DYNAMICO: A Reference Model for Governing Control Objectives and Context Relevance in Self-Adaptive Software Systems

Norha M. Villegas[1,4], Gabriel Tamura[2,3,4], Hausi A. Müller[1], Laurence Duchien[2], and Rubby Casallas[3]

[1] University of Victoria, Victoria, Canada
{nvillega,hausi}@cs.uvic.ca
[2] INRIA - LIFL - University of Lille 1, Lille, France
{gabriel.tamura,laurence.duchien}@inria.fr
[3] University of Los Andes, Bogotá, Colombia
rcasalla@uniandes.edu.co
[4] Icesi University, Cali, Colombia

Abstract. Despite the valuable contributions on self-adaptation, most implemented approaches assume adaptation goals and monitoring infrastructures as non-mutable, thus constraining their applicability to systems whose context awareness is restricted to static monitors. Therefore, separation of concerns, dynamic monitoring, and runtime requirements variability are critical for satisfying system goals under highly changing environments. In this chapter we present DYNAMICO, a reference model for engineering adaptive software that helps guaranteeing the coherence of (i) adaptation mechanisms with respect to changes in adaptation goals; and (ii) monitoring mechanisms with respect to changes in both adaptation goals and adaptation mechanisms. DYNAMICO improves the engineering of self-adaptive systems by addressing (i) the management of adaptation properties and goals as control objectives; (ii) the separation of concerns among feedback loops required to address control objectives over time; and (iii) the management of dynamic context as an independent control function to preserve context-awareness in the adaptation mechanism.

1 Introduction

The necessity of a change of perspective in the engineering of software systems has been widely discussed during the last decade by several researchers and practitioners in different software application domains [1,2,3]. In particular, Truex *et al.* posited that software engineering has been based in part on an incorrect set of goals, from the assumption that software systems should support rigid and stable business structures and requirements, have low maintenance, and fully fulfill these requirements from the initial system delivery [4]. In contrast to this static and "stable" vision, they proposed a new set of goals based on permanent analysis, dynamic requirements negotiation and incomplete requirements

R. de Lemos et al. (Eds.): Self-Adaptive Systems, LNCS 7475, pp. 265–293, 2013.

specification. Their proposal is aligned with the vision of self-adaptive systems, where dynamic adaptation is necessary to ensure the continuous satisfaction of their functional requirements while preserving the agreed conditions on Quality of Service (QoS) levels. These QoS levels are usually represented in the form of Service Level Agreements (SLAs), and their enforcement mechanisms are based on contracts and policies, among others [5,6]. To achieve the continuous satisfaction of changing requirements, the development of this kind of systems requires adaptation mechanisms able to perform short-term adaptations on them, and manage their long-term evolution [7]. As part of this adaptation and evolution, system analysis must be performed at runtime, and its requirements satisfaction must be monitored and regulated by continuously adjusting or enhancing its behavior [8,3].

Although the feedback loop model of *control theory* has been used as a reference in many self-adaptive systems in different application domains, the visibility of the feedback loop as the crucial architectural element to govern software adaptation remains often hidden. In many cases, the managed application is intertwined with the adaptation mechanism, rendering it as hard to analyze, reuse, and manipulate [9,8,10]. In other cases, such as those following the multi-layer architectures (e.g., ACRA [11], FORMS [12] and Kramer and Magee's [13]), their designs assume a completely closed and controlled context where monitoring requirements are not subject to change, even though several feedback loops can be evidenced in them. However, for many systems it is not affordable to discard unexpected context changes and dynamic changes in adaptation goals and user requirements, such as SLA re-negotiation at runtime. In these cases, statically deployed context monitoring elements are not enough to cope with these levels of dynamics, which are implied by context unpredictability.

Hence, as context information requirements evolve over time, due not only to changes in the execution environment, but also to the evolution of the adaptive system and its requirements, monitoring infrastructures are also required to be self-adaptive. Furthermore, in these cases the adaptation of the monitoring infrastructure implies to update the context analyzer of the target system's adaptation mechanism. Therefore, these changes must be coordinated by an independent feedback loop, that is, the one that manages changing control objectives and adaptation goals at runtime, thus preserving context-awareness in the system evolution.

In this chapter we present DYNAMICO (Dynamic Adaptive, Monitoring and Control Objectives model), a reference model for engineering context-based self-adaptive software composed of three types of feedback loops. Each of these feedback loops manages each of the three levels of dynamics that we characterize for self-adaptation: (i) the control objectives feedback loop, (ii) the target system adaptation feedback loop, and (iii) the dynamic monitoring feedback loop. As a reference model (i.e., a standard decomposition of a known kind of problems into distinguishable parts, with functionalities and control/data flow that are well defined [14]), DYNAMICO calls self-adaptive system designers to be aware whether the objectives, the system, or the monitoring infrastructure must be

adapted. In this sense, our reference model can be used to check if these dimensions are being considered in the designs. Moreover, it defines the elements and functionalities, as well as the control and data interactions to be implemented, not only among the feedback loop elements, but also among the three types of feedback loops. In addition, our characterization of the latter interactions allows our reference model to be applied partially, that is omitting any of its feedback loops, targeting self-adaptive systems where supporting changes in any of the three levels of dynamics is a crucial requirement.

In light of this, we argue that, in order to regulate the satisfaction of adaptation goals and managed application's requirements continuously, (i) each of the feedback loop elements and their interactions must be independently analyzable; and (ii) the monitoring elements must be able to process the different kinds of information that the varying context can produce appropriately. DYNAMICO was inspired by classical control theory and the autonomic element proposed by IBM researchers [15]. With this reference model we aim to contribute to the design of self-adaptive software by making its instances consider these aspects explicitly: (i) the achievement of adaptation goals and their usage as the reference control objectives; (ii) the separation of control concerns by decoupling the different feedback loops required to satisfy the reference objectives as context changes; and (iii) the specification of context management as an independent control function to preserve the contextual relevance with respect to internal and external context changes.

The remainder of this chapter is organized as follows. Section 2 describes an industrial-based application example that we use to explain our reference model and its application. In Sect. 3 we re-visit fundamental ideas and concepts that have shaped the engineering of self-adaptive software in the last years, and from which we distill our reference model. Section 4 presents our proposed reference model including the feedback loop interactions and their governance, as well as some variations that DYNAMICO admits. Finally, Sect(s). 5 and 6 discuss related work and conclude the chapter, respectively.

2 Application Example

This section presents a SOA governance application example based on an industrial case study we conducted in collaboration with the IBM Centre for Advanced Studies (CAS) Canada.[1] In this case study, self-adaptation mechanisms are exploited at runtime to manage service-level agreements (SLAs), and ensure quality of service (QoS) requirements in service-oriented systems [16]. In SOA and cloud-based systems QoS is highly affected by, and dependent on context information. On the one hand, SLAs may be violated at any time during system execution due to changes in the situation of relevant context entities such as computational infrastructure components (i.e., internal context), and users (i.e., external context). On the other hand, as businesses and users' requirements are evolving continuously, contracted QoS conditions (i.e., adaptation goals) may be

[1] `http://www-927.ibm.com/ibm/cas/canada/research/index.shtml`

frequently re-negotiated, thus affecting the effectiveness of monitoring and adaptation mechanisms. This application example is also based in one of our previous papers on *governance feedback loops* [17], where we applied our reference model to the implementation of a runtime governance infrastructure able to change monitoring strategies dynamically, as required by changes in adaptation goals and the adaptive system itself. The proposed self-adaptive governance infrastructure aims to ensure contracted conditions such as performance, reliability and resource consumption in SOA and cloud-based environments, where SLAs are constantly re-negotiated at runtime [18,17].

Software-as-a-Service (SaaS) is one of the business models in cloud computing environments. SaaS provides customers with several benefits such as maintenance and evolution supported by the cloud provider, high availability, pay-per-use, and low operational costs. Suppose an SaaS cloud provider, specialized in large scale e-commerce platforms, is interested in governing the efficiency of the service-oriented infrastructure with the goal of optimizing operational costs. Assume that to guarantee low operation costs and thus contracted conditions, performance governance has been initially defined as the adaptation goal. For this, a performance SLA defines a service level objective (SLO) to guarantee an efficiency measurement of at least 90% for a particular service (e.g., *ProcessingPurchaseOrder*). The metric associated to the SLO is the *time behavior metric (TB)* proposed by Lee *et al.* [19]. We used this metric as an efficiency measure based on the processing time of service interfaces. Let us assume that initially the *ProcessingPurchaseOrder* is composed only of one interface, thus we express the service efficiency as:

$$TB = \frac{ProcessingPurchaseOrder\ interface\ execution\ time}{total\ ProcessingPurchaseOrder\ service\ invocation\ time} \quad . \qquad (1)$$

The denominator, *total ProcessingPurchaseOrder service invocation time*, represents the total time it takes for the service to respond after the corresponding request. The numerator, *ProcessingPurchaseOrder interface execution time*, indicates the time consumed for processing a given interface functionality. *ProcessingPurchaseOrder* is composed only of one task defined initially as one interface implementation, thus the numerator is the processing time required for executing that individual task (i.e., *total ProcessingPurchaseOrder service invocation time* − *waitingtime*). TB is in the range 0..1, where higher values indicate a better measure of performance in terms of time efficiency. Finally, suppose that an action guarantee, defined as part of the SLA, will trigger a self-optimizing feature that performs an on-line architectural reconfiguration to improve the system's efficiency and capacity.

