +w_bat*(1-norm(bat))*, which assumes that the user always prefers high accuracy (e.g., acc) and low battery consumption (e.g., bat) [108]. Finally, a global utility function (which aggregates intermediate utility functions) computes the overall utility of the application. This way, the system can evaluate different configurations and choose the most useful one by exploring the space of possible configurations.

Another joint effort was the DiVA[4] European project, which comprises a set of solutions (e.g., toolkits, components, frameworks and methodology) for developing and executing complex self-adaptive systems. The DiVA approach is divided in two stages: design time and runtime. Design time includes analyzing and modeling the requirements that later drive the development of the adaptation model and the architecture models specify relationships between the variable (i.e. aspect) and static part of the system. At runtime, the adaptation model is processed to produce the system configuration that should be executed. This is performed by a set of runtime technologies that includes: the reasoning framework, the model weaver, that validation component and the causal connection [91].

The use of analytical models to reason about the current state of the system, for example models@runtime [18], fosters a whole new generation of runtime technologies. For instance, Kevoree[5] is an open-source dynamic component model, which relies on models@runtime to properly support the dynamic adaptation of distributed systems [52]. Models@runtime basically pushes the idea of reflection [92] one step further by considering the reflection layer as a real model that can be uncoupled from the running architecture (e.g., for reasoning, validation, and simulation purposes) and later automatically resynchronized with its running instance. Kevoree has been influenced by previous work carried out in the DiVA European project [92] and it is currently used by the ongoing HEADS[6] European project.

In order to provide a proper support for distributed models@runtime paradigm, the Kevoree Modeling Framework (KMF) [53] is mainly based on the following four concepts:

- *Component*: the *component* represents a particular functionality and implements a Producer/Consumer relation. Multiple components can be assigned to a node.

- *Node*: the *node* concept allows modeling the infrastructure topology of the runtime system. A node can host multiple components.

- *Channel*: The *channel* concept allows for multiple communication semantics between remote *components* deployed on heterogeneous *nodes* [51].

[4] https://sites.google.com/site/divawebsite/
[5] http://kevoree.org/
[6] http://heads-project.eu/

- *Group*: The *group* concept allows the user to model the semantics of inter-node communication during synchronization of the reflection model among *nodes*.

All Kevoree concepts (Component, Node, Channel, Group) obey the object type design pattern to separate deployment artifacts from running artifacts [52]. Kevoree supports multiple kinds of execution node technology (e.g., Java, Android, MiniCloud, FreeBSD, Arduino).

Aschoff and Zisman in [6] present ProAdapt a QoS-driven proactive adaptation approach for service composition. They define proactive adaptation of service composition as the detection of the need for changes and implementation of changes in a composition, before reaching an execution point in the composition where a problem may occur.

1. **Change dimension**: the cause of adaptation in [6] is triggered by four different classes of problem: 1) a problem that stop the service composition, 2) a problem that reduce the performance of the service composition, 3) emergence of a new requirement, and 4) emergence of a better service.

2. **Temporal dimension**: From our perspective, the temporal dimension of this approach is not necessarily proactive, because what triggers the adaptation is a reaction in front of a given problem/situation. This approach fits more the self-healing property of self-adaptive systems. For this reason we classified it as a hybrid approach.

3. **Control dimension**: the control dimension in ProAdapt [6] is unclear. However they describe a set of steps that allow them to do proactive adaptation of service composition, namely: i) identification and prediction of problems, ii) analysis of the problems triggered by prediction, iii) decision of actions to be taken due to the problems, and iv) execution of the actions. Taking into account this structured control flow we classified this approach has fixed control structure.

4. **Realization dimension**: ProAdapt implemented the adaptation mechanism using component-based approach. The main components in their system are: (i) composer, (ii) execution engine, (iii) adaptor, (iv) service discovery, and (v) monitor. ProAdapt is based on Exponentially Weighted Moving Average (EWMA) that modeled the service operation response time. A limitation fo this approach is the use a single prediction technique in the adaptation process because a single prediction technique can converge overtime and overfit the data.

Cooray et al. [31] describe RESIST, which uses information from several sources, such as monitoring internal and external software properties, changes in the structure of the software, and contextual properties to continuously provide refined reliability predictions at runtime. Next, these predictions are used to decide about changing the configuration of the software to improve its reliability in a proactive fashion.

1. **Change dimension**: In RESIST the source of change can be either due to *component failure* or *process failure*. Component failure is caused by a fault with in the component's implementation. Process failure happens when one of the component running as a thread within a process exits prematurely, causing all the process to fail [31].

2. **Temporal dimension**: In RESIST the temporal dimension is proactive. RESIST mitigates the uncertainty due to the context and simplifying assumptions through constant learning. Moreover, slight changes in the reliability are modeled as probability distributions indicating the noise.

3. **Control dimension**: In RESIST the adaptation process is clearly organized as a feedback control loop that continuously monitors, analyzes and adapts the system at runtime. This approach implements adaptive control mechanism as it has a configuration selector use quality attributes (e.g. performance) in the selection process of a new reconfiguration.

4. **Realization dimension**: RESIST [31] relies on dynamic learning techniques, specifically Hidden Markov Models (HMMs) to provide continuous reliability refinement and Discrete Time Markov Chains (DTMC) to stochastically estimate the time spent in failures. They define *reliability* as the probability that a system performs its required functions under stated conditions for a specified period of time. According to our classification RESIST has a dynamic decision-making and learning-based characteristics.

Rule/Policy-Based Adaptation

Rules and Policy based approaches for engineering self-adaptive systems propose one paradigm to express the adaptation logic. Their reasoning and decision-making process include formalisms such as probabilistic reasoning (e.g., Markov decision process, Bayesian networks), Event-Condition-Action (ECA) rules [58], and Goal-Based Optimization rules [28].

An ECA rule system typically describes for a particular context to select: when *event* if *condition* choose *action*. A goal-based model typically describes how features impact the quality of service (QoS) properties and when QoS should be optimized (e.g., for a violation of service level agreement). Returning to the metaphor of comparing the autonomic system as a autonomic nervous system [70], ECA rules can be seen as a kind of reflex, since they do not involve major reasoning capabilities. When the system is in a critical context they can quickly reconfigure the system into an acceptable configuration. Whereas goal-based decision techniques match the definition of long-term planning: depending on the resources and the time allocated for reasoning, the system can evaluate different configurations and find the one that offers the best trade-offs.

ECA-rules and goal-based rules have complementary benefits and drawbacks. On the one hand, ECA rules can efficiently be processed at runtime. However, it rapidly becomes difficult to fully specify a self-adaptive system using ECA rules. On the other hand, goal-based rules allows specifying the adaptation logic at a higher level of abstraction. However, processing these rules at runtime is often more costly. Ideally , it should be possible to combine several reasoning algorithms in order to leverage their respective advantage, while limiting their respective drawbacks [91]. In our approach we combine ECA rules and goal-based optimization to drive the adaptation logic.

This category of adaptation is triggered by an occurrence of a violation of a service level agreement (SLA) or service level objective (SLO) in the managed system. Below we discuss prominent research approaches that use this adaptation mechanism.

Hielscher et al. [69] present PROSA, an online testing proactive self-adaptation approach for service-based applications. The authors exploit online testing techniques to detect changes and deviations in functionality or quality of service before they can lead to undesired consequences. The authors propose two strategies to determine the test cases: reusing test cases from the design phase and regression testing. These two strategies have some limitations, test cases from the design phase do not include all possible services situations that can happen during run-time. Regression testing only checks whether changes of (parts of) a system negatively affect the existing functionality of that system and implies re-running previously executed test cases, which causes overhead. We categorize this approach according to our previously selected dimensions as follows:

1. **Change dimension**: In Hielscher et al. [69] the cause of adaptation is a change in the context or a deviation from the expected functionality or quality of service. This source of change is detected by executing generated/selected test cases that run in parallel from the actual software system. Thus, they consider internal configuration and external context of both functional and non-functional properties of the software system. However, the authors do not handle uncertainty in terms of probability of future parameter values in time.

2. **Temporal dimension**: PROSA is introduced as a proactive approach. However the authors admit that they reused test cases from the design (i.e. off-line) phase. Moreover, PROSA is bounded to regression testing, which typical involves to re-run previously executed test cases, thus PROSA does not use predictions of future parameter values to anticipate unknown services states at runtime.

3. **Control dimension**: PROSA approach has a fixed control structure because it involves a structured four steps process: 1) test initiation, 2) test case generation/selection, 3) test execution and 4) adaptation triggering. This online testing process is perform in parallel with the operating applications, the authors admit they have not considered the possible impact that the execution of test cases on the performance of the application may have [69].

4. **Realization dimension**: PROSA approach has an specific adaptation requests explicitly assigned to individual test cases. Thus is a one-to-one mapping of a test case to an adaptation request. PROSA does not consider dynamic decision-making at runtime.

Herbst et al. [68] present a Workload Classification and Forecasting (WCF) technique based on time series analysis to drive resource allocation in a data center environment. The authors propose a novel self-adaptive approach that selects suitable forecasting methods for a given context based on a decision tree and direct feedback cycles together with a corresponding implementation. The user needs to provide only his general forecasting objectives. For building their proactive mechanism the authors rely in a sophisticated spectrum of forecasting methods based on time series analysis, and demonstrated that the ensembles of these methods achieved higher accuracy than individual forecasting methods. Even though we share the notion of prediction models ensembles, in our work we consider applying not only forecasting techniques for times series, but also classification, regression and clustering to a wider extent of data types.

1. **Change dimension**: In WCF approach what causes an adaptation is an anticipation of a given Service Level Agreement (SLA) violations (e.g. average response time) using forecasting techniques. Therefore this approach is focused on functional and non-functional types of change. WFC is strongly validated using real data and is focused on evaluating forecasting methods on internal quantitative parameters (e.g. workload and performance data). However, WFC does not fully anticipate external sources of change (e.g. strong hourly/daily/weekly seasonal pattern) or process qualitative parameters.

2. **Temporal dimension**: WFC anticipates SLA violations using workload intensity behavior (WIB) predictions based on historic values. In the experimental scenario the forecast results are provided continuously over the experiment duration. Thus, WFC handles uncertainty and their decision-making process is dynamic.

3. **Control dimension**: WCF's control dimension is reconfiguring control due to the fact that it proposes a collection of forecasting methods that can be selected at runtime (e.g. naive moving averages, trend interpolation and decomposition and seasonal patterns). The predictions are evaluated and forecast accuracy is feedbacked into the selection mechanism.

4. **Realization dimension**: WFC implements its reconfigurations based on considering the most common forecasting approaches based on the time series analysis (e.g. ARIMA, ARMA, ETS, tBATS) [68]. However, it does not mention other reinforcement learning activities such as clustering or classification for external environmental parameters.

Leitner et al. [84] propose the PREvent framework, which is a system that integrates event-based monitoring, prediction of SLA violations using machine learning techniques, and automated runtime prevention of those violations by triggering adaptation actions in service compositions. In particular, PREvent uses a multilayer perceptron, which is a variant of artificial neural network for predicting service levels violations. A limitation of this approach is that early predictions are less accurate. The authors solve this problem by using estimates, that represent data that is not yet available in a checkpoint, but can in some way be estimated (e.g., the response time of a service that is to be invoked later in the composition).

1. **Change dimension**: the source of change in the PREvent framework is service level agreement (SLA) violations that are detected using machine learning techniques (e.g. regression) from monitored runtime data and then trigger adaptations in the service composition.

2. **Temporal dimension**: PREvent is a proactive adaptation approach due to the fact that its decision mechanism is based in predictions and acts before the SLA violations take place. It also handles uncertainty over time as it manage predictor metrics such as: mean predictor error, prediction error standard deviation and prediction error thresholds.

3. **Control dimension**: the control dimension in PREvent is structured in three phases: 1) monitoring of runtime data, 2) prediction of SLA violations, and 3) identification of preventive actions and application of this actions. According to our classification this is a classic predictive control mechanism. However a limitation is that the prediction of violations is calculated

only at defined checkpoints, rather than continuously. Another limitation is that this approach is based on a single predictor model (i.e. multilayer perceptron).

4. **Realization dimension**: the decision-making process in the PREvent approach is static, due to the fact that it implements the adaptation actions using an XML dialect that are predefined at design time. Therefore PRevent engineers self-adaptation from scratch rather than learning from a running system.

Overall, after going through the literature review we consider that the existing proactive approaches are fragmented and in their initial stages of development. From our perspective, some approaches claim to be proactive [69, 6], however they fit more with the description of a self-healing software system. This is due to the fact that what triggers the adaptation is the response to a problem or a change in the system itself or its environment. We argue that in a proactive system, what triggers the adaptation should be the prediction of the system's parameter or forecasting of future situations [30, 84].

3.2.2 Artificial Intelligence

Software engineering and artificial intelligence (AI) are two fields of the computer science that are compared and contrasted in terms of the problems they attempt to solve, the methods they employ, and the tools and techniques that they use [10]. M. Harman [66] pointed out the relationship between artificial intelligence and software engineering, in particular the author highlighted three broad areas of AI that address challenges that lie in the development of autonomic computing and self-adaptive systems [66]. Those areas are: *Search-based software engineering*, *probabilistic methods* and *machine learning* (e.g. classification, learning and prediction). In this thesis we focus on learning-based mechanisms to achieve proactive self-adaptation, thus below we briefly describe selected related work from our literature review.

Another important concept that can be used in self-adaptive software is the way software *agents* model their domains, goals, and decision making attributes. In particular, *Multi-Agent Systems (MAS)* depend on coordinated models and distributed optimization techniques. In such systems, local and global goals needs to be coordinated [109, 115]. However, as recognized by N. R. Jennings [73] there are two major drawbacks associated with the very essence of an agent-based approach: (i) the patterns and the outcomes of the interactions between agents are inherently unpredictable, and (ii) predicting the behavior of the overall system based on its constituent components is extremely difficult (sometimes impossible) because of the strong possibility of emergent behavior [73].

Reinforcement-Learning Adaptation

G. Tesauro [114] presents *Reinforcement learning* (RL) as an approach for developing effective policies for real-time self-management system. RL aims at learning effective management policies in the absence of explicit system models with little or no domain-specific initial knowledge.

Figure 3.3 illustrates RL's normal operation where an agent learns effective decision-making policies through an online trial-and-error process in which it interacts with an environment. A

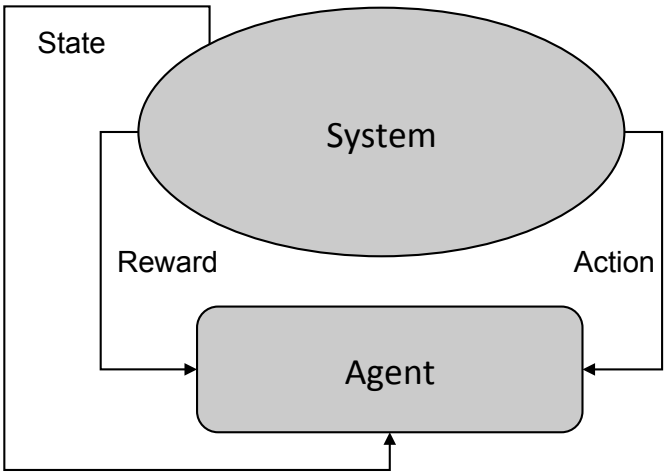


Figure 3.3 – The standard *reinforcement learning* (RL) interaction loop [114]

typical approach is to represent the environment using *Markov decision process* (MDP) with lookup tables used to represent the value function in performing sole action in a state S_t, then receiving a reward r_t followed by an observed transition to a new state s_{t+1} [114].

Learning-based approaches address the problem of uncertainty by quantifying it over time. However, from our own perspective, we categorize [41, 42, 83, 77] as having a reactive adaptation nature, due to the fact that they do not leverage the advantage of historical data to predict future value of parameters in order to avoid conflicting situations. By contrast, other research work clearly take into account predictions of future value of parameters to drive the decision-making process [79, 83, 77].

However, a limitation of reinforcement learning acknowledged by G. Tesauro [114], is when performing training online in a live system, any poor decision RL makes before it has learned a good policy can result in quite poor rewards, and the cost of this can prohibit an online training approach. Another issue is that RL methods generally need to include a certain amount of exploration of actions believe to be suboptimal, purely to facilitate better learning of value functions and policies. These poor initially learned policies and exploratory actions can be too costly in live systems, specially in the domain of safety critical systems.

Elkhodary et al. [41] describe FeatUre-oriented Self-adaptatION (FUSION), a learning-based approach to engineering self-adaptive systems. FUSION uses machine learning, namely *Model Trees Learning* (MTL) to self-tune the adaptive behavior of the system to unanticipated changes. This allows FUSION to mitigate the uncertainty associated to changes in the context of software systems as it gradually learns the right adaptation behavior in the new environment. The results of learning is a set of relationships between the system's adaptation actions and the quality attributes of interest (e.g. response time, availability).

1. **Change dimension**: the source of change in FUSION is a violation of the system's goals. The target of their approach is to reduce interruption by adapting the system only when a goal is violated.

2. **Temporal dimension**: FUSION's temporal dimension in terms of our classification is considered reactive, because the authors admit that FUSION adaptation strategy is "if the systems works (i.e. satisfies the user), do not change it; when it breaks, find the best fix for only the broken part". This resembles the implementation of the self-healing property of self-adaptive systems.

3. **Control dimension**: the control structure in FUSION is based on two complementary cycles: learning cycle and adaptation cycle. The learning cycle relates the measurements of quality attributes to the adaptation actions. The learning cycle constantly monitors the environments to find possible errors in the learned relations. In terms of our classification this is an adaptive control strategy.

4. **Realization dimension**: FUSION implements a learning-based mechanism that enables dynamic decision-making. In particular, they implemented the M5 which is a model tree (MT) algorithm using the WEKA toolkit [64], which provides an open source implementation of a number of learning algorithms.

Esfahani et al. [42] present POssIbilistic SElf-aDaptation (POISED), which is a quantitative approach for tackling the complexity of automatically making adaptation decisions under uncertainty. It builds on possibility theory and fuzzy mathematics to assess both the positive and negative consequences of uncertainty. The goal of POISED is to improve the quality attributes of software systems through reconfiguration of its components to achieve a global optimal configuration.

1. **Change dimension**: the source of change in POISED approach is the violation of resource constraints. POISED focus on internal uncertainty, which is related to the system's internal quality objectives. Therefore they neglect external uncertainty, which is the uncertainty associated with decisions aimed at satisfying the domain objectives [42].

2. **Temporal dimension**: POISED's temporal dimension is hybrid, due to the fact that it is not clear to determine whether the reconfigurations are triggered before problem happens. This approach considers an utility function and calculates the resource usage estimates. Then the system aims at satisfying the worst case (most pessimistic) formulation of the resource constraint. Next, POISED assigns weight to all objectives and selects solutions that do not violate the resource constraint (e.g. memory).

3. **Control dimension**: POISED's control structure is fixed control. The approach was evaluated in the context of robotic software, testing different predefine configurations and alternatives for components of the robot.

4. **Realization dimension**: The adaptation logic of POISED was realized on three steps: 1) generate a probabilistic linear programming (PLP) problem of the system, 2) solve it using conventional linear programing solvers, and 3) change the runtime model using XTEAM [40] API.

Kim and Park [79] use reinforcement learning techniques to enact dynamic adaptation plans at runtime. They propose two planning phases: off-line planning and on-line planning. Off-line planning has the limitation that it uses fixed relationships between situations and configuration of adaptation. On the other hand, on-line planning enables the system to autonomously find better relationships between them in dynamic environments. Finally, they propose an approach to design architecture-based self-managed systems based on Q-Learning, in which for any given situation an appropriate adaptation is selected.

1. **Change dimension**: the source of change in [79] is the detection of environmental changes. Using monitoring techniques the system observes long-term states of the current architecture of the system. If an abnormal situation is detected and adaptation is triggered and this data is passed to the planning phase.

2. **Temporal dimension**: in [79] temporal dimension is given in terms of reacting to changes in the environment. This approach considers goals and scenarios discovery process, then assign conditions (i.e. stimulus) and behavior (i.e. reaction) that can represent a possible state of the system. Thus, contrary to our approach, the work of [79] has a reactive nature.

3. **Control dimension**: the control structure in [79] consists of five phases: detection, planning, execution, evaluation and learning phase. In the detection phase the system monitors the current state of the environment. In the planning phase the system chooses an action to adapt itself to the state. In the execution phase the system applies the action which is chosen (e.g. adding, removing, replacing components). In the evaluation phase the system evaluates the previous action by observing the reward from the environment. Finally, in the learning phase the system uses Q-Learning to accumulate the experiences (i.e. the reward values). Therefore, according to our classification this approach has an adaptive control structure.

4. **Realization dimension**: Kim and Park [79] implement their approach by repeating the process (execution, accumulation, leaning and decision-making). In this way the system can identify better mappings between conditions (i.e. stimulus) and behavior (i.e. reaction), thus improving plans by repeated learning.

Tesauro et al. [116, 114] propose a hybrid approach that combines reinforced learning (RL) (e.g. queueing network) with model-based policies to make resource allocation decisions in data centers. In particularly, the authors use neural networks, a multi-layer perceptron, and claim that many other function approximates could also be used for reinforcement learning (e.g. regression trees, SVMs, regression splines, etc.).

1. **Change dimension**: the source of change in Tesauro et al. [116] is violations on the model-based policies of (e.g. hardware upgrades, changes in SLA).

2. **Temporal dimension**: Tesauro et al. [116] present an approach that involves off-line training on data collected while an externally predefined policy based model makes management decisions in the system. This approach has a reactive nature and does not consider uncertainty or future parameter values.

3. **Control dimension**: the control structure in Tesauro et al. [116] is a fixed control because it considers an external set of fixes policies for managing the system.

4. **Realization dimension**: Tesauro et al. [116] achieve adaptation by using reinforcement learning mechanisms. This approach does not require an explicit model (e.g. workload, traffic) of the computing system being managed. For this reasons the decision-making process is consider dynamic.

Probabilistic Software Analysis

Probabilistic models provide very useful and expressive power to specify uncertain and unpredictable behavior in a quantitative manner. A common approach for modeling the context variability is using Markov Decision process (MDP) or Discrete time markov chains (DTMC) models. These probabilistic models consist of a set of states and transitions between the states that represents the alternative choices to be made according to certain probability tables [114].

However, a limitation of MDP [114] is that real-world problems might not be strict MDPs there can exhibit incomplete observability, history dependence, and non-stationary, so there is great demand for approaches that deals with such non-markovian processes.

PRISM [83] is a probabilistic model checking tool that provides support for building discrete and continuous-time Markov decision processes, and extensions of these models with rewards. Markov Reward Model Checker (MRCM) [77] is another tool for verifying properties over probabilistic models. However, the goal of probabilistic model checking tools is not to try to predict system behavior, rather than that its goal is to formally prove that all possible executions of the system conform to the requirements. Probabilistic model checking focuses on proving correctness of stochastic systems (i. e. systems where probabilities play a role) [96].

Probabilistic software analysis aims at quantifying the probability of a target event to occur during a program execution. There is a set of approaches exploiting symbolic execution to compute the constraints on the inputs leading to the occurrence of a target event; the solution space for such constraints is then quantified given a probabilistic usage profile, which characterizes each input variable by a probability distribution over its possible values [85].

3.2.3 Control Theory/Engineering

Control theory/engineering similar to self-adaptive software is concerned with systems that repeatedly interact with their environment through a sense-plan-act loop [45, 24]. As mentioned in Section 2.4 a feedback loop typically involves four key activities: collect, analyze, decide and act [37].

According to [37], sensor or probes collect data from the executing environment system and its context about its current state in the *collect* phase. In the *analyze* activity the accumulated data are then cleaned, filtered and pruned, and finally stored for future reference to portray an accurate model of past and current states. The *decide* phase then diagnoses the data to infer trends and identify symptoms, and subsequently attempts to plan the future actions to decide how to *act* on the executing system and its context through actuators or effectors [24].

Model-Predictive Control

Model predictive control (MPC) is a specific form of close-loop controller, which is particularly well suited for multi-input, multi-output (MIMO) control problems, were significant interactions occurs between manipulated inputs and outputs. However, MPC needs a predefined model of the system and a standard quadratic programming solver to solve the optimization (or constraint) problem online [102] as shown in Figure 3.4.

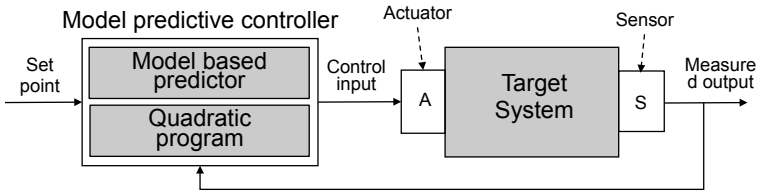


Figure 3.4 – The Model predictive control process [102]

Zhang et al. ,[127] present a control-theoretic solution for dynamic capacity provision which minimizes the energy cost while meeting the performance objectives in a data center environment. This approach uses the Model Predictive Control (MPC) [56] to find the optimal control policy. MPC enables proactive decisions because it employs predictive models for forecasting the future system behavior. However, there is a limitation in the off-line estimation of the prediction parameters, this makes MPC inappropriate for highly changing conditions [8].

1. **Change dimension**: the source of change in [127] is looking forward predictions of usage of each resource type (e.g. CPU, memory usage). The goal is to compute the future trajectory of resource variables to optimize the dynamic provisioning of the system [127].

2. **Temporal dimension**: the temporal dimension in [127] is proactive, which means it is based on forward looking parameters values to optimize the performances of the software system. This approach [127] uses a dynamic model and predicts future behavior of the system to come up with the decisions to optimize.

