SA³—a tool for supporting adaptable software architecture generation for embedded systems

Nary Subramanian\textsuperscript{a,}\textsuperscript{*}, Lawrence Chung\textsuperscript{b}

\textsuperscript{a}Americas Wireless Business Unit, Anritsu Company, 1155 E Collins Blvd. Suite 100, Richardson, TX 75081, USA
\textsuperscript{b}Department of Computer Science, University of Texas at Dallas, Richardson, TX, USA

Abstract

Adaptability is emerging as an important attribute or nonfunctional requirement (NFR) for embedded systems. Intuitively, adaptability is the ability of a software system to accommodate changes in its environment. However, in order for the embedded system to be adaptable its architecture, the first step in the development, should itself be adaptable. It will be helpful if the generation of this architecture could be automated. In this paper, we describe a tool called the Software Architecture Adaptability Assistant (SA³) that helps generate adaptable architectures. SA³ uses the properties of the NFR Framework, in particular its knowledge base properties, to create a catalog of various constituents of architecture such as components and connections, and then searches the catalog to find adaptable constituents for the architecture. In this paper, we describe the theory behind SA³ and then describe the tool itself. We then demonstrate the architecture generation capability of the tool for embedded systems. We also discuss why these architectures are adaptable as well as point out directions for further improvement.

© 2002 Elsevier Science B.V. All rights reserved.

Keywords: Nonfunctional requirements; Adaptability; Software architecture; NFR Framework; Knowledge base

1. Introduction

Adaptability is emerging as an important attribute or nonfunctional requirement (NFR) for embedded systems. Briefly stated, adaptability is the ability of a software system to accommodate changes in its environment. However, in order for the embedded system to be adaptable its architecture, which is the first step in the development of software solution, should itself be adaptable [1,2].

The process of adaptable software development could be hastened if the development of adaptable architectures could be automated. In this paper, we present a tool called the Software Architecture Adaptability Assistant (SA³) that generates adaptable architectures. SA³ uses the principles of the NFR Framework [3,4] to systematically consider synergies and conflicts among NFRs during the process of architecture development. One of the properties of the NFR Framework is its knowledge base property—the NFR Framework’s ontology is easily stored in a knowledge base. This property of the Framework is exploited by the tool. The goal of SA³ is to interactively generate the constituents of an adaptable architecture: components, connections, patterns, constraints, style and rationales [5,6]. In this paper, we demonstrate the
ability of SA³ to interactively generate the components and connections for an adaptable architecture. The developer can then complete the architecture, if necessary. The work on the remaining architectural constituents is underway.

SA³ is similar in intent to other tools built for automatic architecture development, though the approach taken by SA³ is different. The NFR Assistant [7] is a tool developed to analyze quality requirements in a design; the OME series of tools [8] provides support for agent-oriented requirements analysis; tools for Quality Function Deployment (QFD) [9] have also been used to analyze quality requirements in software—however, the SA³ uses the NFR Framework’s knowledge base properties to develop adaptable architectures.

Section 2 introduces the concept of adaptability followed by a brief discussion on software architecture. Section 3 introduces the NFR Framework and its knowledge base properties. Section 4 presents the tool SA³ and discusses its use. Section 5 explains the use of SA³ in architecture generation while Section 6 gives the conclusions.

2. Adaptability and architecture

A survey of literature [10] has shown that there are numerous definitions of the NFR adaptability. In order to provide a common reference, we give our definition of adaptability in this section. Furthermore, we also discuss the basics of software architecture and introduce the constituents of an architecture.

2.1. Adaptability

Adaptability is the ability of a software system to accommodate changes in its environment. In order to accomplish this, the software system should be able to do the following tasks:

1. detect environment changes
2. recognize changes to be made to itself (i.e., the software system) to accommodate changes in the environment
3. perform changes to itself (i.e., the software system) so that a new system is created that meets the needs of the new environment.

The ‘meeting of the needs of the new environment’ involves the two tasks of validation and verification, which confirm that the changed system indeed meets the needs of the changed environment.

