Process-Oriented Metrics for Software Architecture Adaptability

Lawrence Chung
Department of Computer Science
University of Texas, Dallas
Richardson, TX
chung@utdallas.edu

Nary Subramanian
Applied Technology Division
Anritsu Company
Richardson, TX
narayanan.subramanian@anritsu.com

Abstract

Adaptability is important to the survival and success of just about any software system, especially due to the rapid changes in technology, organizational structure, human perception and needs. Measurement of the degree to which a software system is adaptable to such changes is often times of a critical concern to software practitioners.

This paper proposes a framework, POMSAA (Process-Oriented Metrics for Software Architecture Adaptability), which aims to provide numeric scores representing the adaptability of a software architecture as well as the intuitions behind these scores. In this framework, the intuitions behind the architectural adaptability scores are traced back to the "whys" of the architecture, namely, the requirements for which the architecture exists in the first place. POMSAA achieves the needed tracing by adopting the NFR Framework, which is a process-oriented qualitative framework for representing and reasoning about non-functional requirements (NFRs).

In this paper we show how POMSAA can be used to calculate adaptability metrics for an architecture of a software system, how it helps detect weaknesses and strategic strengths in the architecture, how it helps to understand the reasons for the weaknesses/strengths and how to make improvements (that will help improve the metrics), and how to recalculate the metrics for the new architecture fast and intuitively. This paper also describes how POMSAA was used in supporting the development of an adaptable architecture for a real-system, a test and measuring instrument used for testing cell phones.

1. Introduction

Adaptability is important to the survival and success of just about any software system, especially due to the rapid changes in technology, organizational structure, human perception and needs [3,11,12,13,14]. Measurement of the degree to which a software system is adaptable to such changes will be useful to the software engineering community.

Measurements of non-functional requirements (*NFRs*) such as adaptability can occur at different phases in the lifecycle of a software system. Thus there are several metrics defined for such NFRs that take measurements on the requirements for the software system [1,2], the architecture of the software system [5], the source code of the system [6], and metrics that take measurements during the maintenance phase of the software system [6,7]. In general, the earlier we are able to detect the metrics for the software system under development, the better will be control on the quality of the software developed.

In this paper a framework, called POMSAA (Process-Oriented Metrics for Software Architecture Adaptability), is proposed which aims to provide metrics for the adaptability of a software architecture besides providing the reasoning behind these metrics. This framework justifies the metrics for adaptability of an architecture by tracing them back to the requirements of the software system for which the architecture was developed. POMSAA achieves the needed tracing by adopting the NFR Framework [8,9], which is a process-oriented qualitative framework for representing and reasoning about NFRs.

The NFR adaptability has been defined variously by different sources [10,11,12,13]: in [10] adaptability has been defined as the ease of system/component modification, in [11] adaptability is defined as modification of behavior in response to environment changes, in [12] adaptability is defined as adjustment to changing requirements while in [13] it is defined as easy changeability of programs. Accommodating these differing notions, we give a fairly general definition of adaptability. The architectures that we have considered in this paper are in the layered style. Also, we have assumed that message passing is used to connect components and this is indicated by a ⇨.

In this paper we show how POMSAA can be used to

1. calculate adaptability metrics for an architecture
2. detect weaknesses/strategic strengths in the architecture and its reasons
3. how to improve adaptability metrics for the architecture by making modifications

This paper also describes how POMSAA was used in supporting the development of an adaptable architecture for a real-system, a Radio Communication Analyzer (*RCA*), which is a test and measuring instrument used for testing cell phones.

Section 2 defines adaptability, some of the important concepts of the NFR Framework, and Vocabulary Evolution System (*VES*), whose architecture this paper develops (as a sub-system of RCA). Section 3 develops the POMSAA framework for measuring adaptability with examples of its application. Section 4 discusses the implementation of the architectures in a real system, the RCA, and the observations (including feedback) on using POMSAA, Section 5 gives the conclusion, Appendices A and B give the algorithms used to compute the metrics of the various softgoals in the SIG, and Appendix C gives a combined SIG for the two architectures considered in this paper.

2. Definitions

2.1 Adaptability

Adaptation means change in the system to accommodate change in its environment. More specifically, adaptation of a software system (*S*) is caused by change (δ_E) from an old environment (*E*) to a new environment (*E'*), and results in a new system (*S'*) that ideally meets the needs of its new environment (*E'*). Formally, adaptation can be viewed as a function:

Adaptation: E x E' x S $\rightarrow$ S', where meet(S', need(E')).

A system is *adaptable* if an adaptation function exists. *Adaptability* then refers to the ability of the system to make adaptation. Adaptation involves three tasks:

1. ability to recognize δ_E *(environment change detection)*
2. ability to determine the change δ_S to be made to the system *S* according to δ_E *(system change recognition)*
3. ability to effect the change in order to generate the new system *S'* (*system change*).

The *meet* function above involves the two tasks of validation and verification, which confirm that the changed system (*S'*) indeed meets the needs of the changed environment (*E'*). The predicate *meet* is intended to take the notion of goal satisficing of the NFR Framework, which assumes that development decisions usually contribute only partially (or against) a particular goal, rarely "accomplishing" or "satisfying" goals in a clear-cut sense. Consequently generated software is expected to satisfy NFRs within acceptable limits, rather than absolutely.

In the NFR Framework, the above three tasks for adaptation become softgoals to be achieved by a design for the software system. An adaptable component of a software system should satisfice these softgoals for adaptation. Figure 2 shows these softgoals for each component of a Vocabulary Evolution System.

2.2 The NFR Framework

The NFR Framework [8,9] requires the following interleaving tasks, which are iterative:

1. Develop the NFR goals and their decomposition
2. Develop architectural alternatives
3. Develop design tradeoffs and rationale
4. Develop goal criticalities
5. Evaluation and Selection

The graph that results from the application of steps 1, 3 and 4 above is called the Softgoal Interdependency Graph (SIG). The SIG will be used to develop the metrics for software architectures developed at step 2 above. The SIG will also help us develop better architectures.

The NFR Framework uses the concept of satisficing. There are different degrees of satisficing and the degrees are indicated by color codes in this paper. The codes that this paper will use are given in Table 1.

Table 1: Color Codes for Satisficing Degrees.

Strongly Positive Satisficing
Positive Satisficing
Negative Satisficing
Strongly Negative Satisficing

While the NFR Framework gives a qualitative framework to compare architectures and choose the better one, in this paper we will show the use of this qualitative framework to develop quantitative numbers for the NFRs, especially that of adaptability. In order to illustrate the POMSAA concepts, we will be focusing our attention on a particular type of software system - that of the VES [14], a sub-system of RCA, a brief description of which is given below.