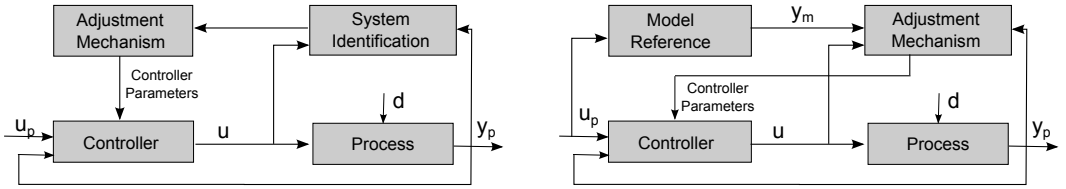
3. **Control dimension**: the control structure in [127] is predictive control mechanism [56].

4. **Realization dimension**: to implement this approach, the authors use the Auto-Regressive Integrated Moving Average (ARIMA) model to predict the time series G_k^r of the usage of resource type r in all the machines at time k [127].

Adaptive/Reconfiguring Control

Y. Brun et al. [24] emphasize that feedback loops should be elevated as first class entities and that they are essential for understanding all types of adaptations. Feedback loops provide the generic mechanism for self-adaptation. Positive feedback occurs when an initial change in a system is reinforced, which leads towards an amplification of the change. In contrast, negative feedback triggers a response that counteracts a perturbation.

The control structure proposed in [24] is adaptive control. In control theory adaptive control involves modifying the model or the control law of the controller to be able to cope with slowly occurring changes of the controlled process. In [24] there are two main control structures: the Model Identification Adaptive Control (MIAC) and the Model Reference Adaptive Control (MRAC), illustrated in Figure 3.5a and Figure 3.5b respectively.



(a) Model Identification Adaptive Control (MIAC) (b) Model Reference Adaptive Control (MRAC)

Figure 3.5 – Two standard schemes for adaptive feedback control loops [24]

"The MIAC strategy builds a dynamical reference model by simply observing the process without taking reference inputs into account. This MIAC system identification element takes the control input u and the process output y_p to infer the model of the current running process (e.g., its unobservable state). Then, the element provides the system characteristics it has identified to the adjustment mechanism which then adjusts the controller accordingly by setting the controller parameters. This adaptation scheme has to take also into account that a disturbances d might affect the process behavior and, thus, usually has to observe the process for multiple control cycles before initiating an adjustment of the controller. [24]."

"The MRAC strategy relies on a predefined reference model (e.g., equations or simulation model) which includes reference inputs and is suitable for situations in which the controlled process has to follow an elaborate prescribed behavior described by the model reference. The adaptive algorithm compares the outputs of the process y_p which results from the control value u of the Controller to the desired responses from a reference model y_m for the goal u_p, and then adjusts the controller model by setting controller parameters to improve the fit in the future. The goal of the scheme is to find controller parameters that cause the combined response of the controller and process to match the response of the reference model despite present disturbances d [24]."

Feedback loops are used in many engineered devices to bring about desired behavior despite undesired disturbances. Below we briefly describe two proactive approaches that use similar feedback control mechanisms.

Patikirikorala et al. [100, 101] propose a Multi-Model Switching-Tuning (MMST) adaptive control to resolve resource allocation problems for workloads operating conditions.

1. **Change dimension**: Patikirikorala et al. [100, 101] consider both internal and external sources of change. For instance, e-commerce systems may face intensive workloads when promotional offers are running or when referenced by high-traffic sites (the so called slash-dot effect). Internally, a software system may change due to bugs fixes, component failures or replacements. The MMST adaptive control is a concept inspired by biological systems. Biological systems have the ability to select an appropriate action for a specific situation from

a collection of behaviors. MMST uses the same concept by selecting the most suitable controller for the current environment that the system is in.

2. **Temporal dimension**: In Patikirikorala et al. [101] the temporal dimension is of reactive nature because the system focuses on adjusting control inputs under-loaded and overloaded characteristic in time. This can be seen as a reaction to varying conditions with an intelligent switching control approach to provide control under these conditions

3. **Control dimension**: The control dimension of MMST is reconfiguring control. However MMST has some limitations. One is the need to come up with proper reconfiguration schemes prior information about the system and environmental conditions. Chattering is another issue that can occur in reconfiguring control. Chattering occurs when a system frequently changes between controllers or different loop configurations without providing desired control. This could lead to drastic performance degradations [101].

4. **Realization dimension**: the implementation of the MMST is done in four types of schemes: 1) Type 1: all adaptive models, 2) Type 2: all fixed models, 3) Type 3: one adaptive model and one fixed model, and 4) Type 4: two adaptive model and two fixed models. However, the authors [101] admit that MMST should only be chosen when a single fixed or adaptive controller cannot provide the effective performance in the entire operating time. Since MMST is a reconfiguring control scheme, it is subject to some limitations such as performance, overhead and chattering.

3.3 Synthesis

In this chapter we have presented the state-of-the-art of proactive self-adaptive systems. The chapter was conceptually divided in two parts. In the first part we presented a taxonomy that allowed us to classify the different self-adaptation approaches. This taxonomy defined a classification of modeling dimensions that should be considered when modeling proactive self-adaptive software systems.

Other similar taxonomies exist, for instance Dobson et al. [37] provide a survey of techniques applied to autonomic communications. Buckley et al. [25] define a taxonomy for software change that does not focus on runtime adaptation but focus on the sources of software change. Salehie and Tahvildari [109] survey on autonomic computing and self-adaptive systems inspired some of the dimensions and their values considered in our work. Figure 3.1 presents a taxonomy of the approaches present in the state of the art. In the following chapter we reuse this taxonomy in order to position our work regarding this well-established taxonomy.

In the second part of this chapter we presented a number of research approaches related to proactive adaptation. Table 3.2 presents a deeper analysis on the challenges introduced in the motivation scenario and transforms them into requirements for our approach. These targeted features are: *proactivity*, *predictability* and *reliability*. We analyze the related state-of-the-art with respect to this criteria as follows.

Approach	Proactivity	Predictability	Reliability
Software architecture-based adaptation	Adaptation is based on self-healing property. A problem occurs and the system address it	Decision process based on instant utility. Monitoring instant values of sensors and gauges	Fixed set of predefined adaptation strategies [57]
Extensive model-based adaptation	No proactivity (if not used in combination with a dedicated approach to self-adaptation)	Analysis based on reflection of the current running configuration hindered by model-drift uncertainty [43]	Good management of variability and explosion of the number of possible reconfigurations
Rules/Policy-based adaptation	Adaptation triggered for violation of "hard wired" rules in the model (e.g., ECA)	No predictability (if not used in combination with a specific forecasting technique e.g., workload forecast)	A limited management of the possible reconfigurations usually predefined at design time
Reinforcement learning adaptation	Adaptation triggered for violation of system's goals (e.g., SLAs)	Environment is typically formulated as a *Markov decision process* (MDP) decisions based on *reward* and *regret* criteria	Possible configurations are given in a set of states; a set of actions; and rules of transitioning between states
Model-Predictive control	It is a proactive and self-optimization technique	Use prediction of future behavior of the system to optimize decisions	Attractive in multi-objective scenarios and to derive close to optimal decisions [102]
Adaptive and Reconfiguring control	Reactive adaptation based on input parameters	No predictability (if not used in combination with an specific predictive controller e.g., MPC)	The controller can tune and adjust their algorithm at runtime (e.g., MMST) [101]

Table 3.2 – Summary of features in related approaches

Chapter 4

Achieving Proactivity Based on Predictive Analysis

In the previous chapters, we have identified the enablers of proactive adaptation for our approach. In particular, we explored in detail the predictive analysis and autonomic computing paradigms. Also, we highlighted the gap in the-state-of-the-art and the need for a new approach that makes use of the overabundance of data in current pervasive systems to derive forward looking decision-making mechanisms.

This chapter presents the key aspects and the main contribution of our thesis. The remainder of this chapter is structured as follows. Section 4.1 presents our framework to achieve proactive self-adaptation. Section 4.2 presents the supporting tools that enable the implementation of our approach.

4.1 Overview of the Approach

As mentioned earlier, in 2001 IBM releases a manifesto [70] describing the vision of autonomic computing. The purpose is to control the overgrowing complexity of software systems by making systems self-managing. A couple of years later, J. Kephart and D. Chesss [78] presented the MAPE-K autonomic element architecture, which became the de facto reference model for autonomic computing.

In this thesis we propose to enhance dynamic adaptation by integrating a *Predict* phase between the *Analyze* and *Plan* phases of the MAPE-K loop, see Figure 4.1. We leverage ideas and techniques from the area of predictive analytics [126] to operationalize the *Predict* phase. The components that operationalize our predictive framework can be divided into two phases: the design phase and the runtime phase.

The design phase include the off-line setup of the approach. In this phase we perform activities related to the predictive modeling process. This includes activities such as: data collection, data preprocessing, build candidate model, train, test and evaluate predictive models based on past data observations.

The run time phase involves the activities mentioned in the MAPE-K loop, starting with

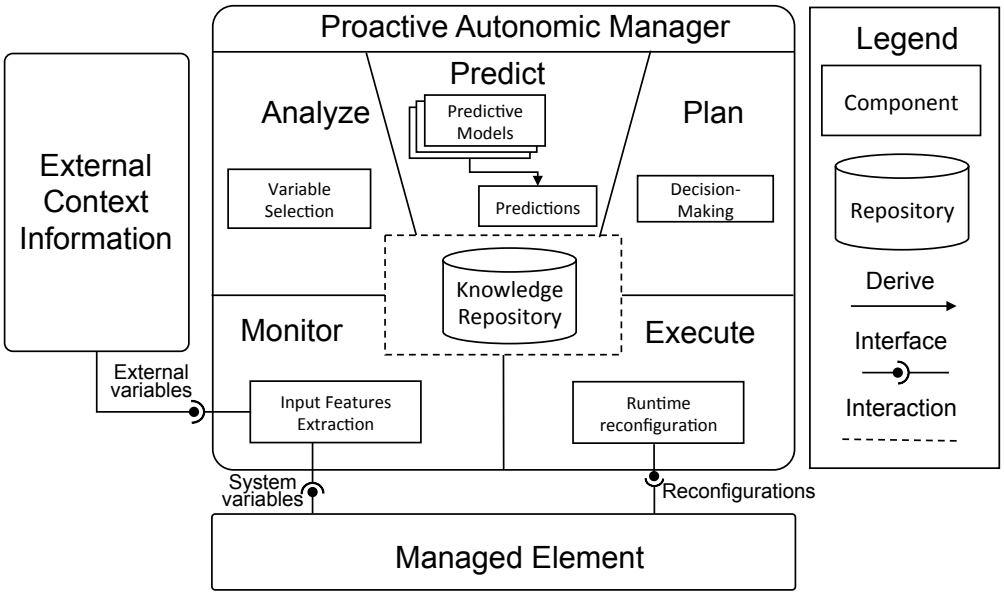


Figure 4.1 – Extension of the MAPE-K autonomic control loop

online monitoring and analysis of the current state of the system. During the on-line phase we evaluate the predictive models defined at design time. This evaluation is performed in two ways: reactive vs. proactive adaptation strategies. Next, taking into consideration the evaluation results we proceed to the decision making process. The on-line phase then carries on the new reconfigurations.

4.1.1 The Design Phase: The Predictive Modeling Process

The process of building predictive models is a craft. This is no different from other software engineering activities such as requirements specification [119], software design [120], or testing [74]. In this section we present a stepwise process to build predictive models and give away some rule of thumb that summarizes our intuition and experience, aiming to guide future practitioners. Figure 4.2 illustrates the predictive modeling process, which is iterative and should be refined over time. As in [126], our approach implements the following steps for building a predictive model.

1. ***Define goals***

   This step is at the heart of the process. Here, it should be clearly stated what the system is trying to achieve, the outcome to be predicted. It is also fundamental to have a clear understanding of the purpose of the model and to determine what is a good model. There exists a variety of quantitative objective functions to measure the model goodness. Wu [126], categorize the prediction approaches in the following categories: (1) linear modeling and regression, (2) non-linear modeling, and (3) time series analysis. For a system to be considered linear the objective function has to obey two properties: additivity (see Equa-

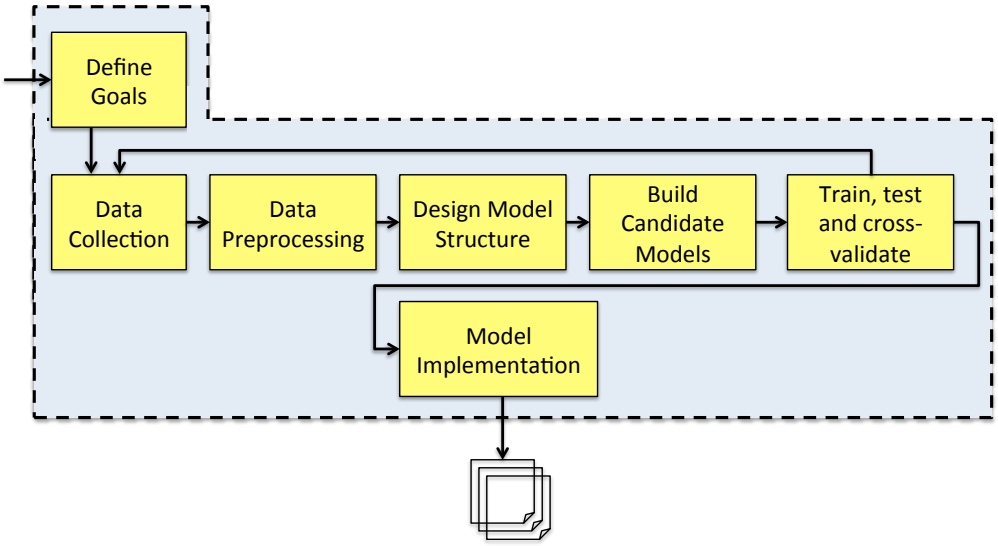


Figure 4.2 – The Predictive Modeling Process

tion 4.1) and homogeneity (see Equation 4.2). Additivity implies homogeneity for any rational α, and for continuous functions, for any real α. Additivity and homogeneity are often combined in the *superposition principle* [126].

$$f(x+y) = f(x)+f(y); \tag{4.1}$$

$$f(\alpha x) = \alpha f(x) \tag{4.2}$$

In a continuous dependent variable, typically, the ultimate goal of predictive modeling is to develop a finely tuned predictor function $h(x)$ (a.k.a. the *hypothesis*). The *learning* part consists in using mathematical algorithms to optimize this function so that, given an input data x about a certain domain (e.g., number of users in the network), it will accurately predict some interesting value $h(x)$ (e.g., predict bandwidth usage).

In practice, x almost always represents multiple free parameters (e.g., number of users is x_1, date x_2, time is x_3, etc). Assuming a single input value is used, the simple objective function has this form:

$$h(x) = \theta_0 + \theta_1 x \tag{4.3}$$

where θ_0 and θ_1 are constants. The goal is to find the perfect values of θ_0 and θ_1 to make our predictor work as well as possible.

Nonlinear models implies more complex polynomial equations (e.g. in the equation $(x+2)^2 = 6$, where x is raised to the power 2). For instance, a binary dependent variable can

be easily illustrated in the three dimensional space. Let us consider a situation where we have two free parameters (x and y) and an *objective function* z-direction rising above an (x,y) plane. If the *objective function* is the model error, without loss of generality we can consider that the goal is to minimize this objective function over the set of all possible values of these free parameters. The objective function is then a surface that rides above this $x-y$ plane, and the goal is to find the minimum, thus having the lowest error [126]. This analysis can be extended to higher dimensions.

A common pitfall when finding the objective function is the lack of clarity around the problem definition. Lack of understanding on how and where the model will be used ends up in solving a problem different from the specifically targeted one. Sometimes key data is not available and should be noticed since the beginning of the process.

2. ***Data Collection***

   This step involves gathering the data. For this purpose its necessary to answer several questions: what data is available and in what form, with what quality, how many data points do we need to perform the task at hand, what is the data type of the records and how far back in time does the data need to go. Together with this, is important to find out what kind of data we are dealing with. Data is heterogeneous and can be highly structured, semi-structured, or totally unstructured.

   Structured data is generally specified in a data modeling notation, which is often in a graphical form (e.g., spreadsheets or a relational data-base). Semi-structured data has some form of structure but its structure is not helpful for the processing task at hand (e.g., books, free-text, images, audio, video, files). Techniques such as natural language processing (NLP) or image processing deal with this kind of data. Totally unstructured data does not have a predefined data model and is not organized in a predefined manner (e.g., the CERN's Large Hadron Collider, where 150 million sensors are capturing data about nearly 600 million collisions per second [63]).

   Machine learning algorithms have the ability to successfully analyze both structured and semi-structured data. However, totally unstructured data still poses a challenge and requires making several trade offs among desired calculability, availability and performance. Finally, a rule of thumb is the more information employed in the model generation the better, however quality of data is key, many observations does not guarantee a good model (e.g., garbage-in, garbage-out effect).) [126].

3. ***Data Pre-processing***

   In this step the data records should be assembled into the appropriate form to serve as input for the predictive model. It is important to highlight that we should not expect the model to do all the difficult work. In practice, it is useful to prepare and encode the inputs as best as possible using expert knowledge and statistical practices [126]. In general, the first task to perform is to do simple data quality examination and exploration of univariate distributions. We can examine each field in the data separately. For continuous fields we can calculate its distribution, its mean, standard deviation, min, max, etc. For categorical

variables we can inspect the number of occurrences of the most common and uncommon values of the field to get an idea of the completeness, coverage, and quality of the data.

Common pitfalls in this step include not cleaning the data or taking the outliers into account, inefficient encoding of categorical variables, not eliminating fields that have inherent bias, missing values for important categories or records, not properly scaling continuous fields (e.g., computing the mean μ_i and standard deviation σ_i). A good practice is to use filters or wrappers to integrate several variables. The impact of not preparing the data is that the predictive model receives noisy data. This directly impacts the predictive model and does not allow it to focus its efforts in the right areas. In general, it reduces the model potential for useful outcome.

4. ***Design Model Structure***

   In this step the decision on the model structure needs to be made. This depends up to a certain way on the data type of the records. For instance, we can build a classification tree model for a categorical outcome, whereas a regression tree is designed to handle continuous variables.

   An important aspect in deciding the model structure is the kind of available data and whether the desired output is known. Next, we can consider applying *supervised learning*, *unsupervised learning* or *reinforcement learning*. In supervised learning the predictive model is trained with example inputs and their desired outputs. The goal is to learn a general rule that maps inputs to outputs. In unsupervised learning no labels are given to the learning algorithm, leaving it on its own to find structure in its input. Unsupervised learning can be a goal in itself to discover hidden patterns in data [126]. In reinforcement learning, the algorithm explores and interacts with a dynamic environment in which it must perform a certain goal without explicitly telling it whether it has come close to its goal or not. Therefore, it learns effective decision-making policies through trial-and-error interactions with its environment based on the concept of reward [114].

   Predictive modeling approaches can be categorized in many ways [15], for instance: (1) *classification*, i.e. predicting the outcome from a set of finite possible values, (2) *regression*, i.e. predicting a numerical value, (3) *clustering* or segmentation, i.e. summarizing data and identifying groups of similar data points, (4) *association analysis*, i.e. finding relationships between attributes, and (5) *deviation analysis*, i.e. finding exceptions in major trends or structures.

   Deciding on a model structure requires experience and knowledge on each technique. Important characteristics to consider in the choice of predictive modeling techniques include continuous or categorical (classification) outcomes, number of records, likely dimensionality and amount of data noise. A good approach is to build a simple prototype linear model as a base line and then try various non linear to see the improvement. Common mistakes on this step include using too simple a model (e.g., univariate) or too complex one that might over fit the objective function.

5. ***Build candidate models***

Model building is the process of finding a functional relationship between inputs and outputs. The stepwise variable selection method is among the most common in practice [126]. In forward selection we start with no selected variables out of the pool of n possible model variables. We then build n separate models, each using exactly one of the possible variables. We select the single variable that alone provides the best model. We then progress by building $n-1$ models, each of them using the previously selected variable along with all the other $n-1$ possible variables. We then select the two variables that provide the best model and proceed to build $n-2$ models using the already selected two variables and all the other possible variables individually, and so on.

A good practice is to start this step by modeling a baseline linear model and then try to improve it using more complex nonlinear models or time series analysis [126]. In previous Section 2.2.2, we introduced the concept of time series analysis.

In this step it is important to assess the model goodness (see below), keeping in mind the defined goals. A common mistake in predictive modeling is to go too deep on a single specialized technique instead of trying a broad spectrum of methods.

6. ***Training, testing and cross-validation***

In the construction of models the goal is to find the best fit of a mathematical expression (e.g., formula, rules, etc) to a set of given data by adjusting free parameters in the model. In this fitting process the objective is to find this set of best parameters according to two things: (1) some quantitative measure of goodness of fit to an objective function, and (2) the need for the model to generalize beyond the particular given data set .

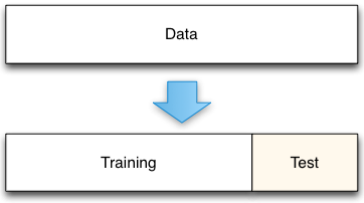
In general, these are competing and somewhat conflicting goals. In practice, we can fit the model exactly to the given data, but when new data comes, sometimes the fit is not as good for this new data. Therefore, the standard practice of separating data into training, testing and validation sets has become obligatory in machine learning process. In this step a rule of thumbs is to separate the data into two sets: one with 70 percent of the source data, for training the model, and one with 30 percent of the source data, for testing the model. This default was chosen because a 70-30 ratio is often used in data mining [87]. Figure 4.3 represents both ways of dividing the data using holdout data split using the 70-30 ratio, or a 5-folds cross validation data split.

Model goodness measures. The standard approach to evaluate the performance of a classifier is to construct a confusion matrix (or contingency table) by comparing the answers from the classifier and manually labeled answers:

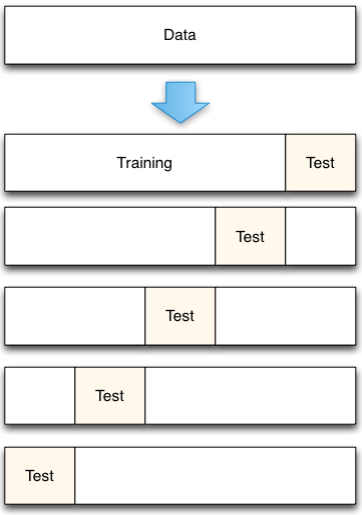
	correct	wrong
predicted	TP	FP
not predicted	FN	TN

- **True positives (TP)** are elements correctly predicted by the classifier;

- **False positives (FP)** are elements incorrectly predicted by the classifier;



(a) Holdout data split [50]



(b) 5-Fold cross-validation data split [50]

Figure 4.3 – Training, testing, and cross-validation data splits

- **True negatives (TN)** are elements correctly not predicted by the classifier;
- **False negatives (FN)** are elements incorrectly not predicted by the classifier

Based on the four previous metrics, we can compute a variety of useful statistical measures [126]:

$$Precision = \frac{TP}{TP+FP} \tag{4.4}$$

$$Recall = \frac{TP}{TP+FN} \tag{4.5}$$

$$F1 = \frac{2 \times Precision \times Recall}{Precision + Recall} \tag{4.6}$$

The number *Precision* represents the probability that the predicted elements are relevant. *Recall* represents the probability that all relevant elements were predicted. For our purposes, Recall is more important, because we can manually delete redundant elements from the predicted domain model. However, in machine learning, the *F1-measure* (a.k.a. F1-score) is used because it represents both *Precision* and *Recall* in a balanced way. Not doing proper training and testing as one examines candidate models may end up in selecting a bad model or overfitting functions.

7. ***Model implementation***

   In this step the model is embedded into the necessary system process, additional steps should be performed to examine the model's performance in ongoing use. Errors in this step include implementation bugs, algorithm mistakes, bad data input streams (e.g., avoid garbage-in, garbage-out effect). More information regarding the implementation of the predictive models in the context of the motivation scenario is presented in Section 5.2.4.

Predictive models are usually a representation of more complicated systems and their objective is to help us understand what is going on, how things interact or to predict what may happen when things change or evolve over time. In this section we presented some guidelines on the main steps of the modeling process. As a summary, the following four components are key in order to build a predictive model [126]: (1) a set of equations or formula with adjustable parameters, (2) a set of data that represents an example of the system that we are modeling, (3) a concept of goodness of fit to the data (e.g., objective function), and (4) a set of rules to tell us how to adjust the parameters to increase the goodness of the model fit.

4.1.2 Integration Points in the Runtime Phase

In order to enable proactive adaptation, there must be integration points of the predictive framework into the autonomic control loop. Figure 4.4 illustrates the potential site for integrating predictions into the autonomic loop.

