Architectural Design Spaces for Feedback Control Concerns in Self-Adaptive Systems

Sandro S. Andrade†§
sandros@ufba.br
§GSORT Distributed Systems Group
Federal Institute of Education, Science and Technology of Bahia (IFBa)
40110-150. Salvador-Ba. Brazil

Raimundo José de A. Macêdo†
macedo@ufba.br
†Distributed Systems Laboratory (LaSiD)
Department of Computer Science
Federal University of Bahia (UFBa)
40170-110. Salvador-Ba. Brazil

Abstract

A lot of current research efforts in self-adaptive systems community have been dedicated to the explicit modeling of architectural aspects related to system self-awareness and context-awareness. This paper presents a flexible and extensible representation of architectural design spaces for self-adaptation approaches based on feedback control loops. We have defined a generic representation for design spaces metamodeling and have instantiated it in order to provide direct support for early reasoning and trade-off analysis of self-adaptation aspects with the aid of a set of feedback control metrics. The proposed approach has been fully implemented in a supporting tool and a case study with a distributed industrial data acquisition service has been undertaken. Whilst preliminary experiences with the proposed approach indicate useful reasoning support when comparing alternative design solutions for self-adaptation, further investigation regarding scalability aspects and automatic handling of conflicting goals has been identified as future work.

Keywords: software architecture design, self-adaptive systems, feedback control, model-driven software engineering.

1. Introduction

The increasing cost of handling complexity and changing environments in modern large-scale distributed systems has motivated significant efforts [15] towards the design and development of self-adaptive systems [7]. Architecture-based approaches with explicit (first-class) modeling of feedback control loops [5, 9, 13] have currently been advocated as a promising research landscape, establishing the foundations of domain-independent mechanisms and enabling early reasoning about self-adaptation quality attributes.

In addition, software engineering researchers have been urged to drive their efforts towards an engineering discipline for software [16], which mostly involves the prospection of theories [8, 18] and organization of knowledge for routine use [3]. In particular, the use of design theories [11] for self-adaptive systems is still on its infancy.

A great number of diverse feedback control schemes for regulating and optimizing distinct software performance variables have been proposed over the last past years [14]. Despite some successful achievements of control goals under disturbances and environment uncertainties, most of these work propose ad-hoc domain-specific feedback control approaches. Software architects willing to endow their systems with self-adaptive behavior still have no systematic guidance about choosing among different feedback control strategies, evaluating alternative design trade-offs, and assessing the resulting self-adaptation quality attributes.

In this paper, we present DuSE†, which is an extensible, navigable and machine-consumable representation of architectural design spaces for self-adaptation feedback control-based approaches. Our approach entails: i) a set of effectors which enable architecture model manipulation when navigating through the design space; ii) a group of OCL (Object Constraint Language) rules to verify the availability of variation points in a given design space dimension; and iii) a set of metrics for evaluating self-adaptation quality attributes in a given design space location.

The proposed architectural design space infrastructure relies on three models which address different supporting aspects: i) an architectural model (logical view) of the system to be endowed with self-adaptive behavior; ii) an auto-regressive with exogenous input (ARX) model [10] representing the system dynamics, and iii) the design space model describing the dimensions and associated architecture manipulations.

† This research is partially supported by grant 006/2012/PRPGI from the IFBa’s Research and Innovation Supporting Program.

http://duse.sf.net
The target system’s architectural model is an UML representation annotated with a specific UML profile which enables the identification of available control parameters and performance variables. Such an UML model provides the means to accomplish automatic black-box system identification, yielding the ARX model which supports the use of self-adaptation quality metrics to guide navigation through design space.

We have implemented the proposed platform on top of the Qt Modeling Framework\(^2\) as a multi-platform tool with a flexible architecture which enables its use in a range of technologies for the target system. The quality and feasibility of the architectural designs resulting from design space navigation have been evaluated in a case study undertaken on top of CIAO (Component Integrated ACE ORB) middleware.

The remainder of this paper is organized as follows. Section 2 presents the most prevalent design dimensions when using feedback control for achieving self-adaptation. Section 3 describes the proposed approach, supporting technologies, and implementation aspects. The evaluation of our approach when supporting reasoning about self-adaptation design alternatives for a distributed system based on CIAO is presented in section 4. Section 5 provides pointers to related work and section 6 draws some conclusions and presents future work.

2. Design Dimensions in Self-Adaptive Systems

Making design decisions concerned to system self-awareness, environment monitoring, adaptation strategies, and enacting mechanisms constitutes a central activity when developing self-adaptive systems. Preliminary studies about design dimensions for self-adaptive systems [1, 5, 7, 14] have identified degrees of freedom related to run-time system representation, system/environment observation, control strategies, adaptation identification, and adaptation mechanisms.