2.3 Vocabulary Evolution System

The VES will operate in the environment shown in Figure 1.

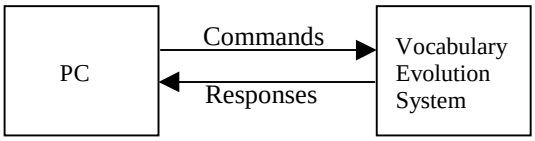

Figure 1: Vocabulary Evolution System

The VES receives commands from an external source, which is usually a PC, in the form of strings of ASCII characters. The link between the VES and the PC could be any physical medium such as Ethernet, General Purpose Interface Bus (*GPIB*), Radio Frequency (*RF*), etc. The VES parses the commands and processes them; if

a command requires a response, the VES sends the responses to the PC over the link. The PC may change the syntax/grammar of the commands it sends to the VES and the VES is expected to adapt itself to this change of environment. Several techniques for this adaptation are discussed in [14]. Usually, VES is part of a much bigger system. In this paper, we will consider VES as embedded in the RCA. This will let us develop adaptability metrics and validate them using the RCA.

2.3.1 Motivation for Using VES

VES is a real component of several systems used in the telecommunication and data-acquisition industry [14,15]. One of the problems in an instrument like the RCA is the evolution of the vocabulary used to communicate with the instrument – this evolution may be due to the development of a more advanced product in the same family of products, a change in the standard used for the syntax, such as SCPI [16], addition of new features to an existing instrument and the like. In this paper we have developed a better architecture for a VES system using POMSAA. An example of vocabulary evolution is given in Table 2 from the current CDMA system to the advanced WCDMA system.

Table 2: Example of Vocabulary Evolution.

Function	CDMA System	WCDMA System
1.Set Address	ADDR 5	ADDR: ETHERNET 5 ADDR: GPIB 5
2.Display Output	OUTPUT "OK"	OUTPUT: LCD "OK" OUTPUT: SPEAKER "OK"
3.Set Time	TIME "12:00:00"	TIME: MAINCPU "12:00:00" TIME: SUBCPU "12:00:00"

2.3.2 Scenarios for a VES System

Scenarios [4, 17] can be used to reason the suitability of an architecture for a system. The scenarios applicable to a VES are given in Table 3.

Table 3. Some Scenarios for a VES

Scenario 1:
PC : sends strings from CDMA System column of Table 2 to VES.
VES: accepts all legal strings belonging to CDMA System column received from PC.
PC: starts sending strings from WCDMA System column of Table 2 to VES.
VES: accepts all legal strings belonging to WCDMA System column received from PC.

Scenario 2:
PC: sends strings from one of the columns of Table 2 to VES.
VES: accepts all legal strings belonging to one of the columns of Table 2.
PC: sends a legal string belonging to another generation of vocabulary *not* in Table 2.
VES: rejects the string from PC declaring error.

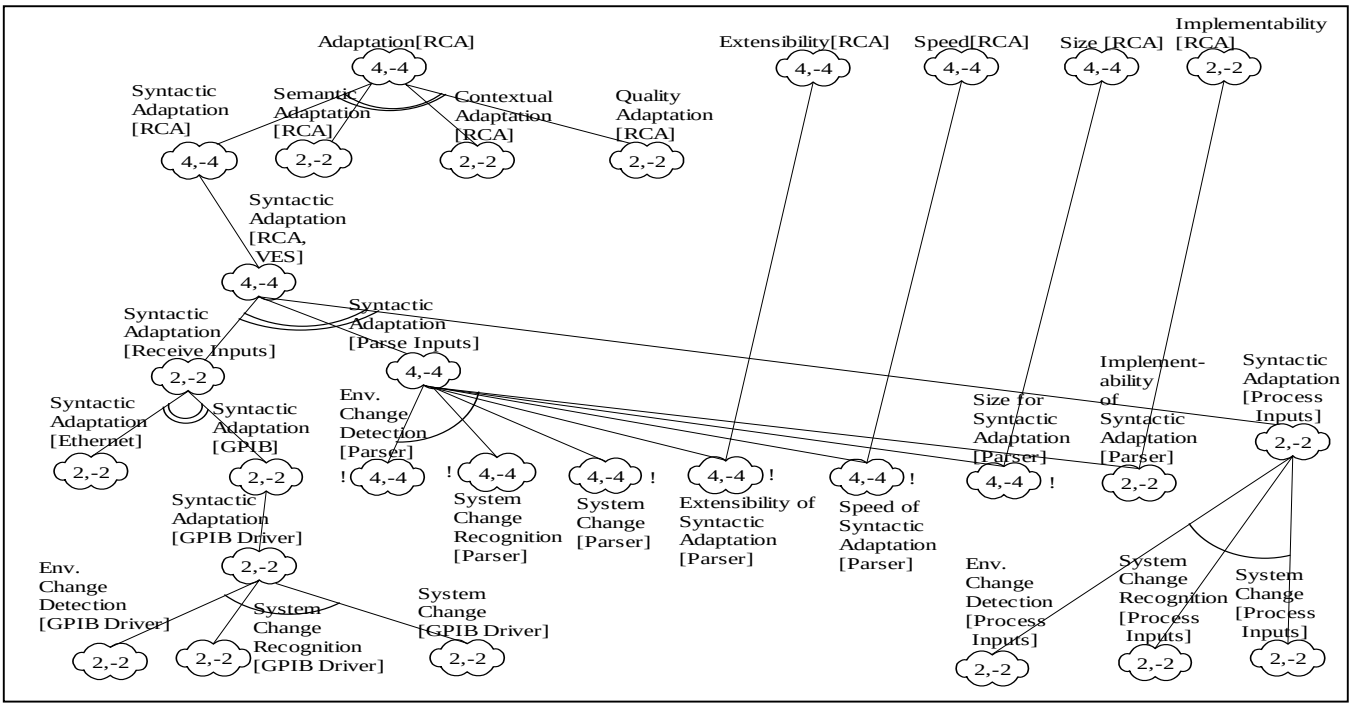

Figure 2: SIG with NFR Decomposition for VES.

3. POMSAA Metrics for Adaptability

3.1 The POMSAA Framework

The POMSAA Framework consists of six major components: a set of *softgoals* for representing NFRs, design components and claims, a set of *contribution types* for relating softgoals to other softgoals, a set of *methods* for refining softgoals into other softgoals, a set of *correlation rules* for inferring potential interactions among softgoals, a *labeling* procedure which determines the degree to which a design component satisfices a softgoal, and a set of *metrification schemes* to map labels to numbers.

