Control Theory-Based Foundations of Self-Controlling Software

Mieczysław M. Kokar
Department of Electrical and Computer Engineering,
Northeastern University

Kenneth Bacławski
College of Computer Science,
Northeastern University

Yönet A. Eracar
Department of Mechanical, Industrial and Manufacturing Engineering,
Northeastern University

Abstract

In spite of the obvious analogy with control systems, the basic paradigm of control has not found its place as a first-class concept in the area of software engineering. The controller concept has been used in software engineering as an architectural style and also as a design pattern. However, the architectural style only considers the possibility that the controller would be implemented in software, while the possibility that the plant itself might be software is not considered. Furthermore, this architectural style does not go beyond the basic feedback loop model; neither adaptive nor reconfigurable control models are considered. The controller design pattern does consider the possibility of a software plant, but it only uses the open-loop model: feedback is not considered at all.

We propose a new paradigm in the development of software in which the issue of self-control of software is addressed explicitly and systematically in the software development process. This new paradigm combines the advantages of control theory and software engineering. A series of progressively more sophisticated software models are introduced, based on corresponding models from control theory. The essence of a control model is one or more feedback loops; we discuss the various loops that occur for software systems. We then introduce a new model called the Self-Controlling Software Model that supports three levels of control: feedback, adaptation and reconfiguration.

Control and adaptation embedded into algorithms are commonplace in software systems. For example,

- Software for dynamic adjustments of the buffering strategy in a database management system;
- Routing algorithms for networks;
- Load balancing algorithms for distributed computer systems;
- Graphical User Interfaces (GUI) that adapt to a specific user;
- Caching strategies for memory management in operating systems.

The main ideas of the control-theory based paradigm for self-controlling software are:

- The basic function of the software system is regarded as a *Plant* to be controlled.
- The behavior of the Plant and the Environment is modeled as a *dynamic system*.
- Measurable inputs to the Plant are identified and split into *control inputs* and *disturbances*. Control inputs are used for controlling the behavior of the Plant, while disturbances alter the behavior of the Plant in an unpredictable way.
- An additional subsystem is added for changing the values of the control inputs to the Plant, called the *Controller* subsystem.
- Yet another subsystem can be added for computing feedback, called the *Quality of Service (QoS)* subsystem. This feedback is used by the Controller to compute control inputs.

The first control approach we consider is called *open-loop control*. In this approach, the control input value is selected according to a *control law* that calculates the control input based upon the values of other inputs. The control law is part of the Controller. The open-loop control model splits the system into a Plant and Controller.

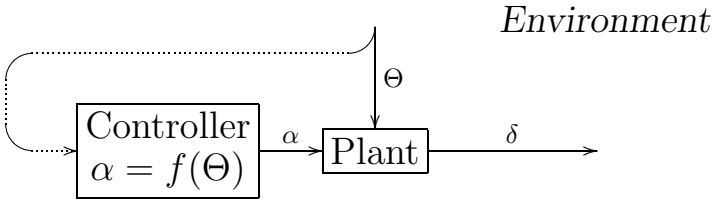

Figure 1: Open Loop Model

The next example of a control model is the *Closed-Loop* or *Feedback* Control Model. In this model the output of the Plant is explicitly and immediately fed back to the Controller as shown in Figure 2.

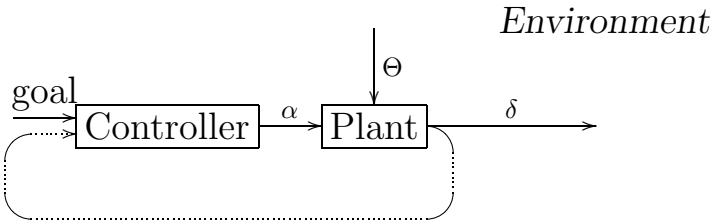

Figure 2: Conventional Feedback Control Architecture

In some cases this value can be computed as a function of input and output variables. In some other cases it might take an external input (e.g., from the user) and then compute the value of feedback based on all available information. In this model, the Controller uses the feedback produced in the QoS to control the Plant. This model is shown in Figure 3.

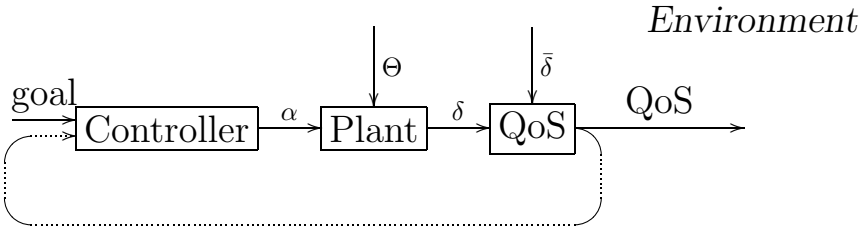

Figure 3: Feedback Model with Quality of Service Subsystem

Figure 4 shows the software model based on the indirect adaptive control approach. This software model includes two subsystems in addition to those

