Citación: Villegas, N., Müller, H.A., & Tamura, G. (2011). On Designing Self-Adaptive Software Systems. *Revista S&T, 9*(18), 29-51.

Artículo de reflexión

On Designing Self-Adaptive Software Systems

Diseño de software autoadaptativo

Norha M. Villegas
Ph.D Candidate
Department of Computer Science
University of Victoria, British Columbia (Canada)
nvillega@cs.uvic.ca

Hausi A. Müller, Ph.D.
Associate Dean of Research Department of Computer Science
University of Victoria, British Columbia (Canada)
hausi@cs.uvic.ca

Gabriel Tamura
Profesor Tiempo Completo
Facultad de Ingeniería
Universidad Icesi, Cali (Colombia)
gtamura@icesi.edu.co

Fecha de recepción: Agosto 16 de 2011
Fecha de aceptación: Septiembre 13 de 2011

Keywords
Self-adaptation; reference models for self-adaptation; self-adaptive software; control loops.

Palabras clave
Autoadaptación; modelos de referencia para auto-adaptación; software auto-adaptativo; bucles de control.

Colciencias tipo 2

Abstract

Self-adaptive systems modify themselves at run-time in order to control the satisfaction of their requirements under changing environmental conditions. Over the past century, feedback-loops have been used as important models for controlling dynamic behavior of mechanical, electrical, fluid and chemical systems in the corresponding fields of engineering. More recently, they also have been adopted for engineering self-adaptive software systems. However, obtaining sound and explicit mappings consistently between adaptive software architectures and feedback loop elements is still an open challenge. This paper, recalling a reference model proposed previously with that goal, discuss key aspects on the design of adaptive software where feedback loop elements are explicitly defined as first-class components in its software architecture. It complements this discussion with an illustration of the process to use this reference model by applying it to a plausible adaptive software example. This paper aims at providing a reference starting point to support software engineers in the process of designing self-adaptive software systems.

Resumen

Ante condiciones cambiantes del entorno, los sistemas autoadaptativos pueden modificarse a sí mismos para controlar la satisfacción de sus requerimientos en tiempo de ejecución. Durante el siglo pasado los sistemas de retroalimentación fueron importantes modelos para controlar el comportamiento dinámico de sistemas mecánicos, eléctricos, de fluidos y químicos, en sus respectivos campos de la ingeniería. Más recientemente fueron adoptados para diseñar software autoadaptativo. No obstante, lograr mapeos coherentes y explícitos consistentemente entre las arquitecturas de software adaptativo y los elementos de sistemas de retroalimentación es aún un reto abierto. Este artículo, sobre un modelo de referencia propuesto con ese propósito, discute aspectos clave del diseño de software autoadaptativo, en que los elementos de sistemas de retroalimentación se definen explícitamente como componentes de primer nivel en su arquitectura. Adicionalmente, ilustra la aplicación de este modelo de referencia a un ejemplo real de software adaptativo. El artículo ofrece a los ingenieros de software un punto de referencia para iniciar el diseño de software autoadaptativo.

I. Introduction

Traditional software engineering has been based on an incorrect set of goals, where software systems are expected to support rigid and stable business structures, have low maintenance, and ensure complete user acceptance. Truex, Baskerville, & Klein (1999) criticized this user-satisfaction emphasis and questioned the economic advantages of lengthy analysis in the engineering of software systems. In light of this, they identified the necessity of new software engineering models based on permanent analysis, dynamic requirements negotiation, and incomplete requirements specification. In this setting, a new kind of software systems is emerging, whose development can be seen as a continuum of short term adaptations and long term evolution (Cheng et al., 2009; de Lemos, Giese, Müller, & Shaw, 2011; Oreizy, Medvidovic, & Taylor, 2008). As part of this continuous adaptation and evolution, system analysis is performed continuously, and in parallel with system operation and maintenance. Furthermore, system requirements satisfaction is regulated and controlled by continuously adjusting or enhancing system behavior (Giese et al., 2009).

Such dynamic systems adapt in response to changes in their environments, either to ensure the continuous satisfaction of their functional and non-functional requirements, or to provide ubiquitous and context-dependent smart services. In any case, this adaptation must be performed while preserving the contracted quality of service (QoS) conditions and desired adaptation properties (Villegas, Müller, Tamura, Duchien, & Casallas, 2011). These quality conditions are usually represented in the form of service level agreements (SLA), and their enforcement mechanisms are based on contracts and policies (Tamura, Casallas, Cleve, & Duchien, 2011).

Over the past century, the feedback-loop model from control theory has been used extensively in many diverse fields and application domains of engineering, with substantial advances and results, for instance automating the control of industrial processes (Ogata, 2010). In the first decade of the new century, IBM researchers proposed the notion of an *autonomic element* as a building block for self-managing systems, in the form of a monitoring-analysis-planning-execution and shared knowledge (MAPE-K) loop that controls a managed computing element (IBM, 2006; Kephart & Chess, 2003). Many researchers have recently proposed similar software architecture constructs and applied these concepts to engineer self-adaptive, self-organizing and self-managing software systems (Salehie & Tahvildari, 2009). Nonetheless and despite the acknowledgment on the importance of the feedback-loop, its visibility as the crucial architectural element to drive adaptation in software systems remains often hidden. This is in part due to the software flexibility, which in general does not enforce separation of concerns between adaptation mechanisms

and managed (controlled) software systems. Thus, it is common that software architectures highlight abstractions related to the functional requirements of the system, having feedback-loop elements blurred in them. In this setting, making and maintaining visible the feedback-loop elements and the feedback-loop itself in self-adaptive software architectures remains an open challenge (Giese et al., 2009; Hellerstein, Diao, Parekh, & Tilbury, 2004; Müller, Kienle, & Stege, 2009).

In this paper we illustrate how to design an adaptive software system where the feedback loop drives its architecture, and moreover, where feedback loop components are explicit in this architecture. This illustration is based on the reference model for the engineering of self-adaptive software systems proposed in a previous work (Tamura, Villegas, Müller, Duchien, & Casallas, 2011). This illustration aims at providing software engineers with a starting point for the engineering of self-adaptive software systems (Frincu, Villegas, Petcu, Müller, & Rouvoy, 2001). Our reference model allows the design of software architectures where the different feedback loops required to guarantee system properties through adaptation are directly implementable, by making explicit: (i) the management of self-adaptive properties as the control reference goals; (ii) the separation of control concerns by decoupling the different feedback-loops required to achieve the reference goals over time; and (iii) the specification of context management as an independent control function to monitor and preserve the contextual relevance, taking into account internal and external context changes.

