The Four Levels of Requirements Engineering for and in Dynamic Adaptive Systems

Daniel M. Berry
School of Computer Science
University of Waterloo
Waterloo, ON, Canada
dberry@uwaterloo.ca

Betty H.C. Cheng
Department of Computer Science and Engineering
Michigan State University
East Lansing, MI, U.S.A.
chengb@cse.msu.edu

Ji Zhang
Department of Computer Science and Engineering
Michigan State University
East Lansing, MI, U.S.A.
zhangji9@cse.msu.edu

ABSTRACT

This paper argues that there are four levels of requirements engineering for and in a dynamic adaptive system: (1) by humans, for the general behavior of the system, (2) by the system itself, whenever it is adapting based on changes to its input, (3) by humans, to decide when, how, and where the system is to adapt, and (4) by humans, doing research about adaptive systems.

1. OVERVIEW

Recently, a significant amount of effort has been devoted to developing technologies to support dynamic adaptive systems (DASs) [e.g., 1; 2; 3]. A DAS is a computer-based system (CBS) that is capable of recognizing that the environment with which it shares an interface has changed and that is capable of changing its behavior to adapt to the changing conditions. Much of the interest in DASs is motivated by the increasing demand for pervasive, mobile, and automatic computing.

1.1 Motivation

We had noticed that RE is always *about* input and responses. That is, RE determines

1. the kinds of input a system may be presented and
2. the system's responses to these inputs.

A DAS, S, is *doing* RE at run time. That is, S is determining, *as it is executing*,

1. the kinds of input S may be presented and
2. S's responses to these inputs.

DEAS '05 Saint Louis, MO, USA

However, this run-time RE is not the *only* RE done about S. Humans are doing lots of RE about S and about S's own RE! Therefore, we offer a categorization of all the various REs that are taking place for and in DASs.

1.2 Adapt-Ready Systems

Let S_{AR} ("AR" for "adapt-ready" [4]) be a DAS operating on domain D (i.e., its input space). A *target* program S_i of S_{AR} is a program exhibiting one of the behaviors that S_{AR} can adopt after adapting. S_i's domain is D_i, and the set of all target programs supported by S_{AR} is S. Let the initial target program of S_{AR} be called S_0. Each index i should be regarded as a name for some target program. The only semantics that can be derived from the numerical order of the indices is the time history of target programs.

1.3 Four Levels of RE

This note argues that there are four levels of requirements engineering (RE) coming into play for and in S_{AR}. They are listed in order of increasing metaness; that is, Level $j + 1$ RE makes decisions about the subject matter of Level j RE. The level indices do not indicate the order of occurrence. Of course, other decompositions into levels are possible.

1. Level 1 RE is that done by humans for all the target programs in S, to determine D_i for each $S_i \in S$ and S_i's reaction to each input in D_i. System invariants, which affect the other levels, should be identified at this level (Space considerations prevent elaboration on this subject.) [5].
2. Level 2 RE is that done by S_{AR} during its own execution in order to determine from the latest input that it must adapt and to determine which $S_i \in S$ to adopt.
3. Level 3 RE is that done by humans to determine S_{AR}'s adaptation *elements*, which allow S_{AR} to do the adaptation embodied in the Level 2 RE.
4. Level 4 RE is that done by humans to discover adaptation mechanisms in general.

Here, elements include detection and monitoring techniques, decision-making procedures, and adaptive mechanisms.

For a given S_{AR}, it is possible that the Levels 1, 3, and 4 REs, done by humans, be done concurrently; that is, the human requirements engineers for S_{AR} will need to determine the set of target

programs, the method for choosing among them, and general monitoring and adaptation techniques simultaneously in order to produce a coherent system.

It is possible also that these human-applied RE levels be revisited during S_{AR}'s life. That is, S_{AR} may be presented totally unanticipated input $I \notin D$, such that S_{AR}'s Level 2 RE fails to adapt. Perhaps, S_{AR} informs the user that S_{AR} cannot adapt to I. Perhaps, the user must notice that S_{AR} is not meeting its requirements. Then, additional Level 1 RE must be done to determine at least one new target program, S_I, that has I in its domain and that responds correctly to I. Additional Level 3 RE must be done to revise S_{AR}'s adaptation mechanism so that when S_{AR} is run again with input I, S_{AR} does a new Level 2 RE in order to adapt to the input I. Perhaps, in addition, some Level 4 RE should be done to determine better ways to deal with unanticipated input.

