A Classification Scheme for Self-adaptation Research

Matthias Rohr, Simon Giesecke, Wilhelm Hasselbring
Software Engineering Group, Carl von Ossietzky University of Oldenburg, 26111 Oldenburg, Germany
Marcel Hiel, Willem-Jan van den Heuvel, Hans Weigand
InfoLAB, University of Tilburg, 5000 LE Tilburg, The Netherlands

The Need for a Classification Scheme

Research on self-adaptation in computer science is distributed across many subdisciplines. Each subdiscipline regards self-adaptation from a restricted point of view and fundamental cross-domain research is essentially missing. This leads to the reinvention of concepts and impairs the ability to establish a structured research program on self-adaptation. Therefore, we introduce a classification scheme that allows to structure self-adaptation research, to compare approaches from different domains, and to identify related research.

What is Self-Adaptation?

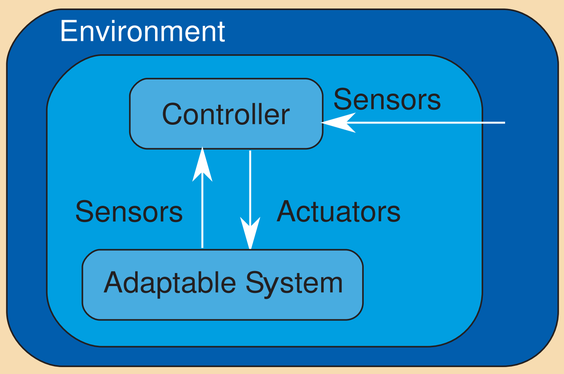
A software entity has the capability of self-adaptation if it can change its structure or behavior, based on observations of the system or its environment. The figure shows the separation between the adaptable system and its controller. The controller observes the system or the environment by sensor and adapts the primary operation under certain circumstances by using actuators.

Goals

Typical goals are to maintain or improve the Quality of Service, dependability, design efficiency, interoperability, or to reduce maintenance costs.

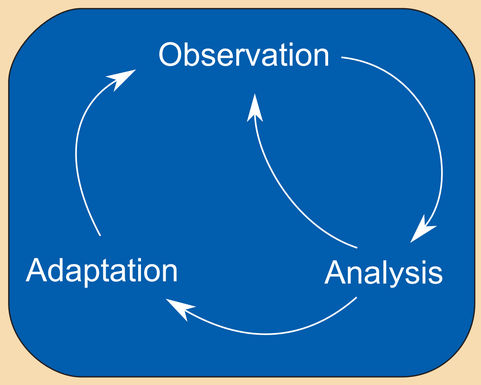
Risks

Self-adaptation can significantly increase the system complexity and the risks for unintended behavior.



The Adaptation Cycle

Self-adaptation can be structured as a cycle of three activities: Observation, analysis, and adaptation.



1. Observation

To perform measurements, a trade-off between information quality (detail and timeliness) and caused performance overhead must be made. Detailed and frequent monitoring is often computationally too expensive for normal operation. Additionally, a large overhead of redundant or useless data can make the evaluation more complex. For a sophisticated maintainable self-adaptive system, it is required to design a monitoring model in a systematic process to ensure that the right measurements are made. The monitoring model should prescribe when, what, and where data is acquired, as both for the interpretation of the monitored data and for the implementation and maintenance of the software, it is important to know the context of monitored data.

When: Time of model update.

What: Object of measurement in terms of metrics (e.g., service response time in ms). A systematic way to derive metrics is the Goal-Question-Metric (GQM) approach.

Where: Points of data collection in the system architecture (or environment).