The remainder of this paper is organized as follows. Section 2 reviews the classical theory of control systems and the feedback loop's structural and behavioural elements. It also describes the illustrative example used throughout the paper. Section 3 introduces the application of feedback loops to the engineering of self-adaptive systems. Sections 4 and 5 present, respectively, an overview of our proposed reference model and its application to the illustrative example. Section 6 discusses similar works on the application of feedback loops to the engineering of self-* systems. Finally, Section 7 concludes the paper and outlines directions for future work

II. Control Theory

In control theory, the feedback loop —or closed loop— is the cornerstone structure for building controllers for different kinds of dynamic systems, such as mechanical, electrical, fluid and chemical. Moreover, they provide the basis for automated control in diverse fields of engineering, and for self-adaptation in computing and software engineering (Müller, Pezzè, & Shaw, 2008). The feedback loop structure, as depicted in the so-called block diagram of Figure 1, performs dynamic control by comparing measured outputs of the target system behaviour to the control objective given as

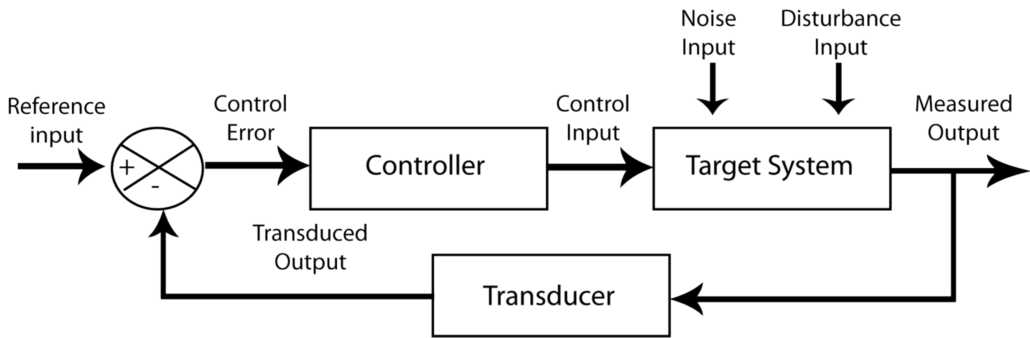


Figure 1. Classical block diagram of a single-input, single output (SISO) feedback control system (Hellerstein, Diao, Parekh, & Tilbury, 2004)

reference input, yielding the control error, and then adjusting the controlling signals accordingly for the target system to behave *closer* to this reference input (Ogata, 2010). The measured output can also be affected by external disturbances or even by the noise caused by the system adaptation. Transducers adapt the signals measured by sensors, as required by the comparison element.

In order 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*, whose main goal is to control the effects of disturbances on the measured output; and (iii) the *optimization*, which permanently seeks to obtain the best value of the measured output, as effectively as possible. These strategies constitute minor variations on the controller element, as they are realizable with no structural changes in the block diagram.

For instance, Figure 2 represents a typical time-response behaviour shape for a temperature control system with a *regulatory control* strategy. The target system behaviour is shown in terms of the signals to be controlled (e.g., the X-axis represents the time dimension and the Y-axis the measured water temperature in Celsius degrees). The system starts in a normal state, that is, stabilized around 26.67 at minute 33 of the industrial process to be controlled. Around minute 35.5, the system is affected by a *disturbance input* heating up the temperature (e.g., hot water is added or cool water is drained), and a new *reference input* is set to 26.92. With this disturbing event, the difference between the measured output and the new reference input is used by the feedback loop controller to drive the system adaptation into these new conditions, by continuously computing suitable *control input* signals and feeding them back to the system until it re-gains a normal, stable state, around minute 45. Transducer elements might be necessary when, for instance, the measured units of the control reference input and the measured system output are different.

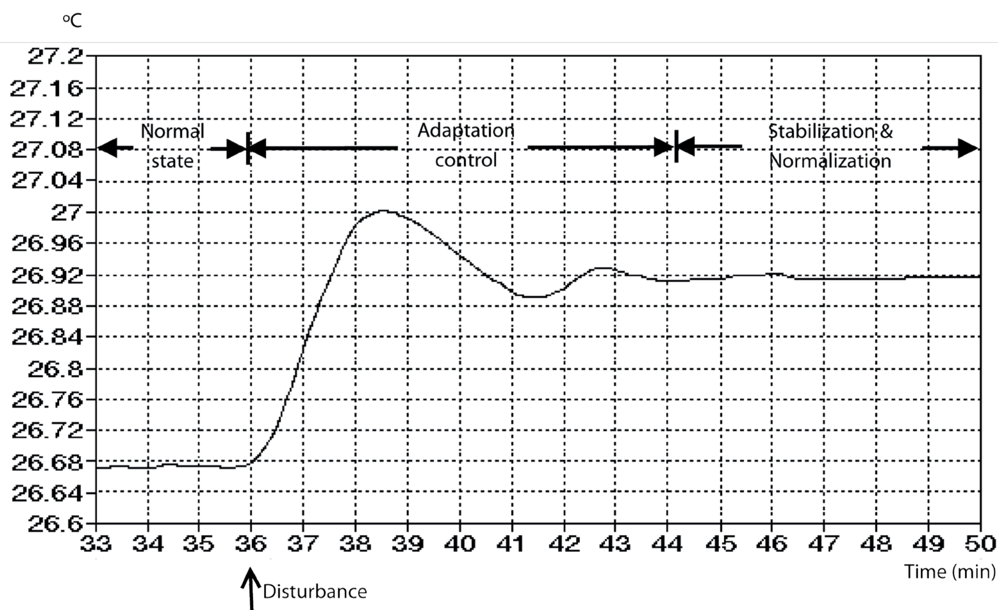


Figure 2. Typical time response for temperature control using a proportional, integral, derivative (PID) controller which is the workhorse of industrial control engineering

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 on the structure of the feedback loop block diagram. Depending on the behavioural characteristics of the system to be controlled, a system transfer function is built with proportional, derivative and integral (PID) terms. The parameters in a PID controller have special significance; there are many sophisticated methods for tuning the constants in this kind of controller.

Although the application of the theory of controlling industrial processes is well understood, its application to controlling software systems has at least two significant challenges: first, its main adaptation mechanism is modeled on continuous mathematics, and second, it relies on measures taken from and actions performed into physical, self-performing artifacts (e.g., sensors, gauges and valves/actuators for temperature, or pressure). As these physical 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 behaviour. Thus, direct sensing must be performed also by CPU-dependent software artifacts, and the adaptation mechanisms must reason on their discrete-time output. Moreover, to fully exploit the possibilities of software adaptation, 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 software actuators on the target software components. Fortunately, there is the theory of linear discrete-time systems, which closely resembles the theory of linear continuous-time systems. Nonetheless, achieving in the software domain the maturity level reached by control theory in modeling the behaviour of physical systems is one of the most challenging problems of self-adaptive software (Hellerstein, Singhal, & Wang, 2009).

To illustrate the elements of our reference model and its application throughout the paper, we next introduce a hypothetical, yet plausible, context-based adaptive system example. In this example we analyze and map the control theory concepts to dynamic software adaptation concepts.

A. An illustrative example

The example we use to discuss the elements of our approach is a smart system for managing hotels or congress centers to host conferences or group meetings. In this case, suppose a conference is held at the Colombian Conference Center, from Sep. 27 to Oct. 1, 2011 in Cartagena. The conference center is supposed to provide a ubiquitous system to assist with the logistics of the conference and the guest's activities during the event. Conferences are registered in the system as soon as their contract is agreed. The registration involves the configuration of important conference information such as start date, duration, agenda, language, community profile, or Internet access requirements. The contract agreement includes conditions related to the system properties, such as performance on rush hours, for supporting the conference activities. In this hypothetical example, the conference organizers anticipate a large turnout. As a result, in order to ensure a comfortable registration process for its delegates, they require guarantees for *maximum processing time per registration request.* The conference organizers accept and subscribe to the service provider offering a service level agreement that guarantees a maximum processing time of three seconds per request from 8:00 am to 10:00 pm, and six seconds from 10:01 pm to 7:59 am, for the two days at the beginning of the conference.