2.2. Architecture

Architecture is the high-level view of a software system. Software architecture has the following constituents [5,6]:

1. components
2. connections
3. patterns
4. constraints
5. styles
6. rationales

Components are the elements from which systems are built; connections are interactions between elements; patterns describe layout of components and connections; constraints are on the components, connections and patterns; styles are an abstraction of architectural components from various architectures; and rationales give reasons for choosing an architecture.

Thus, if SA³ has to be able to generate an adaptable architecture, it should be able to generate the adaptable constituents for the architecture. In this paper, we demonstrate the ability of SA³ to generate components and connections; work is underway with respect to the other constituents of the architecture.

3. The NFR Framework

The NFR Framework [3,4] provides a simple ontology to methodically address quality attributes such as adaptability, reliability, testability, performance, etc. Its language offers expressive constructs for documenting NFRs, and its qualitative, logical reasoning offers a formal system for NFR analysis and design, as well as for software traceability. The main elements of the NFR Framework are the following:

1. softgoals
2. contributions

The softgoals represent NFRs (the NFR softgoals), design elements (the design softgoals) or claims (the
claim softgoals. Each softgoal is named in the convention Type[Topic], where Type is the name of the softgoal and the Topic is the system to which Type applies. A softgoal can have more than one Topic. The contributions represent links between softgoals and they can be of several types: AND, OR, MAKE, HELP, HURT and BREAK (other types are possible as well). The MAKE contribution strongly positively satisfices (this is a term of the NFR Framework which means satisfaction within limits and not absolutely) the parent softgoal; the HELP contribution positively satisfices the parent softgoal, the HURT contribution negatively satisfices while the BREAK contribution strongly negatively satisfices the parent softgoal. The AND contribution indicates that parent softgoal is satisficed provided all the child softgoals are satis-
The OR contribution indicates that the parent softgoal is satisficed if at least one child softgoal is satisfied. The NFR Framework ontology for these elements is given in Fig. 1. The NFR Framework also provides a method to evaluate the effect of child softgoals upon parent softgoals using labels. In addition, softgoals can also have priorities.

An important aspect of the NFR Framework is that it lets prior knowledge be reused by helping the creation of catalogs of methods and correlations. The methods catalog stores the decomposition of the softgoals; the correlation catalog stores the information on interdependency between various softgoals. The creation of catalogs permits the knowledge to be stored in a knowledge base. Fig. 2 shows an example for storing information on some elements of the NFR Framework.

The graph created by the softgoals and the contributions is called the softgoal interdependency graph (or SIG). The SIG stores and displays all the information gained during the process of software development. The SIG can be drawn for any constituent of software architecture. The example SIG of Fig. 3 shows the contributions of a component and a connection for a selected decomposition of the NFR adaptability.

4. The SA³ tool

In this section, we describe the tool that we developed called the Software Architecture Adaptability Assistant (SA³) that helps to generate adaptable architectures. The architectures generated by the tool can be used as is or can be refined further by the developer. The SA³ tool uses the principles of reuse—previously gained knowledge during the process of software development is stored in a knowledge base; based on the NFRs to be satisficed, the tool searches the knowledge base to retrieve the components and connections in the knowledge base that satisfice the given NFRs.

4.1. The architecture of the tool

The architecture of the tool is given in Fig. 4. The tool is based on the client–server style. The server runs on Unix platform. The client runs on Windows workstation; the client communicates with the server using the X-window protocol. The client side has the user interface for populating the knowledge base, providing parameters for searching the knowledge base and for displaying the results of the search. The knowledge base is on the server (KBMS stands for Knowledge Base Management System). The language used for sending messages to/from the server is Telos [11].

4.2. User interface for SA³

The user interface for SA³ is shown in Fig. 5. The user interface consists of five fields: the Menubar, the Icon Panel, the Editor Field, the Protocol Field, and the Status Line. The Menubar provides the various menus supported by the tool. These menus let the tool provide the facilities for architecture generation. The Icon Panel provides icons for commonly used tasks such as copy, paste, tell/query the server, etc. The Editor Field allows the user to enter Telos structures to either populate the server or to query the server. The Protocol Field displays the communication with the server and the result of the communication—whether it was successful or in error. The Status Line currently displays whether the client is connected to the server or not.