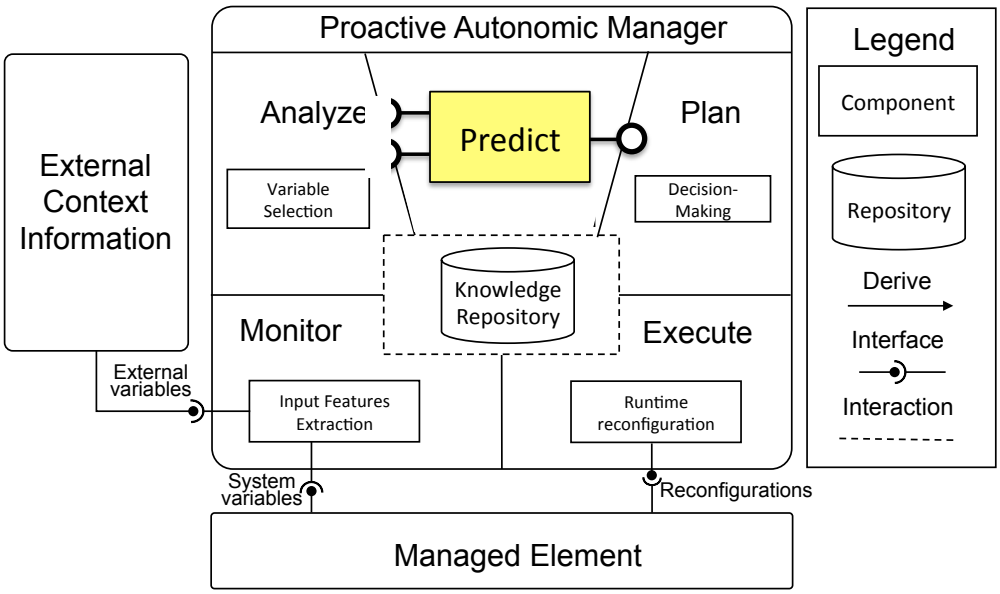


Figure 4.4 – The proactive architecture [98]

1. ***Input feature extractions***: This component is part of the monitoring phase and is in charge of keep record of internal and external properties. It reads system state variables that

hold information about the operating environment (e.g., transmission rate, battery level). Regarding external context information, the monitoring module also observes external information relevant to the system (e.g., weather forecast). In order to make proactive adaptation decisions the data is stored in the knowledge repository.

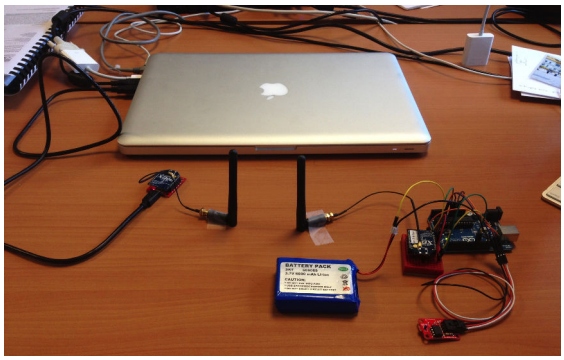


Figure 4.5 – Testbed for data collection phase

As a proof of concept of the feasibility of input feature extraction we developed a testbed built on top of a MacBookPro with four-core, hyper-threaded i7, SSD, 16GB of DDR3 memory. This host has a XBEE antenna receiving sensors readings every 5 seconds sent via XBEE from an Arduino Uno, which was instrumented with a temperature sensor TMP36, as depicted in Figure 4.5. The source code deployed in the Arduino Uno from our testbed is listed in the following Listing 4.1.

```
1  #include <SoftwareSerial.h>
2
3  SoftwareSerial mySerial(10, 11); // RX, TX
4
5  //TMP36 Pin Variables
6  int temperaturePin = 0;
7
8  // set up a new serial port
9  //SoftwareSerial mySerial =  SoftwareSerial(rxPin, txPin);
10
11 void setup()  {
12   // define pin modes for tx, rx, led pins:
13   //pinMode(rxPin, INPUT);
14   //pinMode(txPin, OUTPUT);
15   // pinMode(ledPin, OUTPUT);
16   // set the data rate for the SoftwareSerial port
17   mySerial.begin(9600);
18   mySerial.println("Arduino started sending bytes via XBee"
       );
19 }
```

```
20
21 void loop()                     // run over and over again
22 {
23 //getting the voltage reading from the temperature sensor
24  float temperature = getVoltage(temperaturePin);
25  //converting from 10 mv per degree wit 500 mV offset
26  //to degrees ((volatge - 500mV) times 100)
27 temperature = (temperature - .5) * 10;
28 //converting from F to C degress
29 temperature = ((temperature - 32.0) * (5.0 / 9.0));
30
31  //printing the result
32  mySerial.println(temperature);
33  //waiting 30 seconds
34  delay(30000);
35 }
36
37 float getVoltage(int pin){
38 //converting from a 0 to 1023 digital range
39 // to 0 to 5 volts (each 1 reading equals ~ 5 millivolts
40  return (analogRead(pin) * .004882814);
41 }
```

Listing 4.1 – Listing of input data from temperature sensor

2. ***Variable selection:*** This module is in charge of organizing the collected data according to their spatial and temporal dimensions. At run time, this module is in charge of processing the current values that represents the running configuration of the system. It is important to note that this data should be comparable with the one collected in the design phase. For this reason it involves cleaning, filtering, pruning of data and replacing missing values with reasonable estimates (e.g., by interpolation). Re-dimensional analysis can be performed in case the frequency of measurements is different from the prediction scale.

   For instance, Figure 4.6 represents the outdoor temperature during one day, sampled every 30 seconds. As shown in the previous Listing 4.1 we can clearly see in lines 28 and 29 that the temperature measurements are originally given in °F, and we convert it to °C to be comparable with our policy rules. However, this test bed was not sufficient for our real experiments due to the limitations from the lack of historical data. Another constraint was that the measures where taken in Rennes, which is located in the north-west of France. The weather conditions are rather rainy and humid, which are not prone to wild fires breakouts. For these reasons, we need realistic data and this process is further explained in the section 5.2.1.

3. ***Predictions***: This module receives input data from the current state of the system and from external context information through the analyze module in a processed way. This step

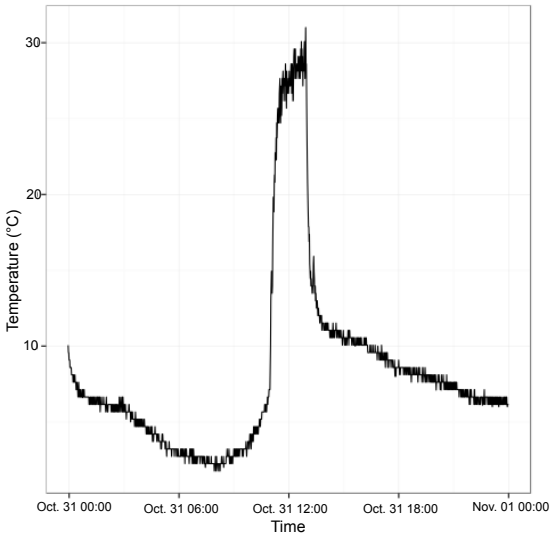


Figure 4.6 – A day of temperature readings from the testbed

involves consuming the predictions and realizing a prediction quality assessment model predictor error. When building prediction models the primary goal should be to make a model that most accurately predicts the desired target value for *new* data. Naturally, any model is highly optimized for the data it was trained on. The expected error that the model exhibits on new data will *always* be higher than that it exhibits on the training data [50].

One way to assess the quality of a predictor model is by calculating the *mean-squared error* (MSE) of the estimator. The MSE of an estimator measures the average of the squares of the "errors", that is the difference between the estimator and what is estimated. This is formally defined in [111]:

If $\hat{Y}$ is a vector of n predictions, and Y is the vector of the true values, then the (estimated) MSE of the predictor is:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(\hat{Y}_i - Y_i)^2. \tag{4.7}$$

An MSE of 0 means that the estimator $\hat{\theta}$ predicts observations of the parameter θ with perfect accuracy, which is an ideal impossible in practice [111].

Next, an important point is the way predictions will be consumed by the adaptive system. These methods can be on different ways:

- *Consume predictions periodically*: Predictions are generated and provided continuously to the running system at a fixed rate (e.g., hourly, daily, weekly).
- *Consume predictions on demand*: Predictions are queried by the running system on demand (e.g., to estimate the remaining battery life).

- *Consume predictions sporadically:* Predictions are triggered every time something change in the environment (e.g., when the monitored parameter crosses a threshold).

4. ***Decision-making****:* The *plan* module is in charge of defining the adaptation policies that may drive the reconfiguration of the system. Most of the current self-adaptive approaches use purely reactive decision techniques. Proactive policies are able to handle uncertainty by carrying aggregate information of previous historic data plus the foresight into the prediction horizon. The following ECA rule describes the reasoning behind the triggering of a new reconfiguration (e.g., increase or decrease in the transmission rate of sensor nodes) based on 7-observations into the future.

   Using the temperature prediction, we have implemented the following ECA rule:
   Event: An increase or decrease in temperature.
   Condition: If the temperature crosses the following threshold in ascending or descending manner.

   - High (e.g., t >= 25 ˚C),
   - Medium (e.g., 15 ˚C >= t < 25 ˚C) and
   - Low (e.g., t < 15 ˚C).

   Action: The system will increase or decrease the transmission rate one level up or down accordingly.

5. ***Runtime reconfiguration*** The goal of this component is to deploy the new system configuration decided by the planning module on the running system. As a proof of concept, it is possible to use Kevoree [52], which is an open source tool that implements the *models@runtime* paradigm to provide a reflection model of the running system. Then, it allows edition of this model and generation of a new target model. Next, the target model can then be redeployed and synchronized with the running system.

4.1.3 Application Scenarios

The goal of self-adaptive software is to modify its own behavior in response to changes in its operating environment, such as end-users input, external hardware devices and sensors, or program instrumentation [96]. However, one significant challenge lies in the realization issues of engineering self-adaptive systems. These issues deal with selecting the correct mechanisms for incorporating adaptivity into such system. Therefore, the paradox is that systems must become more complex in order to achieve self-management capabilities.

A proactive adaptation strategy provides a number of benefits, but at the same time it raises several computational challenges. In order to exemplify such benefits and challenges we refer to the following scenarios:

Scenario 1: Daily observations at a shopping mall that offers free WiFi reveal some curious patterns. Students from nearby university flood the coffee shops during the class breaks, while

the professionals working in the area visit the shops early in the morning, during lunch time, and late at evening. Based on the computer usage in the main lounge area of different groups, the wireless bandwidth availability at the shopping mall may show repeatable patterns.

Scenario 2: A personal laptop computer running on battery has several scheduled background tasks (e.g., updates, backups, virus scans, etc.) that run daily, weekly, etc. Often the execution of these administrative tasks can not be interrupted or suspended by the user once they have started, but the time and length of the execution is known exactly in advance. The resource utilization imposed by such task (e.g., power consumption) certainly has known behavior and can be predictable.

Scenario 3: World class events usually happen periodically (e.g., monthly, yearly) and attract many fans from all around the globe. Sport events such as the ATP Tennis Tour, the NFL Super Bowl, and the FIFA World Cup, greatly increase the ISP bandwidth traffic and usually generates peaks due to the number of followers, even more during the transmission of the final event. Studies have been conducted to analyze the workload characterization during such popular events [5].

Benefits

Some benefits of the proactive strategy might include:

A. *Avoiding unnecessary adaptation and oscillatory behavior*

The context in which adaptive systems evolve is very dynamic and can potentially change more rapidly than the adaptive system itself. Resources such as bandwidth can fluctuate and depend on many external parameters. Similar to the shopping mall scenario many of these external parameters are out of control, including: number of users in the network, size and quality of data exchanged, quality of the network, etc. However, those parameters can be modeled in terms of time series data to extract the pattern in its occurrence.

Now, let us assume a user is browsing an adaptive news website similar to Znn.com[1] proposed in the Rainbow approach [58]. In the case where the bandwidth fluctuates around a predefined threshold, the website triggers a reconfiguration in the news page display format (e.g., video/photo/text). However, this could make the page to continuously oscillate between two configurations. Even worse, if the fluctuations of the bandwidth are quicker than the adaptation process, this could set the news system always in a reconfiguration that is lagging off and not updated to the current context. In that case, it would be preferable to switch the bandwidth parameter to a safe mode option. Here, the safe mode would be to consider that the bandwidth is always low, even if it is sometimes high for short periods of time.

B. *Managing allocation of exhaustible resources:*

This benefit is directly related to the second scenario when we are dealing with perishable, non-renewable resources such as battery. Managing allocation of exhaustible resources (e.g. battery) with proactive adaptation enable us to make provision for future time when the

[1] `http://rainbow.self-adapt.org/RainbowZnn`

resource is scarce. In the context of wireless sensor network a proactive strategy can outperform a reactive strategy in terms of power consumption due to a lower number of triggered reconfigurations [98].

C. *Proactivity in front of seasonal behavior*

Identifying a known pattern component is challenging and might require domain-specific knowledge. Scenario three presents a real example of recurrent events. In the case we have to predict the number of attendants to the next Roland-Garros tennis tournament, we could simple collect historical data from last thirty years and make a linear regression. A more complex study might include correlations between this variable and economic indicators.

For particular cases more suitable for our purpose, we can tackle the detection of seasonal components using a time series decomposition analysis. This analysis extract three main components from the observed data: trend, season and noise. Proactive adaptation to seasonal behavior offers the benefit of adapting the systems before a significant change in the environment (e.g., increase in client requests), thus avoiding potential disruption by adapting the system, when the system is already under heavy load.

Challenges

However, a proactive system needs to compute and communicate predictions at runtime. Which means, there are operational challenges and an increase in computational complexity associated with providing these functionalities. In particular, the provider and the consumer of predictions need to coordinate on the syntax and semantics of the predictions. Both parties need to agree on a small number of important parameters that can help fully and adequately describe predictions [106].

A. *Prediction horizon*

The prediction horizon involves the future interval of time units over which predictions are desired. For instance, if we are interested in a 7-day forecast of weather, then the prediction horizon is 7 days. It is important that the provider of the predictions and the consumer of predictions agree on the value of this parameter in order to ensure that the predictions are useful to the consumer. Indeed, if a prediction consumer needs a prediction of the bandwidth for the next 10 minutes, then providing a prediction for the next 5 seconds will not be useful for the consumer.

B. *Prediction scale*

Prediction scale or granularity is the time window of a single prediction in a sequence of predictions. A tourist on a week-long holiday trip might need a daily weather forecast with a 7-day horizon, while a rock climber might need an hourly forecast with a 36 hours horizon. In the first case, the scale of prediction is 24 hours, while in the second case it is 1 hour. A prediction at a daily scale will not be acceptable for the climber.

C. *Prediction detail*

Prediction detail refers to the amount of information used to describe the probability distributions of the predictive uncertainty at runtime. On paper, predictors can be described analytically, (e.g. using mathematical functions). At run time, it may be more appropriate to use discrete data structure representations to express and communicate predictions. When converting predictions expressions using continuous functions into a discrete representation, the system providing predictions needs to make approximations, resulting in loss of detail. It is important for the prediction communication mechanism to define parameters that control the detail of approximation so that the consumers and the providers of predictions can synchronize.

Supporting these three parameters in a prediction API helps narrow the space of useful predictions from the perspective of the consumer. It is likely that the prediction provider can only support a limited range of these parameters; therefore, the API must allow for a negotiation between the consumer and the provider to agree on the acceptable values of these parameters. In Section 5.2 we describe the implementation of the API, and demonstrate how it addresses the operational issues in the context of the environmental monitoring motivation scenario.

4.2 Implementation of the Approach

In order to implement our approach we used several open source technologies, including a graphical predictive analytics platform (KNIME), a standard XML language for predictive modeling (PMML) and various open source machine learning libraries/tools (R, Weka, Octave, jpmml).

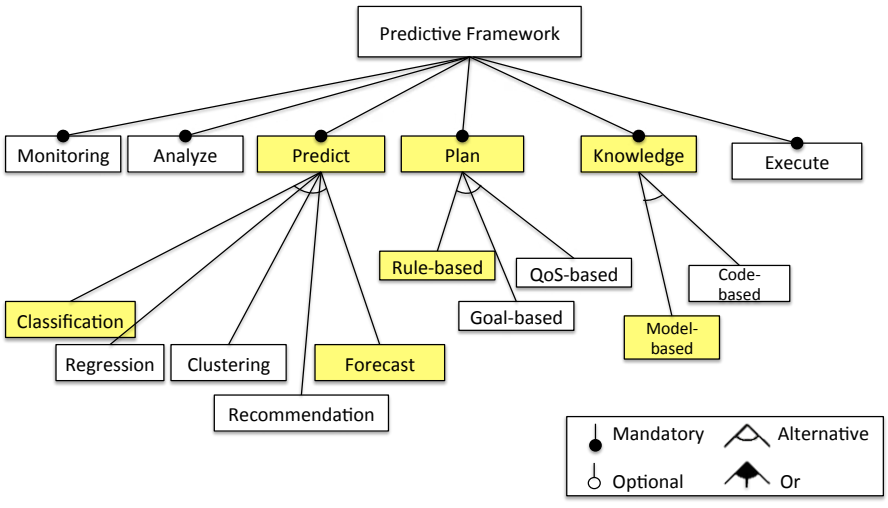


Figure 4.7 – A general implementation of our approach

4.2.1 Predict Module

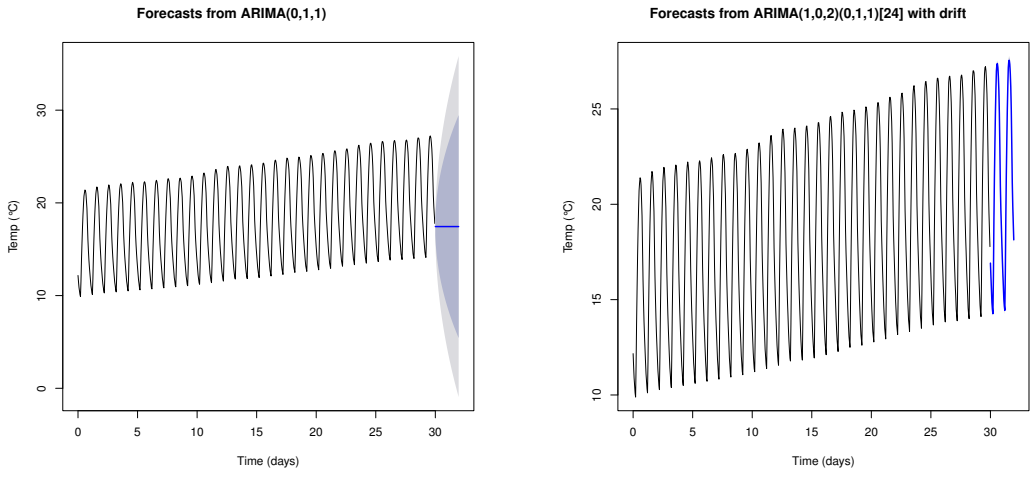
Classification and Forecast

Currently, there are two existing, but not necessarily mutually exclusive approaches to forecast using time series analysis [111]. These two approaches are commonly identified as: *time domain* approach and *frequency domain* approach.

The *time domain* approach is generally motivated by the presumption that correlation between adjacent points in time is best explained in terms of a dependence of the current value on past values. Conversely, the *frequency domain* approach assumes the primary characteristics of interest in time series analyses are related to periodic or systematic sinusoidal variations found naturally in most data [111]. Thus, the best way to analyze a data set is often to use both approaches in a complementary way. For that reason we performed both analysis.

On the one hand, Figure 4.8 illustrates the *time domain* approach analyzing the temperature variable. The models and predictions are generated with the R statistical suite of tools [113], using the Forecast [124] and FitAR [1] packages. These packages offer a suite of methods that permit to automatically generate ARIMA, Holt-Winters and AR_2 models.

For instance, Figure 4.8 (a) presents the forecast for the next 48 hours using a basic *Autoregressive Integrated Moving Average* (ARIMA) univariate model with particular settings (p,d,q). Here p is the number of autoregressive terms, d is the number of nonseasonal differences needed for stationarity, and q is the number of lagged forecast errors in the prediction equation [111]. After applying ARIMA(0,1,1) the forecast result is plotted as a blue line, where the 80% prediction interval is shown as a dark-blue shaded area, and the 95% prediction interval is represented as a light-blue shaded area. However, intuitively we can clearly identify a problem with the model, because the forecast result is apparently a flat line.



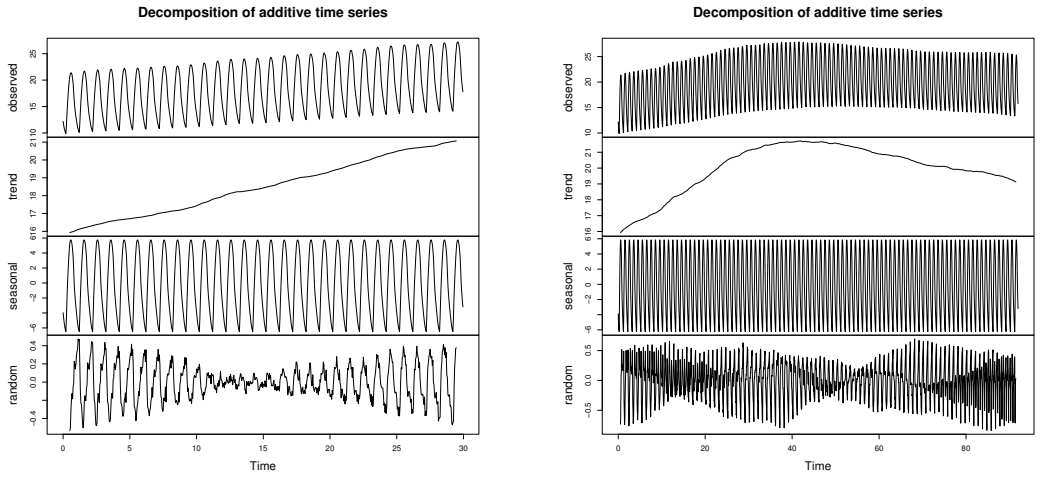
(a) A forecast using ARIMA(0,1,1) (b) A forecast using AUTO.ARIMA function

Figure 4.8 – Forecast analysis of the temperature variable

One way to solve this problem is to redefine the settings of the model. Next, we implement

the same 48 hours forecast using the AUTO.ARIMA funtion, which returns the best ARIMA model according to the *Akaike's Information Criterion* (AIC), *Corrected Akaike's Information Criterion* (AIC_c) and the *Bayesian Information Criterion* (BIC) [72]. Figure 4.8 (b) presents the forecast result plotted as a blue line. This time we can clearly notice that the forecast data follows the sinusoidal variations founded in the observed training data. This straightforward comparison show the importance of not only selecting a good model, but also using the right specifications. More detailed information regarding the ARIMA and AUTO.ARIMA functions can be found in Section A.5.

On the other hand, Figure 4.9 illustrates the *frequency domain* approach for time series analysis. Here, we represent a decomposition of the temperature variable taken from our motivation scenario. For instance, Figure 4.9 (a) depicts one month period of temperature readings from the MERIDIAN NAS ISH station during the month of June 2010. Whereas, Figure 4.9 (b) represents a three months period of temperature readings for the same location during the months of June, July and August of 2010. In both cases, we can appreciate how a time series can be decomposed into its three main components: *season*, *trend* and *remainder*.



(a) One month period

(b) Three months period

Figure 4.9 – Time series decomposition analysis of the temperature variable

The decomposition analysis helped us to visualize the cyclical behavior of the variables. Also, based on the trend component, we elaborated adaptation policies, taking as reference the maximum and minimum values during certain months of the year.

4.2.2 Knowledge Module

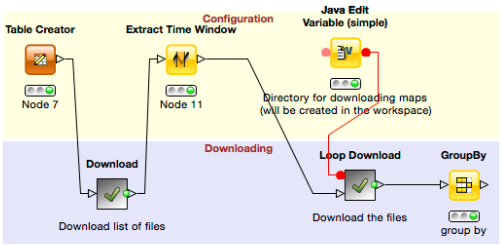
KNIME: A Predictive Analytics Platform

As the underlying platform for the implementation of our approach we used KNIME[2], which is a graphical open source analytic tool that enable us to model the workflow of our approach.

There are two simple reasons for choosing KNIME. Firstly, it helped us move from code-based data mining towards workflow-based analytics, which makes it easier to understand and explain. Secondly, KNIME workflows are reusable because they can be saved in an Eclipse as a workspace and they are easy to maintain. Moreover, KNIME has more than a thousand independently developed plugging (e.g., nodes) , which is an evidence of its great support by the research community working behind this idea. Figure 4.10 illustrates both code-based and workflow-based approaches for data analysis.

```
library(devtools)
# Create data UrlAddress object
UrlAddress <- http://bit.ly/Ss6zDO
Data <- source_GitHubData(url = UrlAddress)
Rfunction["source_GitHubData", Rcode[ <-
function(url, sep = ",", header = TRUE)
{require(httr), request <- GET(url), sthandle <-
textConnection(content(request, as = 'text')),
on.exit(close(handle)) \n
  read.table(handle, sep = sep, header = header)}]
scan(file = "", what = double(), nmax = -1, n = -1,
sep = "", quote = if(identical(sep, "\n")) "" else
"'\"", dec = ".", skip = 0, nlines = 0, na.strings
= "NA", flush = FALSE, fill = FALSE, strip.white =
FALSE, quiet = FALSE, blank.lines.skip = TRUE,
multi.line = TRUE, comment.char = "", allowEscapes
= FALSE, fileEncoding = "", encoding = "unknown",
text) op_for_status(request)
```

(a) Code-based approach



(b) Workflow-based approach

Figure 4.10 – Code-based vs workflow-based data analysis approaches

The KNIME workbench is organized as a workflow. A workflow is composed of nodes, which are the basic processing units. These nodes can be dragged from the *Node Repository*. Each node has a set of input and/or output ports. Data is transferred over a connection from an output port to the input port of another node. Nodes can be configured once inside the workflow and the necessary settings set up with dialog boxes [81].

Once the node is configured, it can be executed and the result of this node will be available in the output port. Ports on the left side of the node are called input ports, while the output port provides data for the following node. Nodes are typed, such that only ports of the same type can be connected. Table 4.1 presents a common set of nodes from the KNIME node Repository. However, they are just an example of the hundreds of nodes available at the KNIME node repository.