2.1 The Need for Dynamic Context Monitoring

Using DYNAMICO, runtime SOA governance can be optimized by supporting adaptive monitoring strategies to address changes in monitoring requirements. Variations in monitoring requirements can be generated by changes in either the

governance objectives (i.e, adaptation goals), the target system, the adaptation mechanism, or relevant context entities. The following two use cases illustrate the need for supporting dynamic monitoring, to preserve the context-awareness of the adaptation mechanism upon changes in the target system (i.e., changes in internal context entities) and adaptation goals.

Use Case 1: changes in internal context entities. Suppose that due to self-adaptation, the *ProcessingPurchaseOrder* service is replaced by a set of distributed services intended to enlarge the order processing capacity of the e-commerce platform. Consequently, the efficiency metric presented in (1) must be applied to every new service interface. With traditional static monitoring mechanisms, the governance of the performance SLA is compromised as the monitoring infrastructure was originally implemented to monitor the time efficiency of the *ProcessingPurchaseOrder* service interface only. The monitoring of the new interfaces is not supported without manually implementing the required sensors and monitors. This implies that every time monitoring conditions or the set of context entities to be monitored change, the monitoring instrumentation must be adjusted manually. Moreover, the effectiveness in performing these changes depends on the effectiveness in reporting them. Using our DYNAMICO, our SOA governance infrastructure is able to deal with changes in monitoring requirements at runtime. Once the new services for purchase order processing are deployed, adaptation mechanisms will trigger the adaptation of the monitoring strategy to monitor the new service interfaces. Our monitoring infrastructure exposes autonomous capabilities to configure and deploy new sensors and monitoring conditions at runtime. Implementation details regarding the dynamic capabilities of the implemented monitoring infrastructure are discussed in our MESOCA paper [17].

Use Case 2: changes in adaptation goals. Suppose now that the initial SLA is re-negotiated. A new service level objective (SLO) on throughput is added to the efficiency SLO defined originally as the contracted condition of the performance SLA (cf. (1)). The new throughput SLO defines two different throughput levels, depending on the applicable context situation, as summarized in Table 1 below.

The original monitoring infrastructure is implemented so that the initial performance SLA supports only the monitoring of the individual *ProcessingPurchaseOrder* interface. Once the SLA is re-negotiated, the adaptation mechanism is no longer effective as the new monitoring requirements imposed by the new throughput SLO are not supported. Dynamic changes in the monitoring infrastructure may occur at different levels. They may imply either the deployment of new sensors and new monitoring condition algorithms, or the modification of existing monitoring thresholds and conditions. In any case, without supporting changes in monitoring strategies at runtime, the adaptation mechanisms must be adjusted manually to ensure their relevance with respect to new adaptation goals. In this example, the new throughput SLO detailed in Table 1 will trigger the adaptation of the existing monitoring strategy. Two new sensors and corresponding monitoring conditions must be added. The first one is to monitor

Table 1. The throughput SLO is defined after the performance SLA has been renegotiated. The contracted conditions depend on different context situations that must be monitored.

Throughput SLO of the Performance SLA		
Throughput level	Monitoring Condition	Relevant Context Entities
Medium load	No. of likes on an offer $\leq$ 200,000	A special offer on a social network
Highest peak load	Is Black Friday or Christmas season?	Day of the year

the acceptance of a special offer placed on a social network integrated into the e-commerce platform. The second one is to keep track of the season. In both cases, the monitored information is used to anticipate the expected system load and thus modify the e-commerce platform capacity accordingly.

3 Design Drivers in the Engineering of Self-Adaptive Software

3.1 Feedback Loops

Feedback loops are the cornerstone of control theory, and as such, they provide the basis for automation in many fields of engineering and in particular for self-adaptation in computing and software engineering [20]. In this theory, the feedback loop or closed loop, as depicted in Fig. 1, is the *model* used to automate the control of dynamic systems. These control mechanisms are realized by comparing the *measured outputs* (A) of the target system behavior to the control objectives given as *reference inputs* (B), yielding the *control error* (C), and then adjusting the *controlling inputs* (D) accordingly for the target system to behave as defined by the reference input [9]. The measured output can also be affected by external *disturbances* (E), or even by the *noise* (F) caused by the system adaptation itself. *Transducers* (G) translate the signals coming from sensors, as required by the comparison element (H).

To keep objectives controlled in a target system, several strategies have been proposed. The three most common strategies are (i) the *regulatory* control, which ensures that the measured output is as close as possible to the reference input; (ii) the *disturbance rejection*, to control the effects of disturbances on the measured output; and (iii) the *optimization* control, which continuously seeks to obtain the best value of the measured output, as effectively as possible [9]. These strategies imply variations on the controller element but are realizable with the general structure of the block diagram. To compute the controlling signals, there are several possible mechanisms. In control theory, the representative mechanism is the *system transfer function,* a mathematical model built upon the physical properties and characteristics of the target system. Depending on these characteristics, the transfer function can be built, for instance, with proportional,

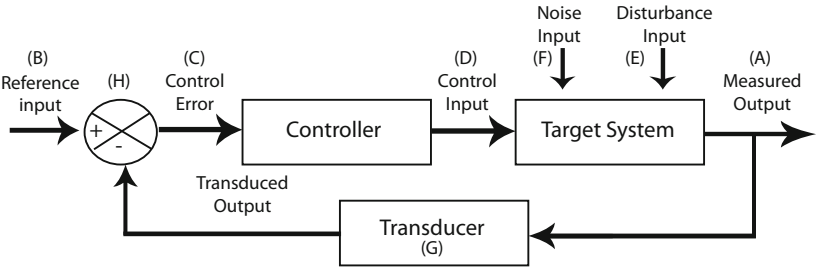

Fig. 1. Classical block diagram of a feedback control system [9]

derivative and integral (PID) terms. The parameters in a PID controller have special significance given that there exist precise and sophisticated methods for tuning their associated parameters.

Even though the application of control theory to industrial processes is well understood, its application to the control of software systems has at least two significant challenges: first, control theory is based on continuous mathematics, and second, it relies on measurements taken from, and actions performed into, physical, self-contained and self-performing artifacts (e.g., sensors, gauges and valves/actuators for temperature, pressure and other variables). As their associated variables are in the continuous-time domain, the use of continuous mathematics in this theory fits perfectly. In contrast, software systems are composed of intangible artifacts with discrete-time behavior and not always well characterized properties. Thus, direct sensing must be performed by CPU time-consuming software artifacts, and the adaptation mechanisms must reason on the target system's discrete-time output. Moreover, to exploit the possibilities of software adaptation fully, the output of the adaptation mechanism must be more structured than controlling signals to be transduced by electro-mechanical devices. This output may take the form, for example, of a plan of ordered actions to be instrumented by the software actuators on the target software components. Fortunately, there exists also the theory of linear discrete-time systems, which closely resembles the theory of linear continuous-time systems.

A reference model should not prescribe any particular software self-adaptation type of control. Instead, we propose DYNAMICO to characterize experimentally the effect that the controller actions produce in the observed behavior of the target system.

3.2 Visibility of Feedback Loops

The benefits of integrating feedback loop-based models into the engineering of self-adaptive software systems have been pointed out by several research papers [7,20,8,3]. Oreizy *et al.* define runtime adaptation in the form of two processes that exploit feedback loops to manage adaptation and system evolution, respectively. The evolution management process feedback loop is in charge of monitoring the consistency between architectural models and the actual system

implementation. Whenever this consistency is no longer satisfied, the evolution management process feeds monitored information back to the adaptation management process feedback loop, which is in charge of reconfiguring the system's architecture [7]. Müller *et al.* outline on the benefits of specifying the feedback loops and their major components explicitly and independently. Furthermore, they articulate the usefulness of defining the interactions among the elements of a feedback loop explicitly, from analysis and design to implementation [20]. Giese *et al.* also argue for the decoupling of feedback loops in control-based reference architectures to address the satisfaction of quality attributes (control objectives), the management of the context complexity, and the interactions among multiple feedback loops and their elements [8]. Cheng *et al.* also emphasize the importance of making explicit not only the feedback loops, but also their elements and properties [3]. In fact, Müller *et al.* [20], as well as Kramer and Magee [13] attest that even though feedback loops have been recognized as fundamental design elements for self-adaptation, the related design documents and research publications usually hide the visibility of both the adaptation controller and the feedback loops. As a result, there currently exists no explicit methods for analysis, validation and verification useful to measure the effectiveness of adaptation mechanisms in software systems [21]. Based on these remarks, we aim in our reference model to increase the visibility of the feedback loop components by making them explicit entities of software architecture design and, thus, directly analyzable, assessable and comparable.