The Menubar provides the interesting facilities of the tool. Thus, upon clicking on Populate KB item on the Menubar the window as in Fig. 6 pops-up. And upon clicking on Add NFR Softgoal item of Fig. 6, the window as in Fig. 7 pops up. Using the template of Fig. 7, the knowledge base can be populated with NFR Softgoals. The various elements of the NFR Framework can be populated by using such templates provided by the tool.
4.3. Using SA³ to generate architectures

A software developer uses the following steps to generate architectures using SA³:

1. Create components and connections for a particular application domain.
2. Create SIGs for these components and connections with the NFR softgoals relating to adaptability.
3. Populate the knowledge base with these SIGs.
4. Start the search with an adaptability related NFR; another input required is whether the degree of satisficing should be strongly positively satisficing or just positively satisficing.
5. The tool searches for those components/connections that satisfice the chosen NFR and displays them using the icons of Fig. 8. In case, no components/connections satisfice the NFR, the tool simply says that no architecture could be generated.
6. Using these components/connections as the starting point, the developer completes the architecture for the system.

The completed architecture will be adaptable as the components and connections satisfy an adaptability-related NFR softgoal.

5. Generating architectures using SA$^3$

In this section, we describe the architecture generation ability of SA$^3$. As mentioned in Section 4.3, the first step is to identify the domain whose components and connections populate the knowledge base. The components and connections are derived from the functional requirements for the domain. In order to validate the functioning of the tool, we chose the domain of embedded systems. Within this domain, we concentrated on a class of embedded systems called the Vocabulary Evolution Systems or VES [12]. VES is a type of embedded system that communicates with an external controller through a port such as ethernet or serial. The external controller sends commands through the physical connection that controls the working of the VES. The VES can also respond to some commands to the external controller using the connected port. In a VES, these commands can evolve with time. As an example, a cell phone communicates with the base station using the radio frequency medium for connection. We used the functional requirements for VES as a starting point for populating the knowledge base.

The functional requirements for VES are given below:

1. it should be able to receive commands through a physical port
2. it should be able to parse the commands
3. it should be able to execute the commands
4. it should be able to see the effect of the commands

Given the above functional requirements it becomes clear what some of the components should be:

1. interface driver
2. parser
3. command executer
4. system changer

The interface driver and parser handle communication through the interface. The command executer executes the parsed commands—it changes the values of parameters or reports back the values of parameters to the external controller. The system changer is a component that can affect changes to other system components based on commands received from the external controller.

The connections for the VES can be chosen from the following:

1. message passing
2. procedure invocation
3. implicit invocation
4. interrupt

![Fig. 8. Icons used by SA$^3$ for displaying components and connections.](image-url)
Each of these connections has pluses and minuses. Message passing supports multiple thread execution while procedure invocation usually does not; implicit invocation hides the identity of the receiver from the sender of events while interrupts support asynchronous operation.

5.1. Populating the knowledge base

In order to populate the knowledge base with the above components and connections, we have to draw the SIGs for these components and connections and then populate the knowledge base with the SIGs. The NFR softgoals of these SIGs should be related to adaptability, as in Fig. 3. The satisficing of the various NFR softgoals by these components and connections and their correlations are encoded in the SIGs. Then using the Populate KB menu of SA³, the knowledge in the SIGs is transferred to the knowledge base.

5.2. Searching the knowledge base

Once the knowledge base has been sufficiently populated, it becomes possible to reuse that knowledge in future projects. This is done by clicking on Generate Architecture menu on the Menubar. This results in a pop-up box asking for the architecture constituent to be searched for: component or connection; upon selecting one of them, another pop-up box appears asking for the adaptability-related NFR softgoal to be satisficed as well as the extent of satisficing (the default is strongly positively satisficed). Upon choosing both of these, the search through the knowledge base begins and the components or connections that satisfice the NFR softgoal are displayed. The developer can then complete the architecture using these displayed constituents as the starting point.