[2] http://www.knime.org/
[3] http://www.openstreetmap.org/

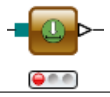
Node	Description
GroupBy 	The most common type of nodes are those that communicate by data port (a white triangle) which transfers flat data tables from node to node (e.g., GroupBy node).
HTTP Connection 	Nodes executing commands connecting by HTTP/FTP protocol can be recognized by their dark cyan square (e.g., HTTP Connection). Parameters like host, port, username and password can be setup in the configuration dialog of the node.
Download 	This node downloads a file or folder from the server specified by the connection information and outputs a table with the references to the local files.
R Learner 	KNIME allows execution of R commands to build a R model in a local R installation. The model is returned in the output port and can be used by the R Predictor node to predict unseen data [81].
Decision Tree Learner 	This node induces a classification decision tree in main memory. Numeric splits are always binary (two outcomes), dividing the domain in two partitions at a given split point. Nominal splits can be either binary or they can have as many outcomes as nominal values [81].
OSM Map View 	This node provides an interactive view on Open Street Maps[3]. Optionally some points of interest (map markers) can be rendered onto it.
MetaNode 	Meta nodes are nodes that contain subworkflows, (i.e. in the workflow they look like a single node), they can contain many nodes and also more meta nodes.

Table 4.1 – Example of nodes from KNIME's Node repository

PMML: Standard Predictive Modeling Language

The Predictive Model Markup Language (PMML)[4] is an XML-based file format developed by the Data Mining Group[5] to provide a way for applications to describe and exchange models produced by data mining and machine learning algorithms. It supports common models such as logistic regression and feedforward neural networks [36].

Since PMML is an XML-based standard, the specification comes in the form of an XML schema. A PMML file can be described by the following components:

- **Header**: contains general information about the PMML document, such as copyright information for the model, its description, and information about the application used to generate the model such as name and version. It also contains an attribute for a timestamp, which can be used to specify the date of model creation.
- **Data Dictionary:** contains definitions for all the possible fields used by the model. It is here that a field is defined as continuous, categorical, or ordinal (attribute optype). Depending on this definition, the appropriate value ranges are then defined, as well as the data type (e.g. string or double).
- **Data Transformations:** transformations allow for the mapping of user data into a more desirable form to be used by the mining model. PMML defines several kinds of simple data transformations [36].
  - Normalization: map values to numbers, the input can be continuous or discrete.
  - Discretization: map continuous values to discrete values.
  - Value mapping: map discrete values to discrete values.
  - Functions (custom and built-in): derive a value by applying a function to one or more parameters.
  - Aggregation: used to summarize or collect groups of values.
- **Model:** contains the definition of the data mining model. E.g., A multi-layered feedforward neural network is represented in PMML by a "NeuralNetwork" element that contains attributes such as:
  - Model Name (attribute modelName)
  - Function Name (attribute functionName)
  - Algorithm Name (attribute algorithmName)
  - Activation Function (attribute activationFunction)
  - Number of Layers (attribute numberOfLayers)

[4] http://www.dmg.org/v4-1/GeneralStructure.html
[5] http://www.dmg.org/

This information is then followed by three kinds of neural layers that specify the architecture of the neural network model being represented in the PMML document. These attributes are NeuralInputs, NeuralLayer, and NeuralOutputs. Besides neural networks, PMML allows for the representation of many other types of models, including support vector machines, association rules, Naive Bayes classifier, clustering models, text models, decision trees, and different regression models [36].

- **Mining Schema:** a list of all fields used in the model. This can be a subset of the fields as defined in the data dictionary. It contains specific information about each field, such as [36]:
  - Name (attribute name): must refer to a field in the data dictionary
  - Usage type (attribute usageType): defines the way a field is to be used in the model. Typical values are: active, predicted, and supplementary. Predicted fields are those whose values are predicted by the model.
  - Outlier Treatment (attribute outliers): defines the outlier treatment to be used. In PMML, outliers can be treated as missing values, as extreme values (based on the definition of high and low values for a particular field), or as is.
  - Missing Value Replacement Policy (attribute missingValueReplacement): if this attribute is specified then a missing value is automatically replaced by the given values.
  - Missing Value Treatment (attribute missingValueTreatment): indicates how the missing value replacement was derived (e.g. as value, mean or median).

- **Targets**: allows for post-processing of the predicted value in the format of scaling if the output of the model is continuous. Targets can also be used for classification tasks. In this case, the attribute priorProbability specifies a default probability for the corresponding target category. It is used if the prediction logic itself did not produce a result. This can happen, e.g., if an input value is missing and there is no other method for treating missing values.

- **Output:** this element can be used to name all the desired output fields expected from the model. These are features of the predicted field and so are typically the predicted value itself, the probability, cluster affinity (for clustering models), standard error, etc. The latest release of PMML, PMML 4.1, extended Output to allow for generic post-processing of model outputs. In PMML 4.1, all the built-in and custom functions that were originally available only for pre-processing became available for post-processing too [36].

Machine Learning Tools and Libraries

In previous Section 2.2.3, we mentioned the importance of machine learning for finding patterns that appear not only in the data at hand, but in general, so that what is learned will hold true in new situations not yet encountered. Hereafter we discuss the suitability of some popular machine learning libraries and the limitations used in our approach.

The R statistical suite [113] is an open source well established programming language focused on statistical computing and graphics. Even though there is no intuitive graphical user interface, there is an IDE environment called R Studio, which is well suited for developing workbenches. Functions in R are grouped into packages, a number of which are automatically loaded when you start R. However for some specific machine learning algorithms one needs to download additional packages to obtain other useful functions. R is particularly good for initial prototyping and implementation of specific reusable code in R (e.g., time series decomposition analysis). However, for more complex workflow it becomes tedious and hard to maintain text-based projects. On the positive side, there are a large number of existing examples and numerous books with applications in R [87, 111].

In the same vein as R, we can find GNU Octave[6], which is an interpreted language, primarily intended for numerical computations. Octave is normally used through its interactive command line interface. It is mainly used for academic purposes. Octave has been mainly built with MATLAB compatibility, however some syntax and keywords may be different.

Weka[7] is a collection of machine learning algorithms for data mining tasks, supported by the University of Waikato, New Zealand. Weka is a Java based library with a graphical user interface that allows you to run experiments on small datasets. This is a good introductory tool to test the classification, clustering, and regression algorithms and to get an idea of what is possible with machine learning. However, the API is poorly designed, the algorithms are not optimized for production use and the documentation is often lacking.

Another useful open source library is jpmml[8], which is a Java API for the *Predictive Model Markup Language* (PMML). The class model consists of two types of classes. There is a small number of manually crafted classes that are used for structuring the class hierarchy. They are permanently stored in the Java sources directory /pmml-model/src/main/java. Additionally, there is a much greater number of automatically generated classes that represent actual PMML elements. They can be found in the generated Java sources directory /pmml-model/target/generated-sources/xjc after a successful build operation.

In conclusion, the main criterion used for the selection of the supporting technologies was based on the premise of the open source availability. In particular, a graphical open source predictive analysis tool, such as KNIME, was chosen over a code-based analytical tools (e.g., R, Python), because of its graphical component, the ability to generate workflows in an Eclipse likewise environment (i.e., plugging management) and its relatively easy learning curve.

4.3 Summary of the Approach

In this chapter we have presented an overview of our approach to guide proactive self-adaptation based on predictive analysis. In Figure 4.11 we characterized our approach using the taxonomy proposed by Salehie et Tahvildari [109].

This framework spans over design time and runtime. Therefore, we have described the separation of the problem of proactive adaptation into two parts: (1) what is needed in terms of

[6] https://www.gnu.org/software/octave/
[7] http://www.cs.waikato.ac.nz/ml/weka/
[8] https://github.com/jpmml/

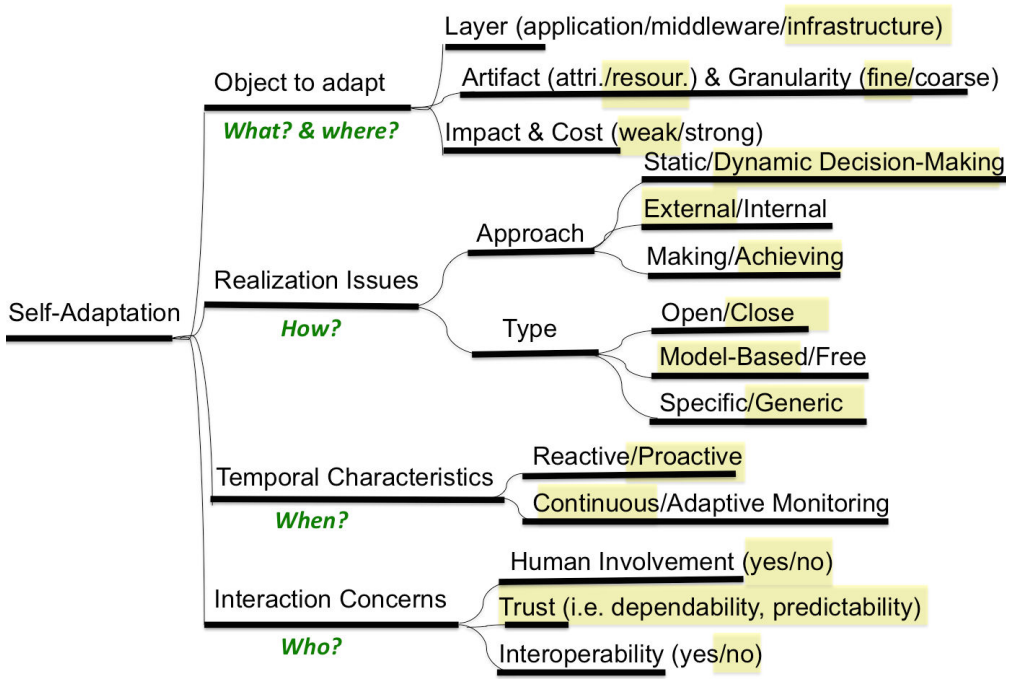


Figure 4.11 – Characterization of our approach using the self-adaptive systems taxonomy

the predictive modeling process at design time was described in Section 4.1.1, and (2) in Section 4.1.2, it was presented how to integrate the predictions from the environment to enable proactive adaptation. We have also described the supporting technologies that enable our approach in Section 4.2. By doing so, we have set the stage for the next chapter, which describes the implementation of the motivation scenario presented earlier in the thesis in Section 2.1.

Chapter 5

Implementation and Evaluation

In this chapter we explain the implementation and evaluation process of our proactive approach based on predictions.

5.1 Requirements of the Forest Monitoring Scenario

As mentioned in the motivation scenario presented in Section 2.1, we consider a wireless sensor network deployed at a location prone to wild fires outbreaks. There is a total of n node sensors in the field and they are assumed to be uniformly distributed within the field. In short, the main requirements of the environment monitoring scenario are:

1. R1- The system should provide feedback on potential fire risks to its users (e.g., environmental guards, fire department) allowing them to act proactively and anticipate possible critical situations.

2. R2- The system should support the coordinated reconfiguration (e.g., increasing/decreasing transmission frequencies) of sensor nodes aiming at avoiding service failure due to battery exhaustion, in order to extend the life time of the system as much as possible.

We tackled these requirements in the following way. During the design phase we collected data regarding the environmental conditions before a fire outbreak and developped predictive models to anticipate potentially critical scenarios, in order to satisfy requirement R1. Figure 5.1 illustrates the use case diagram of R1.

To tackle the second requirements (R2), we built a power consumption model based on a set of given characteristics of a wireless sensor network. Next, we calculated the impact of a reactive strategy vs. proactive strategy on the life time of the system. Figure 5.2 illustrates the use case diagram of R2.

5.2 Design Phase of the Approach

In this section we explain in detail the activities of predictive analysis and adaptation in the design phase. We elaborate on the description of the activities in the context of the motivation

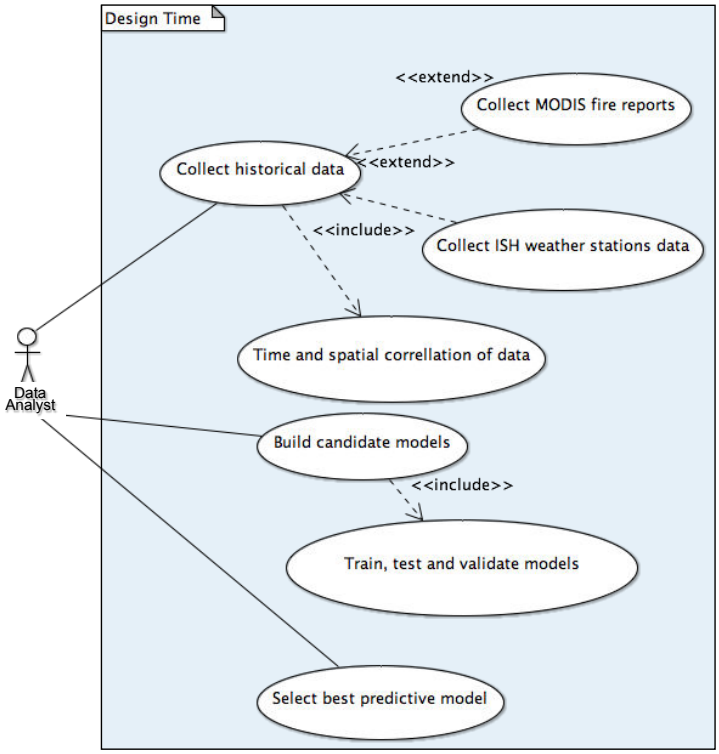


Figure 5.1 – Use case diagram of functional requirement R1

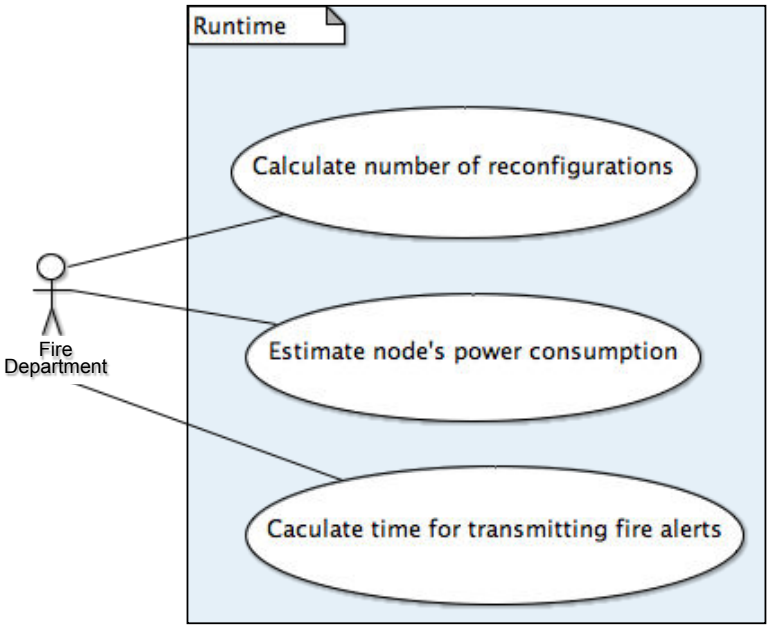


Figure 5.2 – Use case diagram of non-functional requirement R2

scenario of environmental monitoring. The source code of the KNIME workflows implemented for our motivation scenario can be found in the github repository.[1]

5.2.1 Data Collection

The purposes of the data collection process was to gather information on the variables of interest. This data collection was performed in a systematic way that can be reproduced using the specific settings (e.g., location, time period). This enables us to perform data analysis and build research questions, test hypotheses, and evaluate outcomes.

We initiated the implementation of the environmental monitoring scenario with the precondition of finding enough historical environmental data for a given location for a selected period of time. Initially, we considered United States as a broad location (this was later redefined) and we selected the year of 2010 as the time frame period. So, in order to obtain real data about external context information, we investigated several existing sources of fire detection data to feed our environmental monitoring system. Figure 5.3 illustrates the implementation of the part of the workflow that downloads the fire reports. We achieved the goal of the data collection process and collected quality evidence through the following points:

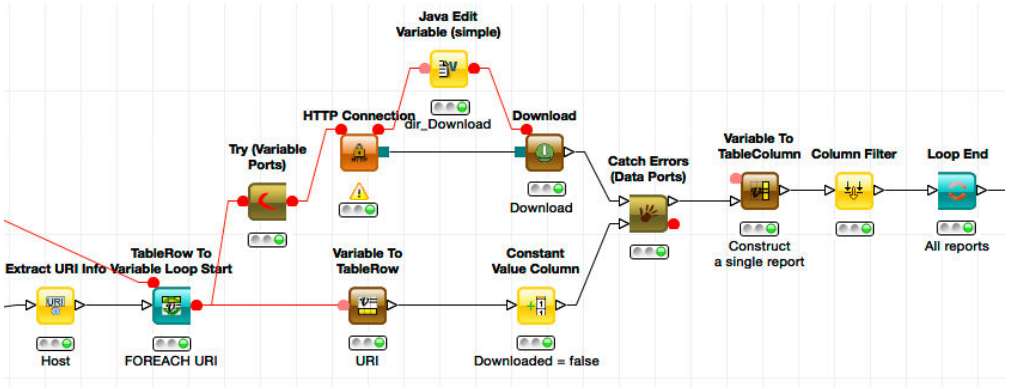


Figure 5.3 – Workflow for downloading fire reports from the USDA Forest Service [49]

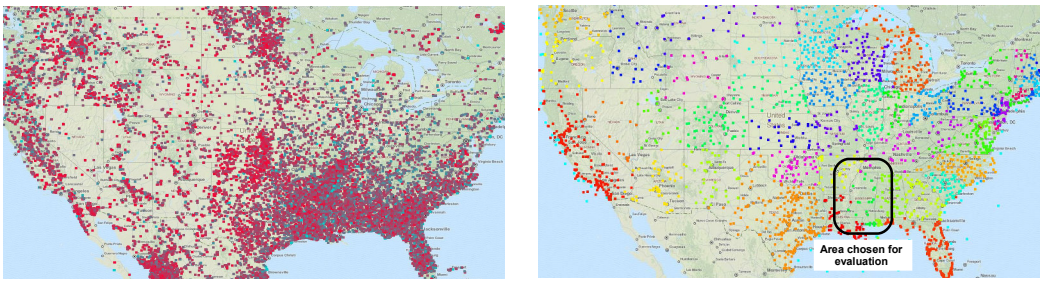
1. We collected a dataset of the wild fires reports with its precise location (e.g., latitude, longitude) for the selected period of time, i.e. year 2010. This workflow starts by setting the URI string that indicates the host server where the data was stored. We downloaded these datasets from the USDA Forest Service website[2]. Next, we created a loop to iterate according to the dates and downloaded all the files from 2010 using a wildcard (e.g., conus_2010mmdd.kmz, see Figure 5.4 a).

   The fire detections were obtained using the Moderate Resolution Imaging Spectro-radiometer (MODIS) and processed as a cooperative effort between the USDA Forest Service Remote Sensing Applications Center, NASA-Goddard Space Flight Center and the University of Maryland. Section A.2 presents a more detailed description of this data dataset.

[1] https://github.com/IvanPaez/proactive-example.git
[2] http://activefiremaps.fs.fed.us/googleearth.php

2. In order to get data of the weather reports we investigated the *Integrated Surface Hourly (ISH)* datasets[3] provided by National Climatic Data Center (NCDC). This *Integrated Surface Hourly (ISH)* weather stations include their specific location (e.g., latitude, longitude) for the selected period of time (i.e., year 2010), see Figure 5.4 b.

   Next, we collected hourly weather readings from ISH land-based stations near (i.e., within a 50 km radius) from the spots where there were confirmed fire detections. Land-based, or surface, observations include data such as: temperature, dew point, relative humidity, precipitation, wind speed and direction, visibility, atmospheric pressure, and types of weather occurrences such as hail, fog, and thunder collected for locations on every continent [49].



(a) MODIS fire detections in 2010 in USA (b) ISH weather stations active in 2010 in USA

Figure 5.4 – Data extracted from by the National Climatic Data Center (NCDC)

5.2.2 Data Processing

Preprocessing the data involves cleaning, tuning and pruning the data. The measured data is organized representing different features of the systems. Some steps to transform the raw data into the processed data include:

- **Grouping** The selected data can be grouped together. These groups determine the number of variables that are returned in the end. Grouping can be optional.
- **Down-sampling** It is common for the large dataset retrieved by a query to have been sampled at a much higher rate than is desired for display, for instance if there is need to display a full year of data that was sampled every second. Display limitations mean that it is impossible to see anything more than about 1–10,000 data points. This makes plotting much faster as well.
- **Aggregation** Data for particular time windows are aggregated using any of a number of pre-specified functions such as average, sum, or minimum.
- **Interpolation** The time scale of the final results regularized at the end by interpolating as desired to particular standard intervals. This also ensures that all the data returned have samples at all of the same points.

[3] http://www.ncdc.noaa.gov/oa/climate/surfaceinventories.html

- **Rate conversion** The last step is the optional conversion to input/output rates.

In the context of the environmental monitoring scenario, we combined both data sources previously described. In this way, we connected individual fires to weather stations in geographical space and time. In other words, the outcome of this correlation is the historical information of the weather conditions near the spots where a wild fire outbreak occurred.

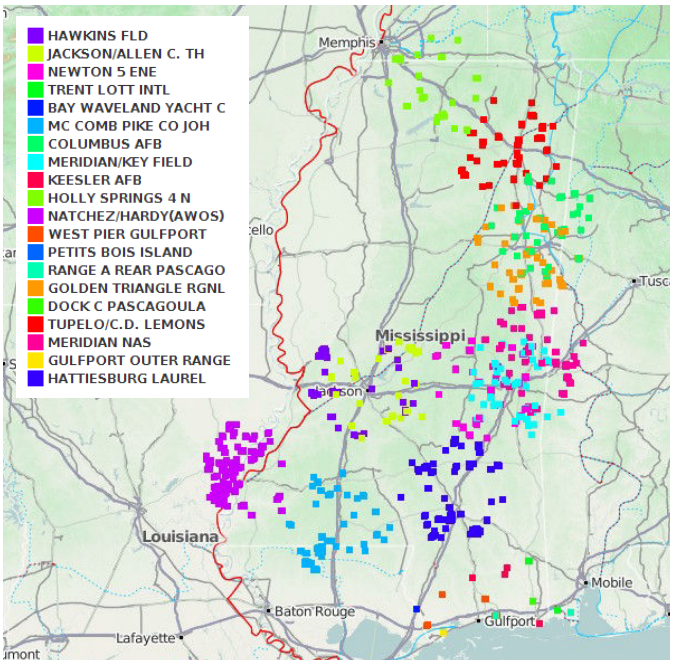


Figure 5.5 – Fires detected in Mississippi by MODIS during 2010

As mentioned earlier, Figure 5.4a (a) illustrates all fire detections in year 2010 for the geographic area covering the continental USA including a 50 km buffer around the periphery. The fires detected by MODIS are not uniformly distributed throughout the USA. Therefore, we chose to reduce the sampling and focus on a specific geographical area that contains a large amount of fires, namely the state of Mississippi, USA (see Figure 5.4a (b)).

Figure 5.5 presents the fires detected in Mississippi by MODIS during year 2010. Colors represent the nearest ISH station that would detect the fire. In order to generate the previous figure we used the Open Street Maps map view plugin from the KNIME nodes repository. The data points were collected in KMZ file format, which is the compressed format of Keyhole Markup Language (KML) and its used for placemark file by Google Earth.

Figure 5.6 illustrates the number of fires within a 50 km radius from an ISH station. There were alltogether 12330 fire detections during year 2010 around the preselected ISH stations. In Section A.3, Table A.2 contains a detailed list of the ISH weather stations from Mississippi, USA used in this study.

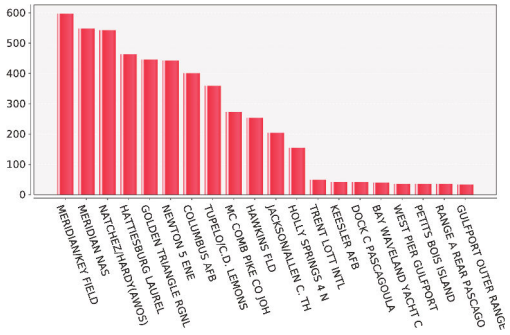


Figure 5.6 – Fires within 50 km radius of an ISH station [98]

5.2.3 Building Predictive Models

In the forest monitoring use case we use prediction models to predict the risk of fire in the near future. The prediction computes a series of data points representing the "future view". In our use case, given a time series of hourly temperature readings, we predicted the fire potential value for the next N hours. For instance, predicting the temperature level in $t+1, t+2, \ldots, t+N$ hours. We selected a classification problem with 13 input attributes: the current and the last 9 hours of temperature readings, and average temperature from the past day, week and month.

In order to develop appropriate evidence to support that we got the right model with the right settings, we selected a broad set of classifiers. These classifiers produce a prediction for the level of temperature using the following rule:

- High (e.g., $t \geq 25$ ˚C),
- Medium (e.g., 15 ˚C $\geq t < 25$ ˚C) and
- Low (e.g., $t < 15$ ˚C).

This set of classifiers represents a large panel of existing training-based techniques for automated classification of items into categories. Table 5.1 presents height classification models selected for modeling our problem. In order to compare the aforementioned classification models, we needed training and testing data, as well as a scoring method.

5.2.4 Training and Cross-Validation of Predictive Models

In this step we separate data into the desire training, testing and validation sets. Next, we evaluated the effectiveness of the previously selected classification models using well established metrics (e.g., precision, recall, F1-measure).