III. Feedback Loops and Adaptive Software Systems

Self-adaptive software systems are systems able to adapt their behaviour or structure at run-time (Oreizy et al., 2008; Salehie & Tahvildari, 2009). Self-managing systems are instances of self-adaptive software systems that adapt themselves according to administration goals (Kephart & Chess, 2003). Context-aware, ubiquitous and user-centric applications are examples of self-adaptive systems where the system behaviour is tailored according to users locations and preferences (Coutaz, Crowley, & Dobson, 2005; Chignell, Cordy, Ng, & Yesha,

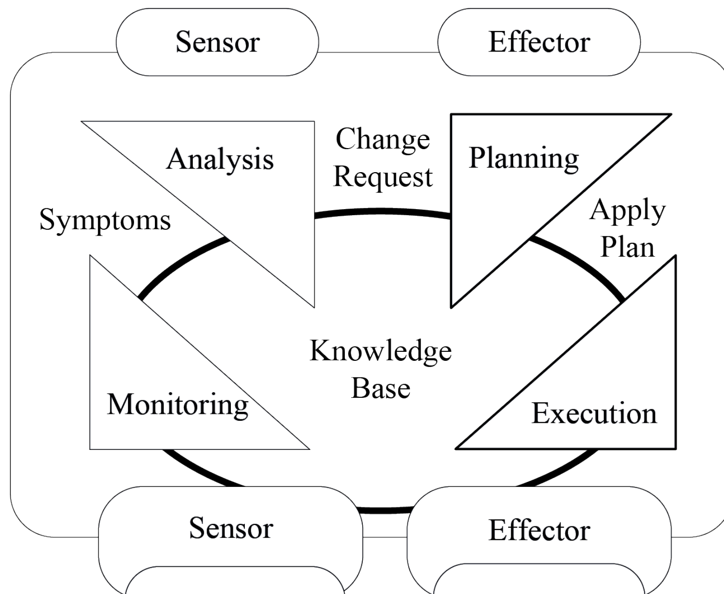


Figure 3. The MAPE-K loop (IBM, 2006; Kephart & Chess, 2003)

2010; Villegas et al., 2011a). To realize continuous adaptation and evolution, system analysis is performed continuously and in parallel with system operation and maintenance, and system requirements satisfaction is regulated and controlled by continuously adjusting or enhancing system behaviour according to environmental conditions. Self-adaptation can be static or dynamic. In static self-adaptation, adaptation mechanisms are explicitly defined by the designers for the system to choose from, during execution; whereas, in dynamic self-adaptation, adaptation plans and monitoring requirements must be produced and selected by the system at run-time (Müller et al., 2009).

As mentioned, in the first decade of the new century, IBM researchers proposed the notion of an *autonomic element* as a building block for self-managing systems in the form of a monitoring-planning-execution and shared knowledge (MAPE-K) loop (cf. Figure 3). This loop is the basic block to support self-adaptation in controlling a managed computing element (IBM, 2006; Kephart & Chess, 2003). In the first phase of the MAPE loop —monitoring, monitors gather and process context information from the environment that is relevant to the adaption process. This monitored information is sent then to the second phase, analysis, where analyzers correlate context information to infer symptoms about the execution environment and the

system behaviour. In the third phase, planning, planners use analyzed symptoms to define adaptation plans accordingly. In the last phase of the loop —execution, executors implement and execute plans to adapt the actual system and obtain the desired behaviour. The monitoring phase continuously feeds the adaptation loop back, restarting the cycle. A knowledge base supports the information flow required throughout the whole cycle.

The necessity of making explicit feedback-loop elements in the engineering of self-adaptive software systems has been identified by several recent investigations in the software engineering community (Cheng et al., 2009; Giese et al., 2009; Müller et al., 2008). On one hand, Müller et al. (2008) analyze the benefits of specifying the feedback loops and their major components explicitly and independently, as well as the necessity of defining explicitly the interactions among feedback loops and among their elements, from analysis and design to implementation. Giese et al. (2009) also argue on the benefits of decoupling 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. Cheng et al. (2009) also emphasize the importance of making explicit not only the feedback loops but also their elements and properties. On the other hand, the behaviour of the adaptive system and its components is highly dependent on the changing context information that defines the execution environment. Thus, improving context-awareness in self-adaptation by understanding, managing, and controlling context information via run-time monitoring has been identified as a promising research direction toward the engineering of reliable and efficient adaptive software systems (Lemos et al., 2011; Villegas & Müller, 2010).

Self-adaptation in software systems can take different forms (Villegas et al., 2001b). Nevertheless, feedback loops, although generally defined implicitly, are at the core of self-adaptive software systems as they adapt their behaviour to keep objectives controlled based on either regulatory control, disturbance rejection or optimization requirements (Müller et al., 2009b). For this, feedback loops provide instrumentation to sense the execution environment, model the system behaviour in that environment, and execute actions to change the environment or the system behaviour. Moreover, as depicted in Figure 4, the architectural components of the MAPE-K loop can be mapped to the elements of the feedback loop studied in control theory, but for controlling dynamic adaptation in a software system (Hebig, Giese, & Becker, 2010; Müller et al., 2008; Tamura et al., 2011b) The elements in Figure 4 clearly resemble the ones of the MAPE-K loop, but with their roles and interrelationships made explicit. This mapping constitutes the foundational structural and behavioural aspects for our reference model (Tamura et al., 2011b).

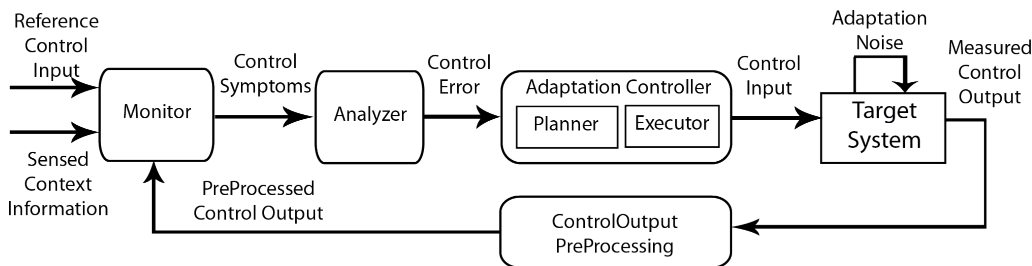


Figure 4. SISO feedback control block diagram with explicit functional elements and corresponding interactions to control dynamic adaptation in a software system.

IV. The Reference Model for Self-Adaptive Software Systems

Figure 5 illustrates the reference model for designing self-adaptive software systems presented in Tamura et al. (2011b). An adaptive system can be defined as a set of cooperating feedback loops that ensure the achievement of the system objectives. Thus, according to our reference model, a self-adaptive system is designed as a collection of cooperating control objective managers, adaptation control loops, and context manager control loops to control the adaptation process.

Our reference model can be combined with other reference models for adaptive systems. In particular, the IBM architectural blueprint provides the ACRA model to orchestrate control loops hierarchically for autonomic systems (IBM, 2006; Kephart & Chess, 2003). Similarly, our reference model can be organized hierarchically to orchestrate control and include manageability endpoints to facilitate communication between levels. Furthermore, the distribution of the feedback loops and their elements themselves in different machines should be possible. Shared knowledge bases, as the ones proposed for the MAPE-K loop, also facilitate interactions among control loops. Such knowledge bases could store historical information such as symptoms, as well as internal and external context facts required for the analyzers in each type of control loop. Moreover, these persistence mechanisms help maintain contracts and policies to achieve the control objectives. Finally, contracts enable the controllers to reason about goals and agreements to determine the control actions.

The following subsections summarize selected design drivers for the application of our reference model.