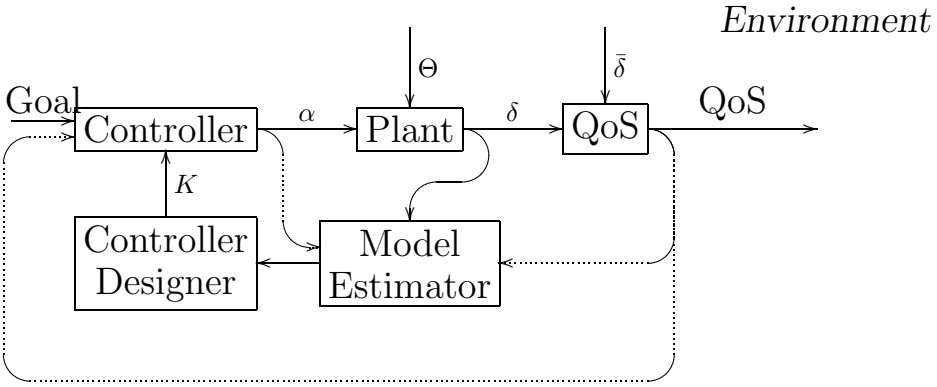

Figure 4: Indirect Adaptive Control Model

that appear in the feedback control model: *Model Estimator* and *Controller Designer*. In this model, the Controller parameters are adjusted based upon the Plant's model that is being updated during execution.

Adaptive control, although more flexible than conventional feedback control, has its own limitations. *Reconfigurable control* is a relatively new model in the design and implementation of control systems. The driving force behind the development of this approach was the need for controlling Plants that change their dynamics structurally in an unpredictable fashion. The software model based upon a reconfigurable Controller is represented in Figure 5.

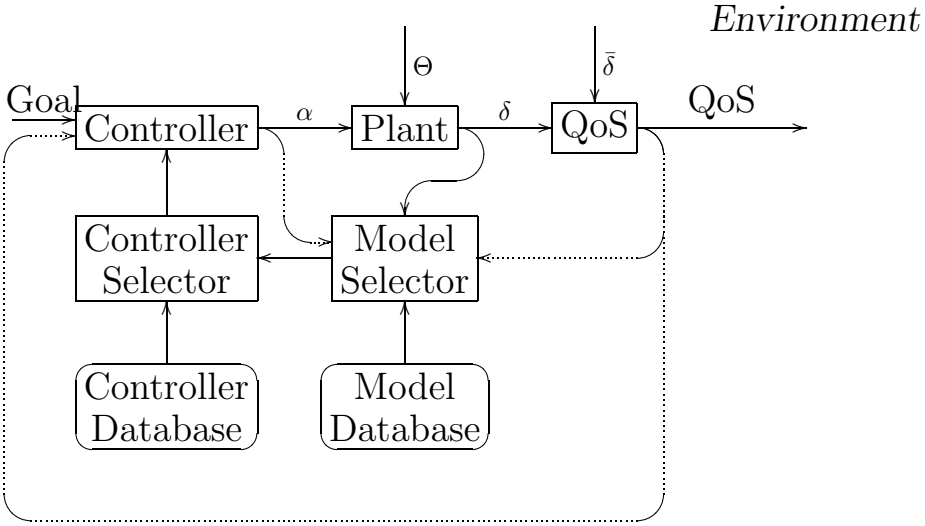

Figure 5: Reconfigurable Control Model

We now introduce a software control model that combines and generalizes the features of the various control models introduced above. This new model is called the *Self-Controlling Software Model*, as shown in Figure 6.