Training and testing data

In general, a supervised machine learning approach requires a representative amount of data usually divided into (i) training dataset, (ii) development dataset and (iii) testing dataset. To achieve good performance, the classifier should be trained on a training set that sufficiently

Classifier	Training Settings
Multi-Layer Perceptron$^{\pi}$	(0..1) normalization, 3 layers, max. 30 neurons/layer, 300 iterations
Fuzzy Rules	linear sampling (3000 samples), min/max fuzzy norm
Probabilistic Neural Net	Z-Score norm., Theta -0.1/+0.9
Logistic Regression$^{\pi}$	–
SVM$^{\pi}$	Polynomial kernel, power=0.5
Naive Bayes$^{\pi}$	–
Random Forest	100 trees
Functional Trees	30 boosting iterations

$^{\pi}$ available as a PMML-based models

Table 5.1 – Selected prediction models (classifiers)

captures the variability of data. During the development process, when the features are designed, the development test set is used to assess the performance of selected features by computing statistical measures such as F1 explained below. To avoid overfitting, the classifier's performance is evaluated against the unseen testing dataset.

Since there is no specific data partition ratio rule, many practitioners have suggested the following partition: 70% for training and 30% for testing. In this study, we realized 70% for training, 20% for developing, and 10% for testing, and we have found it to be more suitable than other ratios.

Cross-validation

Often the training data is scarce (as in our case), and then "hold out" methods are used for evaluation. The most popular is the k-fold cross validation. The data is divided into k subsets (folds) of equal size. Then, we can perform k measurements where one fold is considered as the test set and the other folds are considered as the training set. It means that in every iteration, one of the folds is unseen to the classifier. Although this approach is slightly biased, because we saw all the samples when designing the features, we expect that in practice the classifiers will be reevaluated on domain-specific training data and a new feature set can be selected to suit the new environment.

The implementation of the training and cross-validation analysis was performed using KNIME. Figure 5.7 illustrates the workflow for training and validating the aforementioned classification models.

For the training dataset we used the MERIDIAN NAS temperature readings and for the testing dataset the temperatures from NACHES/HARDY (AWOS) station. The geographical distance between these two stations is approximately 276 km, which gives us confidence that the results are independent and unbiased.

Another reason for choosing these stations is that there are enough fire detections close to them (for both, approx. 550 fires within the radius of 50 km (see Figure 5.6), which allows us to later simulate and evaluate the actual running systems. However, these classifiers do not care about fire detections, but rather predict the risk of fire potential based on historical temperatures.

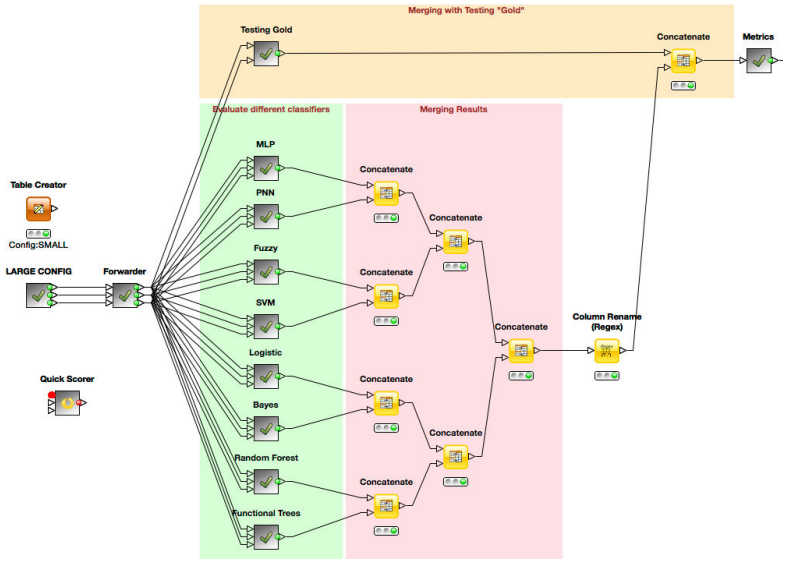


Figure 5.7 – Workflow for training and validating different regression-based models

We run the train-test evaluation loop 43 times for each classifier while changing the number of hours in future for the fire potential prediction (t + 1, t + 2, . . . , t + 24[1d], t + 48[2d], . . . , t + 480[20d])). Each iteration yields several accuracy measures (scores).

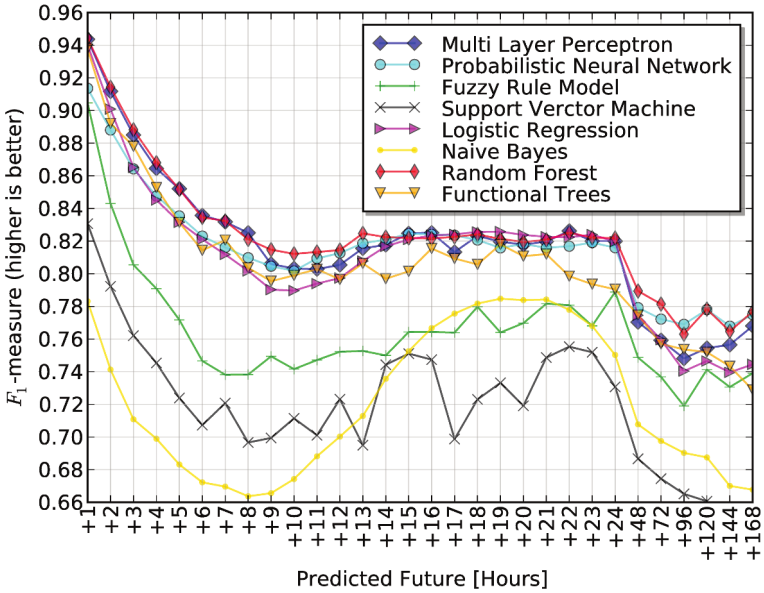
As commonly accepted in the machine-learning community, we used the F1-measure to compare our classifiers (*F1-measure* is a harmonic mean of *Precision* and *Recall*). By the end of the evaluation, each classifier is characterized by a series of F1-measures depicted in Figure 5.8. The outcome of this analysis allowed us to have a ranking of the best predictor models.

We clearly see that most of our classification models achieve a pretty high prediction performance. For instance, predicting 5 hours in the future can be achieved with F1 = 85% accuracy, as depicted in Table 5.8. Moreover, even 20 days in future can be predicted with more than 75% F1. In our experiment, we only used predictions from 1, 2,. . . , 7 hours in the future.

In contrast to the best performing Random Forest (RF) model, Multi Layer Perceptron (MLP) is available as a PMML model. We finally selected the MLP model because of the following reasons. Training MLP requires more time than RF. However it must be done only once, and running predictions using MLP is faster than RF. A trained MLP model consists of 63 neurons organized in 3 layers, with a network of 13 inputs and 3 outputs.

5.3 Runtime Phase of the Approach

Having explained the activities performed at design time in previous section we now focus on the activities at runtime phase. These activities include on-line monitoring, variable selection, model evaluation performed in a reactive and a proactive way, dynamic decision making and deploy reconfigurations. Figure 5.9 illustrates such activities.



Training: *MERIDIAN NAS*, Testing: *NATCHEZ/HARDY*, Distance between locations: *276 km*

Figure 5.8 – Comparing different classification models using data from distant stations.

	MLP	PNN	LR	RF	FT
FirePotential(+1)	94%	91%	94%	94%	94%
FirePotential(+2)	91%	89%	90%	91%	89%
FirePotential(+3)	89%	86%	87%	89%	88%
FirePotential(+4)	86%	85%	85%	87%	85%
FirePotential(+5)	85%	84%	83%	85%	83%

Table 5.2 – F_1-measures of 5 best performing classification models predicting fire potential outcome for 1 to 5 hours in future.

5.3.1 On-line Monitoring

Due to the limitations of real experimentation test bed, we choose to perform simulations of the runtime components of our approach. This offers major advantages regarding cost and feasibility versus replicating a real-world configuration.

To tackle the second requirement R2 presented in Section 4.1.3, we assume the following conditions regarding the organization, communication and develop a basic power consumption model of the system.

Communications in Wireless Sensor Networks

From our working example, a forest is equipped with wireless sensor network, where each node has several sensing capabilities (e.g., temperature, humidity, smoke). Sensor nodes can com-

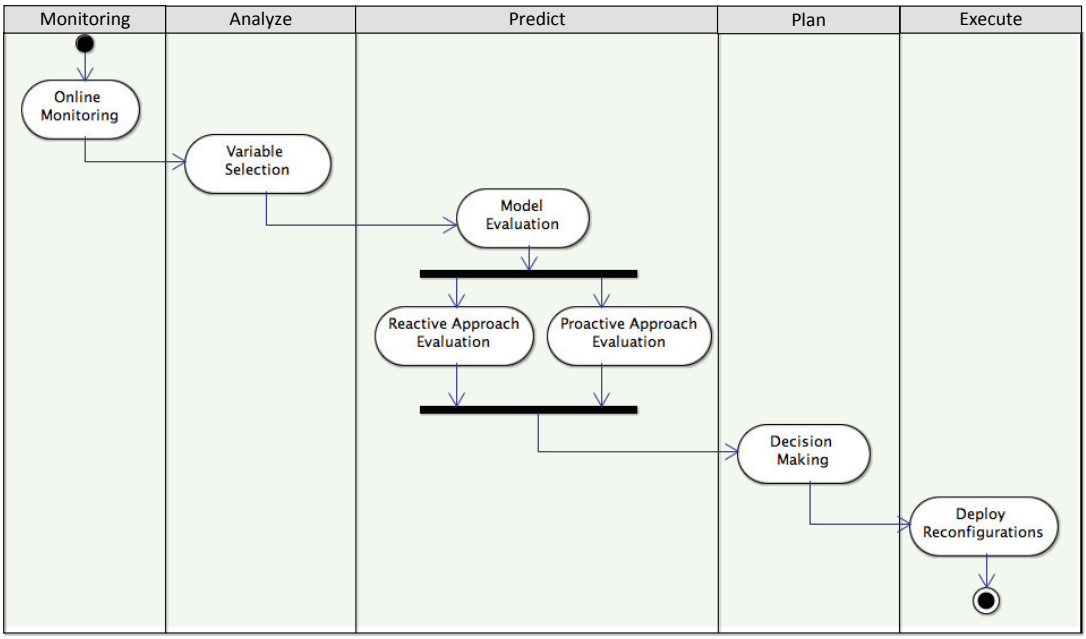


Figure 5.9 – Runtime activities of our proactive approach

municate among themselves (i.e. intra-node communication) and with external data collector devices (i.e., extra node communication) to transmit the raw data. We assume all sensors are aware of the location of the receiver via some type of pre-configuration. However, there are different ways of collecting the sensing data.

The simplest is *direct transmission*, where each sensor directly sends gathered information to the remote receiver independently of each other [67]. This approach does not require any communication between sensors. However, this approach has an inherent scalability problem considering the fact that this is a many-to-one communication where the number of sensors can be potentially huge. In addition, this also puts a limit on how far away from the sensing field the remote collector can be since each sensor will need to be able to reach the collector [38].

A second approach is via *multi-hop routing*, which has been extensively studied for both generic ad hoc routing networks as well as wireless sensor networks, e.g. [67, 86]. Such routing protocols can be designed to realize different goals, e.g. minimize energy consumption. However, these protocols are typically evaluated assuming a random traffic pattern, and it is not clear how they would perform under the scenario where communications are mostly all-to-one or all-to-few (i.e., there can be a small number of collectors). Further investigation on using multi-hop routing within this context is part of our ongoing research [38].

A third approach is *clustering*, where sensors form clusters dynamically with neighboring sensors. One of the sensor in the cluster will be elected cluster head and be responsible for relaying data from each sensor in the cluster to the remote receiver/collector. This approach localizes traffic and can potentially be more scalable. In addition, the cluster heads naturally become points where data fusion and data compression can occur considering the potential correlation

among data from neighboring sensors [118]. In this study we assume a configuration where all sensor nodes are connected to at least one data collector.

Power Consumption Model

The way energy models are developed greatly depends on the platform-specific implementation of the wireless sensor networks. In addition to that, the power consumption rate for sensors varies depending on the protocols used for communications. In this section we derive a power consumption model for the communication subsystem of a wireless sensor network device. For this model, the physical communication rate is constant and assumed to be b bits per second. In addition, we initially assume that the communication bandwidth is low low enough so that interference and transmission collisions can be easily avoided by using simple protocols without significant power consumption penalty.

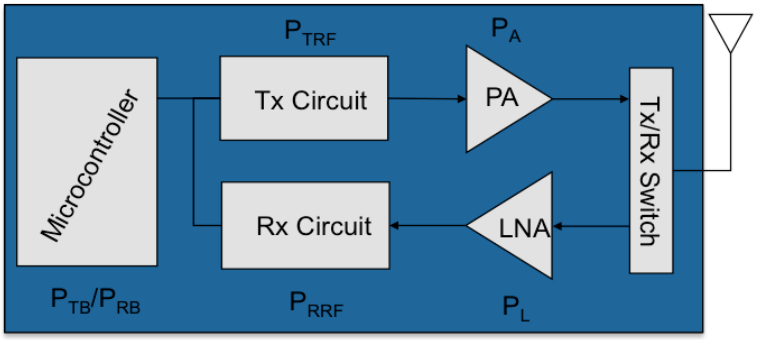


Figure 5.10 – Communication Structure (based on [122])

A wireless sensor node consists of a micro-controller, a radio transceiver, sensors and other peripherals. It is generally assumed that the radio transceiver is the dominant energy consumer in a wireless sensor node. Also a micro-controller is also considered as a major power consumption component in a wireless sensor node. Figure 5.10 illustrates the internal structure of a communication module found in a typical WSN node. Table 5.3 defines the power consumption of each component.

A. *Power Consumption in Receive Mode*

The power consumption of the receiver mode can be modeled by

$$P_R = P_{R0} \tag{5.1}$$

where P_{R0} includes the power consumption of micro-controller, the front-end circuitry of the receiver and the low noise amplifier (LNA).

B. *Power Consumption in Transmit Mode*

The power consumption of the transmitter mode is given by

$$P_T(d) = P_{T0} + P_{PA}(d) \tag{5.2}$$

Variable name	Description
P_{TB}/P_{RB}	Power consumption in micro-controller for transmitting or receiving (mW)
P_{TRF}/P_{RRF}	Power consumption in front-end circuit for transmitting or receiving (mW)
P_A	Power consumption of the power amplifier (PA) for transmitting (mW)
P_L	Power consumption of the the low noise amplifier (LNA) for receiving (mW)
P_{tx}	Radio frequency (RF) output power
η	Power efficiency
P_{RO}	Power consumption of the receiver i.e., $P_{RB} + P_{RRF} + P_L$
P_{TO}	$P_{TB} + P_{TRF}$
P_{PA}	Power consumption of the power amplifier i.e., $\frac{P_{tx}}{\eta}$

Table 5.3 – Variables in energy consumption model

where P_{T0} accounts for the power consumption of the micro-controller (P_{TB}) and the front-end circuit of the transmitter (), and P_{PA} denotes the power consumption of the power amplifier (PA) which is a function of the transmission range, d.

The PA receives a signal at an input power and produces an amplified radio frequency (RF) signal for transmission by the antenna. The PA is driven by a direct current (DC) input voltage, provided for example by a battery in the transmitter, and the efficiency of the power amplifier is given by the ratio of the output power to the DC input power [122]. Thus, the power consumption of the PA can be written as

$$P_{PA} = \frac{P_{tx}}{\eta} \tag{5.3}$$

where η is the power efficiency and P_{tx} is the RF output power. However, the RF power amplifiers are generally designed to provide maximum efficiency at the maximum output service.

To explain this point, we take into consideration the parameters for a well-known commercially available low-power transceiver frequently used in sensor networks such as the CC2420 [117]. Figure 5.11 illustrates the CC2420 power consumption in terms of the transmit power (P_{tx}). It can be seen that if the power amplifier produces an output power that is less than the maximal output power, the efficiency of the power amplifier may be significantly reduced. In this study we assume that the amplifier it working at its maximum output services (e.g., when $R_{tx} = 0.8$, power consumption = 53 mW).

C. *Basic Power Consumption Model*

In order to validate the power model a network model is needed. In our case we assume a single-hop communication between the source node S and a destination, data collector node D.

$$P = P_{RO} + P_{TO} + P_{PA} \tag{5.4}$$

where P_{RO} is the power consumption in receiver mode, P_{TO} is the power consumption in transmit mode, and P_{PA} is assumed as a constant designed to provide maximum efficiency at the maximum output service.

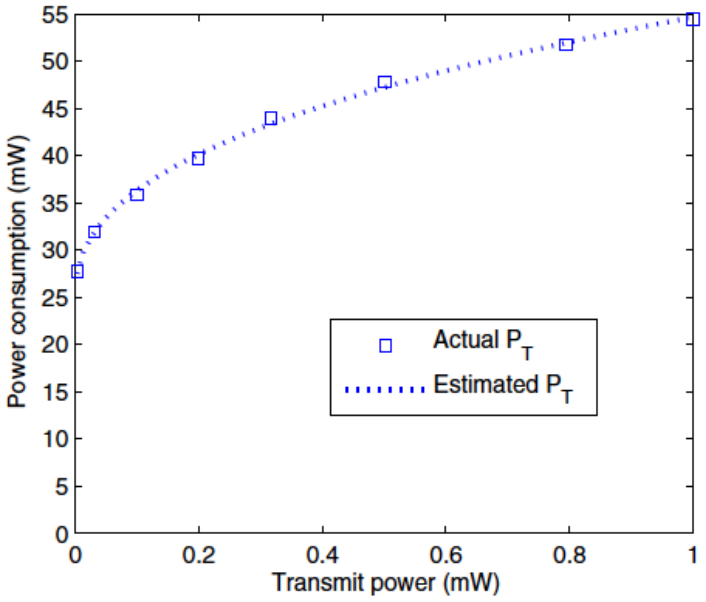


Figure 5.11 – Power consumption in transmit mode (CC2420) [128]

It should be noted that this power consumption model is meant to provide a generic framework for the first order validation of our proactive algorithm. This model should not be considered as platform-specific model. More detailed energy models can be built, including the lower layers of the protocol stack (e.g., radio models, channels, package propagation). It is possible to build such kind of models using sensor networks simulators/emulators available (e.g., OmNet++, MiXim, Castalia). However, such detailed representation is beyond the scope of this study.

5.3.2 Variable Selection

The goal of this module is to analyze the current configuration of the running system. For this reason we assume the following constraints in our fire potential monitoring system:

- Sensing rate is a constant in the system.
- The smallest time unit in the system is 1 hour. Therefore, the time for redeploying a reconfiguration is 1 hour;
- There are three frequency levels of transmission rates:
  1. High (every hour)
  2. Medium (every 8 hours)
  3. Low (every 24 hours)
- A *round* means an iteration, it implies that the system reads its sensors and sends the data to the data collector. Thus, we can estimate the number of data transmissions in a year based on the previous frequencies. This values is denoted by T.

We further elaborate to be able to compare between the different configuration settings of the system. There are four variants of the current condition in the system:

Variants	**Code**	**Description**
Predictive Up	*"U"*	The system runs the prediction only if changing from lower transmission rate to higher transmission rate. This setting targets a better accuracy strategy.
Predictive Down	*"D"*	The system runs the prediction only if changing from higher transmission rate to lower transmission rate. This setting targets an energy saving strategy.
Predictive Standard	*"d"*	The system runs the prediction before any triggered event. This setting combine the previous two strategies.
Interval check	*"i"*	runs prediction for all hours in the interval t+1, t+2, . . . , t+F, so that there is a higher chance to cancel the adaptation.

Table 5.4 – Variants in the behavior of the system

When a positive fire potential is detected by the system, we compute the number of hours that the fire should have been reported to the base station. We call this measure "Late Fire Hours" λ_i.

$$L = \sum_{i=1}^{h} \lambda_i \tag{5.5}$$

where h is the total of hours in the year (i.e., 8760 hours).

Each system variant described in Table 5.4 operates in a certain interval that is either constant or varies over time, depending on the type of a system. If the interval changes, we consider it as

an adaptation. For each system, we compute the number A as the number of adaptations in the year.

Using the power model defined in Equation 5.4, we define the yearly power consumption metrics as

$$P = P_{RO} * A + P_{TO} * T + P_{PA} \tag{5.6}$$

where P_{RO} is the power consumption in receiver mode, A is the number of adaptations over a year, P_{TO} is the power consumption in transmit mode, T is the total number transmission over one year and P_{PA} is a constant designed to provide maximum efficiency at the maximum output service. We also compute relative versions of the metrics A_R, T_R that are relative to the reactive system. As mentioned in Section 2.1 Motivation scenario, we described that the autonomic manager can be considered as a cloud-based dedicated server. For this reason the stochastic models runs in the server and therefore there is no extra computation cost over the nodes modeled in the power consumption equation.

5.3.3 Decision-making and Deploying Reconfigurations

The goal of the planing component is to take charge of the decision-making mechanism. This decision making process implies a trade-off based on the length of the decision-making horizon. The decision-making horizon is given by the units of the prediction scale. In other words, accurate predictions must be available as far ahead into the future as the decision-making horizon (e.g. middle or long term). It is important to identify the best prediction scheme like prediction on-demand, or every certain periods of time (e.g., when the environment has changed).

In our case, we adopted the decision-making paradigm ECA rules explained earlier. Using the temperature prediction, we have implemented the following ECA rule:

Event: An increase or decrease in temperature in all the future 7 hours.

Condition: If the temperature crosses the threshold in ascending or descending manner during the coming 7 hours.

Action: The system will increase or decrease the sensing rate accordingly.

Let us consider the following scenario that lasts 3 time units, where we represent the system state with a (sensing rate, temperature) tuple as follows:

1. (medium transmission rate, medium temperature)
2. (medium transmission rate, high temperature)
3. (high transmission rate, medium temperature)
4. (medium transmission rate, medium temperature)

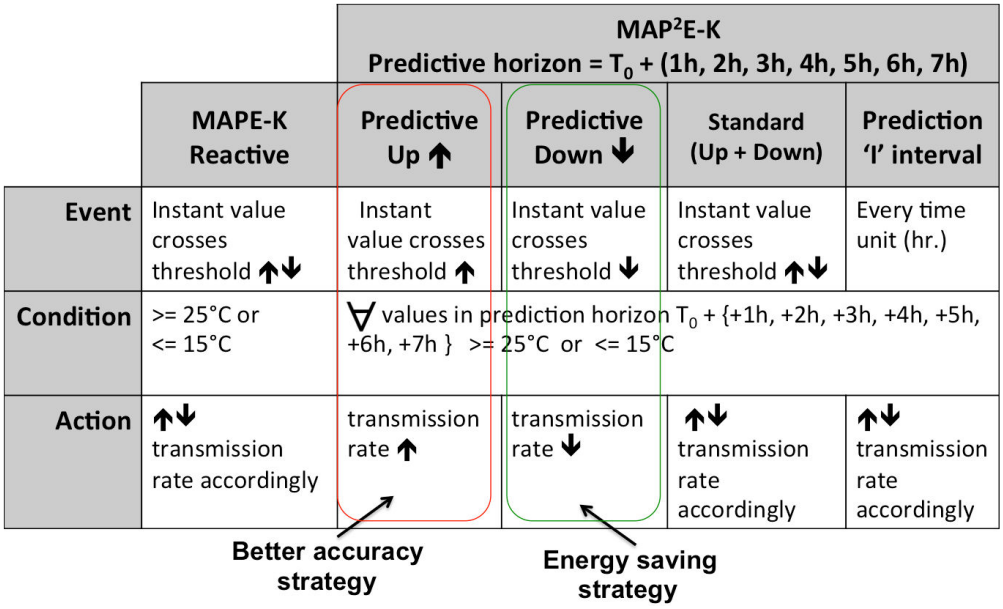


Figure 5.12 – Summary of the different implementations of the business rule

In the case of the reactive approach, the system triggers a reconfiguration at point 2, because the previous temperature reading moves from medium to high. However by the time the reconfiguration has taken place the temperature has dropped back to medium level, making this reconfiguration useless and consuming extra energy.

In the case of the predictive approach, the system does not take decisions on instantaneous values of environmental variables, but it considers the trend and seasonal components of the historical data, plus the forecast of the variables (e.g., 7 hours into the future), thus avoiding this useless reconfiguration. The Figure 5.12 summarizes the different implementations of the business rule previously introduced in Table 5.4.

5.4 Empirical Evaluation

5.4.1 A Brief Introduction to the Goal/Question/Metric (GQM) Paradigm

The Goal/Question/Metric (GQM) paradigm is a mechanism for defining and evaluating a set of operational goals, using measurement. It represents a systematic approach for tailoring and integrating goals with models of the software processes, products, and quality perspectives of interest [11]. In other words, the GQM supports a top-down approach to define the goals behind the model with measures of software processes and products. Assigning these goals to questions helps to decide precisely what to measure (choosing metrics). More precisely, the three main components of the GQM approach are:

Goal: The goal represents the conceptual level and is defined for an object, for a variety of reasons, with respect to various models of quality, from various points of view, relative to

a particular environment.

Question: The questions represent the operational level. A set of questions is used to characterize the way the assessment/achievement of a specific goal is going to be performed based on some characterizing model. Questions try to characterize the object of measurement with respect to a selected quality issue and to determine its quality from the selected viewpoint.

Metric: Represents the quantitative level, which is a set of data associated with every question in order to answer it in a quantitative way. The data can be: (1) objective, if they depend only on the object that is being measured and not on the viewpoint from which they are taken; or (2) subjective, if they depend on both the object that is being measured and the viewpoint from which they are taken.

We performed an experimental study on the fire monitoring scenario to validate our approach. We defined the experimental design of our study using the GQM method described above. Our goal can be defined as following:

Purpose: Improve

Issue: the global effectiveness and reconfigurations

Object: the wireless sensor network

Context: proactive self-adaptive software system

To fulfill this goal, we will focus on answering the three following research questions:

RQ1: Does a proactive adaptation approach trigger less reconfigurations than a reactive approach under seasonal behavior conditions?

RQ2: Does a proactive adaptation approach improve power consumption compared with a reactive adaptation approach?

