ПЛОВДИВСКИ УНИВЕРСИТЕТ “ПАИСИЙ ХИЛЕНДАРСКИ”, БЪЛГАРИЯ
НАУЧНИ ТРУДОВЕ, ТОМ 37, КН. 3, 2010 – МАТЕМАТИКА
PLOVDIV UNIVERSITY “PAISSII HILENDARSKI”, BULGARIA
SCIENTIFIC WORKS, VOL. 37, BOOK 3, 2010 – MATHEMATICS

APPROACHES AND TECHNIQUES FOR LEGACY SOFTWARE MODERNIZATION

Anna Malinova

Abstract. This paper aims to provide an overview of the basic approaches to legacy software modernization. Discussed are black-box and white-box modernization techniques. Modernization towards an SOA environment and its realization through wrapping is also considered. A reference to a real world use case of modernization of legacy software in the domain of plasma physics and simulation of metal vapour lasers is also provided.

Key words: legacy software, modernization, wrapping, modernization towards an SOA

Mathematics Subject Classification 2000: 68N01

1. Introduction

Existing software systems need to evolve in order to face the evolution of technologies and the frequently changing business requirements. According to [17] software systems become legacy systems when they begin to resist modification and evolution. Assuming that these systems still provide significant business value, they must be modernized or replaced.

The aim of this paper is to provide an overview of the basic strategies, activities and techniques for modernization of legacy software. This research is connected with the modernization process we have performed in order to reuse and integrate legacy scientific codes in the domain of plasma physics and simulation of metal vapour lasers. The approaches we have used are based on wrapper techniques,

which are discussed in the following sections. The discussion makes references to some of the results of our work as well.

In Section two are presented the basic approaches to dealing with a system that turns out to be a legacy one. In Section three are discussed a number of primary legacy modernization techniques.

3. Modernization of legacy software – basic strategies and activities

System evolution covers a broad range of development activities - from adding a line of code to completely re-implementing the system. In [22] and [23] system evolution activities are divided in three categories: maintenance, modernization, and replacement. According to [17]:

- **Maintenance** is an incremental and iterative process in which small changes are made to a system. These changes are often bug fixes or small functional enhancements that do not involve major structural changes.
- **Modernization** involves more extensive changes than maintenance but conserves a significant portion of the existing system. These changes may include restructuring the system or enhancing functionality.
- **Replacement** requires rebuilding the system from scratch. Systems can be replaced either all in one by using the "big-bang" approach, or incrementally.

In this regard, Lehman's first law [9] states that software must be continually adapted or it will become progressively less satisfactory. Thus software maintenance and modernization help to keep applications up-to-date and in use. Modernizations generally refer to large-scale changes which help to extend the software's lifetime.

Depending on the required level of system understanding, modernization strategies can be classified into two different categories: "black-box" modernization and "white-box" modernization [17].

Black-box modernization requires knowledge of the external behaviour of the legacy system and involves examining of its inputs and outputs to understand the system interfaces. A common black-box method is "wrapping".

Wrapping can be defined as "surrounding the legacy system with a software layer that hides the unwanted complexity of the old system and exports a modern interface" [17]. In [5] is discussed the reengineering pattern "Present the right interface", which is aimed at wrapping a legacy system in order to export the right abstractions, even if they are not reflected in the existing implementation. Possible problems, hints, pros, and cons are also considered. The proposed solution is

connected with identifying the abstractions that are needed in the new system and wrapping up the old software in order to emulate the new abstractions.

In [18] is discussed that wrapping can be accomplished at multiple levels corresponding to the levels at which one can access the legacy software application: process level, transaction level, program level, module level, and procedural level. In [19] the process level is considered the simplest form of encapsulation, while procedural level is regarded as "the most challenging form of wrapping" since an internal procedure is invoked as if it were a separately compiled module. In [19] it is also stated that wrapping legacy software is normally done in three steps: 1) the wrapper should be constructed; 2) the target programs should be adapted; 3) the interaction between the wrapper and the target program should be tested. Here, adaption involves performing changes to the target system. In contrast, in our work we have performed wrapping without making any changes to the legacy code, as presented in [6], [12] and [13].

White-box modernization is more extensive and complex than black-box approach. It requires understanding of the legacy system internals and is also known as "software reengineering".

Reengineering of legacy systems is defined in [3] as "examination and alteration of a subject system to reconstitute it in a new form and the subsequent implementation of the new form".

Although the reasons for reengineering a system may vary, the actual technical problems are typically very similar and may include: dividing monolithic systems into separate parts for easier marketing; improving maintenance, portability, etc.; increasing efficiency; migration to a different platform; adoption of new technologies.

The reengineering process includes three phases: forward engineering, reverse engineering, and reengineering [5]. *Reverse engineering* reconstructs higher-level models and artefacts from code to achieve program understanding. Reverse engineering involves such activities as re-documentation and design recovery [3]. In contrast, *forward engineering* can be understood as a process of moving form high-level abstractions and logical, implementation independent designs, to the physical implementation of a system [3]. In this context, *reengineering* is a process that transforms one low-level representation into another. The actual code transformations during reengineering are performed through a number of techniques that involve restructuring. According to [3] restructuring is "the transformation from one representation form to another at the same relative abstraction level, while preserving the systems external behaviour". A typical example of restructuring is the transformation of unstructured "spaghetti" code to a structured one. *Refactoring* is restructuring within an object-oriented context. It is defined in [7] as "process of

changing a software system in such a way that it does not alter the external behaviour of the code yet improves its internal structure". This may be renaming (fields, variables, classes), changing the physical organization of code (e.g. moving packages and classes), changing the logical organization of code at class level (e.g. moving methods or fields from a class to a subclass or superclass), changing the code within a class (e.g. turning local variables into class fields), etc.

