Investigating the feasibility and worth of migrating legacy systems

Candidate: Patrick Naish - pn3g10@zepler.net
Supervisor: Prof. Michael Butler - mjb@ecs.soton.ac.uk

UNIVERSITY OF Southampton

Introduction

Mainframe computing has long been the backbone of technological industry. Despite having fallen out of favour in deference to the modern client-server style of interaction, mainframe software still underpins a vast amount of modern systems. Despite the fact that there were an estimated 200 billion lines of COBOL in use in 2008 [3], COBOL is widely considered to be a dead language, and the fact that young computer scientists are primarily taught languages such as Java [8] and Python means that many are unwilling to learn older languages. Those developers who do know the languages required to maintain legacy mainframe systems are rapidly reaching the age of retirement and leaving the industry. Thus, it is important to find a means by which to modernise legacy systems to preserve their functionality, while allowing for practical future maintenance.

Approaches to modernisation

The four main approaches suggested by Almonaies et al. [1] are:

Replacement The original system is rewritten in a new language or replaced with an off-the-shelf solution.

Reengineering Modern functionality is reverse-engineered into the legacy system.

Wrapping An interface is created to access legacy components as services without changing them.

Migration The legacy system is transitioned into a modern environment while retaining functionality.

The most common end-goal for modernisation is to be able to access the system as a **Web Service**[a].

[a] See `http://www.w3.org/TR/wsdl`

Tools

There are various tools available for aiding the modernisation effort. The examined tools fell into two categories: those for **analysis** of systems, and those for the actual **migration**.

An example of a tool for analysis is **COBAudit**, which collects metrics about COBOL programs [7]. These metrics can then be used to assess the complexity of a program, and said program's suitability for modernisation. **ARMIN** is a somewhat more specialised tool, which reconstructs a program's architecture from a static analysis of the code [6]. This can be used to identify dependencies within programs, which can then inform the creation of a migration strategy.

One set of tools for migration is **COB2WEB**, which comprises COBAudit along with COBStrip, COBWrap and COBLink [9]. **COBStrip** separates out a program into 'slices', which each represent a path through the program performing a distinct function. These are used to create cut-down versions of the original program performing only those functions, allowing for each piece of functionality to be represented by a separate Web Service. To this end, **COBWrap** adds an I/O layer to sit between the legacy code and the web, and **COBLink** generates Web Service Description Language (WSDL) interfaces for the same.

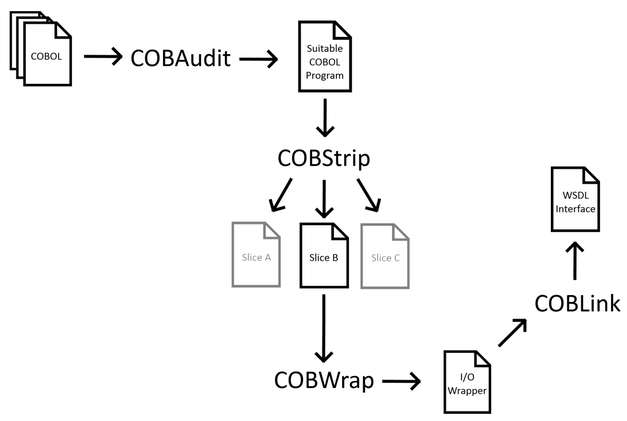

Figure 1: An illustration of the COB2WEB process

Techniques

Several techniques have been developed and tested for applying tools and practices to real-world systems. Most of those examined had been tested in at least a case study environment, although not all.

Among these techniques is the Service-Oriented Migration and Reuse Technique (**SMART**), a process for evaluating the potential for legacy components to become services [5]. This considers not only the complexity of the system, but also factors such as the target environment and user base. This gives a more complete overview of the system and its role in the company environment than a single tool such as COBAudit.

Another technique is **CelLEST**, developed as part of the CelLEST project [10]. This differs from many other techniques in that, rather than attempting to understand a system's architecture through code analysis, a model of the behaviour of the system is constructed based on user interface interaction. Requiring more direct user interaction than other techniques, it does have the advantage of reducing the necessity for large-scale software engineering processes.

The Ubiquitous Web Applications Design Framework (**UWA**) is a technique for reengineering [4]. It involves reverse-engineering of the legacy system to create 'meta-models', which can then be used to create concrete models for the original system, which can in turn be used to engineer a web version of the original system.

Finite State Automata have also been discussed as a potential tool for modelling user-system interactions [2]. This is to address the issue introduced when handling multiple connections as a Web Service – namely, the underlying system may be in an unknown state when a request is received, leading to unexpected behaviour. With a finite state automaton keeping track of the state of the system, the currently available I/O operations can be modelled in the wrapper layer, thereby mitigating the issue.

Conclusion

The main issues presented to the task of system modernisation and migration are those of cost-effectiveness, maintainability and automation. The difficulty lies in assessing whether it is possible to effectively modernise a system, and the actual benefits of doing so. A business must be able to find the most appropriate approach for their situation, and balance the up-front costs of more expensive approaches, such as migration, against the poor maintainability of cheaper solutions such as wrapping. Finding a means to automate (to some degree) the process of striking said balance is vital to the future of the field, and is likely to be where the bulk of new research and development is focused.

Overall, the available tools and techniques provide a significant basis for those wishing to attempt the modernisation of their systems, although as few have been tested widely in real-world scenarios some modifications may be needed to make them practical. New tools based on the currently existing ones are likely to emerge in future – and, in turn, new processes based around the use of these tools.

References

[1] A Almonaies, James Cordy, and Thomas Dean. Legacy system evolution towards service-oriented architecture. In *International Workshop on SOA Migration and Evolution (SOAME 2010)*, pages 53–62, 2010.

[2] G. Canfora, A.R. Fasolino, G. Frattolillo, and P. Tramontana. Migrating interactive legacy systems to Web services. *Conference on Software Maintenance and Reengineering (CSMR'06)*, 2006.

[3] Datamonitor. COBOL - continuing to drive value in the 21st Century. Technical report, 2008.

[4] Damiano Distante, Gerardo Canfora, Scott Tilley, and Shihong Huang. Redesigning Legacy Applications for the Web with UWAT+: A Case Study. In *Proceedings of the 28th International Conference on Software Engineering*, ICSE '06, pages 482–491, New York, NY, USA, 2006. ACM.

[5] G. Lewis, E. Morris, and D. Smith. Service-Oriented Migration and Reuse Technique (SMART). *13th IEEE International Workshop on Software Technology and Engineering Practice (STEP'05)*, pages 222–229, 2005.

[6] L. O'Brien, D. Smith, and G. Lewis. Supporting Migration to Services using Software Architecture Reconstruction. *13th IEEE International Workshop on Software Technology and Engineering Practice (STEP'05)*, pages 81–91, 2005.

[7] Harry M. Sneed. A pilot project for migrating COBOL code to web services, 2009.

[8] Harry M. Sneed and Katalin Erdoes. Migrating AS400-COBOL to Java: A Report from the Field. In *2013 17th European Conference on Software Maintenance and Reengineering*, pages 231–240, 2013.

[9] H.M. Sneed. COB2WEB a toolset for migrating to web services. *2008 10th International Symposium on Web Site Evolution*, 2008.

[10] E. Stroulia, M. El-Ramly, P. Sorenson, and R. Penner. Legacy systems migration in CELLEST. *Proceedings of the 2000 International Conference on Software Engineering. ICSE 2000 the New Millennium*, 2000.