Valuable papers have been published making significant advances in the area. For instance, the feedback control architecture for adaptive systems proposed by Shaw decouples the elements of a feedback loop (i.e., comparison, plan correction, and effect correction), and identifies the importance of context relevance for the adaptation process [22,20]. In the same way, the autonomic manager (MAPE-K loop) presented in Fig. 2, and the autonomic computing reference architecture (ACRA) are important contributions of IBM that also make the feedback loops in autonomic systems explicit [11]. On the one hand, as explained

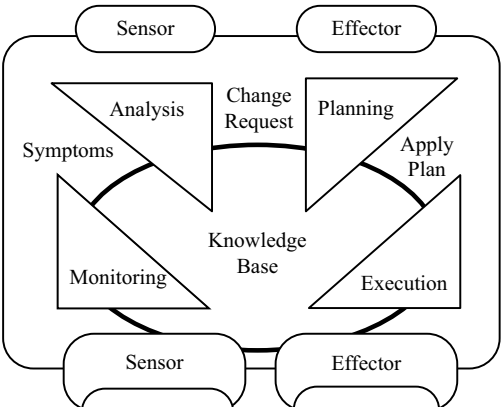

Fig. 2. The MAPE-K loop [15]

in [10], the autonomic manager is an implementation of the controller element in the generic control feedback loop depicted in Fig. 1. At the same time, the autonomic manager controls the managed element by implementing an intelligent control loop composed of the monitor, the analyzer, the planner, the executor, and the knowledge base elements. This knowledge base is an important element to share information along the loop. Moreover, it provides persistence for historical information and policies required to correlate complex situations. On the other hand, ACRA provides a reference architecture as a guide to organize and orchestrate an autonomic system. Autonomic systems based on ACRA are defined as a set of hierarchically structured building blocks composed of autonomic managers, knowledge sources and manageability endpoints (management interfaces).

Nonetheless, despite ACRA and the MAPE-K loop that have helped considerably improve the visibility of feedback loops, the internal components of each control loop, and the control loop itself, still remain hidden inside the autonomic manager. Certainly, the specification of the autonomic manager, provided in the IBM architectural blueprint for autonomic computing, characterizes the manager as *a component that implements an intelligent control loop* [11]. Moreover, even when the ACRA architecture drivers are clearly the feedback loops in the form of autonomic managers, their internal elements (i.e., the elements of the MAPE-K loop) are highly coupled. Therefore, even though the multiple feedback loops defined in an ACRA-based model can be distributed—for instance to improve the system scalability—this distribution is limited by the autonomic manager boundaries. Each autonomic manager implements the entire cycle to collect and aggregate information from the environment (monitor), to correlate the collected information and identify symptoms for supporting the adaptation decision making (analyzer), to plan the adaptation process (planner), and to perform the adaptation plan (executor).

The separation of concerns between the monitoring process, the adaptation controller, and the management of control objectives (adaptation goals) is still an open challenge. This challenge is crucial for governing the consistency between adaptation mechanisms and control objectives, while preserving the relevance of context monitoring of the adaptation mechanism. In light of this, we concluded that a loose-coupling schema is preferable to a tight-coupling one for the integration and communication among the feedback loop elements. However, we retain the idea of composing instances of feedback loops similarly as specified by the generic hierarchical structure described in ACRA. Finally, while the autonomic manager, as an implementation of the feedback loop, is the architecture driver for ACRA, our architecture drivers are the independent MAPE-K loop elements, their explicit interactions, and the separation of these elements in three main groups, as explained in the following section.

3.3 The Three Levels of Dynamics

We identify three levels of dynamics that must be controlled in the engineering of context-driven self-adaptive software systems: (i) the management of

changing control objectives, (ii) the dynamic behavior of the adaptation mechanism controlling the target system, and (iii) the management of dynamic context information. Each of these levels of dynamics plays an important role in governing the dynamic nature of the other two levels. In the case of the first level, as business goals and corresponding control objectives that must drive the behavior of the adaptive system evolve continuously, context monitoring mechanisms (the third level of dynamics), and adaptation controllers (the second level) are required to change accordingly. Furthermore, the management of control objectives may be affected as a result of monitored observations at the third level of dynamics. For instance, whenever the system identifies that even though the adaptation mechanism is performing properly, control objectives may be reviewed to modify the adaptation and/or monitoring mechanism due to changes in context situations.

These three levels of dynamics may be clearly illustrated using the application example described in Sect. 2. The first level, the management of changing control objectives, corresponds to the software instrumentation required to identify changes in adaptation goals. In our example, these changes correspond to the re-negotiation of the performance SLA by adding a new throughput SLO to the initial efficiency SLO. The second level, the dynamic behavior of the adaptation mechanisms, refers to the capability of adaptation strategies to adapt according to changes in either adaptation goals, or context situations. In the application example used as illustration in this chapter, the adaptation mechanism does not expose dynamic behavior. That is, the adaptation strategy is always the same. The third level, the management of dynamic context information, refers to the instrumentation required to support changes in monitoring strategies at runtime. In the application example, the dynamic reconfiguration of the monitoring strategy is triggered by two different situations. In the first case, new sensors are deployed at runtime to monitor the new service interfaces that have been added with the new set of distributed services for processing purchase orders (cf. Sect. 2.1, Use Case 1). In the second case, new sensors and monitoring conditions are deployed dynamically due to changes in adaptation goals (i.e., the new throughput SLA). The negotiation of a new throughput SLO requires from the monitoring infrastructure to keep track of two new context entities, a special offer placed on a social network and the day of the year (cf. Table 1).

4 DYNAMICO: Our Reference Model

Bass *et al.* define a reference model in software engineering as a standard decomposition of a known kind of problems into clearly distinguishable parts [14]. Each of these parts has assigned a well defined functionality, and the data flow among these parts is explicitly specified. Reference models serve as starting points for software architecture and high-level design specifications.

Following this definition and based on the analysis of the seminal research presented in Sect. 3, we distill in our reference model the characteristics that have been commonly discussed and used in other representative research in the

engineering of self-adaptive software. We started by considering the general feedback control loop block diagram presented in Fig. 1. In this diagram, the target system to be controlled, its controller and corresponding transducers are represented as rectangles. The elements for setting the reference input (set point) and perform the comparison against the system measurements are combined in a crossed circle. This block diagram reflects the relative simplicity of the "autonomous" but independent elements used in control engineering. This simplicity hides the very specific and natural electro-mechanical properties (e.g., resistance, capacitance, inductance) of these elements. In contrast, in the MAPE-K model the elements are interdependent and their functions are specified in a general way. Concerning the characteristics of the different control strategies, control theory takes advantage of exactly the particular complex properties of the matter that constitutes both the controller elements, as well as the system to be controlled. In the case of software artifacts, even though they lack the physical properties analyzed in control engineering, these artifacts are given particular properties of behavior by their particular design. Nonetheless, and because of this, it is practically impossible to generalize them.

Therefore, given the characteristics of software systems (i.e., the systems to be controlled), we find that the combination of a general specification for the common elements of both feedback-loops and MAPE-loops together with a loose coupling scheme, are the best options for DYNAMICO. Figure 3 captures these decisions, which represents the general component of our reference model. This diagram clearly results from the merging of the classical feedback-loop and the MAPE loop model (cf. Fig. 1 and Fig. 2 respectively).

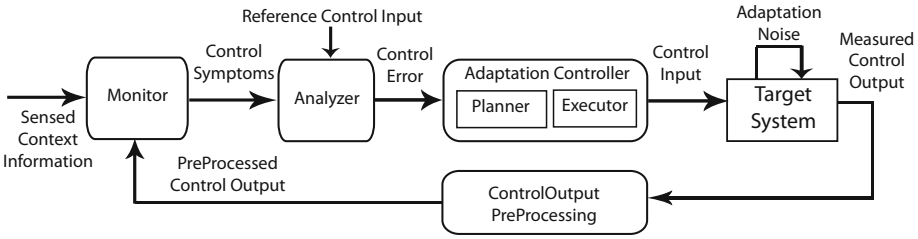

Fig. 3. General components of DYNAMICO. Feedback control block diagram with explicit functional elements and corresponding interactions to control dynamic adaptation in a software system.

4.1 Addressing Separation of Concerns

Analyzing Fig. 3 from both the control theory and software architecture perspective, for a software system (*target system*) to become effectively context-driven self-adaptive, it should incorporate at least three subsystems: (i) a *control objectives manager*, (ii) an *adaptation controller mechanism*, and (iii) a *context manager* or *monitoring infrastructure controller mechanism*. This design separates the concerns with respect to the three levels of dynamics we have proposed

as design drivers for the engineering of context-driven self-adaptive systems: (a) the regulation of the target system's functional and non-functional requirements satisfaction; (b) the continuous accomplishment of adaptation goals and the preservation of the target system's properties under changing conditions of execution; and (c) the relevance of the context monitoring infrastructure according to the varying execution environment (dynamic context monitoring). This separation of concerns leads us to abstract the block diagram presented in Fig. 3 into the block diagram presented in Fig. 4. In this diagram, which constitutes our reference model as such, each of the three feedback loops, the *control objectives feedback loop* (CO-FL), the *adaptation feedback loop* (A-FL), and the *monitoring feedback loop* (M-FL), is an instance of the model depicted in Fig. 3.