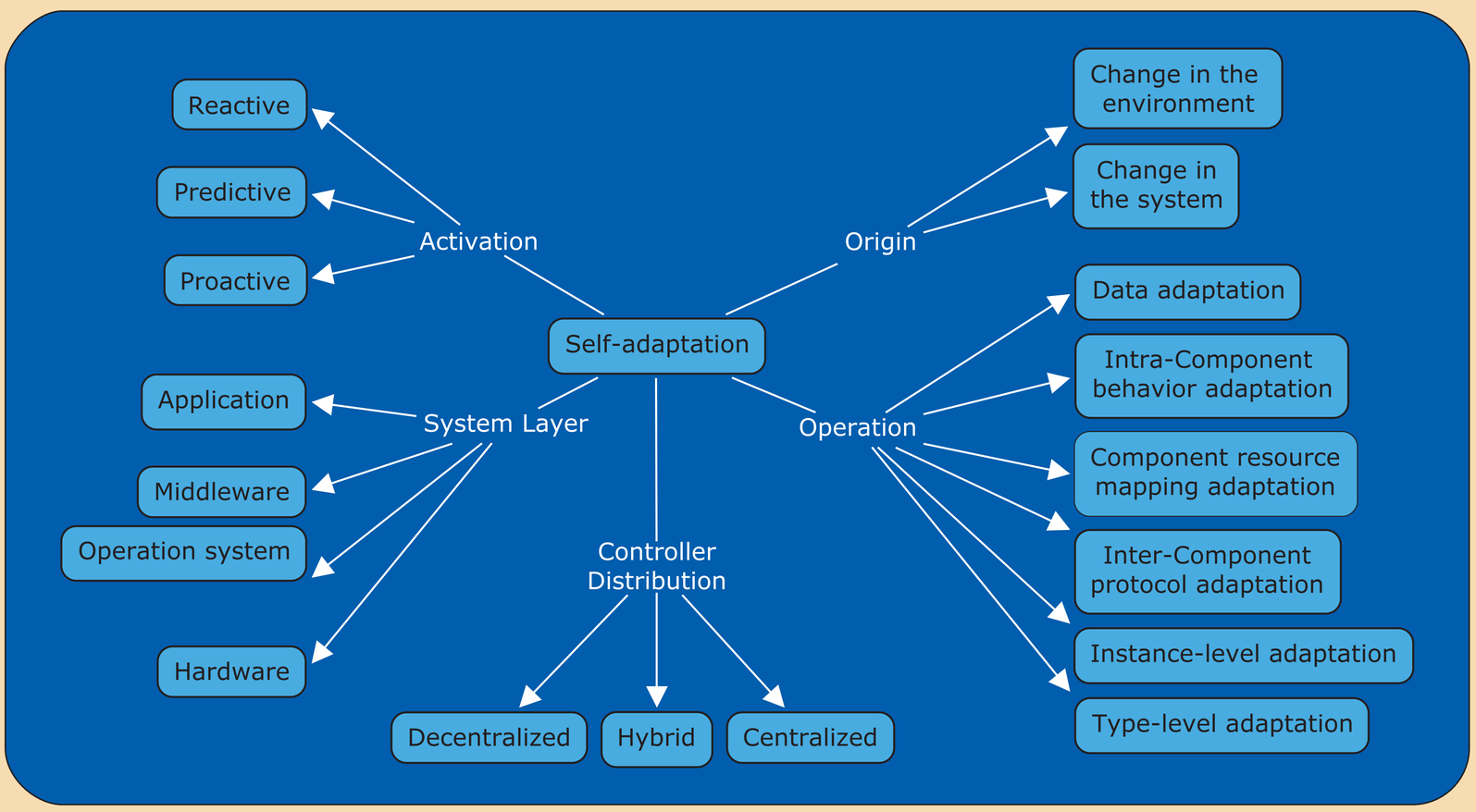
In practice, monitoring logic is often mixed with primary application logic. As this reduces the maintainability of the source code, the instrumentation method should honor that monitoring is a cross-cutting concern of system behavior. Two major alternative instrumentation techniques are middleware interception and source code instrumentation (e.g., via aspect oriented programming (AOP)). Middleware interception manipulates or extends the middleware platform (for instance, the application server or the virtual machine), so that component interaction is automatically observed.