Whilst defining a comprehensive set of design concerns is paramount to further understand solution opportunities and analyze trade-offs, in this work we focus on sub-dimensions related to control strategies, control schema, and their impacts on overall system quality attributes. When dealing with such control aspects, architects should make decisions, for instance, about target system modeling (e.g., first-principles vs black-box models), control strategy (e.g., fixed-gain PID, adaptive model identification, model predictive), and control schema (e.g., single loop, nested control, hierarchical control, decentralized control) [10].

Although any guideline for selecting an effective feedback control solution from that design space would still require a careful assessment against local domain/requirements drivers, we believe the use of flexible and powerful tools to support trade-off analysis of alternative control designs may further advance the use of effective model-based approaches.

3. The Proposed Approach

The feedback control architectural analysis environment we propose defines a model-based meta-architecture which includes: i) a self-adaptation design space representation; ii) changes to be enacted in target system’s original model when navigating through the design space; iii) OCL rules which define valid variation points in each dimension; and iv) a set of metrics to guide architects during design trade-off analysis.

As depicted in Figure 1, DuSE provides a technology-independent framework for reasoning about feedback-control approaches by combining run-time target system dynamics (represented by a first-order ARX model), target system’s structural architecture XMI descriptions, and a design space representation.

A design space [4, 17] is represented as a tuple \(DS = <DD, M>\), where \(DD\) is a set of design dimensions and \(M\) is a set of metrics defined for that design space. A design dimension \(DD_i\) is a set of variation points \(VP_{ij} = <CT_{ij}, CH_{ij}>\), where \(CT_{ij}\) is a set of OCL constraints that must be satisfied in the target system’s original architecture model in order to that design dimension point be available for use, and \(CH_{ij}\) is a set of changes to be enacted in the target system’s original model \(SM_0\).

Therefore, a specific design space location \(<VP_{1i}, VP_{2j}, ..., VP_{DD_i}|m>\) is only defined for \(SM_0\) if \(SM_0\) satisfies the set of constraints \(CT_{1i} \cup CT_{2j} \cup ... \cup CT_{DD_i}|m\). Likewise, the changes to be enacted in the original model \(SM_0\) for a given design space location \(<VP_{1i}, VP_{2j}, ..., VP_{DD_i}|m>\) are the merge of all architectural contributions from each design dimension location: \(CH_{1i} \cup CH_{2j} \cup ... \cup CH_{DD_i}|m\). We denote by

\[\text{Figure 1. DuSE overview}\]
SM_{i,\ldots,m} the architectural model resulting after applying contributions from all specific locations in each design space dimension. A metric M_i is a tuple <$M_{E_i}, MG_i>$, where $M_{E_i}$ is the metric evaluation expression and $MG_i$ describes if the metric is intended to be maximized (1) or minimized (−1). Metrics $M_1$ and $M_3$ can be evaluated at any design space location defined for the original $SM_0$ model. Metric $M_2$ is undefined when $DD_1$ component is $VP_1$ since setting time information is not available off-line.

### 3.1. Analysing Feedback Control Strategies

In order to support early reasoning and analysis of feedback control strategies, we have instantiated our design space model in terms of design dimensions, architectural changes, metrics, and constraints directly related to most prominent feedback control concerns.

As presented in Table 1, we have defined two initial design dimensions related to feedback control loops in self-adaptation. Dimension $DD_1$ captures the robustness of feedback loops and provides two variation points: i) $VP_1$; fixed-gain, which uses a set of pre-defined parameters systematically derived to work on a specific region; and ii) $VP_2$; MIAC, a sort of adaptive control that continuously identifies the target system and reconfigures control parameters. Deciding about such dimension involves a trade-off analysis between robustness vs control overhead. Dimension $DD_2$ represents the nesting degree of feedback loops. Hierarchically interacting loops have successfully been applied to support self-management in different scopes, simplifying the handling of non-linearities and widening control possibilities. On the other hand, hierarchical feedback control requires a more careful analysis so that the time-scales of loops at different levels do not interfere each other and control scalability is kept at accepted levels. The corresponding OCL rules and architectural changes are also presented in Table 1.