The identification of these subsystems as independent feedback loops allows us to independently analyze, design, implement, and assess the instrumentation required to address the complexity of changing requirements at each of the three levels of dynamics. In this way, and depending on the nature of the adaptive system, this instrumentation can be easily temporal and spatial distributed and maintained. In addition, the entire software system would be less affected by

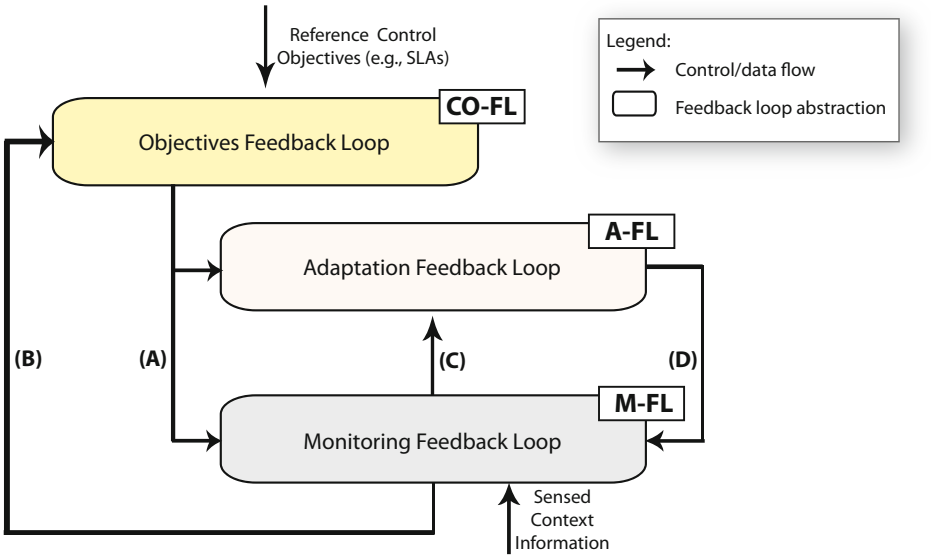

Fig. 4. The three levels of dynamics that must be controlled in context-driven self-adaptive software systems. The control objectives feedback loop, (CO-FL), controls changes in adaptation goals and monitoring requirements to ensure their fulfillment. The adaptation feedback loop, (A-FL), controls the adaptive behavior of the target system and the adaptation mechanism, according to control objectives and taking into account monitored context events. The dynamic monitoring feedback loop, (M-FL), manages context information for preserving context relevance of the adaptation mechanism. Labels (A), (B), (C) and (D) highlight the control/data flow among the feedback loops, which would require the implementation of the appropriate method interfaces.

the computational effort of each of the three subsystems. The separation of concerns made explicit by the DYNAMICO model is particularly crucial for cases such as the cloud-based e-commerce platform presented in our application example. In this example, the automatic reconfiguration of the monitoring strategy would not be feasible without having the context manager as an independent implementation of the adaptation mechanism. In the same way, the explicit control of changes in SLAs requires separate instrumentation. In the case where a dynamic adaptation mechanism is necessary, having a self-contained adaptation strategy (i.e., planner and executor) will contribute to the preservation of desired properties. The chapter "On Patterns for Decentralized Control in Self-Adaptive Systems", by Weyns *et al.* in this book, presents useful architectural patterns, that can be combined with our reference model, for implementing distributed and decentralized feedback loops in self-adaptive software systems.

By applying the separation of concerns introduced in our reference model, it is possible to support three different types of adaptation, depending on the different interactions implemented among the feedback loops: *preventive, corrective* and *predictive*. In preventive adaptation, the dynamic monitoring feedback loop notifies the adaptation feedback loop about context events (*context symptoms*) that, even when they are causing no effects yet in the target system behavior, they eventually will. This is the case of the monitoring condition that evaluates the number of *likes* of an offer placed on a social network. As the offer becomes popular, that is, the number of likes is close to 200,000, a predictive adaptation process can be started to take the system to its medium-load capacity (cf. Table 1 in Sect. 2). Consequently, even though the adaptation subsystem has not detected any disturbances yet for triggering adaptation in the target system, based on this context information, it can minimize the risks of the goal satisfaction to be violated by performing a system adaptation in advance.

Corrective is the usual type of adaptation that takes place when monitoring mechanisms supporting the adaptation feedback loop detect adaptation goals are no longer satisfied. In our application example this can occur when the monitoring feedback loop identifies an SLO violation in either the efficiency of the *ProcessingPurchaseOrder* service(s), or the expected minimum number of purchase orders processed per unit of time (cf. Table 1). Any of these situations requires from the adaptation controller to perform another, perhaps more aggressive, system reconfiguration, or to apply restrictive mechanisms of use to prevent the system from collapsing before a new adaptation is performed.

Predictive adaptation takes advantage of both, historical information to anticipate risks of goal violation, as well as the identification of plausible symptoms that provide evidence to necessitate adaptation eventually. These symptoms may be presented in the form of patterns of correlated events that potentially become significant advice for adaptation. An example of this latter case in our application scenario is the detection of a low but constant degradation of the *ProcessingPurchaseOrder* service efficiency around significant dates, but without reaching the critical levels that trigger corrective adaptation. Using this historical information, the dynamic monitoring feedback loop can trigger an alert event to indicate

or notify the operators that the negotiated performance SLA should be reviewed to keep the system operation in a safe state.

Finally, it is worth noting that in Fig. 4 despite this separation of concerns, the control objectives feedback loop (i.e., CO-FL in the figure), the adaptation feedback loop (i.e., A-FL, including the target system), and the dynamic monitoring feedback loop (i.e., M-FL) together, also constitute a feedback loop. Figure 5 presents the detailed view of the reference model where each level of dynamics is designed as an instance of the general feedback loop with explicit components required for controlling the self-adaptation in software systems.

4.2 The Control Objectives Feedback Loop (CO-FL)

In DYNAMICO, the regulation of requirements satisfaction and the preservation of adaptation properties are objectives controlled through the collaboration of the A-FL and the M-FL. We define requirements and adaptation properties as system variables to be controlled. Throughout the chapter, we refer to these variables as *control objectives* and *adaptation goals* interchangeably. These requirements can be functional and non-functional, and the target system must satisfy them, depending for this on the adaptive capabilities of the overall system. Adaptation properties refer to the properties that are inherent in self-adaptive software, and thus, all adaptation mechanisms should expose these properties [21]. As mentioned in Sect. 3.3, these control objectives are subject to change by user-level (re)negotiations at runtime and therefore must be addressed in a consistent and synchronized way by the adaptation mechanism and the context manager. There may be several causes for these changes. In a first case, service level agreements with dependencies on context situations can imply changes in control objectives at runtime. In our application example, this is the case of the throughput SLO (cf. Table 1). This SLO defines two different thresholds. The medium load threshold is applicable to those cases where special product offers are placed online (e.g., on a social network integrated to the business e-commerce platform). After placing the offer, it must be monitored to apply preventive adaptation with the goal of adjusting the system capacity according to the popularity of the offer. Popular offers are expected to affect the e-commerce platform load considerably. Similarly, time context must be monitored to keep track of the shopping seasons to apply preventive adaptation to guarantee the system operation when the system load reaches its highest point (cf. Table 1). In another case, when the system is in execution, the initial SLA conditions can be re-negotiated. An instance of this case occurs in the second use case of our application example. After the contracted services for the e-commerce platform have been in production, a new throughput SLA is added to the efficiency SLO agreed initially. Both the throughput and efficiency SLOs are managed explicitly as the control objectives for the adaptive system. Thus, both reference inputs, the A-FL reference control input, and the M-FL reference context input, should be derived automatically from changes in control

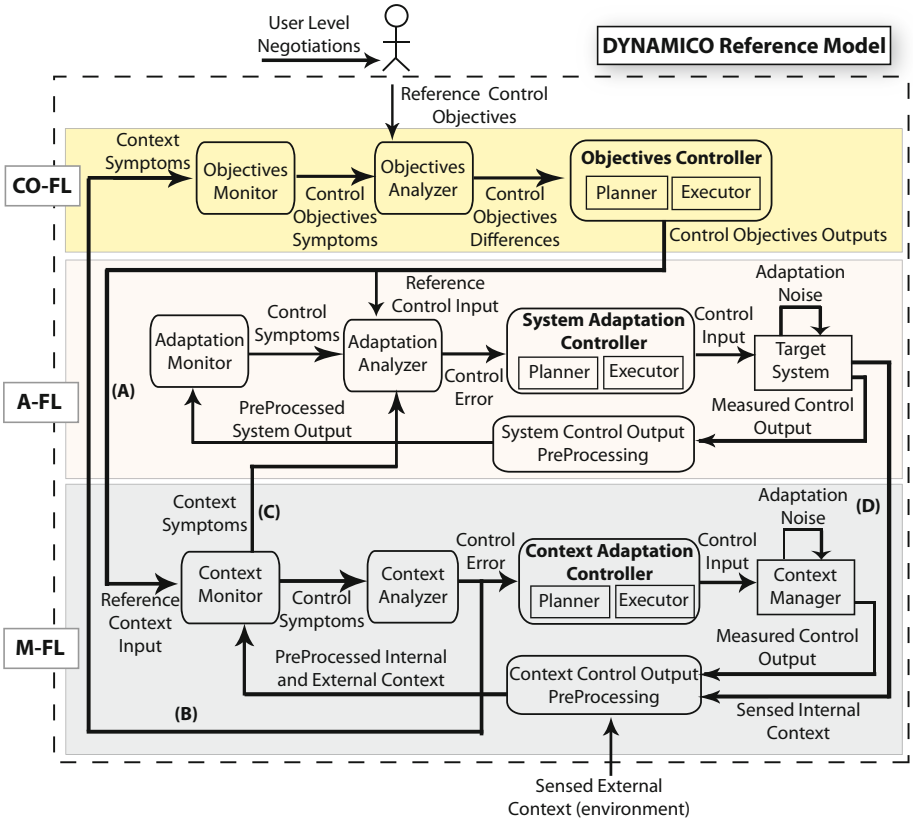

Fig. 5. Our DYNAMICO reference model with a detailed view of the controllers for the three levels of dynamics presented in Fig. 4 realized as the control objectives feedback loop (CO-FL), the adaptation feedback loop (A-FL), and the monitoring feedback loop (M-FL), respectively

objectives and fed into the corresponding feedback loops, as illustrated by interaction (A) in Fig. 5. All of these changes between SLOs and SLAs, which are treated as changes in reference inputs, are governed by the CO-FL.