RQ3: Does a proactive adaptation approach reduce the delay in transmitting fire alerts in comparison with a reactive approach?

5.4.2 Empirical Evaluation of Proactive *vs.* Reactive Strategies

In general, there are three approaches to asses the quality of a solution: modeling, simulation and physical evaluation. However, utilization of real sensor networks limits the scale of experiments and makes the reproduction of results an extremely difficult task that involves real-time feedback from a safety critical system. For that reason, in this study we performed validation by modeling and simulating the system based on real data collected from public organizations such as the USDA Forest Service[4] and the National Climatic Data Center (NCDC)[5].

[4] http://www.fs.fed.us/

[5] http://www.ncdc.noaa.gov/

In our case, the environment is composed entirely of computer models, with which human interaction is reduced to a minimum. This offers major advantages regarding cost and feasibility of replicating a real-world configurations. The simulations were carried out on a MacBookPro with four-core, hyper-threaded i7, SSD, 16GB of DDR3 memory.

To evaluate the previous research questions we consider the following systems:

- *Reactive system.* This is our first baseline system that adapts its transmission rate based on the current temperature.
- *Predictive systems.* This system operates similarly to the reactive system. However, before each adaptation it uses a classifier to predict fire potential at time $t+F$ (t is current time, F is the number of hours in future). The adaptation is canceled whenever

$$FirePotential_{(t-1)} = FirePotential_{(t+F)} \tag{5.7}$$

Figure 5.13 illustrates an extract of the evaluation workflow developed in KNIME and it can be found in the Git repository[6] (i.e., 07 - Simulate Systems).

During this workflow the historical data of fire reports, together with the conditions of the local area prior to the fire outbreak are connected to actual weather conditions in the simulation of the running example. Only fires around Mississippi are taken into account (using a geocoordinate filter) to speed up the filter joining. Next, for each station separately we enrich its hourly table with the following variables:

- Number of fire reports per hour (i.e., fire count) by joining the fires with the stations using geo-location (CFG_FIRE_RADIUS = 50km). In other words, all the fires reported within a radius of 50 km form the weather station.
- We calculate the predictions of fire potential using a *Multilayer Perceptron* (MLP) model. This model predicts 7 hours into the future. By shifting this new time series we get predictions for $+6$, $+5$, $+4$, $+3$, $+2$, $+1$ hours. The decision of using a MLP model has been explained in Section 5.2.4.
- The model is trained on MERIDIAN NAS weather station and simulates predictions for all other stations,
- the simulated predictions are stored in columns "FP+1", "FP+2",... "FP+7".

Figure 5.14 describes the fire potential output table. This table serves as input for the simulation of the Simple, Reactive and Predictive systems. Listing 5.1 shows the Java code for the Predictive system. The Predictive system changes its sending period based on the current fire potential at time "t", however, only if the predicted $FirePotential_{(t+Nhours)}$ differs from the $FirePotential_{(t-1hour)}$. This way, it avoids spikes, which reduces the number of required adaptations.

[6] https://github.com/IvanPaez/proactive-example

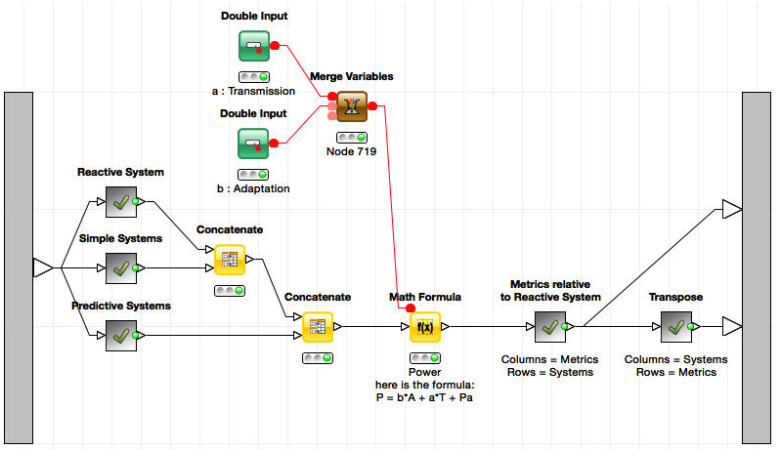


Figure 5.13 – Evaluation of Fire Potential Classification

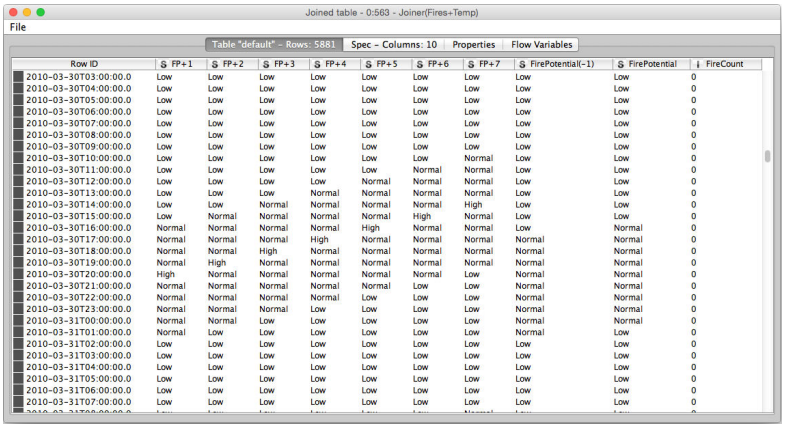


Figure 5.14 – Fire potential output table

```
1  package proact.eval.knime;
2
3  public class PredictiveSystem extends AbstractSystem {
4
5    private int hoursInFuture;
6    private boolean checkWholeInterval = false;
7    private boolean speedupWithoutPrediction = false;
8    private boolean slowdownWithoutPrediction = false;
9
10   public void setHoursInFuture(final int hoursInFuture) {
11     this.hoursInFuture = hoursInFuture;
12   }
13
14   public void setCheckWholeInterval(boolean checkWholeInterval)
       {
15     this.checkWholeInterval = checkWholeInterval;
16   }
17
18   public void setSlowdownWithoutPrediction(boolean
       slowdownWithoutPrediction) {
19     this.slowdownWithoutPrediction = slowdownWithoutPrediction;
20   }
21
22   public void setSpeedupWithoutPrediction(boolean
       speedupWithoutPrediction) {
23     this.speedupWithoutPrediction = speedupWithoutPrediction;
24   }
25
26   @Override
27   protected boolean isPeriodChangeAllowed(final int
       proposedPeriodSize) {
28
29     // speedup
30     if( getPeriodSize() > proposedPeriodSize &&
       speedupWithoutPrediction )
31       return true;
32
33     // slowdown
34     if( getPeriodSize() < proposedPeriodSize &&
       slowdownWithoutPrediction )
35       return true;
36
```

```
37      if(in_lastHourFP == null)
38        return true;
39
40      if(checkWholeInterval) {
41        for(int i = 0; i < hoursInFuture - 1; ++i) {
42          if(pastSameAsFuture(i))
43            return false;
44        }
45      }
46
47      return ! pastSameAsFuture(hoursInFuture - 1);
48    }
49
50    private boolean pastSameAsFuture(final int idx) {
51      return in_lastHourFP.equals( in_predictedFP[ idx ] );
52    }
```

Listing 5.1 – Listing of PredictiveSystem.java

5.4.3 Evaluation in Terms of Number of Reconfigurations

In the first analysis, we evaluated the different variants of the system, base-model, reactive and proactive strategies presented in Table 5.4, comparing them in terms of the number of triggered reconfigurations. Figure 5.15 expands the Predictive systems meta-node presented in Figure 5.14. In this workflow we can clearly identify the default (d), slowing-down (D), speeding-up (U) and interval check (i) variants of the system behavior.

This first analysis addressed the following research question (RQ1): Does a proactive adaptation approach trigger less reconfigurations than a reactive approach under seasonal behavior conditions?

From this result, we can observe that all 28 systems based on proactive adaptations are achieving a smaller number of reconfigurations than a pure reactive system. We are able to save up to 20% of the number of reconfigurations of the system with the Predictive(1) system. These results highlight that using predictive information can help to reduce the number of reconfigurations of a system. Figures 5.16 show the number of reconfigurations for each self-adaptive system.

5.4.4 Evaluation in Terms of System Lifetime

In the second analysis, we evaluated the different variants of the system, base-model, reactive and proactive strategies presented in Table 5.4, comparing them in terms of the system's life time using the power consumption model previously defined in Equation 5.6

This second analysis addressed the second research question (RQ2): Does a proactive adaptation approach improves power consumption compared with a reactive strategy? As evaluation results we use as input the previously generated power consumptions model from Section 5.3.1.

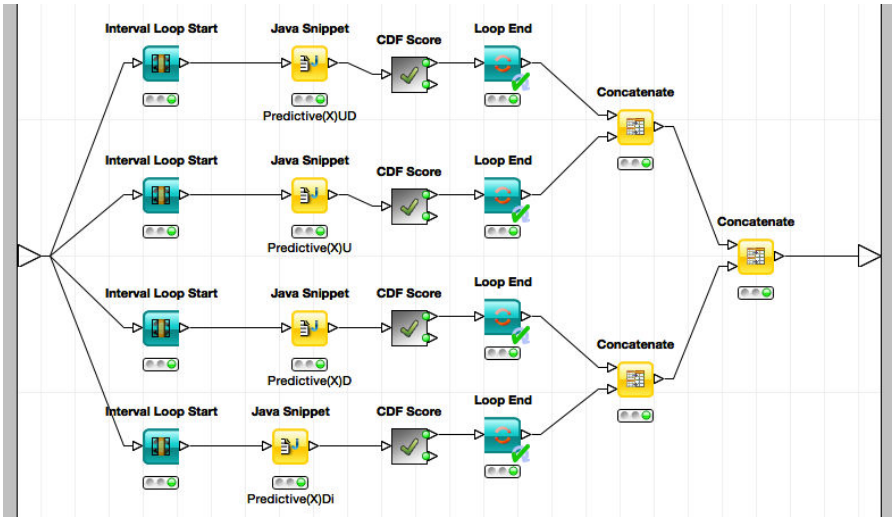


Figure 5.15 – Implementation of the Predictive systems

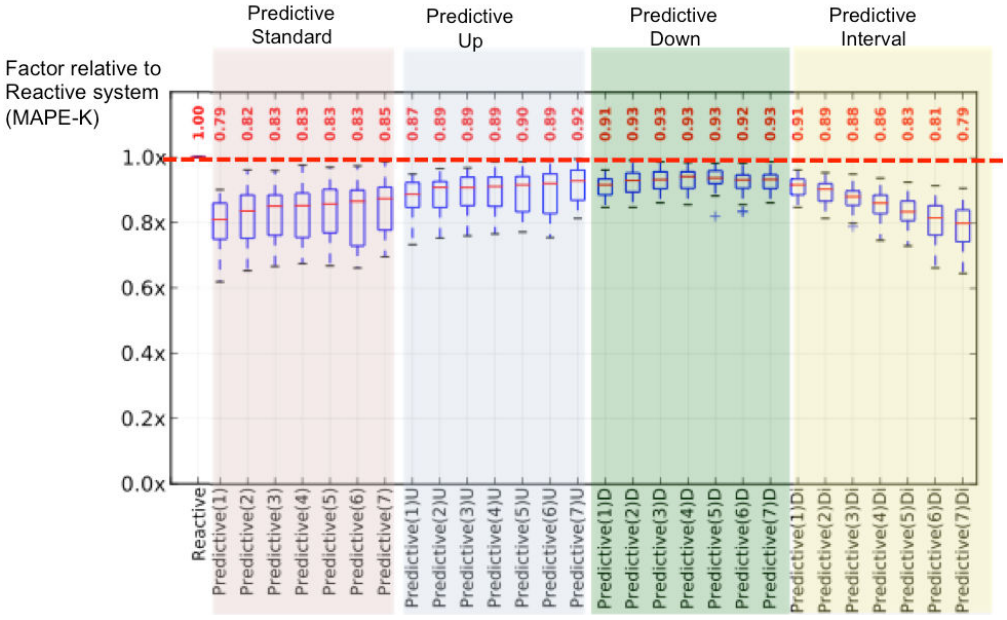


Figure 5.16 – Number of needed reconfigurations relative to the reactive system

Figure 5.17 shows the total power consumption for each type of a systems. We use this metric to quantify the lifetime of the system as the lifetime is directly correlated to the power consumption model.

For this metric, all the Predictive systems reduced the power consumption with respect to Reactive system. The lowest power consumption has been measured in the Predictive(*)D (i.e., slowing-down) and Predictive(*)Di (i.e., interval check) group. In other words, when choosing an adequate adaptation strategy, these results show that using prediction information allows for an increase in system lifetime.

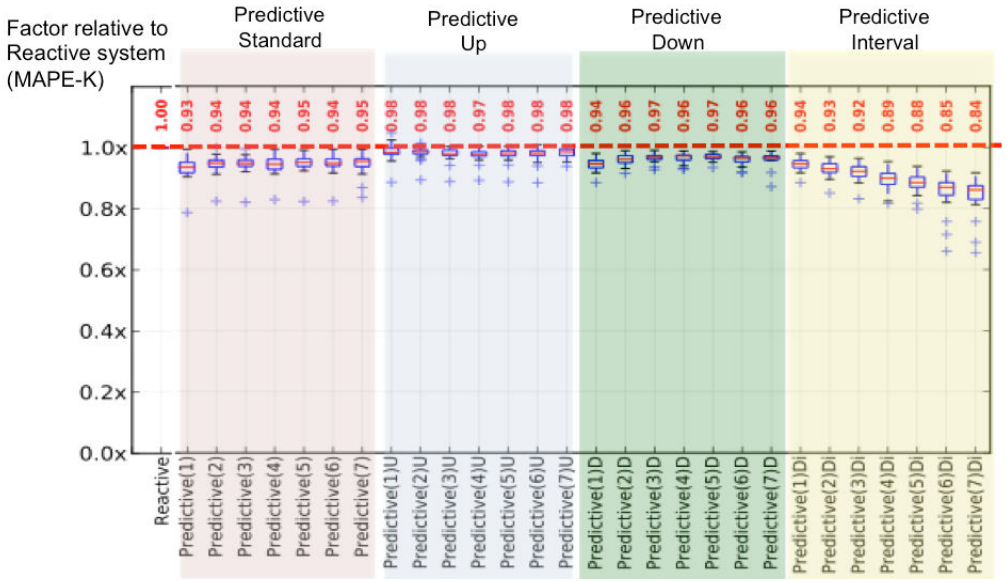


Figure 5.17 – Power consumption relative to the reactive system

5.4.5 Evaluation in Terms of Late Fire Report

In the third analysis we evaluated the different variants of the system, base-model, reactive and proactive strategies presented in Table 5.4, comparing them in terms of the hours taken for the system to reports a potential fire situation which we denominated late fire report.

This analysis deals with the third research question (RQ3): Does a proactive adaptation approach reduce the delay in transmitting fire alert in comparison to a reactive approach?

Figure 5.18 presents the delay introduced in reporting fire alerts for each self-adaptive system. On these results, we can observe that the Predictive(*)Di (i.e., interval check) techniques are exhibiting larger delays than the other approaches. This is due to the fact that this proactive adaptation strategy tends to slow down the adaptation process.

From these results, we can observe that most of the predictive techniques show fire detection delays similar to the reactive approach technique. This result is valuable since proactive techniques such as Predictive(3)U (i.e., speeding-up) were able to decrease the number of system reconfigurations, while providing a smaller delay for detecting fire.

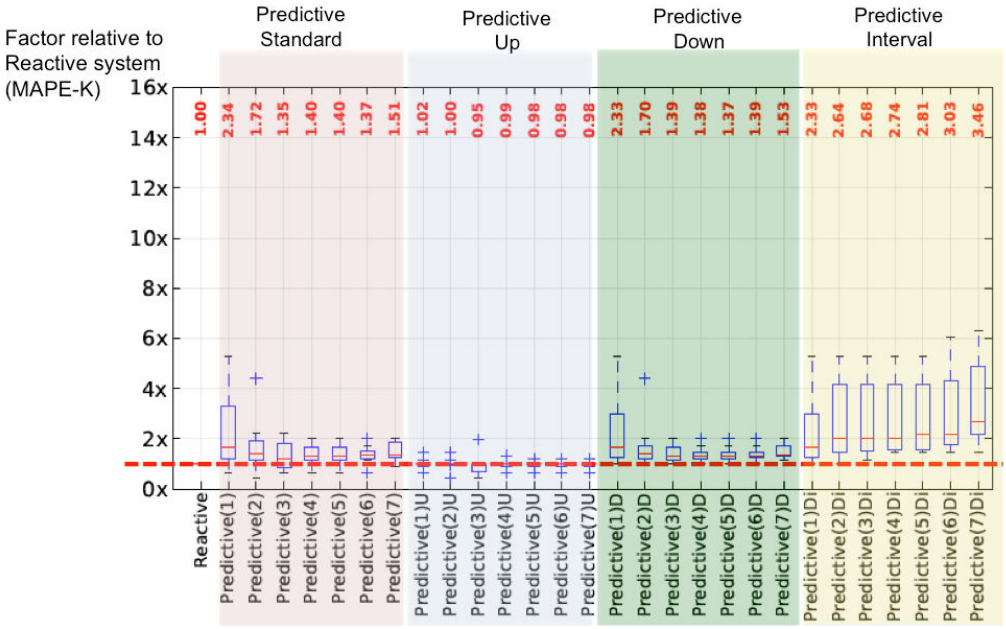


Figure 5.18 – Number of hours when a fire detected by the node was waiting for transmission to the data collector. The number is relative to the reactive system

5.5 Discussion

This chapter covers the third objective of this thesis by explaining the implementation details of our proactive adaptation approach. In our work we have demonstrated and explained the design decisions we took, other considered alternatives were described and we justified our decisions.

Firstly, we analyze the number of reconfigurations based on an ECA rule that decides whether to trigger or not a new reconfiguration according to the predictive horizon that it is available. In a reactive adaptation strategy, the predictive horizon is one unit of time, which represents the current state. For the proactive strategy, we design, train, test and validate different predictive models and we selected the best scoring according to well established measures of model goodness. The Figure 5.19 presents a summary of the encountered results organized according the request question: number of triggered adaptations, energy power consumption and delay in transmitting a fire alert. These characteristics are compared versus the different adaptation strategies, namely: reactive, predictive standard, predictive up, predictive down and predictive interval.

Secondly,we evaluate the impact of implementing a reactive strategy vs. a proactive one in the context of the motivation scenario. We used historical weather conditions and fire reports based on realistic data extracted from publicly recognized organizations. These data was correlated in time and geographical location, which allowed us to model realistic scenarios.

Thirdly, we demonstrated through a series of experiments the different advantages and limitations of the reactive and proactive strategies. The validation of our approach on the fire monitoring case study have shown interesting results that confirm the idea that proactive adaptation

based on predictive analysis is a promising research direction.

It is important to highlight that there is no single strategy that have a better performance regarding all the characteristics. For example, the strategy Predictive standard performed better regarding the number of reconfigurations. In second place, the Prediction interval strategy performed better regarding the energy consumption characteristic. Last, the reactive strategy together with the predictive up, both presented the minimum time delay to report a critical condition in the systems.

		MAP²E-K Predictive horizon = T_0 + (1h, 2h, 3h, 4h, 5h, 6h, 7h)			
	MAPE-K Reactive	**Predictive Up ↑**	**Predictive Down ↓**	**Standard (Up + Down)**	**Prediction 'I' interval**
# of reconfigurations	Maximum	11.2% less than reactive	7.5% less than reactive	17.5% less than reactive ✓	14.8% less than reactive
Energy Consumption	Maximum	2.2% less than reactive	4% less than reactive	5.8% less than reactive	10.7% less than reactive ✓
Delay in transmitting alert	Minimum 59 min. ✓	59 min. ✓	1h 35min.	1h 35min.	2h 21min.

Figure 5.19 – Summary of the different results for the different adaptation strategies

5.5.1 Limitations and Threats to Validity

Indeed, the benefits of predictive analysis combined with proactive adaptation techniques outweighs its overhead when compared with a purely reactive approach. However, there exist some limitations and our approach can be improved in several ways. Drawing general conclusions from empirical studies in software engineering is difficult because any process depends on a potentially large number of relevant context variables that are particular for each domain and there remain threats to validity and generalization [11, 12].

- *Internal validity* relates to the extent to which the design and analysis of the study might have been compromised by the existence of confounding variables and other unexpected sources of bias [80]. Some factors that could affect the characteristics of our design are: selection bias in the variable of study, confounding variables (e.g., an extra factor that affects the relationship between the independent variable and dependent variable), the measurement process changed or the testing process changed.

  In our case study, as a matter of fact, we considered historical data from fire reports and weather stations sensor readings. This data was then correlated in space and time to generate the positive occurrences of fire. A possible limitation is that we focused on predicting

temperature readings. Other variables such as type of soil and vegetation were not considered into this analysis.

- *External validity* relates to the extent to which the hypotheses capture the objectives of the research and the extent to which any conclusions can be generalized [80].

  Our proposed approach is not restricted to the case study we have chosen to evaluate and can be applied to a wide range of domain areas. Preconditions for applying our approach are directly related to the amount of historical data (e.g., time series) available at the design phase, because this will hinder the predictive modeling process. We argue that other application domains are those who have well defined input parameters that can be represented on as mathematical functions (e.g., bandwidth demand forecast).

Chapter 6

Conclusions and Perspectives

6.1 Conclusions

This thesis presented a proactive self-adaptation approach based on predictive analysis for self-adaptive pervasive systems [98]. Following an stepwise process, this approach leverages the advantages of predictive analysis both at design-time and at runtime to support the dynamic decision-making of self-adaptive pervasive systems. We evaluated our approach in a realistic scenario for environmental monitoring.

More in particular, we used predictive analysis machine learning techniques into the MAPE-K autonomic control loop to enable proactive adaptation in self-adaptive pervasive systems. *Predictive analysis* (PA) leverages on extracting information from existing data sets in order to determine patterns, predict future outcomes and trends. It is important to remark that PA does not tell what will happen in the future. It forecasts what might happens in the future with an acceptable level of reliability. This allows engineers and designers to include *what-if* scenarios and do risk assessment for such scenarios [112].

Regarding the integration of predictive analysis in software engineering, in the past there has been great interest in modeling and predicting software costs as part of project planning [19], or to predict the number of faults to be encountered in a new software version [54]. However, these activities were performed mostly at the early stages of the software life-cycle. In this thesis we developed a proactive approach to dynamic adaptation at runtime.

The benefits of combining predictive analysis with self-adaptive approach can be summarized as follows:

- avoiding unnecessary adaptation and oscillatory behavior,
- managing allocation of exhaustible resources, and
- being proactive to take seasonal behavior into account.

Let us continue by positioning our contributions regarding the objectives presented in the Introduction Chapter (Section 1.3).

O1: Explore proactive adaptation approaches that leverage on existing context and environment streams of data in order to anticipate erroneous behavior of the system by predicting conflicting situations.

In Chapter 3, Section 3.2 we explored the state-of-the-art related to different proactive adaptation approaches. We took a broad range of approaches coming from multidisciplinary areas such as software engineering, artificial intelligence and control theory/engineering. Within these areas we had a closer look at architecture-based self-adaptation, extended model-based self-adaptation, reinforcement learning adaptation, adaptive control and reconfiguration control techniques.

O2: Study the activeness (e.g., reactive *vs.* proactive) of self-adaptive systems, analyze the potentials and limitations of system's parameters predictability, and investigate solutions to deal with parameters' future value uncertainty.

In Chapter 4, we presented a comparison analysis of both adaptation strategies (i.e., reactive, proactive), including their benefits and challenges in three different scenarios.

O3: Develop a methodological framework of predictive modeling using machine learning techniques to enhance the effectiveness of proactive adaptation in self-adaptive software systems.

In Chapter 4, Section 4.1 we developed a method for integrating predictive analysis into the autonomic control loop of self-adaptive systems. We defined a stepwise predictive modeling process, which can be implemented at design time. We also pointed out the integration points with the running system at runtime. In Chapter 5, Section 5.4 we developed an empirical evaluation of our approach and identified the most effective predictive strategies regarding different concerns.

As we mentioned in the previous Evaluation Chapter Section 5.5. It is important to highlight that there is no single predictive strategy that has a better performance for all the evaluated characteristics. In other words there is no silver bullet. For example, the strategy Predictive standard performed better regarding the number of reconfigurations. In second place, the Prediction interval strategy performed better regarding the energy consumption characteristic. Last, the reactive strategy together with the predictive up, both presented the minimum time delay to report a critical condition in the systems.

Therefore, the concept of trade off is key to evaluate the impact of implementing a proactive strategy complementary to a reactive one. However, in the context of environmental monitoring, like the one of our motivation scenario, proactiveness it is a requirement due to the fact of the penalty cost of not anticipating a significant fire condition.

The approach presented in this thesis is effectively supported by predictive workflow developed in data analysis tools (i.e. KNIME) in the context of the ITN RELATE Project. These workflows are open to the public in a GIT repository [1]. The use of standard predictive models, such as PMML, and preliminary feedback from participants of the RELATE project indicate that our approach can be used in an industrial context.

[1] `https://github.com/IvanPaez/proactive-example`

6.2 Lessons Learned

In this subsection we critically reflect on the entire thesis. We also discuss the design decisions we made while developing our approach. We describe alternatives that we considered, compare our decisions with the alternatives, and justify our decisions. We describe the lessons learned through out the developing of our approach.

Model of preference. In our work we use Multi-layer perceptron (see Section 5.2.4) with the following settings: (0..1) normalization, 3 layers, maximum 30 neurons/layer, and up to 300 iterations. This model was used for classification of the significant fire potential conditions.