1. Softgoals can be of several types – the NFR softgoals (depicted by a cloud), the design softgoal (depicted by a dark cloud), and the claim softgoal (depicted by a dotted cloud). The design softgoal represents a design component, while a claim softgoal represents a claim (for any item of the Framework).
2. Contribution types connect various softgoals – the links may connect several softgoals to one softgoal in an AND-decomposition (depicted by single arc) or in an OR-decomposition (depicted by double arc).
3. Methods are ways to refine or decompose one softgoal into offspring softgoals for purposes of clarity and achievement of better designs.
4. Correlation rules help determine the interactions between different NFRs for a design component.
5. Labels indicate the degree to which their associated softgoal (or links) are satisficed – the various satisificing degrees are mentioned in Table 1.
6. Metrification schemes map qualitative labels into quantitative scores for a given architectural design. Labels of NFR softgoals, design softgoals, claim softgoals and links, in some combination (either only one of these, any two of these, any three of these or all of these), may be converted to numbers. There are several different metrification schemes, including: 6a) *Max and Min Values*: In this scheme the max and min values are computed for the labels; 6b) *Single Values*: Here one value is computed for the labels; 6c) *Range of Values:* Here a range of values is computed for the labels. In this paper, for the metrification scheme, we compute max and min values for NFR softgoals only and also compute single values for NFR softgoals only (in order to illustrate the ideas). Once the metrics for labels are computed, either an automatic algorithm or a semi-automatic algorithm is used to propagate numbers up the SIG to the NFR whose metric is to be computed.

The NFR softgoals that we consider in this paper are adaptability related; the design softgoals that are considered are also adaptability related. We need to capture the knowledge of decomposition and satisficing in the methods and the knowledge of correlations in the correlation rules.

3.2 Application of the POMSAA Framework

As mentioned earlier, the NFR Framework requires decompositions for the NFRs of interest. Figure 2 shows the decomposition for the VES that will be used in this paper. Adaptation of RCA can be along several dimensions – syntactic, semantic, contextual or quality adaptation. Each cloud is a softgoal in the NFR Framework. Each softgoal has a name following the convention

Type[*Topic1*, *Topic2*, …],

where *Type* is a non-functional aspect (e.g., adaptability) and *Topic* is a system to which the *Type* applies (e.g., VES), and the refinement can take place along the *Type* or the *Topic*. Hence the original softgoal Adaptation[RCA] is OR-decomposed (shown by the double arc) into the four softgoals – Syntactic Adaptation[RCA], Semantic Adaptation[RCA], Contextual Adaptation[RCA] and Quality Adaptation[RCA]. Since we are interested in a VES, which is syntactically adaptable, Syntactic Adaptation is further refined into Syntactic Adaptation[RCA, VES] which indicates that we are interested in the VES sub-system of RCA. The syntactic adaptation of VES can occur in one of three ways – due to the input receiving component, the input parsing component or the input processing component. This translates into an OR-decomposition of the softgoal Syntactic Adaptation[RCA, VES] into the three softgoals Syntactic Adaptation[Receive Inputs], Syntactic Adaptation[Parse Inputs] and Syntactic Adaptation[Process Inputs]. The inputs may be received over ethernet or GPIB and so the softgoal Syntactic Adaptation[Receive Inputs] is OR-decomposed into Syntactic Adaptation[Ethernet] and Syntactic Adaptation[GPIB]. Since we assume that RCA communicates with PC using GPIB, the softgoal Syntactic Adaptation[GPIB] is further refined into Syntactic Adaptation[GPIB Driver], since the adaptation of the input port GPIB depends on the adaptation of its driver. Finally the definition of adaptation from Section 2.1 is applied to get the three adaptation characteristics as leaf softgoals for the GPIB Driver, viz., Environment Change Detection[GPIB Driver], System Change Recognition[GPIB Driver] and System Change[GPIB Driver] (in Figure 2, "Env." has been used instead of "Environment"), and since the GPIB Driver has to satisfy all these tasks in order to be adaptable, it is AND-decomposed (indicated by the single arc) into these three sub-softgoals.

Likewise, the softgoal Syntactic Adaptation[Parse Inputs] is further refined into the three adaptation softgoals applied to the parser; in addition it has softgoals derived from a combination of two NFRs-the extensibility, speed, size and the implementability of the parser. However, since the softgoal Syntactic Adaptation[Parse Inputs] requires all its subgoals to be

adaptable in order for itself to be adaptable and since the parsing is done by a parser, it is AND-decomposed (indicated by the single arc) into the five sub-softgoals of Env. Change Detection[Parser], System Change Recognition[Parser], System Change[Parser], Extensibility of Syntactic Adaptation[Parser], Speed of Syntactic Adaptation[Parser], Size for Syntactic Adaptation[Parser] and the Implementability of Syntactic Adaptation[Parser]. Lastly the softgoal Syntactic Adaptation[Process Inputs] is decomposed along the three components of adaptation, like for the GPIB Driver.

In Figure 2 we also show additional symbols such as '!' – these are used to indicate the priority of the softgoals. Since we are considering the VES part of the RCA, the critical softgoals are those related to parsing and adaptation, which are all marked with '!'. However, implementability of the parser is not considered a priority item, as the parser by itself is not a complex software item to implement – though some parsers are more easily implemented than others. Severely critical softgoals are marked with '!!'.

The values for metrics that we will be coming up with will be just one of the many possible schemes for translating from SIG to numbers. What is unique to POMSAA is the justification for the numbers and the ability to change the design to come up with better design and hence corresponding numbers. POMSAA lets users trace metrics to requirements and develop different architectures to better suit the requirements; else refine requirements to clarify anything unclear. This also permits historical record-keeping for later reference. The particular number scheme that we will be using in this paper is given in Table 4.

Figure 2 also gives the maximum and minimum values (the numbers inside the clouds) for the metrics of different softgoals based on the values in Table 4. The

Table 4: Numerical Values for Satisficing Degrees.

Strongly Positive Satisficing	+2
Positive Satisficing	+1
Neutral	0
Negative Satisficing	-1
Strongly Negative Satisficing	-2

procedure for calculating the maximum and minimum values is given in Appendix B, which is an interactive algorithm. This algorithm and the one in Appendix A (which is also interactive) are just two of the numerous schemes (either algorithms or semi-automatic procedures) possible for propagating numbers up the SIG hierarchy. There could be chains of arguments (claims) and computation of the metrics for contribution links (from left to right or from right to left) will work similarly to the bottom-up propagation followed by the algorithms in the Appendices.

3.3 Calculating Metrics for an Architecture

In this section we will illustrate the use of POMSAA to calculate the metrics for an architecture. The architecture is for the VES part of the RCA. The only RCA component used is the LCD Display, which throws up the input command string on the LCD. The architecture A1 considered is given in Figure 3.