Nonetheless, this explicit management of control reference inputs has two important implications: (i) it is required to model and express the corresponding properties quantitatively in terms of quality attributes, and (ii) it is necessary to have a mechanism to measure and update these reference inputs at runtime whenever they change. Concerning these two implications, we proposed a comprehensive evaluation framework composed of a set of adaptation properties and adaptation goals, and corresponding quality attributes [21]. This catalog is useful for the assessment of self-adaptive software based on the accomplishment of control objectives and suitability of adaptation mechanisms. Concerning the second implication, the dynamic adaptation of control reference inputs (control objectives) is addressed by *closing* the CO-FL. In the context of the main loop

(the one composed of the three feedback loops), the A-FL receives symptoms from the M-FL through interaction (C), which in turn adapts its behavior according to changes in control objectives to guarantee monitoring relevance along the adaptation process. Under more dynamic scenarios, the A-FL controller may be required also to change its adaptation strategy according to changes in control objectives. Furthermore, as an important concern in service provision is the fulfillment of SLAs as specified in contracts, a plausible way to express and manage these reference goals quantitatively is through *contract management* and its explicit modeling.

4.3 The Adaptation Feedback Loop (A-FL)

The adaptation feedback loop, A-FL, serves as a guarantor for regulating the target system's requirements satisfaction and preserving the adaptation properties. Recalling our application example, the efficiency and throughput SLOs represent system's requirements. Due to the changing nature of SLAs and context situations, the satisfaction of these requirements depends on the adaptive capabilities of the e-commerce platform. Among the adaptation properties applicable to the adaptation mechanism of the application example are settling time, small overshoot, stability, and reconfiguration termination. In particular, settling time, the time it takes for the adaptation mechanism to complete the e-commerce platform reconfiguration, is crucial to guarantee the contracted conditions. Our SEAMS 2011 paper provides a comprehensive catalog of adaptation properties and corresponding quality attributes and metrics [21].

A-FL follows the separation of concerns criteria of the previous section. In turn, these criteria conform to the general protocol of control theory, which relies on quantitative expressions to measure the error in the controlled system variables, and respective reference control inputs for these variables. The A-FL gathers these measurements continuously from the target system through context monitors. These monitors notify control symptoms for adaptation to the A-FL analyzer, which determines whether a system adaptation is required (cf. analyzer in Fig. 3). The simplest case for this occurs when the measured variables under control, compared to their corresponding reference control inputs, indicate that some control objective is no longer satisfied. Whenever it is relevant, the A-FL analyzer notifies this fact with the corresponding information to the system adaptation controller. With this information, the planner element selects a strategy to adapt the system for it to re-establish the fulfillment of the violated control objective. A possible result of this strategy is to compute and send a list of system architecture reconfiguration actions to the executor (e.g., a set of distributed services to replace the original *ProcessingPurchaseOrder* service). The executor translates these actions to the specific runtime platform and executes them in the target system, thus closing the main control loop. DYNAMICO and its A-FL can take advantage of any strategy to perform the target system adaptation.

4.4 The Monitoring Feedback Loop (M-FL)

The role of the monitoring feedback loop, M-FL, as an independent feedback control loop is crucial for addressing the dynamic nature of context information. In a *context-based self-adaptive system*, a context manager must be able to make decisions based on past, current and foreseeable future states of context. It must analyze context symptoms and facts to support the system adaptation and the management of control objectives, as explained in Sect. 4.2. Moreover, the monitoring mechanism must adapt itself to support new context management requirements as the common control objectives are re-negotiated, or the adaptive system evolves. For instance, the context manager for the application example must be able to deploy new context management instrumentation. In the first use case, the deployment of the new set of distributed services, caused by the adaptation of the e-commerce platform, will trigger the deployment of a new set of time behavior sensors to keep track of the new service interfaces (cf. Sect. 2.1, Use Case 1). In the second use case, the re-negotiation of the performance SLA (cf. Sect. 2.1, Use Case 2) will trigger the deployment of the monitoring infrastructure required to keep track of two new types of context information, the shopping season (i.e., time context according to our *Smarter-Context* taxonomy [23]), and the special product offer (i.e., artificial context).

The M-FL in Fig. 5 represents a context manager that supports dynamic monitoring. The reference context inputs correspond to the reference context management objectives derived from the CO-FL reference control objectives. Context monitors are in charge of gathering primary context information from the internal and external environment, and the correlation of this information to infer either, context symptoms that can affect the target system adaptation process (provided to the A-FL through interaction (C) in Fig. 5), or control symptoms to decide about the context manager adaptation. This information is pre-processed by the context control output preprocessing element to generate numeric observables from physical and logical sensors, and producing comparable measures by performing basic transformations on them.

The context analyzer performs the context handling process required for the context adaptation controller to decide about adapting the monitoring strategy, and for the CO-FL to decide about changing the system objectives (interaction (B)), as demanded by the current state of the environment and the self-adaptive system requirements. The change of control objectives can be performed fully- or semi-automatically, depending on whether it is necessary to re-negotiate the contracts, and consequently, for the user to intervene (cf. Sect. 4.2). The context adaptation controller is responsible for defining and executing the adaptation plan for the context manager, according to its adaptation strategy.

Finally, the measured control output and the target system's internal context are used to ensure the context manager goals, thus supporting the system adaptation process and the management of the system control objectives.

To explicitly manage the relationship between control objectives and monitoring requirements in our case study, we proposed context-driven SLAs [17]. A *context-driven SLA* is an extension of a traditional SLA where context

requirements are explicitly mapped to SLOs. In this way, changes in SLOs generated at runtime will include changes in the context management strategy specified with the original SLA. Context-driven SLAs are implemented as *contextual RDF graphs* based on the *SmarterContext ontology*. Both contextual RDF graphs and SmarterContext are results of our research on dynamic context management for context-aware self-adaptive software systems [23,17].

From the reference context inputs stated with the SLA, it is possible to generate context models that represent the environmental information relevant for the adaptation process. In our application example, context models are RDF graphs that represent a composition of relevant context entities, context sensors, and monitoring conditions. Whenever new SLAs are defined or existing ones are re-negotiated, the RDF representation of the monitoring strategy for the corresponding SLOs must be updated accordingly. The contextual RDF graph representing the new monitoring requirements is processed by our M-FL analyzer. Then, the planner element of the M-FL generates the adaptation plan that will modify the monitoring strategy by deploying new, or modifying existing sensors and monitoring conditions. The generation of these context adaptation plans at runtime is based on semantic Web inference rules defined as part of our SmarterContext ontology. Further details on the instrumentation of dynamic monitoring strategies for our case study are available in [17].

4.5 Feedback Loop Interactions

In DYNAMICO, not only are the three described feedback control loops well separated, but also the elements within each feedback loop. However, even though control loops are designed independently of each other, they must operate cooperatively to achieve the overall system objectives.

As depicted in Figs. 4 and 5, to regulate the satisfaction of the control objectives, DYNAMICO specifies four interactions among its three feedback loops. These interactions are labeled (A), (B), (C) and (D) in Fig. 5. We classify interactions (A) and (B) as *indirect interactions* because they are realized through the CO-FL, whereas interactions (C) and (D) as *direct interactions* due to their direct connections between the M-FL and the A-FL.

Interaction (A) provides the reference context input (i.e., context manager requirements) for the context manager (M-FL) to (i) maintain its relevance with respect to the actual context situation and contracted conditions; and (ii) decide on context management strategies. In the application example, reference context inputs correspond to the context management requirements defined as part of the SLA in the form of contextual RDF graphs [17].

Interaction (B) enables the control objectives manager (CO-FL) to decide about the changes in the control objectives, whenever the M-FL detects that, given the current context, the current set of control objectives should be adjusted or re-negotiated dynamically. Common control objectives are crucial for governing the interactions between the A-FL and the M-FL. We specify common control objectives in the form of contracts, machine readable SLAs as contextual RDF-graphs to infer both, adaptation and context monitoring objectives

[24,6,17]. Thus, a context management infrastructure (i.e., M-FL) must be able to infer, from contracts and common control objectives, the context management reference inputs, as well as the required monitoring strategies.

Interaction (C) is triggered by context symptoms that are identified and sent from the M-FL context monitor to the A-FL analyzer. These context symptoms, which can be manifested as groups of events presented with different characteristics, are important for decision making in the A-FL. The communication mechanism and the information associated with these symptoms depend on the type of adaptation the system is supporting (i.e., preventive, corrective or predictive). For example, for a predictive adaptation, the M-FL could trigger symptomatic events in advance about whether or not to perform a future adaptation. For a preventive adaptation, the M-FL also sends symptoms, but the adaptation is performed immediately. In contrast, for corrective adaptation, symptoms are either, pushed by the M-FL or pulled by the A-FL depending on who recognizes the need for adaptation —the context manager or the adaptation controller.

Interaction (D) represents the flow of internal context sensed by the M-FL from the adaptive system. Monitoring of internal context information is necessary to assess the system consistency after an adaptation. Moreover, by analyzing internal context information that characterizes the current state of system properties, the M-FL could provide useful information to understand the relationship between context symptoms, achievement of system goals, and the preservation of adaptation properties [21].