<table>
<thead>
<tr>
<th>Design Dimension</th>
<th>Variation Points</th>
<th>OCL Constraints</th>
<th>Architectural Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(DD_1)$ Control Adaptness: refers to the moment (design-time, run-time with predefined alternatives, run-time with full control reconfiguration) in which control tuning is undertaken</td>
<td>$(VP_1)$ Fixed Gain: PID control with pre-defined gain values</td>
<td>$(CT_{111})$ allOwnedElements() -&gt; selectAsType(Component).provided -&gt; exists(extensionMonitorInterface.-&gt;notEmpty())</td>
<td>$(CH_{111})$ nPack.addOwnedType(s = sensorFactory(&quot;QPSensor&quot;, target.mInterface()));</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$(CT_{112})$ allOwnedElements() -&gt; selectAsType(Component).provided -&gt; exists(extensionMonitorInterface.-&gt;notEmpty())</td>
<td>$(CH_{112})$ nPack.addOwnedType(a = actuatorFactory(&quot;QPActuator&quot;, target.chInterface()));</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$(VP_2)$ Model-Identification Adaptive Control (MIAC): on-line model estimation and control tuning</td>
<td>$(CH_{121})$, $(CH_{122})$ and $(CH_{123})$ as defined in $(CH_{111})$, $(CH_{112})$ and $(CH_{113})$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$(CT_{121})$ as defined in $(CT_{111})$</td>
<td>$(CH_{124})$ nPack.addOwnedType(e = new QARXEstimator(s, a));</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$(CT_{122})$ as defined in $(CT_{112})$</td>
<td>$(CH_{125})$ nPack.addOwnedType(t = new QPIDFieldPlacement(e, c, st));</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$(CT_{211})$ as defined in $(CT_{111})$</td>
<td>$&lt;$no changes required$&gt;$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$(CT_{212})$ as defined in $(CT_{112})$</td>
<td></td>
</tr>
<tr>
<td></td>
<td>$(VP_2)$ d nested loops</td>
<td>$(CT_{2d1})$ allOwnedElements() -&gt; selectAsType(Component).-&gt; select(shortestControllableChain().size())=d</td>
<td>$(CH_{2d1})$ var i = 0;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$(CT_{2d2})$ post: target.loop.stime() &gt; target.longestControllableChain().collect(stime()).sum()</td>
<td>while(i &lt; d) {</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>instantiateControlLoop(currentTarget);</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>currentTarget = currentTarget.-&gt;nestingControllableComponent();</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>}</td>
</tr>
</tbody>
</table>

Table 1. Feedback control design space dimensions

Table 2 presents the defined architectural metrics. Metric $M_1$ captures how much of control adaptiveness exists in the system. A highly control-adaptive system exhibits higher robustness, with the cost of increasing control overhead due to continuous system identification and control tuning.

Metric $M_2$ estimates high discrepancies in hierarchical loops. In order to prevent control interference, hierarchical loops should be time-separated and the outer loop time-scale is usually dominated by the slower inner loop stabilization time. $M_2$ measures the highest difference between the slower and faster inner loops at level $d$. Levels with high values in $M_2$ should be further designed if faster self-adaptation is to be achieved.

Finally, $M_3$ measures how much control operation depends upon a single top-level controller. High values for $M_3$ indicates better control scalability although probably demanding a more complex control programming model. A design solution is a function that maps each lowest-level
4. Case Study

The definition of new design spaces, its dimensions, variation points and corresponding OCL constraints and metrics, as well as the analysis of feedback control approaches against a specific target system model are currently available in DuSE-MT - our supporting tool. DuSE-MT supports the workflow depicted in Figure 1 by defining a flexible architecture which enables the use of connector plugins for system identification in a variety of platforms. Each connector plugin enables the probing of target systems developed for that specific platform, gathering input/output relationships between system’s controlled/measured parameters and allowing offline and on-line system identification. DuSE-MT currently provides connector plugins for the CIAO and Qt/DBus technologies.

A case study aimed at analyzing feedback control loops in a CORBA-based data acquisition service has been undertaken. The ARCOS platform [2] is a flexible component-based framework for industrial data acquisition, implemented on top of CIAO middleware. This case study investigated alternative feedback control approaches aimed at controlling service response time in the presence of disturbances, by adjusting three component parameters: thread pool’s size, data cache size, and thread pool priority.

Figure 2 illustrates the initial non-adaptive architectural model provided as input in the case study. The case study’s goal was to investigate to which extent our platform helps the architect when choosing among different control strategies and provides insights for trade-off analysis.

For that purpose, a running instance of the data acquisition server was probed by DuSE-MT and ARX models for the three controllable components (ThreadPool, CacheManager, and DAISLeaderFollowers) have been generated. In that particular scenario there are $2^d$ feedback control strategies available. Each of those solutions exhibits different values for the defined self-adaptation metrics and, therefore, favors different quality attributes. For those cases where a running instance/prototype of the original non-adaptive system is not available, metrics $M_1$ and $M_3$ still provide valuable guidance when choosing among alternative candidate architectures.