Fig. 9. Initial adaptable architecture for VES.

Fig. 10. Another architecture generated using SA³.

<table>
<thead>
<tr>
<th>Date</th>
<th>Date = 00:00:00</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>Time = 00:00:00</td>
</tr>
<tr>
<td>gpib_addr</td>
<td>int = 1</td>
</tr>
<tr>
<td>error</td>
<td>String = &quot;0&quot;</td>
</tr>
<tr>
<td>SetTime</td>
<td>Time</td>
</tr>
<tr>
<td>GetTime</td>
<td>void: Time</td>
</tr>
<tr>
<td>SetDate</td>
<td>Date: int</td>
</tr>
<tr>
<td>GetDate</td>
<td>void: Date</td>
</tr>
<tr>
<td>SetGpioAddr</td>
<td>int</td>
</tr>
<tr>
<td>GetGpioAddr</td>
<td>void: int</td>
</tr>
<tr>
<td>SetError</td>
<td>String: int</td>
</tr>
<tr>
<td>GetLastError</td>
<td>void: String</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Level Controller</th>
</tr>
</thead>
<tbody>
<tr>
<td>level: float = 0.0</td>
</tr>
<tr>
<td>Set(float): int</td>
</tr>
<tr>
<td>SetExtra(float): int</td>
</tr>
<tr>
<td>Get(void): float</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Frequency Controller</th>
</tr>
</thead>
<tbody>
<tr>
<td>frequency: float = 10.0</td>
</tr>
<tr>
<td>Set(float): int</td>
</tr>
<tr>
<td>SetExtra(float): int</td>
</tr>
<tr>
<td>Get(void): float</td>
</tr>
</tbody>
</table>
5.2.1. Example 1
Let the starting softgoal be RemoteInterfaceAdaptabilityVES (as in Fig. 3). Then upon search the result displayed by SA³ will be:

```
<<component>>
InputControllerVES
<<connection>>
MessagePassingVES
```

Using these as starting points the developer could create an adaptable architecture for VES as in Fig. 9.

5.2.2. Example 2
Fig. 10 shows another architecture developed for VES using SA³. The components marked “Adaptable” were identified by SA³.

6. Conclusions
Adaptability is emerging as an important attribute for embedded systems. However, the final software system will be adaptable provided the architecture for the system was adaptable to begin with. We have developed a tool called Software Architecture Adaptability Assistant (SA³) that helps to generate adaptable components and connections for architectures interactively. SA³ uses the principles of the NFR Framework [3,4] to create a catalog of components and connections. Subsequently, during the time of system design, the developer can use SA³ to search the catalog for adaptable components/connections.

There are several areas for further improvement—considering other architectural constituents such as patterns, constraints, components, connections, and rationales; testing SA³ on other types of embedded systems as well as in other domains. However, SA³ gives the developer a tool needed to generate adaptable architectures.

References

Nary Subramanian is employed as Firmware Engineer at Anritsu’s Americas Wireless Business Unit at Richardson, TX. He has been involved in implementing man–machine interface firmware for the test and measuring instruments manufactured by Anritsu. He has been working with Anritsu’s Wireless Business Unit for the past 4 years. Anritsu manufactures test and measuring instruments used in the telecom industry. Nary has an MSEE from Louisiana State University, Baton Rouge, LA.

Dr. Lawrence Chung is currently an Associate Professor in the Department of Computer Science at the University of Texas at Dallas. He received his PhD from University of Toronto, and is currently on the Editorial board of Requirements Engineering Journal. He is the principal author of a book, “Non-Functional Requirements in Software Engineering”, which is being adopted in extending object-oriented analysis to goal-oriented and agent-oriented analysis. His research interests include Requirements Engineering, Software Architecture, Electronic Business Systems and Conceptual Modeling.