4.6 Governing and Controlling Feedback Loop Interactions

According to our reference model, an adaptive system is defined as a collection of cooperating feedback loops that ensure the achievement of the system objectives under changing context conditions. However, DYNAMICO can be combined with other models for adaptive systems. In particular, the IBM architectural blueprint provides the ACRA model to orchestrate control loops hierarchically for autonomic systems [11,15]. Combined with this model, DYNAMICO supports the distribution of functions in a more fine-grained level, that is, at the feedback-loop elements level. More extensive use of knowledge bases, as the ones proposed for the MAPE-K loop, should also facilitate interactions among control loops. Such knowledge bases store historical information such as symptoms, as well as internal and external context facts required by the analyzers in any of the three types of control loop. Moreover, these persistence mechanisms help to fine-tune contracts and policies to achieve the control objectives, and to develop machine-learning based adaptation mechanisms [25]. It is worth noting that having common control objectives enable the three control loops to reason consistently about the system goals, and to determine the coordinated control actions on each of them.

Having common control objectives is important to govern the interactions among the feedback loops. Figure 6 illustrates DYNAMICO abstracted as a control objectives feedback loop. The A-FL (adaptation mechanism), the M-FL (context manager), and the core controlled target system are abstracted

as a whole managed *(super target) system*. This managed system is governed by the CO-FL according to changes in contracted conditions. Reference control objectives (i.e., contracts) are fed into the system through direct user intervention. Changes in these objectives can result from re-negotiations or from context symptoms received through interaction (B) (cf. Fig. 4). According to Fig. 6, whenever the objectives change as a result of symptoms received from the context manager, the control objectives monitor perceives these symptoms as symptoms of changes in the current set of control objectives. Then, the CO-FL analyzer makes decisions on the necessity of producing a new set of reference control inputs. If applicable, the CO-FL controller produces a new set of reference control inputs and reference context inputs to be sent to the adaptation mechanism and the context manager respectively. The measured control objectives feed the system back with information about the achievement of the system control goals. Finally, if the control objectives change as a result of a re-negotiation, the user is responsible for providing the control objectives analyzer with the new SLAs, and their corresponding SLOs and context monitoring requirements.

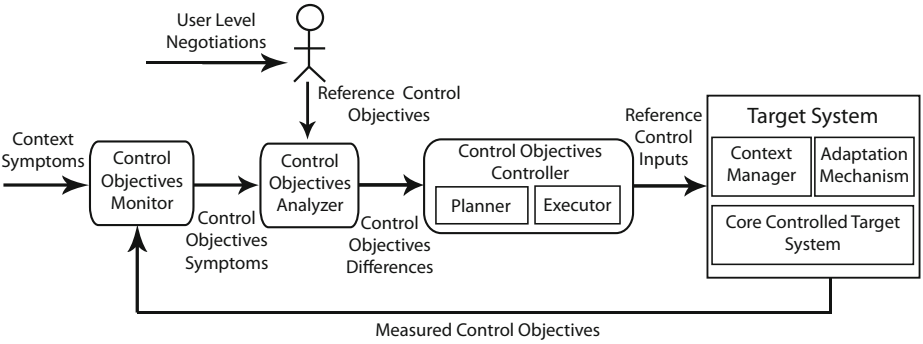

Fig. 6. DYNAMICO abstracted as a feedback loop for governing the dynamic change of the system control objectives

4.7 Possible DYNAMICO Variations

To deal with out-of-kilter environmental behaviors or perturbations, control community has developed several variations to modify the control function, such as the Model Reference Adaptive Control (MRAC) and the Model Identification Adaptive Control (MIAC) mechanisms [26,27]. The main difference between MRAC and MIAC is how the reference model is defined—in MIAC directly inferred from the running process, whereas in MRAC pre-computed using a mathematical model.

These variations are also applicable to DYNAMICO. Both variations can be realized, for instance, using a rule-based or policy-based reconfiguration approach in the planner element of the system adaptation controller, as illustrated in Fig. 7. In this figure, the CO-FL is represented by the control objectives manager. The adjustment mechanism detects, through the measured control output,

whether the target system is facing an out-of-kilter environmental perturbation (e.g., an unusual high number of on-line shoppers during the Black Friday season), or the adaptation strategy is far from being effective. If this is the case, it modifies either the system adaptation planner (in the adaptation mechanism), or the context adaptation planner (in the context manager), depending on the situation. In our application example, the adaptation planner can be adjusted by replacing the reconfiguration rules in the rule-based subsystem using the *controller parameters*. Similarly, the context manager's planner could be modified by replacing the semantic Web rules, defined as part of the SmarterContext ontology, to be used to infer changes in monitoring strategies.

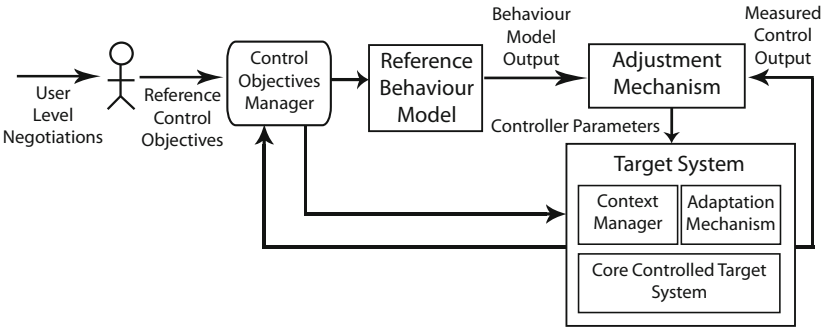

Fig. 7. Reference model variation for supporting adaptive feedback control loops with reference behavior models. The control objectives manager feedback loop is abstracted.

5 Discussion of Related Work

Different research communities, related to dynamic software systems, have proposed several examples of the application of feedback loops to concrete implementations of this type of systems. However, even though in most cases the existence of the feedback loop is evident, their designs lack separation of concerns among the multiple feedback loops required to orchestrate the three levels of dynamics introduced by our reference model (i.e., CO-FL, A-FL, and M-FL). Moreover, the explicit treatment of the interactions among these three levels is not generally addressed by existing implementations. Our reference model is general enough for being applied to different context-driven adaptive systems in many different application domains, where supporting changes in the three levels of self-adaptation dynamics is a crucial requirement.

In this section we discuss how DYNAMICO can be used to optimize context relevance in existing implementations of self-adaptive approaches, as well as the way different models for self-adaptation address the key drivers addressed by our reference model.

5.1 Optimizing Existing Implementations

A first example of concrete implementations is Rainbow, the adaptive framework for implementing self-healing software systems developed by Garlan *et al.* [28]. Rainbow's architecture maps directly to the feedback control architecture proposed by Shaw [22,20]. Our contribution complements Garlan's and Shaw's approaches by making explicit not only the feedback loops, but also their internal components, the interactions among them, as well as the separation of concerns at the three levels of dynamics proposed by our reference model.

A second interesting instance from a different application domain is the context-aware dynamic software product line proposed by Parra *et al.* [29]. They proposed the introduction of context-aware assets that are dynamically incorporated into the product line, depending on context changes. Although their architecture identifies the main feedback loop elements—a context manager (monitor), a decision maker (analyzer and planner), a runtime platform (executor) and a knowledge base—DYNAMICO can be used to improve their architecture by introducing a context monitoring infrastructure governed by an independent feedback loop, and coordinating the respective feedback loop interactions.

Yet another instance from the autonomic computing community is the real-time adaptive control approach for autonomic computing environments proposed by Solomon *et al.* [30]. Their system aims to control the computing infrastructure through a mathematical description of the time variation on the number of users in the system. Based on this function, the system modifies the control structure of the autonomic computing infrastructure by replacing its controller with one that matches the variation of the number of users on given time intervals. Furthermore, their adaptive control is based on a multi-layer architecture similar to ACRA, where the two upper layers correspond, respectively, to the autonomic system adaptation and the autonomic system layers, and the lowest layer corresponds to the managed infrastructure. The autonomic system adaptation layer adapts the autonomic system layer whenever the management objectives are not achieved. In this particular case, DYNAMICO is valuable for addressing the separation of concerns within the adaptation and autonomic management layers, as well as to guarantee the contextual relevance of monitoring mechanisms according to changes in the management objectives.

In the self-organizing systems community, Caprarescu and Petcu proposed a decentralized autonomic manager composed of many independent lightweight feedback loops implemented as agents, where each agent is an implementation of a MAPE-K loop [31]. Control objectives in this approach are specified as policies. Moreover, each feedback loop agent uses just one policy that is shared among all the agents organized in the same group. At the architectural level, this approach is based on the three-layer model proposed by Kramer and Magee [13], which was in turn inspired by the three-layer architectures proposed by the artificial intelligence and robotics community [32]. The system performs its adaptation based on a process of three phases. The first one separates agents into groups according to policies (i.e., self-organization phase); the second one ensures that

only one agent can execute changes at a specific time (i.e., management phase); and the third one keeps the policies of the feedback loop up to date (i.e., policy update phase). Feedback loops adapt the system by modifying their parameters, adding new components or reconnecting components. The application of our reference model to this self-organizing system would help tackle the high degree of coupling among the components of each feedback loop, thus making the system components replaceable, reusable and distributable. An instance of the application of our reference model to this particular domain is the self-healing distributed scheduling platform presented by Frîncu *et al.* [33].