2. Analysis

The analysis activity summarizes several sub-activities related to the processing of the observations and the identification and selection of adaptation operations. A straightforward design could use simple rules (e.g., “if disconnected, then reconnect”). Such rules contain constraints that specify adaptation operations for a particular subset of observations. When an observation violates one of the constraints then a predefined operation will be executed. However, even simple rules can cause complex and unpredictable behavior.

3. Adaptation

The adaptation activity is executes the selected adaptation operation(s). It may happen that non-tested system configuration result from an adaptation. This introduces risks for unexpected system operation. The execution of the operation itself can be technically challenging for the execution environment as it is often desired that the adaptation takes place transparently. This means that the adaptation is executed during runtime, without an interruption of primary system operation. To provide sufficient dependability, it is a good strategy to design the system under consideration of adaptation faults. This could involve the supervision of the execution of the adaptation operation, the creation of recovery points, or the step-wise adaptation on a redundant copy of the system.



Activation

Regardless of the origin of a state change, it is detected through the help of sensors and the system must respond to these changes. We distinguish three different activation types or response types, i.e., reactive, predictive and proactive. To describe the differences between the activation types, we use performance as an illustrative example system attribute:

Reactive: The system adapts only after the performance of the system has dropped under a certain level.

Predictive: If a certain state or a certain observation occurs before a drop of performance, then the system might adapt to this at once rather than waiting for the actual drop.

Proactive: The system has a normal performance level but the system decides to adapt itself to gain a better performance. This may also be termed goal-directed self-adaptation, and is closely related to self-optimization. It may occur at any time.

Origin

The origin is the location of the state change that triggers an adaptation cycle. In biology, this is called the stimulus (state change) and the response (adaptation operation). The stimulus can either originate from inside the system or from the environment.

This separation between system and environment can be made more specific: In a particular system, it can be beneficial to model elements of the environment explicitly if they are the source of many changes. For instance, in a system with a lot of human interaction, it can be helpful to model the possible changes that a user might desire.

Operation

We use an Architectural Description Language (ADL) view on the adaptation operation. Multiple component instances can belong to a component type, and the implementation of a component instance can change without affecting the associated component type. We distinguish six types of adaptation operations according to the architectural perspective:

Data adaptation: Data values are changed in the system data model without effecting the control flow within or between the system components

Intra-component behavior adaptation: Changes the behavior (control flow or quality of service) of a component without directly affecting component execution sequences

Component resource mapping adaptation: Changes the association of software components and resources (data source are here not considered as resources)

Inter-component protocol adaptation: Changes the dynamic communication between components in a fixed structure

Instance-level adaptation: Changes the structure by adding or removing component instances (but not component types)

Type-level adaptation: Changes the component types, which allows to define new component types (the component dependency graph is affected)

Controller Distribution

Three controller distribution styles can be distinguished according to the architectural localization of self-adaptation: centralized, decentralized, and hybrid. The controller distribution style is centralized if the adaptation is controlled and executed from a central point in the system. A decentralized system has no such single point of failure and executes self-adaptation (all activities) locally. Most software agent architectures would be considered as decentralized. Hybrid approaches combine the advantages of a centralized and decentralized architecture by applying parts of the activities locally and other parts globally.

System Layer

It is common to distinguish at least four fundamental layers of computing systems: hardware, operation system, middleware, and application. Self-adaptation is a general concept that can be applied at any layer, e.g. regarding the adaptation of non-functional system attributes such as timing behavior. For simplicity, the system layer classification should focus on the layers where the adaptation operation is applied.

The other aspects of self-adaptation, especially observation, are related to one or more system layers, as well, and additionally, might even operate on different layers than the adaptation operations. For instance, application behavior could be monitored on the application layer, while the reconfiguration follows on the hardware layer. However, the adaptation activity requires usually more effort during development and operation than just observation.

This work is supported by the German Research Foundation (DFG), grant GRK 1076/1