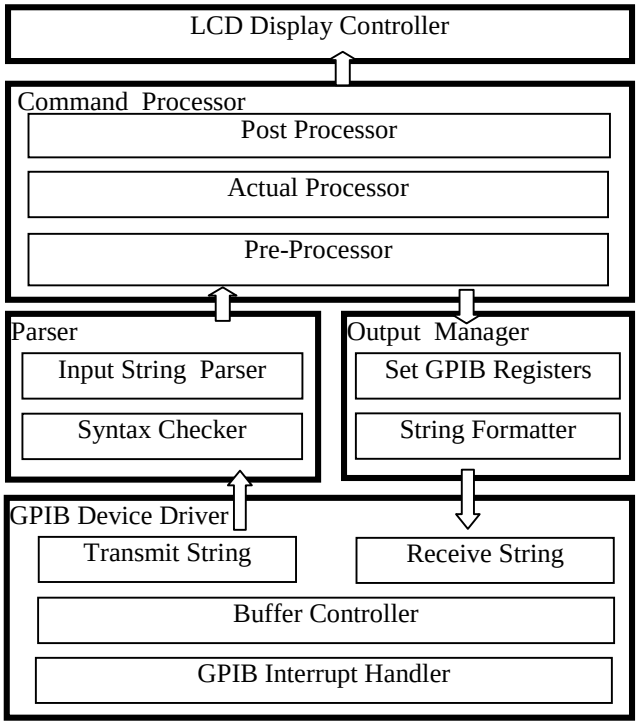

Figure 3: A Candidate Architecture (A1) for VES.

In architecture A1, the LCD Display Controller component controls the displays on the LCD of the RCA. The Command Processor component processes the commands and its sub-components are the Pre-Processor, Actual Processor and the Post Processor. The Pre-Processor component takes any action prior (such as current state checking) to the actual processing by the Actual Processor component, while the Post Processor takes actions after the processing (such as performing dependent actions). The Parser Component consists of the Syntax Checker (to check for illegal characters) and the actual parser – the Input String Parser. The Output Manager is used for sending responses to the external PC and consists of two components – Set GPIB Registers, which sets the registers of the GPIB, and the String Formatter, which formats the strings so that it is suitable for output as per the GPIB standard. The GPIB Device Driver contains the components of Transmit String (which sends the string received from the PC to the Parser), Receive String (which receives the strings from the Output Manager), the Buffer Controller (which controls the input and output buffers) and the Interrupt Handler.

The metrics that the POMSAA calculates for the various NFRs for the architecture A1 are given in Figure 4. In this figure, it should be observed that those softgoals

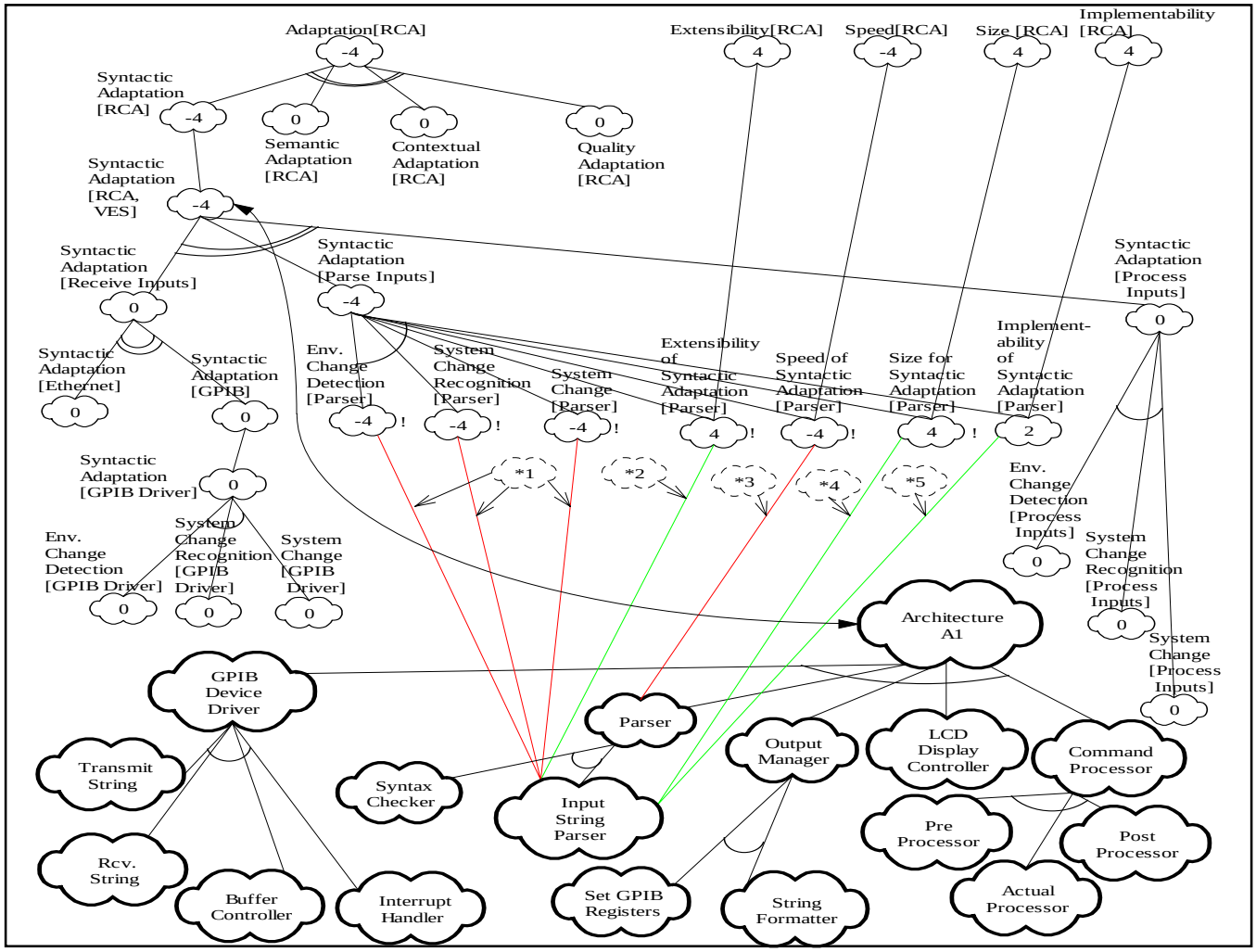

Figure 4: SIG with POMSAA Metrics for Architecture A1.

to which no line is connected are neither satisficed nor dissatisfied (i.e., the architecture is neutral with respect to those softgoals).