5.2 Comparing DYNAMICO to Other Self-Adaption Models

With DYNAMICO we intend to provide software engineers with a simple, but useful guide to (i) identify the minimum components required for implementing highly dynamic adaptive systems (i.e., facing highly changing contexts); and (ii) realize and control effectively the interactions among these components at runtime. Thus, our reference model aims to support software engineers in the implementation of dynamic mechanisms, by calling their attention to the necessity of reasoning about changes at the three levels of dynamics introduced by DYNAMICO. Moreover, our model constitutes a guide to analyze the effect of these changes in (a) the accomplishment of control objectives, (b) adaptation mechanisms, and (c) context relevance along the system evolution. From the perspective of this research, highly dynamic adaptive systems are adaptive systems where changes in control objectives (adaptation goals) are supported at runtime. As a result, adaptation and monitoring mechanisms are capable of adjusting themselves, at runtime, accordingly. To address dynamics, feedback loops, their visibility, and separation of concerns among them and their components constitute key runtime drivers in DYNAMICO.

Several contributions have recognized the importance of these drivers in the engineering of self-adaptive software. Feedback loop models from control theory address separation of concerns by decoupling controllers from target systems. From the perspective of adaptive software, this corresponds to a separation of concerns between adaptation mechanisms and managed systems [9]. The autonomic manager, as defined by IBM in its autonomic computing vision, goes further by increasing the visibility of the components that define a controller in the form of the MAPE-K loop. Moreover, the autonomic manager identifies the knowledge base as an important element for implementing intra-loop communication and data persistence mechanisms. Similarly to feedback loops, the autonomic manager addresses separation of concerns by implementing sensors and effectors as a level of indirection between the adaptation mechanism and the managed element [15]. A more recent reference model is FORMS, defined by Weyns *et al.* [12]. FORMS provides a meta-model based on the MAPE-K model, and combines it with a formal specification of its elements, which supports the composition of self-adaptation mechanisms. The FORMS's static structure diagram specifies the types of elements required to implement adaptation

mechanisms, and the relationships among these elements. Thus, self-adaptive implementations instantiated from FORMS are based on the MAPE-K loop, and rely on computational reflection approaches to affect managed elements. The MAPE components are realized as computations derived from meta-level computations, whereas the K component is realized in the form of models instantiated from meta-level models. Meta-level models are key enablers of adaptation mechanisms, which are supported by meta-level computations [34]. MAPE-K loop implementations, including hierarchical and decentralized compositions of MAPE-K loop components can be instantiated directly from FORMS. Therefore, FORMS addresses separation of concerns and visibility of the feedback loop's components in the same way as addressed by the autonomic manager. The FORMS model seems to be a suitable approach to implement self-adaptation mechanisms by exploiting model-driven engineering technologies. Nevertheless, implementations where the relevance of adaptation mechanisms and monitoring strategies must be controlled at runtime to address changes in adaptation objectives are not currently supported by FORMS.

The main contribution of our reference model refers to the separation of concerns required to deal with the three levels of dynamics in self-adaptation. In DYNAMICO, separation of concerns goes beyond the decoupling of adaptation mechanisms from managed systems. We introduce three different types of MAPE-K loops that must interact among them to address changes in self-adaptive approaches at three different levels: control objectives, adaptation, and monitoring. Our reference model characterizes the elements required to control adaptation mechanisms under highly changing execution conditions. These elements are the components of the three types of feedback loops, the control/data flow among their components, and the control/data flow among the three levels. DYNAMICO relies on the MAPE-K loop to characterize the components that define our control objectives, adaptation and monitoring feedback loops. However, DYNAMICO is independent of the particular strategies and technologies used for implementing self-adaptation. To characterize the elements of our reference model and the interactions among them, we analyzed 34 of the most representative research approaches to self-adaptation [21]. The surveyed approaches range from control theory-based approaches to pure software-based approaches. In control-based approaches, the managed system's structure is generally a non-modifiable structure, and control actions are continuous signals that affect behavioral properties of the managed system. In software-based approaches, the managed system's structure is commonly a modifiable structure, and control actions are discrete operations, supported by software models and reflection, that affect the system's software architecture. We proposed DYNAMICO to guide the design and implementation of dynamic control capabilities along the whole self-adaptive systems spectrum.

A software engineer can use DYNAMICO not only to instantiate, independently, each of the three feedback loops, but also to instantiate the interactions among these three feedback loops. Consequently, a DYNAMICO-based self-adaptive system can support changes in adaptation goals at runtime, as well

as the use of these changes to adapt adaptation mechanisms and monitoring strategies accordingly. Moreover, this adaptive instrumentation can keep track of changes in monitoring strategies that could indicate the necessity of revising adaptation goals. Revisiting the application example used throughout this chapter, an adaptive solution purely based on any the feedback-loop, the MAPE-K loop, or the FORMS model could not adapt automatically the monitoring strategy after re-negotiating the performance SLA, according to the new context monitoring requirements stated with the throughput SLO (cf. Sect. 2.1).

According to Bass *et al.*, the process of designing a concrete software architecture for a system should start either from a reference model or an architectural style, or from both [14]. In either case, the process continues with successive refinement steps, where each step augments the previous one with additional information from further analysis of requirements in the problem domain, as well as global design decisions. In light of this, the application of DYNAMICO must be complemented with specific design patterns, architectural styles, design profiles, frameworks, and even other more specific or domain-dependent reference models for designing self-adaptive software systems. In particular, architectural patterns for interacting control loops such as the ones described in Sect. 4 of the first chapter in this book—the roadmap—constitute a suitable approach applicable to the design and implementation of feedback loop interactions defined in our reference model. Similarly, approaches such as the MAPE-K loop extensions proposed by Vromant *et al.* may be applied together with DYNAMICO, and selected architectural patterns to support intra- and inter-loop coordination during the different phases of self-adaptation [35]. Furthermore, due to its general nature, DYNAMICO supports the engineering of self-adaptive systems independently of concrete architectural considerations, such as the level of centralization or decentralization required by the control mechanism, as exemplified in [34]. UML profiles, such as the one proposed by Hebig *et al.* [36], provide valuable support for the design of UML-based concrete architectures based on our reference model.

6 Conclusions and Future Work

In this chapter we have presented DYNAMICO, a reference model for engineering highly dynamic adaptive software systems. This kind of system must deal with highly dynamic contexts of execution, and effectively respond to, by evaluating their own behaviour at runtime and reconfiguring itself whenever it no longer satisfies its requirements.

A highly dynamic context is characterized by (a) expected and unexpected changes in context conditions such as user location (in mobile software clients), network access point, service throughput and load (in the server side), time and calendar dates, and even user interests associated to specific locations and special dates; (b) dynamic changes in adaptation goals and user requirements, such as renegotiation of QoS levels for specific services; and (c) other sensible changes that affect the satisfaction of system requirements, such as unauthorized intrusions

or faults. In addition, all of these changes are assumed as natural requirements to be satisfied by the self-adaptive system at runtime.

DYNAMICO helps cope with this kind of dynamic requirements by defining three types of feedback loops. Each of these feedback loops manages each of the three levels of context dynamics that we characterized for self-adaptation: (i) the control objectives feedback loop, for managing changes in adaptation goals and user requirements; (ii) the target system adaptation feedback loop, to deal with changes addressable directly at the target system level; and (iii) the dynamic monitoring feedback loop, to manage changes that require the deployment of different or additional monitoring infrastructures to those already configured for execution, thus maintaining its relevance with respect to the changing adaptation goals. As a reference model, DYNAMICO reconciles the many visions and contributions of different approaches for the development of self-adaptive software systems, whether they hide or exhibit the elements of feedback control loops. Nonetheless, our reference model emphasizes the visibility of these control elements and constitutes a guide to design self-adaptive systems in which the system goals, the target system itself, or the monitoring infrastructure must be adapted—assuming this is a crucial requirement for the system to be developed. Depending on these requirements, the model can be applied as a whole, with its three feedback loops, or partially, involving only a subset of them.

We showed the applicability of DYNAMICO using a SOA governance application example based on an industrial case study. In this example, self-adaptation mechanisms are used to guarantee SLAs in a cloud-based infrastructure whose conditions of operation (i.e., efficiency and throughput) are re-negotiated at runtime, potentially compromising the effectiveness of the already deployed monitoring infrastructure. To reestablish the relevance of the monitoring infrastructure, we combined two of the three feedback loops managed in DYNAMICO: (i) the control objectives feedback loop for managing changes in the adaptation goals (i.e., SLAs); and (ii) the dynamic monitoring feedback loop to deploy the required additional monitoring elements.

For future research there are several opportunities for extending and validating DYNAMICO: (i) the use of DYNAMICO in additional validation cases, as part of the IBM CAS project "Managing Dynamic Context to Optimize Smart Interactions and Smart Services",[2] addressing different issues such as the dynamic discovery and adaptation of smart services to enable user-driven web integration, and supporting distributed feedback loops for decentralized adaptation control, as those discussed in Sect. 5.2; (ii) the concrete definition of control objectives as contracts, to support the synchronized cooperation between context management systems and self-adaptation mechanisms; (iii) the development of generalized governance infrastructures to manage feedback loop interactions; and (iv) the definition of a formal framework to evaluate and compare adaptation mechanisms based on the three levels of self-adaptation dynamics that we characterized in Sect. 3.3. For this, our characterization model and adaptation properties can be used as a useful starting point [33].