A. Separation of Concerns

Analyzing our reference model (cf. Figure 4) from a combined control theory and software architecture point of view, for a software system (*target system*) to become self-

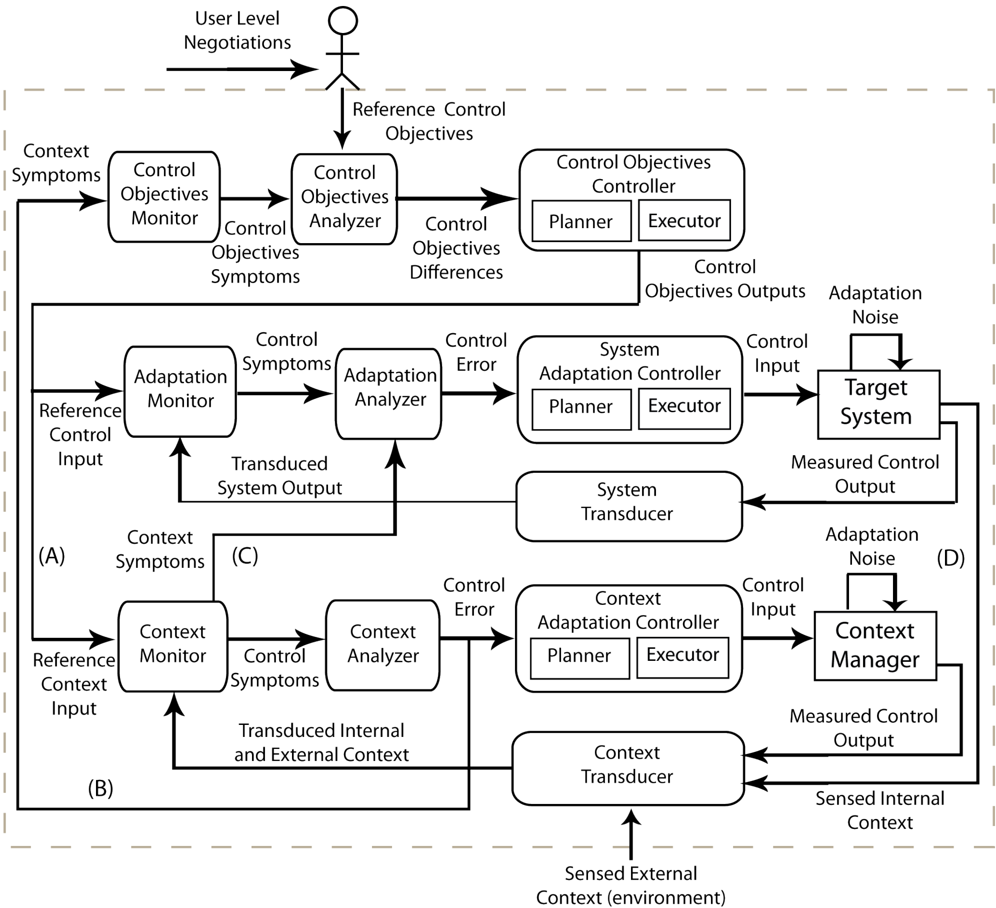


Figure 5. Reference model for context-based self-adaptive systems (Tamura et al., 2011b). The context manager feedback loop supports the adaptation for a system to keep fulfilling its control objectives in a dynamic environment. Signals (A), (B), (C) and (D) highlight the interactions among the feedback loops.

adaptive, it should incorporate at least three subsystems with it: (i) a *control objectives manager*; (ii) a *context manager* (monitoring mechanism); and (iii) an *adaptation mechanism.* This design separates the concerns with respect to (a) the final purpose and properties of the target system (control objectives manager); (b) the continued fulfillment of the system purpose, and the preservation of its properties under changing conditions of execution (system adaptation controller or adaptation mechanism); and (c) the system relevance according to whatever it must take into account from a *new* environment, after a change of execution conditions (responsibility of the context manager).

Applying the reference model to the illustrative scenario outlined in Section II.A, we clearly identify three types of adaptation, depending on the different emphasis of feedback loop interactions: *preventive*, *corrective* and *predictive*. In preventive adaptation, the context monitor notifies the adaptation analyzer about context events (*context symptoms*) that are causing no effects in the target system behaviour yet, but eventually will. For instance reaching the previous day of a registered conference beginning date (a calendar event); that is, even though the adaptation subsystem has encountered no disturbances yet in the target system, based on this context information it could minimize the risks of goal disruption by performing a system adaptation (e.g., to manage a significantly higher than normal number of registration requests per time unit). Corrective adaptation takes place when the adaptation controller detects that the target system is not fulfilling its control objectives (e.g., QoS agreements) by measuring the target system behaviour. This can occur if, even in the example of preventive adaptation, the number of requests per time unit is reaching the limits of the new capacity after performing the adaptation. This situation requires for the adaptation controller to perform an additional reconfiguration or to apply restrictive usage mechanisms 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 fulfillment disruption, as well as symptoms that evidence the necessity of adaptation. These symptoms might be presented in the form of patterns of correlated events that eventually become significant evidence to advice for adaptation. An example of this latter case is the detection of a low but frequent degradation in the system performance near a conference start date, independently of the fact that this date had been previously registered in the system. This continued degradation may be caused by early registration of several groups of conference assistants, that nonetheless is non-sensible enough for the adaptation subsystem to react.

Finally, despite this separation of concerns, the target system together with the control objectives manager, the context manager and the adaptation mechanism, also constitute a general feedback loop.

B. Management of Control Objectives as a Feedback Loop

The system purpose and corresponding properties are objectives whose permanent fulfillment must be controlled through the collaboration of the adaptive and context feedback loops. However, control objectives are also subject to change by user-level negotiations at run-time. Therefore, they must be managed 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, there may be service level agreements that, even if they have been pre-negotiated (or defined statically), they imply changes at run-time in the system objectives (reference inputs). For instance, this

may happen when a pre-negotiated contract defines that the system must guarantee a maximum of six seconds per request from 8:00 am to 10:00 pm, but a maximum of 12 seconds from 10:01 pm to 7:59 am, for the two days before the conference start. In this case, whenever the context manager detects a change of time from 7:59 am to 8:00 am within these two days, it notifies the adaptation mechanism to reconfigure the target system architecture in such a way that the target system can support a maximum of six seconds per request. This is a new reference input to be controlled by the adaptation mechanism, whose achievement must be enforced by reconfiguring the target system, analogous to the change in the set points in our initial example of temperature control. However, the achievement of the reference inputs over time may require a successive application of several rounds of planned actions for system reconfiguration and corresponding measurements, until the target system effectively achieves the required performance level.

In a second case, when the system is in execution, there may be a re-negotiation of conditions on previously negotiated SLAs. An instance of this case would occur if a new SLA is agreed requiring a maximum of only one second per request, for the four hours before the conference opening. In either case, the system purpose and corresponding properties are managed explicitly as the control objectives for the adaptive system. Thus, ideally both, the system adaptation reference control input and the reference context input should be automatically derived from these control objectives and fed into the corresponding feedback loops (cf. interaction (A) in Figure 5). This automatic derivation ensures the consistency and synchronization between these two sets of reference inputs.

In our example, when the time changes from 7:59 am to 8:00 am, the control objectives monitor is notified of this event. Then, the control objectives analyzer determines that the control objectives must change according to the agreed SLAs and the control objectives controller plan, and execute the replacement of the in force SLA of 6 seconds per request by the SLA of 3 seconds per request. This change defines the quality attribute or new control objective (e.g., performance), and therefore the adaptation strategy (e.g., architecture reconfiguration rule for deploying additional scalable software components in spare hosts of a grid computing infrastructure). The new SLA also determines the relevant context information that must be monitored from the environment (e.g., elapsed time between request start and stop time-stamps), and its corresponding context management requirements (e.g., context gathering strategies, sampling rates, measurement precision and resolution). Thus, from this new SLA, the control objectives manager derives the *reference control inputs* (quality attribute: request processing performance; set point: 6; units: seconds) and the corresponding *reference context inputs* (sensor infrastructure type: logical; sensor type: time; precision: tenths of second; minimum resolution: one tenth of second; monitoring sampling rate: every 30 seconds).