One alternative is the approach proposed by Herbst et al. [68]. The authors propose an approach that selects suitable forecasting methods for a given context, based on a decision tree. As input, the user needs to provide his general forecasting objectives. For building their proactive mechanism the authors rely on a sophisticated spectrum of forecasting methods based on time series analysis, and demonstrated that the ensembles of these methods achieved higher accuracy than individual forecasting methods. Even though we share the notion of prediction models, in our work we consider applying not only forecasting techniques for times series analysis, but also classification, regression and clustering to a wider extent of data types.

Workflow versus code-based developing environment. In our work we used a workflow-based developing environment (see Section 4.2.2). As the underlying platform for the implementation of our approach we used KNIME[2], which is a graphical open source analytic tool that enable us to model the workflow of our approach. There are two reasons for choosing KNIME. Firstly, it helped us move from code-based data mining towards workflow-based analytics, which makes it easier to understand and explain predictive process to non data scientist people. Secondly, KNIME workflows are reusable because the workflows can be saved in an Eclipse like workspace and are easy to maintain. Moreover, KNIME has more than a thousand independently developed plugging (e.g., nodes) , which is an evidence of its great support by the research community working behind this idea.

Comparing proactive versus reactive strategies. In our work we evaluated a basic reactive strategy versus a set of four different predictive strategies designed for different objectives (see Section 5.3.3). However, before we compare them, let us discuss the differences of the two configuration strategies. The strategies differ in two respects: (1) the input information that is used to make reconfiguration decisions and (2) the length of the decision-making horizon. The proactive strategies requires predictive inputs and must be available at least as far ahead into the future as the decision making-horizon. Conversely, prediction beyond the decision-making horizon are not needed, because they are irrelevant for the configuration decisions. So both strategies can be distinguished from each other using the parameter of the length of the decision-making horizon.

In our work, we measure the length of the decision-making horizon in units of prediction scale (e.g. 1 hour). The reactive strategy has a decision horizon of one unit of time, while the predictive strategies have an horizon of multiple units, more specifically seven hours into the future as we mentioned in Section 5.3.3. These forward looking input parameters are necessary to make a single decision, while the reactive approach only looks at the present time unit. As

[2] http://www.knime.org/

we mentioned, predictions of future events contain uncertainty. The two strategies also differ in the way they treat uncertainty. The reactive strategy simply ignores such uncertainty by waiting until the future uncertainty has resolved, and then reacting. The proactive strategies leverages quantified uncertainty of future events and plans its decision based on predictions. In this manner, the proactive strategy handles uncertainty instead of ignoring it.

There are many possible traps and difficulties in building and implementing a predictive-based self-adaptation approach. Just to mention a few regarding the modeling steps of the predictive process: (1) lack of clarity around the problem definition, or problem defined too narrowly, (2) gathering data too old or otherwise irrelevant to the broader goal being targeted, (3) using a too complex model to represent a simple problem, the modeling methodology should be appropriate to the nature of the data, (4) not cleaning the data, not identifying outliers, or inefficient encoding of categorical variables, (5) going deep on a single specialized predictive algorithm rather than evaluating and comparing a broad spectrum of methods, finally (6) not monitoring model ongoing performance, inputs, outputs, and predictive power.

In order to avoid the previously mentioned pitfalls, there is a need to follow a methodological process. Typically, good modelers can come from the area of data analysis or computer sciences for the characteristics of their background. However, there is a wide range of other scientific disciplines including mathematics, statistics, engineering and physics, that have the essential baseline skills and trained ability to identify the nature of problems. In concrete, there are two main characteristics required for someone who wants to build and implement proactive adaptations: (1) curiosity and passion for data. Data is at the foundation of all our models and everything we do. It is required to understand where the data came from, what will it be used for, the quality, the robustness and stability of data, (2) technical competence in math and statistics and good programming skills. Although there are commercial software that already implements predictive analytics, there is a need for an in-depth understanding of the possible techniques and possible modifications of existing algorithms.

6.3 Perspectives

In this thesis we have considered a proactive approach that integrates predictive analysis into the autonomic control loop. However, our approach can be improved in several ways and can be applied to other possible scenarios. This section also outlines new directions for future research work.

6.3.1 Improving our approach

A crucial characteristic of the predictive modeling concept is that it is performed at design time by data analysts. In our approach the dynamic adjustment are carried out as parametric adaptations. We believe that we can use the same knowledge in a model-driven approach to guide structural adaptations. The main idea for improvement is to reflect the changes in models at runtime on the systems architecture [52]. These structural adaptations can be deplyoed in the running system using models@runtime tools like Kevoree [51].

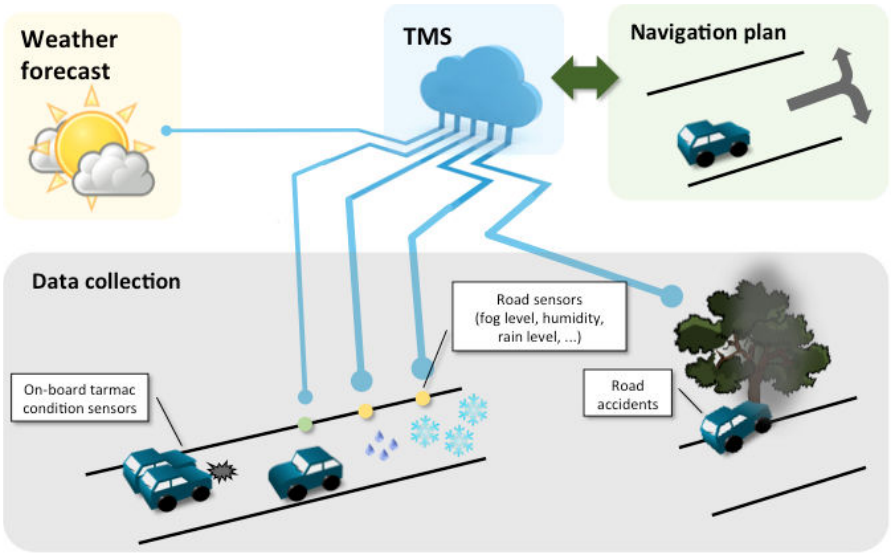


Figure 6.1 – A Traffic Management System (TMS) with proactive adaptation

A possible limitation of our approach that can be considered is the decision-making mechanism. In our case, we adopted a decision-making paradigm based on ECA rules, using the temperature as the main input parameter. As explained earlier, each individual rule is quite easy to specify, understand and implement: it can be seen as *if-then* constructs. However, to fully specify an adaptive system using ECA rules often requires to define numerous rules [47]. More over, one specific context can trigger several rules that can be conflicting at the same time. It is thus required to define additional constraints, priorities and exclusion rules to properly manage the set of ECA rules. A possible option to handle uncertainty is to use fuzzy logic, which deals with approximate values, rather than fixed and exact reasoning. Recently, there are works in extending UML for the modeling of fuzzy self-adaptive software systems [65].

Other possible improvement is to extend our approach towards self-organizing systems. The self-organizing property is bottom-up in contrast to self-adaptiveness, which is typically top-down [109]. Self-organization emphasizes decentralization and emergent functionalities on the interacting elements that are unaware of or have partial knowledge about the global system. In our case, we considered a centralized approach, where the computation is carried out in the proactive autonomic manager component and the predictions are consumed by the managed elements.

6.3.2 Other possible application scenarios

While this thesis has demonstrated the potential of predictive analysis in dynamic environmental monitoring, many opportunities for extending the scope of this thesis remain. This section presents some of these directions.

Cyber Physical Systems Other possible application scenarios include Cyber Physical Systems (CPS). As an example, let us consider the following traffic navigation scenario. Vehicles

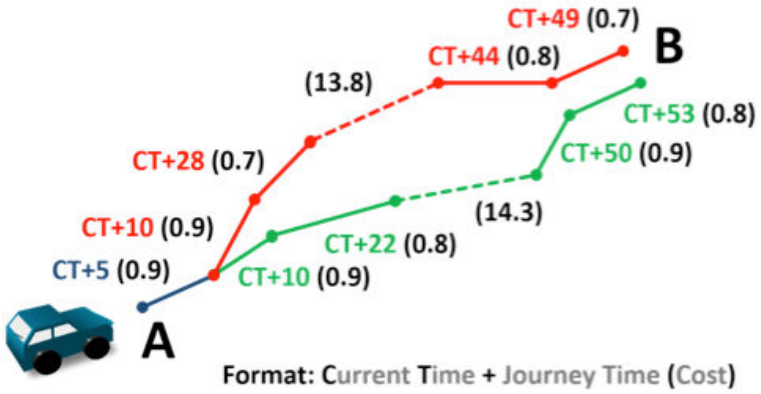


Figure 6.2 – A trajectory selection mechanism based predictive analysis

moving in a city communicate with a Traffic Management System (TMS) to obtain an optimal route to their destinations. TMS is a cloud-based dedicated service that decides on the route for each vehicle, taking into account the following vehicle-related parameters: (i) position, (ii) destination, and (iii) driver's preferences expressed as a cost function of journey time, journey distance, and safety. Figure 6.1 illustrates the previously described scenario.

A vehicle can follow different routes to reach its destination B from its current position A. Each route is a sequence of road segments. Each road segment has an associated cost that is calculated as a function of the route-relevant parameters and constraints, which correspond to the data layers (e.g. precipitation, traffic, speed limits). Then, routing is the process of choosing the route among the possible ones that maximizes the driver's preferences (e.g., journey time, journey distance, safety), represented as a utility vector.

The key idea is that the layers of information contain a projection of location-dependent data into the future, as provided by the analysis and prediction phases. For example, traffic level along a road segment is calculated not by the current level, but on the prediction of the traffic level when the vehicle is expected to reach the segment, see Figure 6.2. This allows, for instance, to penalize certain road segments that are known to become too busy at a certain time of the day.

Elastic Distributed Data Centers In this thesis we have proposed a prediction-based approach for anticipating critical conditions. Elastic distributed data centers are complex systems with a variety of operations and analytics taking place around the clock. Multiple teams need access at the same time, which requires coordination. In order to optimize resource use and manage workloads, system administrators monitor a huge number of parameters with frequent measurements for a fine-grained view. For example, data on CPU usage, memory residency, I/O activity, levels of disk storage, and many other parameters are all useful to collect as time series [39].

The utilization of our approach can contribute to prediction mechanisms that forecast the number of I/O operations, how they affect the systems performance and how to take advantage of it. Once these datasets are recorded as time series, data center operations teams can reconstruct the circumstances that lead to outages, plan upgrades by looking at trends, or even detect

many kinds of security intrusion by noticing changes in the volume and patterns of data transfer between servers and the outside world [39].

6.4 Publications and Dissemination Activities

Publications

- Ivan Paez Anaya. Integrating Predictive Analysis with Self-Adaptive Systems. Report of the GI-Dagstuhl Seminar 14433: Software Engineering for Self-Adaptive Systems. Oct 19-24. 2014.

- Paez Anaya I., Simko V., Bourcier J., Plouzeau N. and Jézéquel J.-M.: A prediction-driven adaptation approach for self-adaptive sensor networks. In Proceedings of 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS'14), Hyderabad, India, May 31, 2014.

- Hamadache K., Zerva P., Polyviou A., Simko V., Dautov R,. Gonidis F., Paez Anaya I.: Cost in the Cloud Rationalisation and Research Trails. In Proceedings of the 2nd International Conference on Advanced Cloud and Big Data (CBD 2014), November 20-22, Huangshan, Anhui, China, 2014

Research Internships

- FZI Forschungszentrum Informatik research center in Karlsruhe, Germany, under the supervision of Dr. Ing. Klaus Krogmann, from September to November 2013.

Presentations

Table 6.1 shows the most representatives presentations and talks regarding the integration of predictive analysis in self-adaptive pervasive systems that I had the opportunity to conduct during my doctorate.

Date	Topic	Place
20.10.2014	Integrating Predictive analysis in self-adaptive systems	Software Engineering for Self-Adaptive Systems, Dagstuhl Seminar, Germany
03.06.2014	Prediction-driven approach for self-adaptive systems	SEAMS'14- Hyderabad, India
06.11.2013	Proactive adaptation approach for pervasive distributed systems	KIT Doctoral round - Computer Sciences Department, Karlsruhe, Germany

Table 6.1 – Presentations and Talks

6.4.1 Projects

The research presented in this thesis was supported by the European Union within the FP7-Marie Curie Initial Training Network (ITN) RELATE[3], under grant agreement number 264840.

[3] http://www.relate-itn.eu/

Appendix A

Implementation Details

A.1 Time Series Analysis Background

In this section, we present a light introduction to the time series theory and the classical decomposition approach. This will serve as a motivation and starting point for defining resource predictor types and operations among them.

In time series analysis, we consider a discrete time model. Let t denote an integer time. A time series is an infinite vector of values indexed by time. We denote time series like this: Y_t, X_t, etc. The classical decomposition breaks up time series Y_t into three components as follows:

$$Y_t = m_t + s_t + X_t \tag{A.1}$$

where Y_t is the original time series, m_t is the trend component, s_t is the known pattern component (called a seasonal component in time series literature), X_t is the remainder, a stationary series.

The trend component captures a steady change, typically an increase, in the series. The trend can be linear, quadratic, exponential, etc. The known pattern component captures periodic patterns in the series, e.g. increases and decreases that repeat reliably over a defined period of time. There may be multiple known pattern components, e.g. hourly, daily, weekly, or monthly. When multiple season components are present, the decomposition formula above will have additional terms.

In the time series analysis using the classical decomposition approach, once the trend and known pattern components are removed, the remaining time series will be stationary, a technical term that guarantees the series has certain properties. A stationary time series can be analyzed using a model that captures serial correlation in the series. Serial correlation, or autocorrelation, is the relationship between he adjacent values of a time series. The objective of this analysis would be to determine a model (model selection) that does a good job of predicting future values based on past observations and solving for the best-fit parameters (parameter inference) of the model.

A historical trace of the series Y_t is first analyzed to determine trend and known pattern components. Then these are removed to obtain a stationary series, X_t. Model selection and parameter estimation is done using X_t.

A.2 MODIS Active Fire Detections for CONUS (2010)

Abstract:

This coverage represents year 2010 MODIS fire detections for the geographic area covering the continental United States including a 50km buffer around the periphery. The detections are obtained using both TERRA MODIS and AQUA MODIS data are collected and processed as a cooperative effort between the USDA Forest Service Remote Sensing Applications Center, NASA-Goddard Space Flight Center and the University of Maryland [49].

Purpose:

These fire detection data are collected for the USDA Forest Service MODIS Active Fire Mapping Program (http://activefiremaps.fs.fed.us). These data are intended to provide a synoptic view of active fires for the past and present over the specified time period. These data are collected at a spatial resolution of 1 kilometer and therefore are only intended for geographic display and analysis at the national and regional levels. No responsibility is assumed by the USDA Forest Service in the use of these data [49]. Figure A.1 illustrates an image of an actual fire taken by the MODIS satellite. Table A.1 describe the data type of the data points of a fire report.

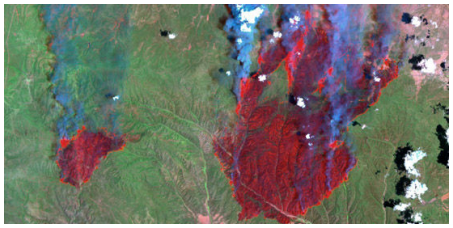


Figure A.1 – An actual fire satellite image taken by MODIS near RODEO LANDSAT

Label	**Definition**
FID	Internal feature number.
Shape	Feature geometry
AREA	Area of feature in internal units squared.
PERIMETER	Perimeter of feature in internal units
LAT	Latitude of fire detection.
LONG	Longitude of fire detection
DATE	Date of fire detection.
JULIAN	Julian date of fire detection
GMT	Time of fire detection.
TEMP	Measured brightness temperature
SRC	Station source of MODIS data and detection

Table A.1 – MODIS fire detections metadata [49]

A.3 ISH/ISD Weather Stations in MS, USA

Table A.2 contains the ISH/ISD weather stations in the Mississippi state of the USA. In total we collected data from 20 ISH/ISD weather stations from Mississippi, USA.

Code	Station Name	Latitude	Longitude	Elevation
722354	HAWKINS FIELD AIRPORT	32.337	-90.221	104.2
722350	JACKSON INTERNATIONAL AIRPORT	32.320	-90.078	100.6
999999	NEWTON 5 ENE	32.338	-89.07	114.0
747688	TRENT LOTT INTL AIRPORT	30.463	-88.532	5.5
998219	BAY WAVELAND YACHT CLUB	30.317	-89.317	5.0
722358	Mc COM/PIKE CO/J E LWS FD AP	31.183	-90.471	125.9
723306	COLUMBUS AFB AIRPORT	33.650	-88.45	66.8
722340	MERIDIAN/KEY FIELD AIRPORT	32.335	-88.744	89.6
747686	KEESLER AIR FORCE BASE	30.417	-88.917	10.1
999999	HOLLY SPRINGS 4 N	34.822	-89.435	147.5
722357	HARDY-ANRES FD NATCHEZ-ADAMS COUNTY AIRPORT	31.617	-91.283	82.9
998234	WEST PIER GULFPORT	30.350	-89.083	7.0
998271	PETITS BOIS ISLAND	30.350	-88.417	5.0
998239	RANGE A REAR PASCAGOULA	30.333	-88.517	5.0
723320	TUPELO REGIONAL AIRPORT	34.262	-88.771	110.0
722345	MERIDIAN NAS/Mc CAIN FD AP	32.549	-88.566	82.6
998233	GULFPORT OUTER RANGE	30.233	-88.983	13.0
722348	HATTIESBURG-LAUREL RGNL AP	31.467	-89.333	89.3

Table A.2 – ISD/ISH Weather Stations with in 50 km radius range from fire detections in Mississippi State USA

A.4 PMML: Predictive Model Markup Language

```
<?xml version="1.0"?>
<PMML version="4.1" xmlns="http://www.dmg.org/PMML-4_1"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.dmg.org/PMML-4_1 http://www.
    dmg.org/v4-1/pmml-4-1.xsd">
    <Header copyright="Copyright (c) 2013 www.inria.fr"
    description="A binary tree model of significant wildland
    fire potential.">
        <Extension name="user" value="ipaezana" extender="
    TreeModel/PMML"/>
        <Application name="TreeModel/PMML" version="1.3"/>
        <Timestamp>2013-11-05 10:00:15</Timestamp>
    </Header>
  <DataDictionary numberOfFields="5">

    <DataField name="temperature" optype="continuous">
      <Interval closure="openOpen" leftMargin="-20" rightMargin
    ="120" />
    </DataField>

    <DataField name="humidity" optype="continuous">
      <Interval closure="closedClosed" leftMargin="0"
    rightMargin="100" />
    </DataField>

    <DataField name="windy" optype="categorical">
      <Value value="true" />
      <Value value="false" />
    </DataField>

    <DataField name="outlook" optype="categorical" >
      <Value value="sunny" />
      <Value value="cloudy "/>
      <Value value="rain" />
    </DataField>

    <DataField name="outcome" optype="ordinal">
      <Value value="above-normal" />
      <Value value="normal" />
      <Value value="below-normal" />
    </DataField>
```

```
34
35  </DataDictionary>
36
37  <TreeModel modelName="firePotential">
38
39  <MiningSchema>
40    <MiningField name="temperature" />
41    <MiningField name="humidity" />
42    <MiningField name="windy" />
43    <MiningField name="outlook" />
44    <MiningField name="outcome" usageType="predicted" />
45  </MiningSchema>
46
47  <Node score="normal">
48    <True/>
49
50  <Node score="normal">
51    <Predicate field="outlook" operator="equal" value="sunny"/>
52
53    <Node score="normal">
54
55      <CompoundPredicate booleanOperator="and">
56        <Predicate field="temperature" operator="lessThan"
    value="90" />
57        <Predicate field="temperature" operator="greaterThan"
    value="50" />
58      </CompoundPredicate>
59
60      <Node score="above-normal">
61        <CompoundPredicate booleanOperator="and">
62          <Predicate field="humidity" operator="lessThan" value
    ="30" />
63          <Predicate field="windy" operator="equal" value="true
    " />
64        </CompoundPredicate>
65      </Node>
66
67      <Node score="normal">
68        <CompoundPredicate booleanOperator="and">
69          <Predicate field="humidity" operator="greaterOrEqual"
     value="30" />
70          <Predicate field="windy" operator="equal" value="
    false" />
```

```
71          </CompoundPredicate>
72        </Node>
73
74      </Node>
75
76      <Node score="normal">
77
78        <CompoundPredicate booleanOperator="or">
79          <Predicate field="temperature" operator="greaterOrEqual
      " value="90"/>
80          <Predicate field="temperature" operator="lessOrEqual"
      value="50" />
81        </CompoundPredicate>
82
83        <Node score="above-normal">
84          <CompoundPredicate booleanOperator="and">
85            <Predicate field="temperature" operator="
      greaterOrEqual" value="90" />
86            <Predicate field="humidity" operator="lessThan" value
      ="30" />
87            <Predicate field="windy" operator="equal" value="true
      " />
88          </CompoundPredicate>
89        </Node>
90
91        <Node score="below-normal">
92          <CompoundPredicate booleanOperator="and">
93            <Predicate field="temperature" operator="lessOrEqual"
       value="50" />
94            <Predicate field="humidity" operator="greaterOrEqual"
       value="30" />
95            <Predicate field="windy" operator="equal" value="
      false" />
96          </CompoundPredicate>
97        </Node>
98      </Node>
99    </Node>
100
101   <Node score="normal">
102     <CompoundPredicate booleanOperator="or">
103       <Predicate field="outlook" operator="equal" value="
      overcast" />
```

```
104       <Predicate field="outlook" operator="equal" value="rain"
      />
105     </CompoundPredicate>
106
107     <Node score="normal">
108
109     <CompoundPredicate booleanOperator="and">
110         <Predicate field="temperature" operator="greaterThan"
      value="50" />
111         <Predicate field="temperature" operator="lessThan"
      value="90" />
112         <Predicate field="outlook" operator="equal" value="
      overcast" />
113         <Predicate field="humidity" operator="greaterThan"
      value="30" />
114         <Predicate field="windy" operator="equal" value="false"
       />
115       </CompoundPredicate>
116     </Node>
117
118     <Node score="below-normal">
119       <CompoundPredicate booleanOperator="and">
120         <Predicate field="outlook" operator="equal" value="rain
      " />
121         <Predicate field="humidity" operator="greaterThan"
      value="70" />
122       </CompoundPredicate>
123     </Node>
124
125   </Node>
126
127   </Node>
128   </TreeModel>
129   </PMML>
```

Listing A.1 – Example PMML TreeModel

A.5 Fit best ARIMA model to univariate time series

Description Returns best ARIMA model according to either AIC, AICc or BIC value. The function conducts a search over possible model within the order constraints provided [72]. The following listing A.2 describe in detail its use with the R language.

Usage

```
auto.arima(x, d=NA, D=NA, max.p=5, max.q=5,
     max.P=2, max.Q=2, max.order=5, max.d=2, max.D=1,
     start.p=2, start.q=2, start.P=1, start.Q=1,
     stationary=FALSE, seasonal=TRUE,
     ic=c("aicc","aic", "bic"), stepwise=TRUE, trace=FALSE,
     approximation=(length(x)>100 | frequency(x)>12), xreg=NULL
    ,
     test=c("kpss","adf","pp"), seasonal.test=c("ocsb","ch"),
     allowdrift=TRUE, lambda=NULL, parallel=FALSE, num.cores=
    NULL)
```

Listing A.2 – Description of the AUTO.ARIMA function

Details Non-stepwise selection can be slow, especially for seasonal data. Stepwise algorithm outlined in Hyndman and Khandakar (2008) except that the default method for selecting seasonal differences is now the OCSB test rather than the Canova-Hansen test.

Usage

```
library(forecast)
fit <- auto.arima(WWWusage)
plot(forecast(fit,h=20))
```

Listing A.3 – Usage of the AUTO.ARIMA function

Argument	Definition
x	a univariate time series
d	Order of first-differencing. If missing, will choose a value based on KPSS test.
D	Order of seasonal-differencing. If missing, will choose a value based on OCSB test.
max.p	Maximum value of p
max.q	Maximum value of q
max.P	Maximum value of P
max.Q	Maximum value of Q
max.order	Maximum value of p+q+P+Q if model selection is not stepwise.
max.d	Maximum number of non-seasonal differences
max.D	Maximum number of seasonal differences
start.p	Starting value of p in stepwise procedure.
start.q	Starting value of q in stepwise procedure.
start.P	Starting value of P in stepwise procedure.
start.Q	Starting value of Q in stepwise procedure.
stationary	If TRUE, restricts search to stationary models.
seasonal	If FALSE, restricts search to non-seasonal models.
ic	Information criterion to be used in model selection.
stepwise	If TRUE, will do stepwise selection (faster). Otherwise, it searches over all models. Non-stepwise selection can be very slow, especially for seasonal models.
trace	If TRUE, the list of ARIMA models considered will be reported.
approximation	If TRUE, estimation is via conditional sums of squares andthe information criteria used for model selection are approximated. The final model is still computed using maximum likelihood estimation. Approximation should be used for long time series or a high seasonal period to avoid excessive computation times.
xreg	Optionally, a vector or matrix of external regressors, which must have the same number of rows as x.
test	Type of unit root test to use. See ndiffs for details.
seasonal.test	This determines which seasonal unit root test is used. See nsdiffs for details.
allowdrift	If TRUE, models with drift terms are considered.
lambda	Box-Cox transformation parameter. Ignored if NULL. Otherwise, data transformed before model is estimated.
parallel	If TRUE and stepwise = FALSE, then the specification search is done in parallel. This can give a significant speedup on mutlicore machines.
num.cores	Allows the user to specify the amount of parallel processes to be used if parallel = TRUE and stepwise = FALSE. If NULL, then the number of logical cores is automatically detected.