Another point to be observed in the SIG of Figure 4 is that architectural components (or design softgoals) are indicated by the dark-bordered clouds. The design softgoals are decomposed in a manner similar to the decomposition of the NFR softgoals (Section 3.2). The architecture A1 (Figure 3) has five basic components – the LCD Display Controller, Command Processor, Parser, Output Manager and the GPIB Device Driver. All these components are included in A1 and hence they each form a softgoal and are connected to the Architecture A1 softgoal by AND-decomposition (indicated by the single arc). The softgoal named Command Processor is further AND-decomposed into the softgoals Post Processor, Actual Processor and Pre-Processor. Likewise other design softgoals have been AND-decomposed based on architecture A1.

The justification for the colors for the lines from the Parser softgoal and the Input String softgoal is given by the claims softgoals, shown in Figure 4 by dotted clouds, and the claims they represent (some of which result from reasoning about scenarios given in Table 3) are given in Table 5. The algorithm for computing the metrics is given in Appendix A and uses values from Table 4. The curly double headed arrow in Figure 4 connecting the design softgoal "Architecture A1" to the NFR softgoal "Syntactic Adaptation [RCA,VES]" indicates that architecture A1 is designed to satisfice the NFR softgoal Syntactic Adaptation[RCA,VES].

3.4 Using POMSAA to Improve Metrics

From Figure 4, it can be seen that Syntactic Adaptation [RCA,VES] has a metric of –4, which is the minimum score that Syntactic Adaptation can get (from Figure 2). We now illustrate how POMSAA fulfills some of the claims made earlier.

1. Detect Architectural Weaknesses:

It is easy to see that the main cause for this low value of metric for Syntactic Adaptation is the weak satisficing of

the high priority softgoals by the parser component of architecture A1.

2. Understand reasons for the weaknesses:

The reason for the weaknesses, in the example, is that the Input String Parser component does not have the intelligence to detect environment change, recognise need for system change, and perform system change, and the Parser component as a whole is slow to adapt.

3. How to change architecture to increase its adaptability:

One one of the ways in which to improve the metric will be using a parser component that is capable of strongly satisficing some of the high priority softgoals. We would like to have a parser that strongly satisfices all

Table 5: Claims for SIG of Figure 4.

*1: No adaptation ability exists; hence no ability to detect environment change, recognize need for system change or make the system change exists.
*2: This parser has unlimited extensibility; for each generation of vocabulary a new parser is built and used.
*3: Since no syntactic adaptation exists, for any evolution in vocabulary the new parser should be built and reloaded into the system; this takes a long time – in the order of minutes (this was measured during implementation – Section 4.1).
*4: Parsers that are changed for any change of vocabulary are optimized for that generation of vocabulary and are usually small in size (about 10000 bytes – discussed in Section 4.1).
*5: Such parsers have little complexity as knowledge of only one vocabulary generation is required.

the softgoals; however, as we know from software engineering, such a parser does not exist. We will therefore develop a parser that satisfices all the high priority softgoals; however, the price that will be paid is that not all softgoals will be satisficed strongly. Some will be satisficing positively; while some, strongly. The architecture A2 that we develop based on the feedback from Figure 4 is shown in Figure 5.

In Figure 5, the Parser component is enhanced with two new components – the Parser Selector and N parsers for different generations of vocabulary. The Parser Selector is further composed of two components – the Vocabulary Generation Detector (detects the generation of vocabulary of the received string) and a component to decide the parser for that generation of vocabulary (called Select Parser for the Generation of Vocabulary). If the input command string belongs to one of the N generations (corresponding to the N parsers) of vocabulary that the VES can handle, then it can parse that input string and process it.

The SIG for this architecture A2 is given in Figure 6. In this figure, some of the design components of architecture A2 (the dark clouds) have not been shown due to lack of space and they are same as in Figure 4. The design softgoals have been decomposed in the same manner as for Figure 4. The algorithm for propagating values up the SIG is given in Appendix A. In Figure 6, the dotted clouds are claims for the color of the lines they point to. The claims are given in Table 6 (some of these claims result from reasoning about scenarios in Table 3). As can be seen from Figure 6, the NFR Syntactic Adaptation[RCA,VES] now has a value of +2, which is an improvement from the previous architecture.

In order to better understand the differences between the two architectures, the relevant information from Figures 4 and 6 have been put in one figure, Figure C1 of Appendix C. In that figure, all the unshaded clouds refer to softgoals of Figure 4, while the shaded clouds refer to softgoals of Figure 6.

4. Initial Experience with POMSAA Metrics

POMSAA has been used in developing the next version of the RCA, which is a test and measuring instrument used for testing cell phones of several current and future cell phone standards. As a ground-breaking step, we used the POMSA to implement a better VES for the RCA.

The architectures A1 and A2 were both implemented – one in the current system and one in the evolved system. The link between the PC and the VES was the GPIB. The VES sent the correct commands to the LCD Display of the RCA; for incorrect commands it displayed an error on the LCD Display. For each implementation, some important parameters were measured.

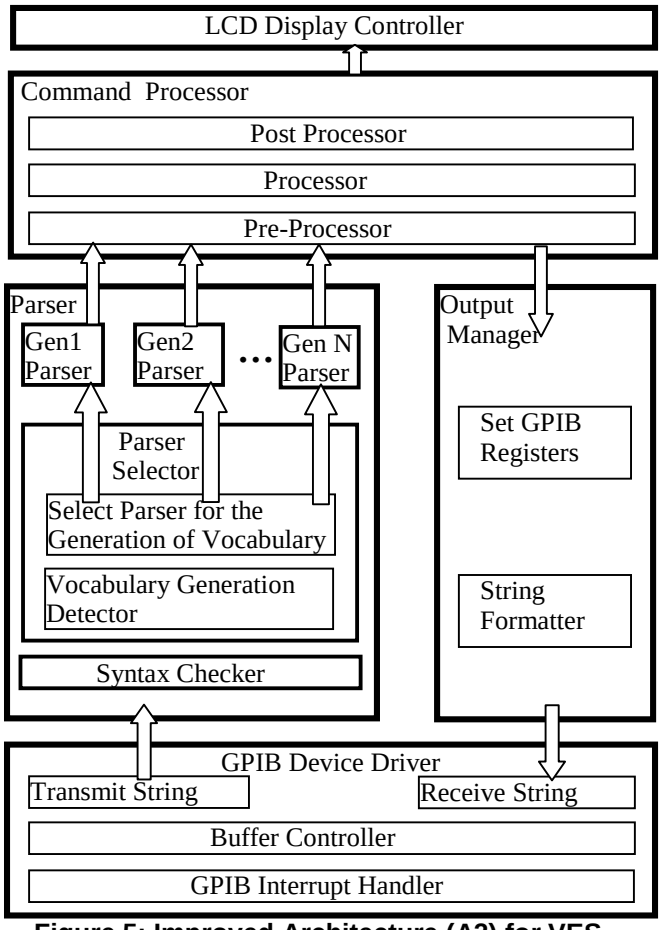

Figure 5: Improved Architecture (A2) for VES.