Figure 3 shows the values for metrics $M_1$ and $M_3$ in the case study. Whilst obviously there may exist no single solution which fully maximizes all involved metrics, the obtained results may serve as helpful subsidies to support and well-inform the adoption of a specific feedback control architec-

\[
\begin{align*}
\text{(M1) Control Overhead} & : ME_1 = \text{allOwnedElements}() \to \text{selectAsType}(\text{QARXEstimator}) \to \text{size}() \\
\text{(M2) Maximum Setting Time} & : ME_2 = \text{product}() \to \text{iterate}(\text{Tuple(first:QPIDController, second:QPIDController)}), \\
& \qquad \text{maxDiff:Integer}=0 \\
& \qquad \maxDiff(\text{first.stime}() - \text{second.stime}(), \text{abs}()) \\
\text{select}() \qquad \text{allNestingLoops}() \to \text{size}()=l \\
\text{QPIDController.stime}()=4/\log_{10}(\max\{p_i\}) \\
\text{allOwnedElements}() \to \text{selectAsType}(\text{QPIDController}) \\
\text{(M3) Control Decentralization} & : ME_3 = \text{allOwnedElements}() \to \text{selectAsType}(\text{QARXEstimator}) \to \text{size}()=0 \\
& \qquad \text{iterate}(\text{c.QPIDController}), \\
& \qquad \text{minDepth:Integer}=1 \\
& \qquad \text{minDepth.min}(1/\text{allNestedLoops}() \to \text{size}()=1) \\
\text{allOwnedElements}() \to \text{selectAsType}(\text{QPIDController}) \\
\end{align*}
\]

Table 2. Feedback control design space metrics
Figure 3. Metrics M1 and M3 in case study

Figure 3. Metrics M1 and M3 in case study

5. Related Work

Over the past years some approaches for early architectural reasoning by explicit modeling of control loops have been proposed. In [20], a reference model for self-adaptation (FORMS) is presented. FORMS provides an unified view which integrates perspectives from reflexive, distributed computing, and MAPE-K [15] technologies. FORMS elements are described in Z specification language, which enables the formal reasoning of self-adaptation properties.

The use of megamodels at run-time to describe self-adaptation behavior is presented in [19]. The proposed notation entails the definition of multiple interacting feedback loops and relies on a model interpreter to dynamically adjust the adaptation logic. An UML Profile for feedback control modeling is presented in [9], along with a set of well-formedness rules to validate control schemas. Guidelines for modeling of multiple loops and elicitation of loop interferences are also presented. Actor-based approaches [12] and notations for self-adaptation with multiple objectives [6] have also been proposed.

Although some of the aforementioned proposals - such as FORMS - provide expressive and rigorous notations for feedback control modeling, they are mostly based on non-standardized and/or low-parsimony languages, provide no tool support, and still heavily depends on architect’s skills, as a consequence of the lack of explicit design space representations.

Our work tries to overcome some of these shortcomings by rather proposing an architectural analysis environment based on MOF (Meta Object Facility) and UML (Unified Modeling Language) standards, an explicit domain-specific design space representation, and a set of effective metrics to assess adaptation-related architectural quality attributes. Finally, we observe that our tool leverages rapid modeling/analysis and the proposed design space representation is also liable to be applied in other application domains since we aim at a proper balance between modeling notation generality and expressiveness.

6. Conclusion and Future Work

The design and development of large-scale distributed systems with flexible and robust self-adaptation capabilities have become a promising approach to cope with continuous increases in system complexity. Furthermore, stringent demands to provide quality of service assurances in unpredictable and uncertain environments introduce additional challenges in such scenarios.

In this paper, we have presented DuSE - a flexible analysis environment for representing and comparing alternative architectural design choices related to feedback control approaches to self-adaptation. We have described the underlying design space model representation and its instantiation devoted to support early reasoning of architectural trade-offs regarding control loop robustness and interaction.

Our reference implementation, DuSE-MT, integrates the mechanisms we have proposed and allows for rapid modeling and analysis of self-adaptation scenarios. While our preliminary experience with DuSE indicates useful reasoning support, several avenues of future work may be identified.

We are currently performing experiments with larger design spaces and metrics sets in order to assess DuSE scalability aspects when dealing with large-scale models. DuSE...
has currently no automatic support for handling conflicting design goals and deciding between metric satisfaction trade-offs. The use of search-based optimization approaches to find out a Pareto-optimal set of candidate architectures is also currently being investigated. The definition of annotated design space navigation traces to document design rationale also currently being investigated. The definition of annotated off. The use of search-based optimization approaches to design goals and deciding between metric satisfaction trade-offs. In Proceedings of the 2nd International Workshop on Self-Organizing Architectures, SOAR ’10, pages 21–28, New York, NY, USA, 2010. ACM.


References