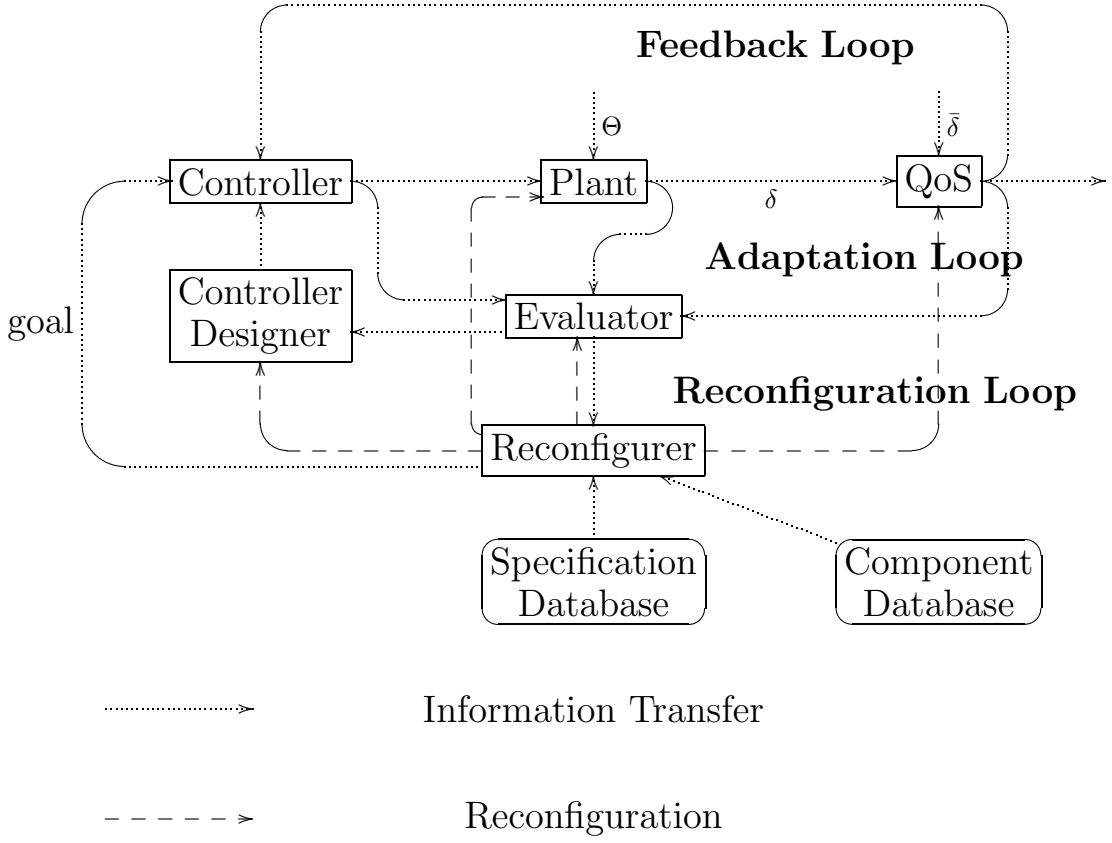

Figure 6: Self-Controlling Software Model

The structure of this model can be best described by the three loops each of which represents a different time-scale for control activity:

1. The *feedback loop* in which the Controller sets parameters to the Plant based upon goal and feedback received from the Quality-of-Service subsystem.
2. The *adaptation loop* in which the Evaluator evaluates the behavior and performance to determine whether the model of the Plant is appropriate, and adapts the model, which in turn triggers a change in the control law.
3. The *reconfiguration loop*, which is a drastic and relatively costly action. This action is performed by the *Reconfigurer* on request of the Evaluator. The reconfiguration can involve structural changes in the Plant model, Quality-of-Service, Evaluator, Controller, Controller Designer, goal, or even the Plant. We assume that the Reconfigurer itself remains fixed. The Reconfigurer in its decision making process uses the Specification Database, which contains a high-level system requirement, including a high-level goal. In the reconfiguration planning process it uses the Component Database to assemble various elements of the system.

Issues in the Use of Control Theory for Software Engineering

Controllability This is the ability to steer the system (Plant) in desired directions.

Observability This refers to the ability to determine the (initial) state of a system from measurements of the system.

Stability In simple terms it is a property of the system which ensures that small changes (disturbances) in an initial state (also called an *equilibrium state* or *invariant set*) eventually have negligible effects upon the behavior of the system.

Robustness This property is the ability of the controller to achieve its objectives even if there are large, unanticipated variations in the Plant.

Generality The generality of any system is limited by its knowledge base. The Self-Controlling Software Model is amenable to any number of control strategies, such as *expert control*, *neural* and *fuzzy control*, *hybrid control* and *learning control*.

Autonomy Since a self-controlling system performs reconfiguration without direct supervision, it exhibits a great deal of autonomy.

Chattering When the environment happens to reach a state that is on the boundary between two control regimes, the system may reconfigure repeatedly between two or more configurations.

Scheduling The scheduling of components is important for any system model.

Proactive Reconfiguration

Efficiency The redundancy necessary for self-adaptability adds a cost to the system.

Conclusion

With very few exceptions, reconfiguration of a software system requires the system to be shut down and must be supervised manually. Software that can be reconfigured or even just upgraded at runtime is unusual, and research on this issue is relatively recent. The systematic use of the control theory based paradigm can lead to software development models that could alter this situation by making it much easier to design and develop software with the following capabilities:

Capability	**Modules in Figure 6**
Knows its purpose and structure.	Controller and Controller Designer
Evaluates its behavior and performance at runtime.	QoS and Evaluator
Can reconfigure itself at runtime in response to a variety of criteria such as a failure to accomplish its objectives due to changes in the inputs obtained from the environment.	Reconfigurer using Component Database
Can generate code at runtime to match a component with another component that needs to use it.	Reconfigurer using Specification Database