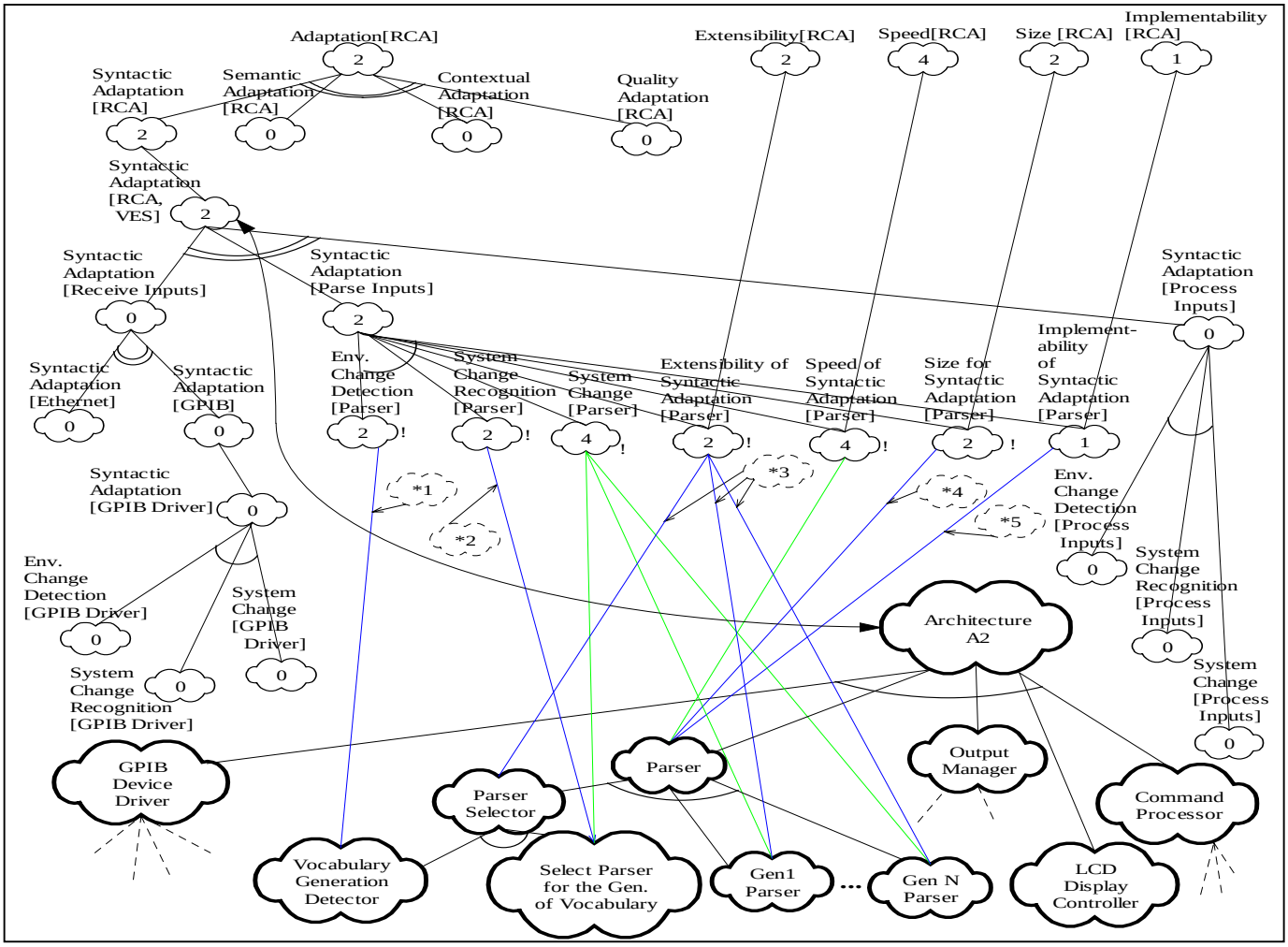

Figure 6: SIG with POMSAA Metrics for Architecture A2.

4.1 Implementation of Architecture A1

For architecture A1, a parser that can parse a given generation of commands was implemented. For any change of the generation, the parser had to be changed, the new code re-compiled and loaded into the RCA, and the RCA had to be re-started. The RCA was then responsive to the new generation. Some of the measurements we took were: parser size 10000 bytes for a 750 string vocabulary (the size of the system was about 750000 bytes), time for adaptation was in the order of minutes, and no adaptation characteristics were exhibited.

4.2 Implementation of Architecture A2

Here N was fixed to 2. The difference between the generations was the presence of a colon ':' in one of the generations of commands, as in Table 2. The generation detector used the colon to decide which of the two parsers to use. Some of the measurements were: total size of the two parsers 20000 bytes (approx.) for 750 string vocabulary each, time for adaptation was about 100 ms, and if the input string belonged to one of the available generations it was parsed correctly (provided the string was legal in that generation).

4.3 Observations

The project in which one of the authors was involved, had to do with the development of an evolved syntax for an advanced member of a product family. The newer product supported the latest 3G cell-phone standard. That author was also the main engineer for developing the VES for the advanced product. The project lasted 3 months, and included 4 people, of which the author was one. In the experience of that author, POMSAA will be extremely useful to develop newer designs for a system. Also from the feedback that was received from the colleagues of one of the authors, POMSAA promises to be an effective framework to determine systems' adaptability. One of the

observations that was made was that detection of system change is perhaps one of the hardest tasks for any software system – POMSAA makes this process easier by

Table 6: Claims for SIG of Figure 6.

*1: Limited environment change recognition is possible; can automatically detect differences between N generations of vocabulary.
*2: Limited system change recognition is possible; can detect which of the available parsers to use and this recognition can be done automatically.
*3: Limited extensibility is possible; the evolution of vocabulary is possible to the extent that parsers for the evolved vocabulary are available (upto N generations of vocabulary evolution are possible).
*4: Size of the parser component will be more than the previous case as different parsers for different generations of vocabulary will have to be present.
*5: Implementability is slightly more difficult than the previous case, as the knowledge of the different parsers will have to be tied to the different generations of vocabulary.

providing a historical record of design decisions and letting the designer make appropriate system changes. Another feedback that we received was the usage of Gaussian distribution as a metrification scheme – POMSAA permits use of any metrification scheme including statistical ones.

