Tools for migrating, building, and maintaining distributed systems with Object-based Standards

P. Devanbu & K. Levitt & R. Olsson & R. Pandey
Computer Science Department,
University of California, Davis CA 95616
devanbu, olsson, levitt, pandey@cs.ucdavis.edu

November 24, 1997

Abstract

With the increasing emphasis on inter-operability, organizations are faced with the task of standardizing their systems to inter-operability standards such as CORBA. This task has two aspects: one is the aspect of bringing legacy systems to the standards compliance, and the second aspect is to create systems whose implementation is agnostic with regards to the specific (vendor's) interpretation of the standards. Our research interest is to develop a framework to characterize the difficulty of these tasks, and to develop tools and methodologies to support various aspects of the standards compliance process.

1 Introduction

We are concerned with the problem of migrating legacy systems to compliance with interoperability standards, and maintaining this compliance as the system evolves. The difficulty of the migration task varies with the type of migration requireed, the properties of the legacy system, etc. This paper is concerned with three issues: a) what are the factors that must be considered when planning for the migration? b) what are the types of incompatibilities that must be addressed during migration? and c) Once migration has been accomplished, how can the system be kept in "true" standards compliance, *i.e.*, retain the ability perform under any implementation of the standard?

2 Migration

2.1 Planning for Migration

During migration, several early planning decisions have to be made regarding the type of migration that is to be undertaken. We would like to develop a framework to support the planning of such tasks. Several important questions (list below) need to be addressed– managers and architects need tools, methodologies, processes etc to answer these questions, and use the information gained to plan the migration.

1. What is the degree of modification that can be tolerated in the legacy system? i.e., How "brittle" is it? How can this be determined?
2. What are the resources (time, personnel) available for the migration task?
3. What level of interoperability is required? i.e., are only the external interfaces targeted for standardization, or do the internal workings of the system also have to exposed to standardization?
4. What are the performance requirements? What will the impact of standardization be on the performance? Can lower performance be traded off against the benefits? Are there methods of developing good performance models that can predict the performance of the legacy system, after standardization?
5. Can this system be standardized in isolation, or are there other systems that will be impacted?

2.2 Designing the Migration

There are several types of incompatibility that have to be resolved prior to migration. They can be broadly classified as control, communication and data issues. Our goal is to explore these incompatibilities in detail and develop tools and methodologies that can be used to support the resolution of these incompatibilities during migration.

2.2.1 Data

Marshalling incompatibilities: representation incompatibilities, adequacy of the legacy "type language" versus the standards IDL, etc. We believe these can be addressed by a "stub-generator-generator" (based on GENOA/GENII [1]) that can be used to create custom stub generators to do data translation.

2.2.2 Communication

RPC/Protocol incompatibilities. Security handling differences. QOS incompatibilities. Can these be resolved? Are existing techniques [6, 4]. applicable?

2.2.3 Control/Architectural Style

What is the old architecture style [5]—client/server? event loop? Motif-style callback functions? Asynchronous messaging? RPC? How does this style match (or mis-match [3]) What is the impact of various legacy styles on the conversion effort?

3 Implementation Agnosticism

Different vendors of CORBA have slightly different implementation of the standards; some provide additional functionality, that are not in the standard, others provide alternative ways of implementing standards functionality in addition to the prescribed ones. It would be advantageous to write software that only uses the functionality that is clearly laid out in the standard, and avoids others. We would like to identify such a subset, and write language based tools (for C++ [2], Java etc) that can identify such non-standard code.

4 Conclusion

Migrating legacy systems to interoperability standards, and upholding the adherence of systems to standards are areas of interest to us. We would like to develop a framework to support the planning of such tasks, and tools and methodologies to support the actual development effort.

References

[1] P. Devanbu. Genoa- a language and front-end independent source code analyzer generator. In *Proceedings of the Fourteenth International Conference on Software Engineering*, 1992.

[2] P. Devanbu and L. Eaves. Gen++ manual, version 1.2 (send mail to `genserver-@research.att.com` for more information). Technical report, AT&T Bell Laboratories, November 1994.

[3] D. Garlan, R. Allen, and J. Ockerbloom. Architectural mismatch, or, why it's hard to build systems out of existing parts. In *Proceedings of the 17th International Conference on Software Engineering*. IEEE Computer Society, May 1995.

[4] J. M. Purtilo and J. M. Atlee. Module reuse by interface adaptation. *Software Practice and Experience*, 21(6), 1991.

[5] M. Shaw and D. Garlan. *Software Architecture: Perspectives on an Emerging Discipline*. Prentice-Hall, 1996.

[6] D. M. Yellin and R. E. Storm. Protocol specifications and component adaptors. *ACM TOPLAS*, 19(2), 1997.