2. EXAMPLE

For example, in the history of the adaptive, assistive e-mail system developed by Fickas *et al* [6, 7] to help brain-injured patients improve their social connectedness, one can see examples of all four levels of RE. For each item below, the parenthesized list gives the sections of reference [7] describing the item's RE level.

- Level 1 RE is the work done by Fickas *et al* to determine all possible e-mail features and user interfaces (UIs) to be supported by any version of the e-mail system for a cognitively disabled person. (Outermost Section 5 and Section 5.5)
- Level 3 RE is the work done by Fickas *et al* to determine the categories of users to be helped by the system, how to recognize a user's category by his or her input, and the appropriate collection of features for each category of user. This RE was done by a combination of interviews of patients and analysis by caretaking experts and computing experts; patient goals were matched to skills need to achieve them and then to features requiring those skills. Doing this RE led to (1) the discovery of the need for e-mail features and UIs not anticipated in the previous Level 1 RE effort and (2) the invention of these additional e-mail features and UIs, i.e., some additional Level 1 RE. (Sections 5.1, 5.4, and 5.5)
- Level 2 RE is the work done during runs of the e-mail system, as it monitors a user's input and determines that it is now time to change the e-mail system's UI and behavior to appear to the user as a new e-mail program. If the e-mail system cannot adapt to a user or Fickas *et al* determine that the user's e-mailing is deteriorating or that the user is behaving in unanticipated ways that are not detected by the run-time monitoring, then Fickas *et al* intervene and do more Level 1 and Level 3 RE, especially that involving personal interviews of the patient. (Sections 5.2 and 5.3)
- Level 4 RE is all the research done by Fickas *et al* in requirements satisfaction monitoring and adaptation, requirements deferment, personal and contextual RE, etc., i.e., what they describe and cite in their papers [6, 7]. (Section 5.5 and References)

Note that in this example and in general, Level 3 RE will happen before Level 2 RE simply because it is Level 3 RE that determines the Level 2 RE that S_{AR} does during its execution.

While in any given S_{AR} the boundaries between Levels 1, 2, and 3 RE are precise, in a history of versions of S_{AR}, as the human requirements engineers understand better the adaptations that need to be made, work may shift from Levels 1 and 3 RE, done by humans, to Level 2 RE, done by the next version of S_{AR}.

3. ANOTHER EXAMPLE

Martin Feather describes a degenerate case of an adaptive tool that he has written for himself as the only user. He has inserted *assert* statements into the code of the tool. Each such assert statement causes a run-time break when its logical expression evaluates to false. Each such assert statement is, in effect, a requirement specification describing an assumed property of the tool's input or of a value calculated by the tool in response to some input. Often, the violation of an assumption points to a requirements change; he is using the tool in a way he had not anticipated and to which the existing code is not prepared to respond in a reasonable way. Occasionally, the violation indicates a feature interaction he did not anticipate. In either case, Feather reacts by analyzing the situation and deciding on new behavior. He implements the new behavior by manually modifying the code. He modifies also the assert statements to reflect the environmental assumptions.

In this case, nearly all of all four levels of RE are done by Feather, the user–implementer himself. The only exception is the part of Level 2 RE that detects that the current input is not in the tool's current domain and that it is time to change the tool's behavior. The rest of Level 2 RE is done off line by Feather. The result is that the Level 3 RE is rather trivial, as it involves only figuring out the logical expressions of the assert statements that monitor requirements changes.

4. LEVELS OF RE

This section describes each level of RE in detail.

4.1 Level 1

Level 1 RE resembles the traditional RE that is done for any CBS. This RE involves

1. eliciting and analyzing information about the domain D of S_{AR},
2. deciding the set of all features of any target program to be adopted by S_{AR} and their functionalities,
3. deciding the set of all target programs to be adopted by S_{AR} and their functionalities, and
4. specifying the functionalities of all target programs presented by S_{AR}.

A wide variety of standard methods are available for this RE [e.g., 8; 9; 10].

4.2 Level 2

Level 2 RE is what S_{AR} does when it gets input not in the domain of its current target program. S_{AR} must determine which target program in S it should adopt next. That this behavior is RE can be seen if one considers what S_{AR} is doing. Suppose S_{AR} currently has adopted the target program S_i, and its current input I is not in D_i. Then, S_{AR} effectively

1. determines from I how its new domain D_{i+1} differs from D_i,
2. determines which of its target programs, S_{i+1}, to adopt next, and
3. modifies its own behavior to adopt S_{i+1} as its current target program.

Of course, S_{AR} must have some monitoring code to keep track of environmental changes as reflected in its input. S_{AR} must have code that determines which of its target programs to adopt as a function of detected environmental changes. Finally, S_{AR} must have somewhere in its code, for each target program S_j, either the code for S_j or code to find the code for S_j, e.g., in a library.