Finally, the dynamic adaptation of control reference inputs is addressed by closing the control objectives manager as another feedback loop, in which the monitor receives symptoms from the context analyzer (control error); the analyzer, planner and executor conform the control objectives manager; and the target system is represented by the triple: context manager, adaptation mechanism, and core controlled system.

C. Adaptation Control Feedback Loop

The adaptation control feedback loop mission is to serve as a guarantor for the continued fulfillment of the target system purposes and corresponding properties preservation. We refer to purpose and properties as system variables to be controlled.

To accomplish this mission, the adaptation control feedback loop follows the separation of concerns criteria described in previous sections. In turn, these criteria conform to the general principle of control theory, which relies on quantitative expressions to measure the error in the controlled system variables, and a reference control input for each of these variables. The adaptation control feedback loop continually gathers these measurements from the target system through its adaptation monitor. This monitor notifies control symptoms for adaptation to the adaptation analyzer, which determines the necessity of performing a system adaptation. 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 not being fulfilled. Whenever it is relevant, the adaptation analyzer notifies this fact with its corresponding information to the system adaptation controller. With this information, the planner element selects a strategy to adapt the system with the goal of reestablishing the fulfillment of the violated control objective. The result of this strategy is sent as an ordered list of system architecture reconfiguration actions, which are performed in the target system by the executor, thus closing the control loop.

D. Context Manager Feedback Loop

The feedback loop depicted in the bottom part of Figure 5 represents the context manager as part of the monitoring mechanism in our reference model. The *reference context input* corresponds to the reference context management objectives derived from the system control objectives. These reference inputs form the basis for the generation of context models that represent environmental information useful for supporting the adaptation process. In our application example, context models represent context information types as well as different levels of granularity, constraints, and relations among context entities, and spatial and scope information that must be managed for guaranteeing the objectives of the ubiquitous conference system (e.g., sensor infrastructure type: logical; sensor type: time; precision: tenths of a second; monitoring sampling rate: every 30 seconds). The *context monitor* refers to the gathering of primary context information from the internal and the external environment

and the correlation of this information to infer either, context symptoms that can affect the target system adaptation process, including the adaptation of the control objectives, or control symptoms to decide about the context manager adaptation. This information is preprocessed by the *context transducer* to generate numeric values from physical and logical sensors, and determine comparable measures by performing basic transformations. Furthermore, the *context analyzer* performs the context handling process required for the context adaptation controller to decide about adapting the context manager, and for the control objectives manager to decide about changing the system objectives, as demanded by the adaptive system and its environment. Changes in control objectives can be performed fully or semi-automatically, depending on whether or not it is necessary to re-negotiate and, consequently, for the user to intervene (cf. Section 4.2). The *context adaptation controller* is in charge of defining and executing the adaptation plan for the context manager, according to its adaptation strategy. Finally, the measured control output and the adaptive system internal context are used to achieve the context manager goal: to support the system adaptation process and the management of the system control objectives.

E. Feedback Loop Interactions

The decoupling proposed by our model not only supports the separation of the corresponding feedback control loops, but also the separation of the elements within each feedback loop. Even though control loops are designed independently of each other, they must operate together to achieve the overall system objectives.

As depicted in Figure 5, in order to ensure the achievement of the control objectives, our reference model specifies four interactions among its feedback loops, labeled (A), (B), (C) and (D). We classify interactions (A) and (B) as indirect interactions because they are realized through the control objectives manager, and interactions (C) and (D) as direct interactions due to their direct connections between the context manager loop and the adaptation loop.

Interaction (A) provides the reference context input (i.e., context manager objectives) for the context manager loop to gauge the relevance of context information; decide how to manage and provision that environmental information; and provide the reference control input (i.e., adaptation objectives) for the adaptation controller to manage the adaptation process for achieve the system objectives.

Interaction (B) enables the control objectives manager to decide about changes in the control objectives, whenever the context manager detects that given the current context, the current set of control objectives should be dynamically adjusted or re-negotiated.

Interaction (C) is realized through context symptoms that are identified and sent from the context monitor to the adaptation analyzer. These context symptoms can

be in the form of events useful for decision making in the adaptation control loop. The communication mechanism and the information associated with these symptoms depend on the supported adaptation type (i.e., preventive, corrective or predictive).

Interaction (D) represents the internal context sensed by the context manager from the adaptive system. Monitoring internal context information is necessary to assess the consistency of the system after an adaptation. By analyzing internal context information that characterizes the current state of system properties, the context manager must be able to infer symptoms about the achievement of system goals.

V. Applying the Reference Model

According to Bass, Clements and Kazman (2003), the process of designing a concrete software architecture for a system should start either from a reference model or from an architectural style, or from both. In either case, the process continues with successive refinement steps, where each step augments the previous with additional information from further analysis of requirements in the problem domain as well as design decisions.

To obtain a concrete self-adaptive software architecture for the conference system of our hypothetical example, we follow this step-wise refinement process, starting from our reference model. An intermediate refinement step is obtained by mapping and incorporating the reference model elements into the conference system components. Figure 6 illustrates this refinement step by using *contracts* as system control objectives, and *SLA*s as reference control inputs. We also assume that in this case the context manager is not required to be self-adaptive.

Through further refinement, we obtain a concrete software architecture for our conference management system, the UML deployment diagram depicted in Figure 7.

To illustrate the system behaviour, assume that the system has been configured to accept registration requests just for two days before the conference start day. Thus, for this conference, the mission of *ControlObjectivesManager* is to preserve four performance SLAs: (i, ii) three seconds per request from 8:00 am to 9:59 pm and (iii, iv) six seconds from 10:00 pm to 7:59 am on Sep. 25/26.

ControlObjectivesManager, once notified through its *contextSymptoms* port of the 8:00 am beginning-of-conference-registration time-event on Sep. 25 by the *ContextMonitor* component, sets the *SLA* of three seconds per request and the corresponding *SLAContextRequirements* reference control inputs for the adaptation and context monitors, *AdaptMonitor* and *ContextMonitor* respectively. Periodically, these two concurrent components monitor the collected system measurements from the *ManagedConferenceSystem*'s *sensorSystPolling* port. However, both components monitor the target system at different rates.

