Formal Specification in Heterogeneous Distributed Software Integration*

Chunmin Yang
Barrett R. Bryant
Carol C. Burt
Computer/Information Sciences
Univ. of Alabama-Birmingham
Birmingham, AL 35294, USA
{yangc, bryant, cburt}@cis.uab.edu

Rajeev R. Raje
Andrew M. Olson
Computer/Information Science
Indiana Univ. Purdue Univ.
Indianapolis, IN 46202, USA
{rraje, aolson}@cs.iupui.edu

Mikhail Auguston
Computer Science
New Mexico State Univ.
Las Cruces, NM 88003, USA
mikau@cs.nmsu.edu

1. Introduction

Distributed computing is a focus of interest in research and application, for which component-based software development is a promising solution [1]. Implementation of Distributed Computing Systems (DCS) with component-based software development requires combining components that adhere to different distributed models. A comprehensive framework is needed to provide seamless access to underlying components and to aid the design of DCS. A Unified Meta-component Model (UMM) and a Unified Approach (UA) based on it are proposed in [4] for distributed component systems.

2. Motivation:

It is desirable to have a formal method to represent the interaction between the components within the system. Components, by definition, are independent of the implementation language, tools and the execution environment. So formal specification is appropriate to represent the logical relation between components. A method of representing the logic in a formal specification to replace the ambiguous natural language, and translating them into designs is needed.

When considering the quality of a software system, the functional output is only part of the criteria and the Quality of Service (QoS) is very important to distinguish a good design from a poor one. So we are not only interested in using formal specification to represent the semantics of a software system, but in representing, verifying, and optimizing the QoS of the integrated system.

3. QoS Parameters in UA

The design of a software system is based on the users' requirement, consisting of functional and non-functional requirements (also called QoS requirement) [2]. Functional requirements define what a software system does, while QoS requirements specify how well it functions. Obviously, the former is much more concrete than the latter and has been earning more attention in system design in practice. In order to produce high quality software and for the purpose of upgrading and integration of the software system, it is realized that we have to take the QoS requirements into consideration in the process of software design and integration.

One reason that QoS requirements were rarely considered is they are usually described in an abstract or vague way and thus hard to be explicitly dealt with, for example, "the system must have satisfactory performance." Obviously, it is almost impossible to judge if a system meets the "satisfactory performance" requirement only based on such a description, since the level of "satisfaction" is usually not quantifiable. Such a description is too ambiguous for system design.

In considering non-functional abstractions, we take into account non-functional attributes, actions and properties. Non-functional attributes model the non-functional characteristics and features in quantified or non-quantified ways, such as reliability, a significant characteristic of which is that it can be decomposed into some more detailed properties. Non-functional actions have effect on non-functional attributes. On the other hand, a non-functional action, which implements or affects one non-functional attribute, may have effect over another non-functional attribute. This is called "correlation." Thus it is important to make sure the non-functional attributes affected by non-functional actions still meet the requirements. Non-functional properties combine the above by modeling the constraint over non-functional attributes [5], like how low the error rate is.

* This research is supported by the U. S. Office of Naval Research under the award number N00014-01-1-0746.

4. Formal Specification in UA

We would like a formal method to represent and verify the effect of the non-functional actions on the non-functional attributes. By this method, if we can verify the validity of each unit of a system, we can be sure about the overall validity of the integrated system even if some non-functional actions or attributes are changed. This expected formal specification would have such characteristics:

- The representation of the QoS is in the form of a class, which consists of the non-functional attributes it describes, and the constraints applied to each of the attributes, and represents the actions on the attributes, including the applied attributes, the effect of the action, and the correlated attributes
- The class supports multiple inheritance. As indicated above, decomposability is an important characteristic of non-functional attributes. Thus, a QoS parameter which constraints one attribute actually constrains all the attributes this attribute is decomposed into. So, if we already have the class to represent some of the constraints, we can efficiently reuse them so as to save the resources and prevent inconsistency. For example, when considering error rate, we need to include lost, delayed, and wrong information. So we can inherit these classes to make the error rate class.
- The effect of the non-functional actions on the attributes is defined within the class, too, including which attribute is affected, how it is affected, and all correlated attributes. An example of this is how an extra client affects the error rate of the system.
- A class defining the relation between the correlated attributes. Obviously, this class has some relationship more than inheritance with the correlated attribute classes. For example, if we have to change the system for throughput, we may need to consider the error rate and security as well.
- It has a high level of abstraction and is independent of the hardware/software implementation or platform.

The above is the syntax aspect proposal of the formal specification. As to the logic, we may take the approach of the 3-state value to represent no effect, in favor, and against respectively, since most of the QoS properties are not quantifiable and what we care about is just the existence of the effect and how it affects the system. The most important and difficult part of the formal method is to represent the interactive effects on the same object (an attribute) from multiple actions, the effects on multiple attributes by a single action, and correlation.

Two-Level Grammar (TLG) [3] is a good candidate for the basis of the formal specification language for this purpose. We can use the two context free grammars in TLG to define the non-functional attributes and the actions performed on them, the combined result could be extended to represent the QoS requirements. The type definition in TLG allows multiple attributes to be defined in a class, which is exactly needed in the QoS attributes. Multiple function signatures could be in one class in TLG, which is exactly similar to multiple actions affecting one attribute.

We can represent the QoS requirements in a formal way and take advantage of the formal specification to verify the functional and non-functional properties of the system at the early stages of design, and benefit in integration. This is especially important for heterogeneous system design in which each individual module is very likely to be running on different platforms. If we can verify the QoS for each individual unit by the formal method, we can be sure about the QoS of the whole system and free of inconsistency.

Like all formal specification languages, this language verifies the logical relation between different modules, for example the class inheritance, object interaction and compatibility, and including all the basic arithmetic functions.

5. Conclusion and Future work:

The approach mentioned above is a proposal to represent and verify the QoS parameters in the heterogeneous distributed software integration, and more work is needed for further research and implementation of this approach. Based on the above analysis, we can be confident to foresee that it would be feasible and useful to develop a formal specification language by applying and extending TLG in such an environment. Another related research task is the interface with Object-Oriented programming languages to develop a tool to generate standard template from the verified logic from the formal method, and thus standardize the programming language coding.

References:

[1] ASTER: Software Architectures for Distributed Systems, http://www-rocq.inria.fr/solidor/work/aster.html, 2001.

[2] Brahnmath, G. J., Raje, R. R., Olson, A. M., Auguston, M., Bryant, B. R., and Burt, C. C., "A Quality of Service Catalog for Software Components," to appear in Proc. $(SE)^2$, the Southeastern Software Engineering Conf., 2002.

[3] Bryant, B and Lee, B, "Two-Level Grammar as an Object-Oriented Requirements Specification Language", Proc. 35th Hawaii Int. Conf. System Sciences, 2002.

[4] Raje, R., Auguston, M., Bryant, B., Olson, A., Burt, C, "A Unified Approach for the Integration of Distributed Heterogeneous Software Components", Proc. Monterey Workshop on Engineering Automation for Software Intensive System Integration, 2001, pp. 109-119.

[5] Rosa, N., Cunha, P., Justo, G., "$Process^{NFL}$: A Language for Describing Non-Functional Properties", Proc. 35th Hawaii Int. Conf. System Sciences, 2002.