Table A.3 – Arguments of the AUTO.ARIMA function

List of Figures

List of Tables

Listings

Bibliography

[1] Ying Zhang A.I. McLeod and Changjiang Xu. Fitar: Subset ar-model fitting, 2013. R package version 1.94. Accessed Feb 2015. http://cran.r-project.org/web/packages/FitAR/index.html.

[2] Mourad Alia, Frank Eliassen, Svein Hallsteinsen, and Erlend Stav. Madam: towards a flexible planning-based middleware. In *Proceedings of the 2006 international workshop on Self-adaptation and self-managing systems*, pages 96–96. ACM, 2006.

[3] Cameron Alverson. Polling and statistical models can't predict the future. *Personal Blog*, 2012. Retrieved on 01 Dec. 2014 from http://www.cameronalverson.com/2012/09/polling-and-statistical-models-cant.html.

[4] Jesper Andersson, Rogerio De Lemos, Sam Malek, and Danny Weyns. Modeling dimensions of self-adaptive software systems. In *Software engineering for self-adaptive systems*, pages 27–47. Springer, 2009.

[5] Martin Arlitt and Tai Jin. A workload characterization study of the 1998 world cup web site. *Network, IEEE*, 14(3):30–37, 2000.

[6] Rafael R. Aschoff and Andrea Zisman. Proactive adaptation of service composition. *2012 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pages 1–10, June 2012.

[7] Kevin Ashton. That 'internet of things' thing. *RFiD Journal*, 22:97–114, 2009.

[8] K. J. Astrom and B. Wittenmark. *Adaptive control*. Addison-Wesley, 1995.

[9] K. Bache, School of Information M. Lichman. University of California, Irvine, and Computer Sciences. (uci) machine learning repository, "2013". Accessed Jan 2015. http://archive.ics.uci.edu/ml.

[10] David Barstow. Artificial intelligence and software engineering. In *Proceedings of the 9th international conference on Software Engineering*, pages 200–211. IEEE Computer Society Press, 1987.

[11] Victor R Basili. The experimental paradigm in software engineering. In *Experimental Software Engineering Issues: Critical Assessment and Future Directions*, pages 1–12. Springer, 1993.

[12] Victor R Basili, Forrest Shull, and Filippo Lanubile. Building knowledge through families of experiments. *Software Engineering, IEEE Transactions on*, 25(4):456–473, 1999.

[13] Nelly Bencomo, Peter Sawyer, Gordon S Blair, and Paul Grace. Dynamically adaptive systems are product lines too: Using model-driven techniques to capture dynamic variability of adaptive systems. In *SPLC (2)*, pages 23–32, 2008.

[14] Michael R Berthold. Mixed fuzzy rule formation. *International journal of approximate reasoning*, 32(2):67–84, 2003.

[15] Michael R Berthold, Christian Borgelt, Frank Höppner, and Frank Klawonn. *Guide to Intelligent Data Analysis*, volume 42. Springer Science & Business Media, 2010.

[16] Michael R Berthold and Jay Diamond. Constructive training of probabilistic neural networks. *Neurocomputing*, 19(1):167–183, 1998.

[17] Joseph P. Bigus, Don A. Schlosnagle, Jeff R. Pilgrim, W Nathaniel Mills III, and Yixin Diao. Able: A toolkit for building multiagent autonomic systems. *IBM Systems Journal*, 41(3):350–371, 2002.

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

[19] Barry W Boehm, Ray Madachy, Bert Steece, et al. *Software cost estimation with Cocomo II with Cdrom*. Prentice Hall PTR, 2000.

[20] George Edward Pelham Box and G.C. Reinsel Jenkins. Time series analysis: Forecasting and control. 1994.

[21] George Edward Pelham Box and Gwilym Jenkins. *Time Series Analysis, Forecasting and Control*. Holden-Day, Incorporated, 1970.

[22] George EP Box and David A Pierce. Distribution of residual autocorrelations in autoregressive-integrated moving average time series models. *Journal of the American statistical Association*, 65(332):1509–1526, 1970.

[23] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[24] Yuriy Brun, Giovanna Di Marzo Serugendo, Cristina Gacek, Holger Giese, Holger Kienle, Marin Litoiu, Hausi Müller, Mauro Pezzè, and Mary Shaw. Engineering self-adaptive systems through feedback loops. In *Software engineering for self-adaptive systems*, pages 48–70. Springer, 2009.

[25] Jim Buckley, Tom Mens, Matthias Zenger, Awais Rashid, and Günter Kniesel. Towards a taxonomy of software change. *Journal of Software Maintenance and Evolution: Research and Practice*, 17(5):309–332, 2005.

[26] Carlos Cetina, Joan Fons, and Vicente Pelechano. Applying software product lines to build autonomic pervasive systems. In *Software Product Line Conference, 2008. SPLC'08. 12th International*, pages 117–126. IEEE, 2008.

[27] Betty HC Cheng, Rogerio De Lemos, Holger Giese, Paola Inverardi, Jeff Magee, Jesper Andersson, Basil Becker, Nelly Bencomo, Yuriy Brun, Bojan Cukic, et al. Software engineering for self-adaptive systems: A research roadmap. In *Software engineering for self-adaptive systems*, pages 1–26. Springer-Verlag, 2009.

[28] Betty HC Cheng, Pete Sawyer, Nelly Bencomo, and Jon Whittle. A goal-based modeling approach to develop requirements of an adaptive system with environmental uncertainty. In *Model Driven Engineering Languages and Systems*, pages 468–483. Springer, 2009.

[29] Shang-Wen Cheng, David Garlan, and Bradley Schmerl. Architecture-based self-adaptation in the presence of multiple objectives. In *Proceedings of the 2006 international workshop on Self-adaptation and self-managing systems*, pages 2–8. ACM, 2006.

[30] Shang-Wen Cheng, Vahe V Poladian, David Garlan, and Bradley Schmerl. Improving architecture-based self-adaptation through resource prediction. In *Software Engineering for Self-Adaptive Systems*, pages 71–88. Springer, 2009.

[31] Deshan Cooray, Sam Malek, Roshanak Roshandel, and David Kilgore. Resisting reliability degradation through proactive reconfiguration. In *Proceedings of the IEEE/ACM international conference on Automated software engineering*, pages 83–92. ACM, 2010.

[32] Keling Da, Marc Dalmau, Philippe Roose, et al. A survey of adaptation systems. *International Journal on Internet and Distributed Computing Systems*, 2(1):1–18, 2011.

[33] Erwan Daubert, François Fouquet, Olivier Barais, Grégory Nain, Gerson Sunye, J-M Jezequel, J-L Pazat, and Brice Morin. A models@ runtime framework for designing and managing service-based applications. In *Software Services and Systems Research-Results and Challenges (S-Cube), 2012 Workshop on European*, pages 10–11. IEEE, 2012.

[34] Jan G De Gooijer and Rob J Hyndman. 25 years of time series forecasting. *International journal of forecasting*, 22(3):443–473, 2006.

[35] Rogério De Lemos, Holger Giese, Hausi A Müller, Mary Shaw, Jesper Andersson, Marin Litoiu, Bradley Schmerl, Gabriel Tamura, Norha M Villegas, Thomas Vogel, et al. Software engineering for self-adaptive systems: A second research roadmap. In *Software Engineering for Self-Adaptive Systems II*, pages 1–32. Springer-Verlag, 2013.

[36] Dmg.org. Pmml 4.2 - general structure, 2015. Accessed 15 Mar. 2015. `http://www.dmg.org/v4-2/GeneralStructure.html`.

[37] Simon Dobson, Spyros Denazis, Antonio Fernández, Dominique Gaïti, Erol Gelenbe, Fabio Massacci, Paddy Nixon, Fabrice Saffre, Nikita Schmidt, and Franco Zambonelli. A survey of autonomic communications. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 1(2):223–259, 2006.

[38] Enrique J Duarte-Melo and Mingyan Liu. Analysis of energy consumption and lifetime of heterogeneous wireless sensor networks. In *Global Telecommunications Conference, 2002. GLOBECOM'02. IEEE*, volume 1, pages 21–25. IEEE, 2002.

[39] Ted Dunning and Ellen Friedman. *Time Series Databases New Ways to Store and Access Data*, volume 1st Edit. Oreilly and Associates, 2014.

[40] George Edwards, Sam Malek, and Nenad Medvidovic. Scenario-driven dynamic analysis of distributed architectures. In *Fundamental Approaches to Software Engineering*, pages 125–139. Springer, 2007.

[41] Ahmed Elkhodary, Naeem Esfahani, and Sam Malek. Fusion: a framework for engineering self-tuning self-adaptive software systems. In *Proceedings of the eighteenth ACM SIGSOFT international symposium on Foundations of software engineering*, pages 7–16. ACM, 2010.

[42] Naeem Esfahani, Ehsan Kouroshfar, and Sam Malek. Taming uncertainty in self-adaptive software. In *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*, pages 234–244. ACM, 2011.

[43] Naeem Esfahani and Sam Malek. Uncertainty in self-adaptive software systems. In *Software Engineering for Self-Adaptive Systems II*, pages 214–238. Springer, 2013.

[44] Robert S Fabry. How to design a system in which modules can be changed on the fly. In *ICSE'76: Proceedings of the 2nd International Conference on Software Engineering*, pages 470–476. IEEE Computer Society Press, 1976.

[45] A. Filieri, C. Ghezzi, A. Leva, and M. Maggio. Self-adaptive software meets control theory: a preliminary approach supporting reliability requirements. *in Proc. ASE*, page 283–292, 2011.

[46] Antonio Filieri, Carlo Ghezzi, Alberto Leva, and R Ole. Reliability-Driven Dynamic Binding via Feedback Control. 2:43–52, 2012.

[47] Franck Fleurey and Arnor Solberg. A domain specific modeling language supporting specification, simulation and execution of dynamic adaptive systems. In *Model Driven Engineering Languages and Systems*, pages 606–621. Springer, 2009.

[48] Jacqueline Floch, Svein Hallsteinsen, Erlend Stav, Frank Eliassen, Ketil Lund, and Eli Gjorven. Using architecture models for runtime adaptability. *Software, IEEE*, 23(2):62–70, 2006.

[49] USDA. Forest Service. Modis active fire detections for conus (2010) - through 12/31/2010 2300 mdt, 2011. Accessed March 10, 2015.

[50] Scott Fortmann-Roe. Accurately measuring model prediction error., May 2012. R package version 5.9. Accessed 30 Mar. 2015.`http://scott.fortmann-roe.com/docs/MeasuringError.html`.

[51] François Fouquet, Erwan Daubert, Noël Plouzeau, Olivier Barais, Johann Bourcier, and Jean-Marc Jézéquel. Dissemination of reconfiguration policies on mesh networks. In *Distributed Applications and Interoperable Systems*, pages 16–30. Springer, 2012.

[52] Francois Fouquet, Brice Morin, Franck Fleurey, Olivier Barais, Noel Plouzeau, and Jean-Marc Jezequel. A dynamic component model for cyber physical systems. In *Proceedings of the 15th ACM SIGSOFT symposium on Component Based Software Engineering*, pages 135–144. ACM, 2012.

[53] François Fouquet, Grégory Nain, Brice Morin, Erwan Daubert, Olivier Barais, Noël Plouzeau, and Jean-Marc Jézéquel. *An eclipse modelling framework alternative to meet the models@ runtime requirements*. Springer, 2012.

[54] John E Gaffney. Estimating the number of faults in code. *Software Engineering, IEEE Transactions on*, (4):459–464, 1984.

[55] João Gama. Functional trees. *Machine Learning*, 55(3):219–250, 2004.

[56] Carlos E Garcia, David M Prett, and Manfred Morari. Model predictive control: theory and practice—a survey. *Automatica*, 25(3):335–348, 1989.

[57] David Garlan. A 10-year perspective on software engineering self-adaptive systems (keynote). In *Software Engineering for Adaptive and Self-Managing Systems (SEAMS), 2013 ICSE Workshop on*, pages 2–2. IEEE, 2013.

[58] David Garlan, Shang-Wen Cheng, An-Cheng Huang, Bradley Schmerl, and Peter Steenkiste. Rainbow: Architecture-based self-adaptation with reusable infrastructure. *Computer*, 37(10):46–54, 2004.

[59] David Garlan, Shang-Wen Cheng, and Bradley Schmerl. Increasing system dependability through architecture-based self-repair. In *Architecting dependable systems*, pages 61–89. Springer, 2003.

[60] David Garlan and Bradley Schmerl. Model-based adaptation for self-healing systems. In *Proceedings of the first workshop on Self-healing systems*, pages 27–32. ACM, 2002.

[61] John C Georgas and Richard N Taylor. Policy-based self-adaptive architectures: a feasibility study in the robotics domain. In *Proceedings of the 2008 international workshop on Software engineering for adaptive and self-managing systems*, pages 105–112. ACM, 2008.

[62] Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, and Marimuthu Palaniswami. Internet of things (iot): A vision, architectural elements, and future directions. *Future Generation Computer Systems*, 29(7):1645–1660, 2013.

[63] Venkat N Gudivada, Ricardo Baeza-Yates, and Vijay V Raghavan. Big data: Promises and problems. *Computer*, (3):20–23, 2015.

[64] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H Witten. The weka data mining software: an update. *ACM SIGKDD explorations newsletter*, 11(1):10–18, 2009.

[65] Deshuai Han, Qiliang Yang, and Jianchun Xing. Extending uml for the modeling of fuzzy self-adaptive software systems. In *Control and Decision Conference (2014 CCDC), The 26th Chinese*, pages 2400–2406. IEEE, 2014.

[66] Mark Harman. The role of artificial intelligence in software engineering. In *Realizing Artificial Intelligence Synergies in Software Engineering (RAISE), 2012 First International Workshop on*, pages 1–6. IEEE, 2012.

[67] Wendi Rabiner Heinzelman, Anantha Chandrakasan, and Hari Balakrishnan. Energy-efficient communication protocol for wireless microsensor networks. In *System sciences, 2000. Proceedings of the 33rd annual Hawaii international conference on*, pages 10–pp. IEEE, 2000.

[68] Nikolas Roman Herbst, Nikolaus Huber, Samuel Kounev, and Erich Amrehn. Self-adaptive workload classification and forecasting for proactive resource provisioning. In *Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering*, pages 187–198. ACM, 2013.

[69] Julia Hielscher, Raman Kazhamiakin, Andreas Metzger, and Marco Pistore. *A framework for proactive self-adaptation of service-based applications based on online testing*. Springer, 2008.

[70] Paul Horn. Autonomic computing: Ibm's perspective on the state of information technology. *IBM*, 2001.

[71] Markus C Huebscher and Julie A McCann. A survey of autonomic computing—degrees, models, and applications. *ACM Computing Surveys (CSUR)*, 40(3):7, 2008.

[72] Rob J Hyndman, Yeasmin Khandakar, et al. Automatic time series for forecasting: the forecast package for r. Technical report, Monash University, Department of Econometrics and Business Statistics, 2007.

[73] Nicholas R Jennings. On agent-based software engineering. *Artificial intelligence*, 117(2):277–296, 2000.

[74] Paul C Jorgensen. *Software testing: a craftsman's approach*. CRC press, 2013.

[75] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. Reinforcement learning: A survey. *arXiv preprint cs/9605103*, 1996.

[76] Kyo C Kang, Sholom G Cohen, James A Hess, William E Novak, and A Spencer Peterson. Feature-oriented domain analysis (foda) feasibility study. Technical report, DTIC Document, 1990.

[77] Joost-Pieter Katoen, Ivan S Zapreev, Ernst Moritz Hahn, Holger Hermanns, and David N Jansen. The ins and outs of the probabilistic model checker mrmc. *Performance evaluation*, 68(2):90–104, 2011.

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

[79] Dongsun Kim and Sooyong Park. Reinforcement learning-based dynamic adaptation planning method for architecture-based self-managed software. In *Software Engineering for Adaptive and Self-Managing Systems, 2009. SEAMS'09. ICSE Workshop on*, pages 76–85. IEEE, 2009.

[80] Barbara A Kitchenham, Shari Lawrence Pfleeger, Lesley M Pickard, Peter W Jones, David C. Hoaglin, Khaled El Emam, and Jarrett Rosenberg. Preliminary guidelines for empirical research in software engineering. *Software Engineering, IEEE Transactions on*, 28(8):721–734, 2002.

[81] KNIME. Knime, open source, open for innovation, 2015. `http://www.knime.org/`. Accessed 15 Mar. 2015.

[82] Jeff Kramer and Jeff Magee. Self-managed systems: an architectural challenge. In *Future of Software Engineering, 2007. FOSE'07*, pages 259–268. IEEE, 2007.

[83] Marta Kwiatkowska, Gethin Norman, and David Parker. Prism: probabilistic model checking for performance and reliability analysis. *ACM SIGMETRICS Performance Evaluation Review*, 36(4):40–45, 2009.

[84] Philipp Leitner, Anton Michlmayr, Florian Rosenberg, and Schahram Dustdar. Monitoring, prediction and prevention of sla violations in composite services. In *Web Services (ICWS), 2010 IEEE International Conference on*, pages 369–376. IEEE, 2010.

[85] Kasper Luckow, Corina S Păsăreanu, Matthew B Dwyer, Antonio Filieri, and Willem Visser. Exact and approximate probabilistic symbolic execution for nondeterministic programs. In *Proceedings of the 29th ACM/IEEE international conference on Automated software engineering*, pages 575–586. ACM, 2014.

[86] Anastassios Michail and Anthony Ephremides. Energy efficient routing for connection-oriented traffic in ad-hoc wireless networks. In *Personal, Indoor and Mobile Radio Communications, 2000. PIMRC 2000. The 11th IEEE International Symposium on*, volume 2, pages 762–766. IEEE, 2000.

[87] Ryszard S Michalski, Ivan Bratko, and Avan Bratko. *Machine learning and data mining; methods and applications*. John Wiley & Sons, Inc., 1998.

[88] Tom M Mitchell. *Machine learning. 1997*, volume 45. McGraw Hill, 1997.

[89] Tom M Mitchell. Machine learning and data mining. *Communications of the ACM*, 42(11):30–36, 1999.

[90] Theophano Mitsa. *Temporal data mining*. CRC Press, 2010.

[91] Brice Morin. *Leveraging Models from Design-time to Runtime to Support Dynamic Variability*. PhD thesis, Rennes 1, 2010.

[92] Brice Morin, Olivier Barais, Gregory Nain, and Jean-Marc Jezequel. Taming dynamically adaptive systems using models and aspects. In *Proceedings of the 31st International Conference on Software Engineering*, pages 122–132. IEEE Computer Society, 2009.

[93] Dushyanth Narayanan. Operating system support for mobile interactive applications. Technical report, DTIC Document, 2002.

[94] Kumpati S Narendra and Osvaldo A Driollet. Adaptive control using multiple models, switching, and tuning. In *Adaptive Systems for Signal Processing, Communications, and Control Symposium 2000. AS-SPCC. The IEEE 2000*, pages 159–164. IEEE, 2000.

[95] Charles Nyce and API CPCU. Predictive analytics white paper. *American Institute for CPCU. Insurance Institute of America*, pages 9–10, 2007.

[96] HA Oldenkamp. Probabilistic model checking: A comparison of tools. *UT*, 2007.

[97] Peyman Oreizy, Michael M Gorlick, Richard N Taylor, Dennis Heimbigner, Gregory Johnson, Nenad Medvidovic, Alex Quilici, David S Rosenblum, and Alexander L Wolf. An architecture-based approach to self-adaptive software. *IEEE Intelligent systems*, 14(3):54–62, 1999.

[98] Ivan Dario Paez Anaya, Viliam Simko, Johann Bourcier, Noël Plouzeau, Jean-Marc Jézéquel, et al. A prediction-driven adaptation approach for self-adaptive sensor networks. In *9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, 2014.

[99] Tharindu Patikirikorala and Alan Colman. Feedback controllers in the cloud. *Swinburne University*, 2011.

[100] Tharindu Patikirikorala, Alan Colman, J Han, and L Wang. Tech-report: Multi-model driven framework to implement self-managing control systems for qos management. *Swinburne University of Technology*, 2010.

[101] Tharindu Patikirikorala, Alan Colman, Jun Han, and Liuping Wang. A multi-model framework to implement self-managing control systems for qos management. In *Proceedings of the 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pages 218–227. ACM, 2011.

[102] Tharindu Patikirikorala, Alan Colman, Jun Han, and Liuping Wang. A systematic survey on the design of self-adaptive software systems using control engineering approaches. In *Software Engineering for Adaptive and Self-Managing Systems (SEAMS), 2012 ICSE Workshop on*, pages 33–42. IEEE, 2012.

[103] Brad Peters. The big data gold rush. *Forbes Magazine*, 2012. Retrieved on 28 Nov. 2014 from `http://www.forbes.com/sites/bradpeters/2012/06/21/the-big-data-gold-rush/`.

[104] John Platt et al. Fast training of support vector machines using sequential minimal optimization. *Advances in kernel methods—support vector learning*, 3, 1999.

[105] Vahe Poladian, David Garlan, Mary Shaw, Mahadev Satyanarayanan, Bradley Schmerl, and Joao Sousa. Leveraging resource prediction for anticipatory dynamic configuration. In *Proceedings of the First IEEE International Conference on Self-Adaptive and Self-Organizing Systems, SASO-2007, 214- 223*. IEEE, 2007.

[106] Vahe V Poladyan. *Tailoring configuration to user's tasks under uncertainty*. PhD thesis, CARNEGIE MELLON UNIVERSITY, 2008.

[107] Martin Riedmiller and Heinrich Braun. A direct adaptive method for faster backpropagation learning: The rprop algorithm. In *Neural Networks, 1993., IEEE International Conference on*, pages 586–591. IEEE, 1993.

[108] Romain Rouvoy, Paolo Barone, Yun Ding, Frank Eliassen, Svein Hallsteinsen, Jorge Lorenzo, Alessandro Mamelli, and Ulrich Scholz. Music: Middleware support for self-adaptation in ubiquitous and service-oriented environments. In *Software engineering for self-adaptive systems*, pages 164–182. Springer, 2009.

[109] Mazeiar Salehie and Ladan Tahvildari. Self-adaptive software: Landscape and Research Challenges. *ACM Transactions on Autonomous and Adaptive Systems*, 4(2):1–42, May 2009.

[110] Robert E Schapire and Yoav Freund. *Boosting: Foundations and algorithms*. MIT Press, 2012.

[111] Robert H Shumway and David S Stoffer. *Time series analysis and its applications: with R examples*. Springer Science & Business Media, 2010.

[112] Eric Siegel. *Predictive analytics: the power to predict who will click, buy, lie, or die*. John Wiley & Sons, 2013.

[113] R Development Core Team. R: A language and environment for statistical computing. r foundation for statistical computing, vienna, austria, 2012, 2012.

[114] Gerald Tesauro. Reinforcement learning in autonomic computing: A manifesto and case studies. *Internet Computing, IEEE*, 11(1):22–30, 2007.

[115] Gerald Tesauro, David M Chess, William E Walsh, Rajarshi Das, Alla Segal, Ian Whalley, Jeffrey O Kephart, and Steve R White. A multi-agent systems approach to autonomic computing. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems-Volume 1*, pages 464–471. IEEE Computer Society, 2004.

[116] Gerald Tesauro, Nicholas K Jong, Rajarshi Das, and Mohamed N Bennani. A hybrid reinforcement learning approach to autonomic resource allocation. In *Autonomic Computing, 2006. ICAC'06. IEEE International Conference on*, pages 65–73. IEEE, 2006.

[117] Texas Instruments. CC2420 Datashet. `http://www.ti.com/product/cc2420`. Online; accessed 15 March 2015.

[118] María Gabriela Calle Torres. *Energy consumption in wireless sensor networks using gsp*. PhD thesis, University of Pittsburgh, 2006.

[119] Axel van Lamsweerde. Requirements engineering: from craft to discipline. In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*, pages 238–249. ACM, 2008.

[120] Hans Van Vliet, Hans Van Vliet, and JC Van Vliet. *Software engineering: principles and practice*, volume 3. Wiley, 1993.

[121] Liuping Wang. *Model predictive control system design and implementation using MATLAB®*. springer, 2009.

[122] Qin Wang, Mark Hempstead, and Woodward Yang. A realistic power consumption model for wireless sensor network devices. In *Sensor and Ad Hoc Communications and Networks, 2006. SECON'06. 2006 3rd Annual IEEE Communications Society on*, volume 1, pages 286–295. IEEE, 2006.

[123] Mark Weiser. The computer for the 21st century. *Scientific american*, 265(3):94–104, 1991.

[124] Rob J Hyndman with contributions from Slava Razbash and Drew Schmidt. forecast: Forecasting functions for time series and linear models, 2015.

[125] Rich Wolski, Neil T Spring, and Jim Hayes. The network weather service: a distributed resource performance forecasting service for metacomputing. *Future Generation Computer Systems*, 15(5):757–768, 1999.

[126] James Wu and Stephen Coggeshall. *Foundations of Predictive Analytics (Chapman & Hall/CRC Data Mining and Knowledge Discovery Series)*. Chapman & Hall/CRC, 2012.

[127] Qi Zhang, Mohamed Faten Zhani, Shuo Zhang, Quanyan Zhu, Raouf Boutaba, and Joseph L Hellerstein. Dynamic energy-aware capacity provisioning for cloud computing environments. In *Proceedings of the 9th international conference on Autonomic computing*, pages 145–154. ACM, 2012.

[128] Jin Zhu. On the power efficiency and optimal transmission range of wireless sensor nodes. In *Electro/Information Technology, 2009. eit'09. IEEE International Conference on*, pages 277–281. IEEE, 2009.