[2] https://www-927.ibm.com/ibm/cas/cassis/viewReport?REPORT=747

Acknowledgments. This work was funded in part by the National Sciences and Engineering Research Council (NSERC) of Canada under the Strategic Networks Grants Program (NETGP 397724-10) and Collaborative Research and Development program (CRDPJ 320529-04 and CRDPJ 356154-07), IBM Corporation, CA Inc., Icesi University (Cali, Colombia), and Ministry of Higher Education and Research of Nord-Pas de Calais Regional Council and FEDER under Contrat de Projets Etat Region (CPER) 2007-2013.

References

1. Northrop, L., Feiler, P., Gabriel, R., Goodenough, J., Longstaff, T., Kazman, R., Klein, M., Schmidt, D., Sullivan, K., Wallnau, K.: Ultra-Large-Scale Systems the Software Challenge of the Future. Technical report, Carnegie Mellon University Software Engineering Institute (2006)
2. United States Air Force Chief Scientist (AF/ST): Technology Horizons a Vision for Air Force Science & Technology During 2010-2030. Technical report, U.S. Air Force (2010)
3. Cheng, B.H.C., de Lemos, R., Giese, H., Inverardi, P., Magee, J., Andersson, J., Becker, B., Bencomo, N., Brun, Y., Cukic, B., Di Marzo Serugendo, G., Dustdar, S., Finkelstein, A., Gacek, C., Geihs, K., Grassi, V., Karsai, G., Kienle, H.M., Kramer, J., Litoiu, M., Malek, S., Mirandola, R., Müller, H.A., Park, S., Shaw, M., Tichy, M., Tivoli, M., Weyns, D., Whittle, J.: Software Engineering for Self-Adaptive Systems: A Research Roadmap. In: Cheng, B.H.C., de Lemos, R., Giese, H., Inverardi, P., Magee, J. (eds.) Self-Adaptive Systems. LNCS, vol. 5525, pp. 1–26. Springer, Heidelberg (2009)
4. Truex, D.P., Baskerville, R., Klein, H.: Growing Systems in Emergent Organizations. Communications of the ACM 42(8), 117–123 (1999)
5. Tran, V.X., Tsuji, H.: A Survey and Analysis on Semantics in QoS for Web Services. In: International Conference on Advanced Information Networking and Applications, pp. 379–385. IEEE (2009)
6. Tamura, G., Casallas, R., Cleve, A., Duchien, L.: QoS Contract-Aware Reconfiguration of Component Architectures Using E-Graphs. In: Barbosa, L.S. (ed.) FACS 2010. LNCS, vol. 6921, pp. 34–52. Springer, Heidelberg (2010)
7. Oreizy, P., Medvidovic, N., Taylor, R.N.: Runtime Software Adaptation: Framework, Approaches, and Styles. In: 30th International Conference on Software Engineering (ICSE 2008), pp. 899–910 (2008)
8. Giese, H., Brun, Y., Serugendo, J.D.M., Gacek, C., Kienle, H., Müller, H., Pezzè, M., Shaw, M.: Engineering Self-Adaptive and Self-Managing Systems. LNCS 5527, 47–69. Springer (2009)
9. Hellerstein, J.L., Diao, Y., Parekh, S., Tilbury, D.M.: Feedback Control of Computing Systems. John Wiley & Sons (2004)
10. Müller, H.A., Kienle, H.M., Stege, U.: Autonomic Computing Now You See It, Now You Don't—Design and Evolution of Autonomic Software Systems. In: De Lucia, A., Ferrucci, F. (eds.) ISSSE 2006-2008. LNCS, vol. 5413, pp. 32–54. Springer, Heidelberg (2009)
11. IBM Corporation: An Architectural Blueprint for Autonomic Computing. Technical report, IBM Corporation (2006)
12. Weyns, D., Malek, S., Andersson, J.: FORMS: a FOrmal Reference Model for Self-adaptation. In: 7th International Conference on Autonomic Computing, ICAC 2010, pp. 205–214. ACM, New York (2010)

13. Kramer, J., Magee, J.: Self-Managed Systems: an Architectural Challenge. In: 2007 Workshop on the Future of Software Engineering (FOSE 2007), pp. 259–268. IEEE Computer Society (2007)
14. Bass, L., Clements, P., Kazman, R.: Software Architecture in Practice, 2nd edn. Addison-Wesley, Reading (2003)
15. Kephart, J.O., Chess, D.M.: The Vision of Autonomic Computing. Computer 36(1), 41–50 (2003)
16. Papazoglou, M.P., Heuvel, W.J.: Service Oriented Architectures: Approaches, Technologies and Research Issues. The Very Large Databases (VLDB) Journal 16, 389–415 (2007)
17. Villegas, N.M., Müller, H.A., Tamura, G.: Optimizing Run-Time SOA Governance through Context-Driven SLAs and Dynamic Monitoring. In: 2011 IEEE International Workshop on the Maintenance and Evolution of Service-Oriented and Cloud-Based Systems (MESOCA 2011), pp. 1–10. IEEE (2011)
18. Villegas, N.M., Müller, H.A.: Context-driven Adaptive Monitoring for Supporting SOA Governance. In: 4th International Workshop on a Research Agenda for Maintenance and Evolution of Service-Oriented Systems (MESOA 2010). CMU/SEI-2011-SR-008, Pittsburgh: Carnegie Mellon University (2011)
19. Lee, J.Y., Lee, J.W., Cheun, D.W., Kim, S.D.: A Quality Model for Evaluating Software-as-a-Service in Cloud Computing. In: 7th ACIS International Conference on Software Engineering Research, Management and Applications (SERA 2009), pp. 261–266. IEEE Computer Society, Washington, DC (2009)
20. Müller, H., Pezzè, M., Shaw, M.: Visibility of Control in Adaptive Systems. In: 2nd International Workshop on Ultra-Large-Scale Software-Intensive Systems (ULSSIS 2008), pp. 23–26 (2008)
21. Villegas, N.M., Müller, H.A., Tamura, G., Duchien, L., Casallas, R.: A Framework for Evaluating Quality-driven Self-Adaptive Software Systems. In: 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, pp. 80–89. ACM, New York (2011)
22. Shaw, M.: Beyond Objects: A Software Design Paradigm Based on Process Control. ACM Software Engineering Notes 20(1), 27–38 (1995)
23. Villegas, N.M., Müller, H.A.: Managing Dynamic Context to Optimize Smart Interactions and Services. In: Chignell, M., Cordy, J., Ng, J., Yesha, Y. (eds.) The Smart Internet. LNCS, vol. 6400, pp. 289–318. Springer, Heidelberg (2010)
24. Bianco, P., Lewis, G., Merson, P.: Service Level Agreements in Service-Oriented Architecture Environments. Technical Report CMU/SEI-2008-TN-021, CMU/SEI (2008)
25. Elkhodary, A., Esfahani, N., Malek, S.: FUSION: a Framework for Engineering Self-Tuning Self-Adaptive Software Systems. In: 18th ACM International Symposium on Foundations of Software Engineering, FSE 2010, pp.7–16. ACM (2010)
26. Dumont, G., Huzmezan, M.: Concepts, Methods and Techniques in Adaptive Control. In: 2002 American Control Conference, vol. 2, pp. 1137–1150. IEEE (2002)
27. Narendra, K.S., Balakrishnan, J.: Adaptive Control Using Multiple Models. IEEE Transactions on Automatic Control 42, 171–187 (1997)
28. Garlan, D., Cheng, S.W., Schmerl, B.: Increasing System Dependability through Architecture-based Self-Repair. In: de Lemos, R., Gacek, C., Romanovsky, A. (eds.) Architecting Dependable Systems. LNCS, vol. 2677, pp. 61–89. Springer, Heidelberg (2003)
29. Parra, C., Blanc, X., Duchien, L.: Context Awareness for Dynamic Service-Oriented Product Lines. In: 13th Intentaional Software Product Line Conference (SPLC 2009), pp. 131–140 (2009)

30. Solomon, B., Ionescu, D., Litoiu, M., Mihaescu, M.: A Real-time Adaptive Control of Autonomic Computing Environments. In: 17th Annual International Conference hosted by the Centre for Advanced Studies Research, IBM Canada Software Laboratory (CASCON 2007), pp. 124–136 (2007)
31. Caprarescu, B.A., Petcu, D.: A Self-Organizing Feedback Loop for Autonomic Computing. Future Computing, Service Computation, Cognitive, Adaptive, Content, Patterns, Computation World, 126–131 (2009)
32. Gat, E.: Three-layer Architectures. MIT Press, Cambridge (1998)
33. Frîincu, M.E., Villegas, N.M., Petcu, D., Müller, H.A., Rouvoy, R.: Self-Healing Distributed Scheduling Platform. In: 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGRID 2011, pp. 225–234. IEEE Computer Society, Washington, DC (2011)
34. Weyns, D., Malek, S., Andersson, J.: On Decentralized Self-Adaptation: Lessons from the Trenches and Challenges for the Future. In: 2010 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2010), pp. 84–93. ACM, New York (2010)
35. Vromant, P., Weyns, D., Malek, S., Andersson, J.: On Interacting Control Loops in Self-Adaptive Systems. In: 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2011), pp. 202–207. ACM, New York (2011)
36. Hebig, R., Giese, H., Becker, B.: Making control loops explicit when architecting self-adaptive systems. In: 2nd International Workshop on Self-Organizing Architectures, SOAR 2010, pp. 21–28. ACM, New York (2010)