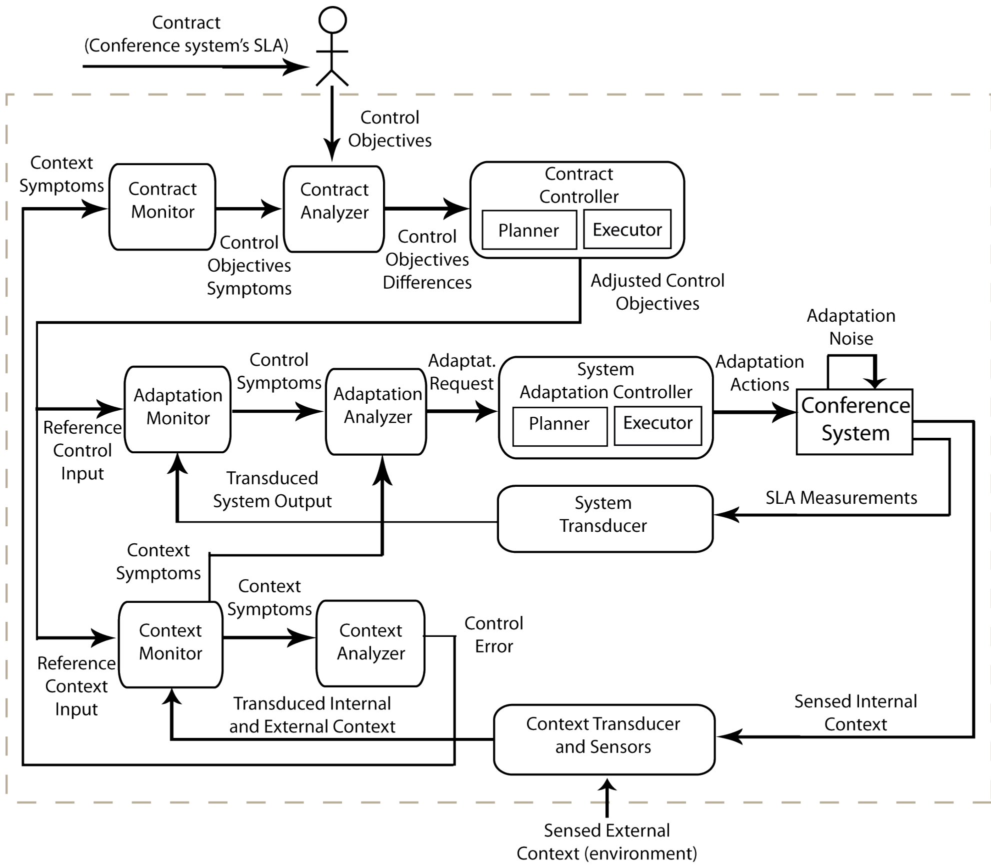


Figure 6. The reference model elements mapped into the conference management system components. Note that the context manager is not self-adaptive.

AdaptMonitor signals *corrective adaptation symptoms* to the *AdaptAnalyzer* based on comparisons between the system measurements and reference control inputs, either in a stabilized and normal system state, or immediately after a system adaptation, during the stabilization phase. *ContextMonitor* provides the *AdaptAnalyzer* with (i) *predictive adaptation symptoms*, based on the recognition of behavioural adaptation; or with (ii) *preventive adaptation symptoms*, based on context events gathered by the *ContextGathering* components.

Whenever *AdaptAnalyzer* determines that a system adaptation is required, it provides the corresponding context information to the *AdaptController* component. With this information, *AdaptController* requests a system architecture *reconfiguration plan* from the *ArchReconfRuleSystem*, and then sends the *reconfiguration actions* to *ReflectionInfrastructure* for execution. To achieve better SLA performance over time, one plausible architecture

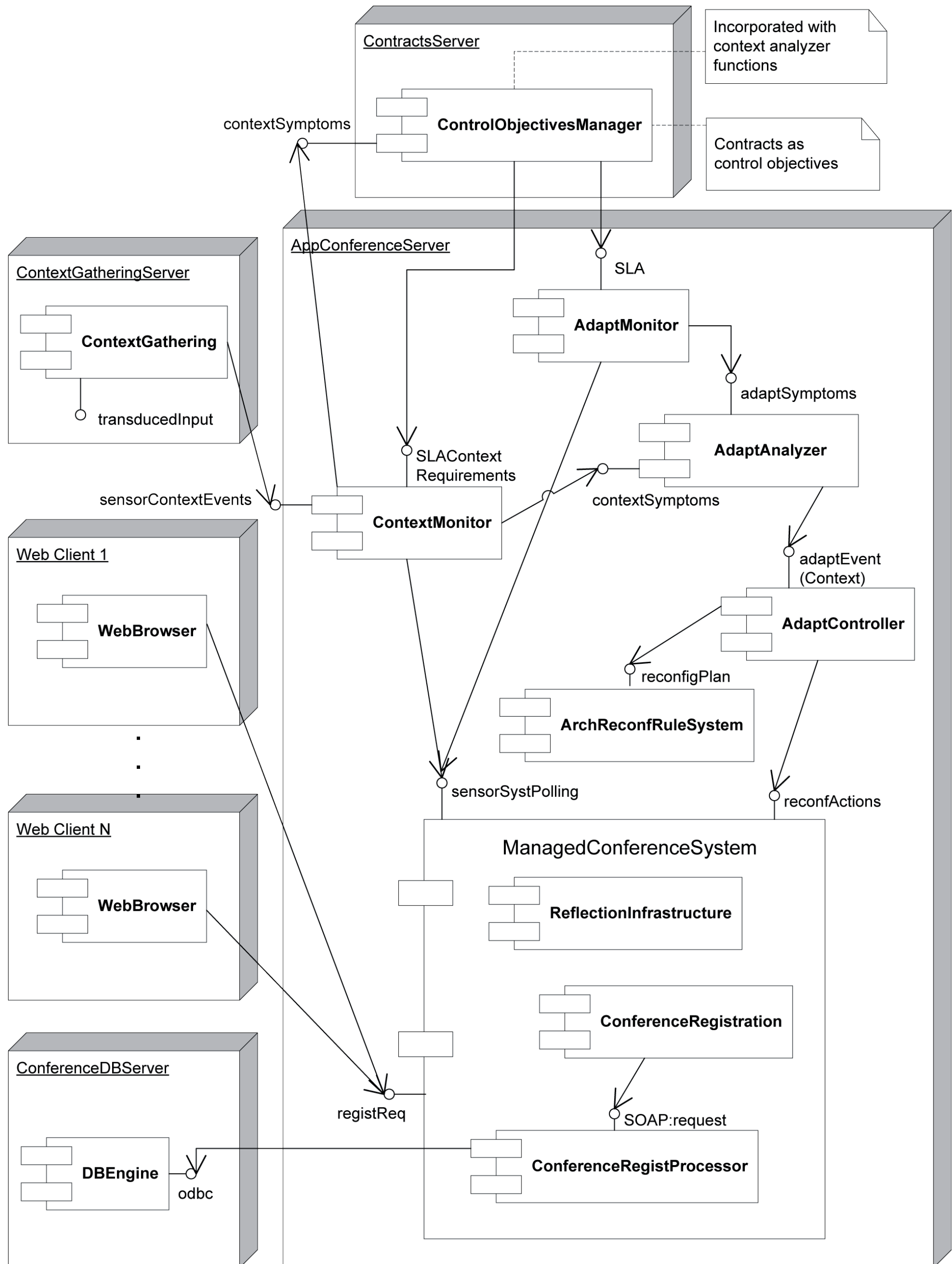


Figure 7. Simplified software architecture for the conference management system derived from our reference model. The visibility of the feedback control loops is clearly maintained and mappable to the detailed design documents and the source code

reconfiguration rule would re-deploy the *ConferenceRegistProcessor* component of Figure 7 and deploy additional *ConferenceRegistProcessor* components, each one on new additional hosts, then reconnecting the corresponding components.

Finally, the *ConferenceRegistration* component receives registration requests from *WebBrowser* clients. The *ConferenceRegistration* component redirects them to the different *ConferenceRegistProcessor* components for processing.

VI. Related Work

This section presents evidence of the application of feedback loops to concrete implementations of adaptive software systems. However, as concrete implementations, they are not necessarily examples of reference models. On the contrary, these cases evidence the necessity of providing reference models with explicit application guidelines, such as the one we illustrate in this paper.

A first example of concrete implementations is the self-healing system developed by Garlan, Cheng and Schmerl (2003). Their system architecture maps directly to the feedback control architecture proposed by Müller et al. (2008), even though not all of the control loop elements are made explicit. Similarly, the separation of concerns and the management of common control objectives are not considered.