5. Conclusion

In this paper we have introduced a process-oriented framework, POMSAA, for calculating the metrics for adaptability. Adaptability is emerging as an important attribute for software systems and its numerical value will help organizations keep track of this NFR's content. The main advantages of POMSAA are:

1. intuitive calculation of metrics
2. traceability of metrics to requirements
3. calculation of metrics at the architectural level
4. ability to analyze reasons for weaknesses/strategic strengths in the metrics
5. ability to visualize the effect of architectural component changes on the metrics
6. historical record keeping for later reference

In this paper we developed an architecture for a vocabulary evolution system, computed its metric for a decomposition of adaptation using the NFR Framework, analyzed reasons for its low metric value, improved its architecture based on this analysis, computed the metrics for this improved architecture and verified to a certain degree that the metrics for the improved architecture is indeed better. We also implemented the architectures in a real system, the Radio Communication Analyzer, and were able to confirm that the resulting system indeed met the requirements. We received positive feedback on the utility of this framework and the variety of metrification schemes (that could be automatic algorithms or semi-automatic procedures) that were possible including the use of different distribution curves.

Furthermore, we considered, for brevity, application of POMSAA on the components of an architecture only. However, POMSAA applies equally well to other constituents of an architecture such as connections, constraints, etc.

There is a lot of work still to be done – the extension of POMSAA technique to other NFRs, using customer feedback to improve the quality of architectures for better metrics, guidelines for developing metrics for design softgoals, and applying POMSAA to other software systems. Although much work still remains to be done, it is our opinion that our preliminary studies show that the POMSAA framework will be of much help to software development organizations in practice.

Acknowledgements

The authors wish to thank Mr. John Freasier, Project Leader, Anritsu Company, for reviewing the paper and providing insightful suggestions. We also wish to thank the anonymous referees of Requirements Engineering Symposium, 2001, who reviewed the original version of this paper and gave detailed comments.

References

[1] L. Rosenberg, T. Hammer, J. Shaw, "Software Metrics and Reliability", *9th International Symposium on Software Reliability*, Germany, Nov. 1998.

[2] L. Rosenberg, T. F. Hammer, L. L. Huffman, "Requirements, Testing, and Metrics", *15th Annual Pacific Northwest Software Quality Conference*, Utah, Oct., 1998.

[3] G. Koutsoukos, J. Gouveia, L. Andrade, J. L. Fiadeiro, "Managing Evolution in Telecommunication Systems", from the web-site of Dr. J.L.Fiadeiro.

[4] J. Ryser, M. Glinz, "A Practical Approach to Validating and Testing Software Systems Using Scenarios", QWE '99, 3rd International Software Quality Week Europe, Brussels, Nov. 1999.

[5] J. C. Duenas, W. L. de Oliveira, J. A. de la Puente, "A Software Architecture Evaluation Model", Lecture Notes in Computer Science (1429), "Development and Evolution of Software Architectures for Product Families", *Proceedings of Second International ESPIRIT ARES Workshop*, Las Palmas de Gran Canaria, Spain, Fee., 1998, pp. 148 – 157.

[6] N. E. Fenton, "*Software Metrics – A Rigorous Approach*", Chapman & Hall, London, 1991.

[7] T. Gilb, "*Principles of Software Engineering Management*", Addison Wesley, England, 1988.

[8] L. Chung, B. A. Nixon, E. Yu, and J. Mylopoulos, "*Non-Functional Requirements in Software Engineering*", Kluwer Academic Publishers, Boston, 2000.

[9] J. Mylopoulos, L. Chung, S. S. Y. Liao, H. Wang and E. Yu, "Exploring Alternatives During Requirements Analysis", *IEEE Software*, January/February 2001, pp. 1- 6.

[10] IEEE Std 610.12-1990, IEEE Standard Glossary of Software Engineering Terminology.

[11] P. Oreizy, M. M. Gorlick, R. N. Taylor, D.Heimbigner, G. Johnson, N. Medvidovic, A. Quilici, D. S. Rosenblum and A. L. Wolf, "An Architecture-Based Approach to Self-Adaptive Software", *IEEE Intelligent Systems*, May/June 1999, pp. 54 – 62.

[12] Report on Adaptability in Object-Oriented Software Development Workshop, July 8, 1996, *10th European Conference on Object Oriented Programming*, July 8-12, 1996, Linz, Austria.

[13] Report on the Workshop on Adaptable & Adaptive Software, *10th Annual Conference on Object-Oriented Programming Systems*, Languages, and Applications, Oct., 1995, Austin, Texas.

[14] N. Subramanian and L. Chung, "Architecture-Driven Embedded Systems Adaptation for Supporting Vocabulary Evolution", Proc. *ISPSE* 2000, IEEE Computer Society, November, 2000, Kanazawa, Japan.

[15] S. Zanetti, "Using Software to Manage Your Networked Measurements", *AutomationView*, Vol. 5, No. 4, Fourth Quarter 2000, pp. 1-5.

[16] *Standard Commands for Programmable Instruments (SCPI)*, Version 1999, May 1999.

[17] A. Anton, M. McCracken and C. Potts, "Scenarios and Goals in Business Process Reengineering", *6th Conference on Advanced Information Systems Engineering*, Utrecht, Netherlands, May, 1994.

Appendix A: Algorithm Used for the Propagation of Metrics Up a SIG

Let M_n denote the metric of softgoal *n*.

For each leaf softgoal *n* in the SIG:
1. $M_n = 0$, if there is no line connected to it.
2. if one colored line is connected to *n*, then
$M_n = +2$, for green line,
$M_n = +1$, for blue line,
$M_n = -1$, for orange line,
$M_n = -2$, for red line.
3. if more than one colored line is connected *n*, then
M_n = *minimum* value of all lines connected to it.
4. if *n* is critical (marked with '!'), then
$M_n = M_n * 2$
5. if *n* is severely critical (marked with '!!'),
$M_n = M_n * 4$
End For;

For each parent softgoal *p* in the SIG:
1. if *p* has only one child *n*, then
$M_p = M_n$
2. if *p* has several children *n1*,…,*nk*, and
$M_{n1} = M_{n2} = \ldots = M_{nk} = 0$, then $M_p = 0$.
3. if *p* has several children *n1*,…,*nk*, and not all M_{ni} $(k \geq i \geq 1) = 0$, and the children are connected by AND, and all children are of same criticality, then $M_p = \min(M_{j1},\ldots,M_{jq})$ such that $k \geq j1,\ldots,jq \geq 1$ and $M_{jr} \neq 0$ for all r, $q \geq r \geq 1$.
4. if *p* has several children *n1*,…,*nk*, and not all M_{ni} $(k \geq i \geq 1) = 0$, and the children are connected by OR, and all children are of same criticality, then $M_p = \max(M_{j1},\ldots,M_{jq})$ such that $k \geq j1,\ldots,jq \geq 1$ and $M_{jr} \neq 0$ for all r, $q \geq r \geq 1$.
5. if *p* has several children *n1*,…,*nk*, and not all M_{ni} $(k \geq i \geq 1) = 0$, and all children are not of same criticality, then
a. if the highest level of criticality in all children is '!', then for each non-critical child softgoal *x*, compute, $M'_x = M_x * 2$, and for each critical child softgoal *y*, $M'_y = M_y$.
b. if the highest level of criticality in all children is '!!', then for each non-critical child softgoal *x*, compute, $M'_x = M_x * 4$, and for each critical child softgoal *y* (marked with '!'), $M'_y = M_y * 2$, and for each severely critical child softgoal z (marked with '!!'), $M'_z = M_z$.
c. Apply step 3 or 4 using the above M' values instead of the M values.
End For;

Appendix B: Algorithm Used for Calculating the Max and Min Value of Metrics in a SIG

Let MAX(n) and MIN(n) denote max and min values of softgoal *n*.