4.3 Level 3

Level 3 RE is probably the most difficult to achieve because it requires assessing what S_{AR} should do at the meta level, that is, how can we make S_{AR} do its Level 2 RE. Level 3 RE involves figuring out *how* to get S_{AR} to

1. determine from I how its new domain D_{i+1} differs from D_i,
2. determine which of its target programs, S_{i+1}, to adopt next, and
3. modify its own behavior to adopt S_{i+1} as its current target program.

Doing this RE requires having determined program-testable correspondences to environmental changes that trigger adaptation. The requirements engineers will have to explore representations for

1. the possible new domains with their corresponding environmental conditions,
2. the possible adaptive reactions to new inputs, and
3. the testable conditions under which each new adaptive reaction is to be applied.

By "representation", we allow any scheme from which specific adaptive reactions can be derived, perhaps by instantiation, parameter application, mapping, reconfiguration [6, 7], table lookup, recomposition of new components [1], formula or specification generation, etc.

4.4 Level 4

Level 4 RE is essentially the research into adaptation mechanisms. Adaptation mechanisms have been developed for the application level [e.g., 4; 11; 12; 13], middleware [e.g., 14; 15; 16; 17; 18], and operating systems [e.g., 19; 20].

5. DISCUSSION

This classification of the RE involved in DASs highlights the fundamental barrier that must be conquered before DASs can become truly adaptable. Since for the foreseeable future, software is not able to think and be truly intelligent and creative, the extent to which a DAS S_{AR} can adapt is limited by the extent to which the adaptation analyst can anticipate the domain changes to be detected and the adaptations to be performed. This limit is called the *envelope of adaptability*. This envelope thus determines the domain D of S_{AR} and the set S of target programs of S_{AR}. This envelope of adaptability cannot exceed our own adaptability. While we are adaptable, we do not know how we are adaptable, and thus we cannot program software to be even *as* adaptable as we are. Therefore, S_{AR} will always be less adaptable than we are.

In other words, it is not likely that we will be able to implement any time soon, the easy adaptability that we see in the android Data on *Star Trek Next Generation* and the holographic doctor on *Star Trek Voyager*.[1] Moreover, this adaptability cannot happen until and unless we humans understand enough about our own thinking that we know *how* we think, create, and adapt, and can translate that knowledge into software that truly thinks, creates, and adapts. Of course, a clear topic for Level 4 RE, i.e., research, is how an adapt-ready system can adapt to unanticipated domain changes on the fly without human intervention.

6. WHAT'S NEXT?

As we move forward with decreasing costs for CBSs; with increasing demand for mobile, heterogeneous, and pervasive computing; and with increasing interest in autonomic systems [e.g., 21; 22], the need for DASs will increase. Currently, much of the effort has focused on how to make legacy systems adaptive. As we move towards an adaptive software paradigm, we anticipate that the adaptability envelope will expand since the RE at Level 1 will expand to include RE at Levels 3 and 4.

As we move into this new era of dynamic adaptation, more attention is needed to establish the correctness of software, before, during, and after adaptation. Thus far, we have largely focused on the enabling technologies that provide adaptive capabilities. We need to step back and ensure that assurance issues are being considered at all 4 levels of RE for DASs. Assurance will contribute also to the decision-making process for determining when, how, and where adaptations should take place.

ACKNOWLEDGMENTS

We thank Martin Feather and Steve Fickas for suggesting the main examples used in the paper. Daniel Berry's work was supported in part by Canada's NSERC under Grant No. NSERC-RGPIN227055-00. Betty Cheng's work is sponsored in parts by the U.S. O.N.R. Grant N00014-01-1-0744, and N.S.F. Grants CCR-9901017, EIA-0000433, and EIA-0130724.

REFERENCES

[1] McKinley, P.K., Sadjadi, M., Kasten, E.P., Cheng, B.H.C.: Composing adaptive software. IEEE Computer (2004) 56–64. For more information, please refer to the companion technical report.

[2] Sousa, J.P., Garlan, D.: Aura: an architectural framework for user mobility in ubiquitous computing environments. In: Proceedings of the third Working IEEE/IFIP Conference on Software Architecture. (2002) 29–43

[1] Even a confirmed trekker must gasp at the dialog on these TV shows that has a person deciding as simply as he chooses his breakfast from the food replicator that a program or android will be adaptable.