A second interesting instance is the context-aware dynamic software product line proposed by Parra, Blanc and Duchien (2009). They proposed the introduction of context-aware assets that are dynamically incorporated into the product line, depending on context changes. Although their architecture alludes to the existence of feedback loop elements (i.e., a context manager (monitor), a decision maker (analyzer and planner), a run-time platform (executor) and a knowledge base), control loop properties and interactions are not completely addressed. However, despite the monitor is implemented in their architecture as an independent context manager using COSMOS (Abid, Chabridon, & Conan, 2009), the monitoring mechanism as a context manager is not designed itself as a feedback look.

From the community of autonomic computing we consider the real-time adaptive control approach for autonomic computing environments proposed by Solomon, Ionescu, Litoiu and Mihaescu (2007). Their system aims to control the computing infrastructure through a mathematical model of the variation of the number of users per unit time. Based on this function, the system modifies the control structure of the autonomic computing infrastructure by replacing its controller with one that matches the model of users variation in time. Furthermore, their adaptive control is based on a multi-layer architecture similar to ACRA, where the two upper layers correspond to the autonomic system adaptation and the autonomic system layers respectively, 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. Even though their approach separates the adaptation and the autonomic management mechanisms into different layers, the concerns are not separated within every layer.

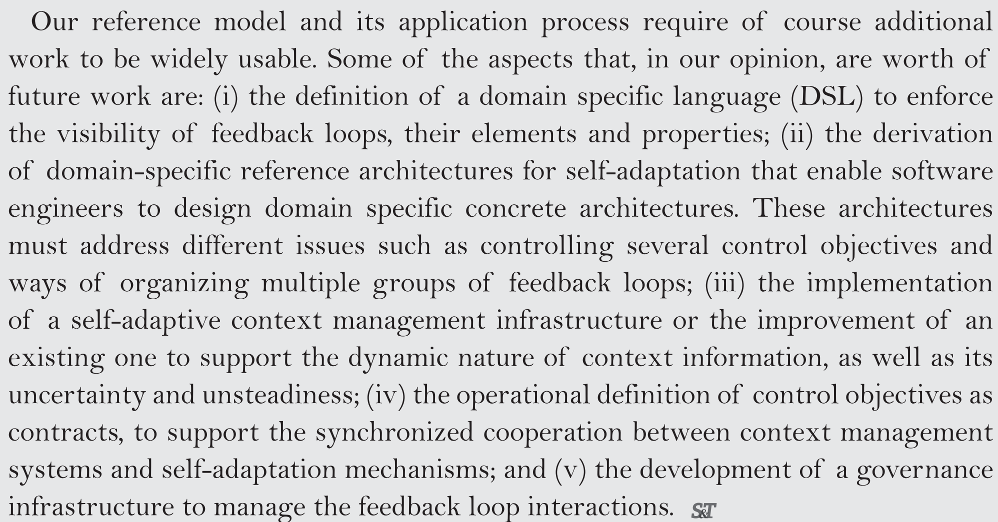
For the implementation of self-organizing systems, Caprarescu and Petcu (2009) proposed a decentralized autonomic manager composed of many independent lightweight feedback loops implemented as agents, where each agent is an implementation of a MAPE-K loop. Control objectives in this approach are specified 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 layers proposed by Kramer and Magee (2007). The system performs its adaptation based on a process of three phases. The first one separates agents into groups, by policy (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. However, although their approach makes the separation of multiple feedback loops explicit, the elements of each loop are highly coupled.

Finally, it is worth noting that none of the analyzed approaches address the preservation of context relevance. In our approach, this critical aspect is achieved by a dynamic self-adaptive infrastructure of monitoring that is explicitly maintained by the context adaptation controller feedback loop.

Conclusions and Future Work

Control theory, as a discipline matured over the past century, has condensed in its feedback loop reference model and corresponding variations the accumulated knowledge and experience of control engineers designing and building automated controllers for physical systems.

The main goal we address in this paper is to illustrate the application of a feedback loop-based reference model to the engineering of self-adaptive software systems. For this, we used a reference model we previously proposed for designing self-adaptive software systems where feedback loops are explicit components of the software architecture. Our reference model emphasizes the visibility of these control elements through the separation of three fundamental concerns: (i) the preservation of the system self-adaptive properties over time; (ii) the management of the dynamic nature of context management for supporting the continued relevance of the system with respect to its control objectives; and (iii) the dynamic system adaptation as the mechanism to guarantee system properties under changing



conditions of context. We illustrated how to apply this reference model to obtain the architecture of an conference management system with ubiquitous characteristics.

Our reference model and its application process require of course additional work to be widely usable. Some of the aspects that, in our opinion, are worth of future work are: (i) the definition of a domain specific language (DSL) to enforce the visibility of feedback loops, their elements and properties; (ii) the derivation of domain-specific reference architectures for self-adaptation that enable software engineers to design domain specific concrete architectures. These architectures must address different issues such as controlling several control objectives and ways of organizing multiple groups of feedback loops; (iii) the implementation of a self-adaptive context management infrastructure or the improvement of an existing one to support the dynamic nature of context information, as well as its uncertainty and unsteadiness; (iv) the operational definition of control objectives as contracts, to support the synchronized cooperation between context management systems and self-adaptation mechanisms; and (v) the development of a governance infrastructure to manage the feedback loop interactions. S&T

Acknowledgments

This work was funded in part by the National Sciences and Engineering Research Council (NSERC) of Canada (CRDPJ 320529-04 and CRDPJ 356154-07), IBM Corporation, CA Inc., and Universidad Icesi (Cali, Colombia).

References

Abid, Z., Chabridon, S., & Conan, D. (2009). A framework for quality of context management. In Quality *of context. First international workshop, QuaCon 2009, Stuttgart, Germany, June 25-26, 2009. Revised Papers* (LNCS 5786) (120-131). Berlín, Alemania: Springer-Verlag. DOI: 10.1007/978-3-642-04559-2

Bass, L., Clements, P., & Kazman, R. (2003). *Software architecture in practice* (2nd ed.). Boston, MA: Addison-Wesley.

Caprarescu, B.A, Petcu, D., (2009). A self-organizing feedback loop for autonomic computing. In *Proceedings Computation world 2009: Future computing, service computation, cognitive, content, patterns* (pp.126-131). Los Alamitos, CA: IEEE Computer Society

Cheng, B.H., Lemos, R., Giese, H., Inverardi, P., Magee, J., Andersson, J.,...Whittle, J. (2009). Software Engineering for self-adaptive systems: A research roadmap. In *Software engineering for self-adaptive systems* (LNCS 5525) (pp. 1-26). Berlín, Alemania: Springer-Verlag. doi: 10.1007/978-3-642-02161-9

Chignell, M., Cordy, J., Ng, J., & Yesha, J.

(Eds.) (2010). *The smart Internet. Current research and future applications* (LNCS 6400). Berlín, Alemania: Springer-Verlag. DOI: 10.1007/978-3-642-16599-3

Coutaz, J., Crowley, J.L., & Dobson, S. (2005). Context is key. *Communications of the ACM (48)*3, 49–53.