For each leaf softgoal *n* in the SIG:
1. if *n* is non-critical, MAX(n) = 2, MIN(n) = -2.
2. if *n* is critical, MAX(n) = 4, MIN(n) = -4.
3. if *n* is severely critical, MAX(n) = 8, MIN(n) = -8.
End For;

For each parent softgoal *p* in the SIG:
1. if *p* has only one child *n*, then
MAX(p) = MAX(n), MIN(p) = MIN(n).
2. if *p* has several children *n1*,…,*nk*, connected by AND, and all children are of same criticality, then MAX(p) = MAX(n1), MIN(p) = MIN(n1).
3. if *p* has several children *n1*,…,*nk*, connected by OR, and all children are of same criticality, then MAX(p) = max(MAX(n1),…,MAX(nk)),

MIN(p) = min(MIN(n1),…,MIN(nk)).
4. if *p* has several children *n1*,…,*nk*, and not all are of same criticality,
a. if the highest criticality is '!', then for each non-critical child softgoal x, MAX'(x) = MAX(x) * 2, and MIN'(x) = MIN(x) * 2, and for each critical child softgoal y (marked with '!'), MAX'(y) = MAX(y), MIN'(y) = MIN(y).
b. if the highest criticality is '!!', then for each non-critical child softgoal x, MAX'(x) = MAX(x)*4, MIN'(x) = MIN(x)*4, for each critical child softgoal y (marked with '!'), MAX'(y) = MAX(y)*2, and MIN'(y) = MIN(y)*2 and for each severely critical child softgoal z (marked with '!!'), MAX'(z) = MAX(z) and MIN'(z) = MIN(z).
c. Apply step 2 or 3 using MAX' and MIN' values instead of the MAX and MIN values.
End For;

Appendix C: Combined SIG

Figure 4 and Figure 6 have been combined into one Figure C1. In Figure C1, the shaded clouds refer to softgoals of Figure 6 (Architecture A2), while unshaded clouds refer to softgoals of Figure 4 (Architecture A1). Only the relevant architectural components have been shown.

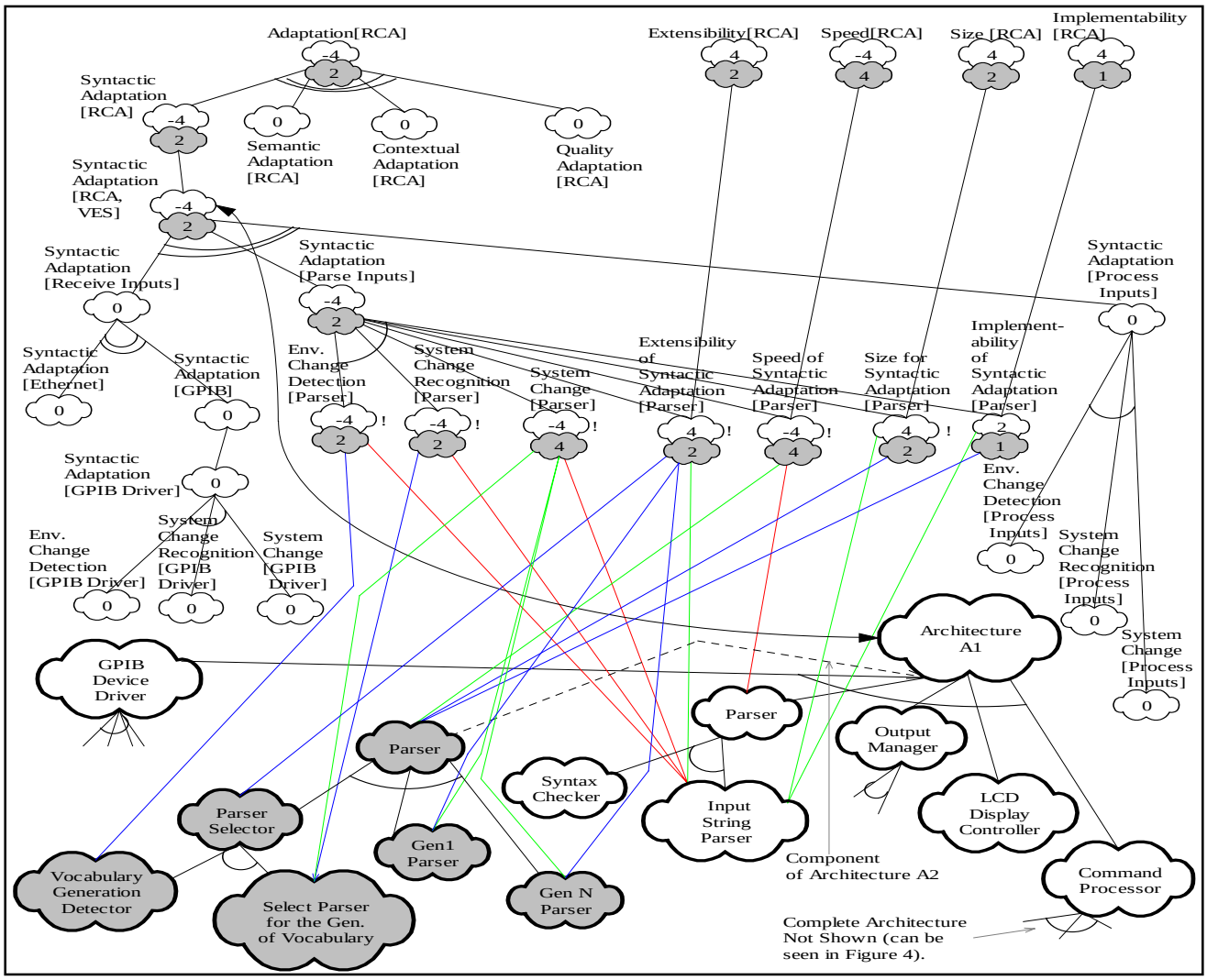

Figure C1: SIG with POMSAA Metrics for Architectures A1 & A2.