[3] Adve, S., Harris, A., Hughes, C., Jones, D., Kravets, R., Nahrstedt, K., Sachs, D., Sasanka, R., Srinivasan, J., Yuan, W.: The illinois grace project: Global resource adaptation through cooperation. In: Proceedings of the Workshop on Self-Healing, Adaptive, and self-MANaged Systems (SHAMAN). (2002)

[4] Yang, Z., Cheng, B.H., Stirewalt, R.E.K., Sowell, J., Sadjadi, S.M., McKinley, P.K.: An aspect-oriented approach to dynamic adaptation. In: Proceedings of the ACM SIGSOFT Workshop On Self-healing Software (WOSS'02). (2002)

[5] Zhang, J., Cheng, B.H.C.: Specifying adaptation semantics. In: ICSE Workshop on Software Architectures for Dependable Systems (WADS05). (2005)

[6] Fickas, S.: Clinical requirements engineering. In: Proceedings of the 27th International Conference on Software Engineering. (2005)

[7] Fickas, S., Robinson, W., Sohlberg, M.: The role of deferred requirements: A longitudinal study. In: Proceedings of the Thirteenth IEEE International Conference on Requirements Engineering. (2005)

[8] Gause, D., Weinberg, G.: Exploring Requirements: Quality Before Design. Dorset House, New York, NY, USA (1989)

[9] Robertson, S., Robertson, J.: Mastering the Requirements Process. Addison-Wesley, Harlow, England (1999)

[10] Larman, C.: Applying UML and Patterns. Second edn. Prentice Hall PTR, Upper Saddle River, NJ, U.S.A. (2002)

[11] David, P.C., Ledoux, T., Bouraqadi-Saadani, N.M.N.: Two-step weaving with reflection using AspectJ. In: OOPSLA 2001 Workshop on Advanced Separation of Concerns in Object-Oriented Systems, Tampa (2001)

[12] Sadjadi, S.M., McKinley, P.K., Stirewalt, R.E.K., Cheng, B.H.: Generation of self-optimizing wireless network applications. In: Proceedings of the International Conference on Autonomic Computing (ICAC-04), New York, NY (2004) 310–311

[13] Wohlstadter, E., Jackson, S., Devanbu, P.: DADO: enhancing middleware to support crosscutting features in distributed, heterogeneous systems. In: Proceedings of the International Conference on Software Engineering, Portland, Oregon (2003) 174–186

[14] Redmond, B., Cahill, V.: Supporting unanticipated dynamic adaptation of application behaviour. In: Proceedings of the 16th European Conference on Object-Oriented Programming. (2002)

[15] Kon, F., Román, M., Liu, P., Mao, J., Yamane, T., Magalhães, L.C., Campbell, R.H.: Monitoring, security, and dynamic configuration with the dynamicTAO reflective ORB. In: Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware'2000). Number 1795 in LNCS, New York, Springer-Verlag (2000) 121–143

[16] Blair, G., Coulson, G., Blair, L., and P. Grace, H.D.L., Moreira, R., Parlavantzas, N.: Reflection, self-awareness and self-healing in OpenORB. In: WOSS02, Charleston, SC (2002)

[17] Zinky, J.A., Bakken, D.E., Schantz, R.E.: Architectural support for quality of service for CORBA objects. Theory and Practice of Object Systems **3** (1997)

[18] Sadjadi, S.M., McKinley, P.K.: ACT: An adaptive CORBA template to support unanticipated adaptation. In: Proceedings of the 24th IEEE International Conference on Distributed Computing Systems (ICDCS'04), Tokyo, Japan (2004)

[19] Kon, F., Campbell, R.H., Ballesteros, F.J., Mickunas, M.D., Nahrstedt, K.: 2K: A distributed operating system for dynamic heterogeneous environments. In: Proceedings of the 9th IEEE International Symposium on High Performance Distributed Computing, Pittsburgh, PA, U.S.A. (2000)

[20] Appavoo, J., Hui, K., Soules, C.A.N., Wisniewski, R.W., Silva, D.M.D., Krieger, O., Auslander, D.J.E.M.A., Gamsa, B., Ganger, G.R., McKenney, P., Ostrowski, M., Rosenburg, B., Stumm, M., Xenidis, J.: Enabling autonomic behavior in systems software with hot-swapping. IBM Systems Journal **42** (2003)

[21] Ganek, A.G., Corbi, T.A.: The dawning of the autonomic computing era. IBM Systems Journal, Special Issue on Autonomic Computing **42** (2003)

[22] Kephart, J.O., Chess, D.M.: The vision of autonomic computing. IEEE Computer **36** (2003) 41–50