Frincu, M.E., Villegas, N.M., Petcu, D., Müller, H.A., & Rouvoy, R., (2011). Self-healing distributed scheduling platform. In *Proceedings IEEE international symposium on cluster, cloud and grid computing, CCGrid 2011* (pp. 225-234). Los Alamitos, CA: IEEE Computer Society

Garlan, D., Cheng, S-W., & Schmerl, B. (2003). Increasing system dependability through architecture-based self-repair. In *Architecting dependable systems* (LNCS 2677) (pp. 61-89). Berlín, Alemania: Springer-Verlag. doi: 10.1007/3-540-45177-3

Giese, H., Brun, Y., Serugendo, J.D.M., Gacek, C., Kienle, H., Müller, H.,… Shaw, M. (2009). Engineering self-adaptive and self-managing systems. In *Applied algebra, algebraic algorithms and error-correcting codes. 18th International symposium, AAECC-18 2009, Tarragona, Spain, June 8-12, 2009. Proceedings* (LNCS 5527) (pp. 47-69). Berlín, Alemania: Springer-Verlag. doi: 10.1007/978-3-642-02181-7

Hebig, R., Giese, H., & Becker, B., (2010). Making control loops explicit when architecting self-adaptive systems. In *Proceedings 2nd international workshop on self-organizing architectures* (pp. 21–28). New York, NY: ACM.

Hellerstein, J.L., Diao, Y., Parekh, S., & Tilbury, D.M. (2004). *Feedback control of computing systems.* Hoboken, NJ: John Wiley & Sons.

Hellerstein, J.L., Singhal, S., & Wang, Q., (2009). Research challenges in control engineering of computing systems. *IEEE Transactions on Network and Service Management (6)*4, 206.211.

IBM Corporation (2006). *An architectural blueprint for autonomic computing* (4th ed.) [Technical Report]. Hawthorne, NY: Autor

Kephart, J.O., & Chess, D.M., (2003). The vision of autonomic computing. *Computer (36)*1, 41–50.

Kramer, J., & Magee, J. (2007). Self-managed systems: an architectural challenge. In *Proceedings: 2007 workshop on the future of software engineering (FOSE 2007)* (pp. 259-268). Los Alamitos, CA: IEEE Computer Society

Müller, H., Pezzè, M., & Shaw, M., 2008. Visibility of control in adaptive systems. In *Proceedings 2nd international workshop on ultra-large-scale software-intensive systems, ULSSIS 2008* (pp. 23-26). New Yok, NY: ACM.

Müller, H.A., Kienle, H.M., & Stege, U., (2009). Autonomic computing: Now you see it, now you don't. Design and evolution of autonomic software systems. In *Software engineering. International summer schools, ISSSE 2006-2008, Salerno, Italy, Revised tutorial lectures* (LNCS 5413). (pp. 32-

54). Berlín, Alemania: Springer-Verlag. doi: 10.1007/978-3-540-95888-8

Ogata, K. (2010). *Modern control engineering* (5th ed.). Boston, MA: Prentice Hall.

Oreizy, P., Medvidovic, N., & Taylor, R.N., (2008.) Runtime software adaptation: framework, approaches, and styles. In *Proceedings 30th international conference on software engineering, ICSE 2008* (pp. 899-910). New Yok, NY: ACM.

Parra, C., Blanc, X., & Duchien, L. (2009). Context awareness for dynamic service-oriented product lines. In *Proceedings 13th intentaional software product line conference, SPLC 2009* (pp.131.140). New York, NY: ACM.

Salehie, M., & Tahvildari, L. (2009). Self-adaptive software: Landscape and research challenges. *ACM Transactions on Autonomous and Adaptive Systems (4)*2, 14:1-14:42. doi: 10.1145/1516533.1516538

Solomon, B., Ionescu, D., Litoiu, M., & Mihaescu, M. (2007). A real-time adaptive control of autonomic computing environments. In *Proceedings 17th annual international conference hosted by the Centre for advanced studies research, IBM Canada Software Laboratory, CASCON 2007* (pp. 124-136). New York, NY: ACM

Tamura, G., Casallas, R., Cleve, A., & Duchien, L. (2011a). QoS Contract-aware reconfiguration of component architectures using e-graphs. In *7th International workshop on formal aspects of component software, FACS 2010* (LNCS 6921) (pp. 34-52). Berlín, Alemania: Springer-Verlag.

Tamura, G., Villegas, N.M., Müller, H.A., Duchien, L., & Casallas, R. (2011b). A control-engineered reference model to optimize context relevance in self-adaptation. Recuperado de: https://connex.csc.uvic.ca/access/content/group/eac7abb3-27a0-4a53-be0f-10525cabe46e/Papers/control-based-reference-model-for-self-adapt.pdf

Truex, D.P., Baskerville, R., & Klein, H. (1999). Growing systems in emergent organizations. *Communications of the ACM (42)*8, 117–123.

Villegas, N.M., & Müller, H.A. (2010). Managing dynamic context to optimize smart interactions and services. In *The smart Internet: Current research and future applications* (LNCS 6400) (pp. 289-318). Berlín, Alemania: Springer-Verlag.

Villegas, N.M., Müller, H.A., Muñoz, J.C., Lau, A., Ng, J., & Brealey, C. (2011a [in press]). A dynamic context management infrastructure for supporting user-driven web integration in the personal web. In *Proceedings the 2011 conference of the Center for advanced studies on collaborative research, Canada (CASCON 2010)* (pp. 1-15). New York, NY: ACM.

Villegas, N.M., Müller, H.A., Tamura, G., Duchien, L., & Casallas, R. (2011b). A framework for evaluating quality-driven self-adaptive software systems. In *Proceeding 6th international symposium on Software engineering for adaptive and self-managing systems* (pp. 80–89). New York, NY: ACM.

Currículum vitae

Norha M. Villegas

Ph.D. Candidate under the supervision of Dr. Hausi A. Müller, Department of Computer Science, University of Victoria, Canada. She is a CAS student at the Center for Advanced Studies at the IBM Toronto Laboratory (2010-2011). Her dissertation focuses on the application of dynamic context management to the optimization of self-adaptive software systems. Her research interests include control theory, autonomic computing, dynamic context management, context-awareness, semantic web, and service-oriented systems. Norha Villegas received a Diploma Degree in Systems Engineering and a Graduate Degree in Organizational Informatics Management in 2002 and 2004, from Universidad Icesi, Cali, Colombia

Hausi Müller, Ph.D.

Professor, Department of Computer Science and Associate Dean of Research, Faculty of Engineering at University of Victoria, Canada. He is a Visiting Scientist at the Center for Advanced Studies at the IBM Toronto Laboratory (CAS), CA Canada Inc., and the Carnegie Mellon Software Engineering Institute (SEI). Dr. Müller 's research interests include software engineering, self-adaptive and self-managing systems, context-aware systems, and service-oriented systems. He serves on the Editorial Board of Software Maintenance and Evolution and Software Process: Improvement and Practice (JSME). He served on the Editorial Board of IEEE Transactions on Software Engineering (TSE) 1994-2000, 2005-2009). He is Chair of the IEEE Technical Council on Software Engineering (TCSE). Dr. Müller received a Diploma Degree in Electrical Engineering in 1979 from the Swiss Federal Institute of Technology (ETH), Zürich, Switzerland and MSc and PhD degrees in Computer Science in 1984 and 1986 from Rice University in Houston, Texas, USA.

Gabriel Tamura, M.Sc.

PhD student in co-supervision between University of Los Andes, Bogotá, Colombia, and University of Lille 1, Lille, France, and a member of the INRIA-USTL-CNRS team-project ADAM (Adaptive Distributed Applications and Middleware) and the Software Construction Research Group. Gabriel Tamura obtained his M.Sc. degree in Systems and Computing Engineering from Universidad de Los Andes, Bogotá, Colombia, in 1996, and his professional degree in Computing Engineering from Universidad Javeriana, Cali, Colombia. His current research interests include the engineering of context-aware self-adaptive software systems and the evolution of component-based and service-oriented computing.