Although white-box and black-box approaches suggest wrapping as an alternative strategy to reengineering and redevelopment, quite often wrapping is introduced as one of the techniques to carry out the reengineering [5], [15], or it is defined as a "black-box reengineering task" [17], [4]. This assumes broader understanding of the reengineering process that depends on the level of abstraction at which wrapping has been performed. For instance, the wrapping techniques and practical experience presented in [6], [12] and [13] show that most often wrapping is not entirely a "black-box" approach and requires some level of reverse engineering for better understanding of the wrapped legacy interfaces, class hierarchy, or objects interrelations. In this process a need for re-documentation and design recovery may appear. In addition, in our work, after completion of the wrapping process, a subsequent process of forward engineering has been performed over the wrappers in order to extend the functionality of the legacy system, add safety or new features in wrappers by the use of the new technologies that became available as a result of the overall reengineering process.

3. Legacy modernization techniques

According to [14] one of the main difficulties of software evolution is that all artefacts produced and used during the entire software life-cycle are subject to changes, ranging from early requirements over analysis and design documents, to source code and executable code. In [14] is also stated that this fact automatically spawns many subdisciplines in the research domain of software evolution, some of which are: requirements evolution, architecture evolution, data evolution, runtime evolution, Service-Oriented Architectures (SOA), language evolution. Furthermore, in [4] is discussed that legacy systems may be modernized at functional (logic), data, or user interface level. In this context, it is obvious that a collection of different modernization techniques is needed for each of these modernization levels and areas of software evolution. For instance: a common technique for user interface modernization is "screen scrapping" which provides old (usually text-based) interface with new (graphical or web) one; data modernization may be connected with an XML integration; modernization at functional level may include techniques for object-oriented wrapping or component wrapping.

Among the existing number of modernization techniques, several can be pointed out as primary [8], [20], [21], [24]:

- **Automated migration** – migration of languages, databases and platforms using software tools like automated parsers and converters. Some examples are: migration of legacy applications and data that use legacy databases and file systems to relational databases, program restructuring (dead code elimination, GOTO elimination), etc. Automated migration suggests that the transformation process is "algorithmic" in nature and does not require injection of human intelligence into the transformation process [20].
- **Re-hosting** – running the legacy applications on a different platform. The business logic and data of legacy applications remain intact in the new platform. Re-hosting is often used in combination with other modernization techniques, such as automated migration.
- **Package implementation** – replacement of legacy applications with commercial-off-the-shelf (COTS) packages [20]. This option focuses on building a portfolio with the best packages and components available from third-party vendors. However, reuse of existing legacy business logic is not possible with this approach [21]. Some level of reengineering or customization of packages and rewriting business logic may be involved in this process as well.
- **Reengineering/Re-architecturing** – the most efficient modernization technique to transform legacy applications. It works by gathering requirements from existing legacy applications and redeveloping them on newer platforms using new technologies [21]. A typical example is the adoption of modern technology with new architectural paradigms like Service-Oriented Architecture through reengineering.
- **SOA Integration** – expose business logic and data embedded in legacy programs as well-defined, reusable services. As discussed in [20], the simplest way to address the legacy modernization is to "wrap" existing application interfaces through SOA wrappers, thus creating SOA services that can be registered to an SOA management facility on a new platform, but are implemented via the existing legacy code.

Modernization towards an SOA environment is a major trend in legacy software evolution. As listed above, it can be achieved through reengineering activities, as well as trough wrapping techniques, depending on whether the legacy code will be changed or not. Web services-based SOA addresses many of the legacy modernization issues, providing interoperability, application integration, reusability, and flexibility, which are motivated mainly by the loosely coupled nature of the web

services. Modernization towards an SOA adds to the discussed general modernization strategies activities, such as [2]:

- Identifying the candidates for services – what can be defined as a service and then choose the services with the greatest business value and the least business cost;
- Salvaging the legacy code - locate that code and determine its worthiness for reuse, extract it and reassemble it as a separate module with its own interface;
- Wrapping the salvaged code - the final aim of the wrapping process is producing Web Services Description Language (WSDL) interface for the legacy code;
- Linking the services into a business process.

Hence, within white-box and black-box modernization strategies, *wrapping* and *componentization* can be specified as basic techniques aimed at achieving service orientation. Wrapping provides legacy components with a new WSDL interface, making them easily accessible by other software components and thus facilitates the SOA principle of interoperability. Wrapping concentrates on the interface of the legacy system, hiding the complexity of its internals [1] and thus provides for flexibility. Componentization involves restructuring in order to group together functionality and data into components. This provides for fulfilment of the SOA principles of loose coupling and reusability.

In [10], [11] and [12] is presented a modernization towards an SOA in order to perform numerical simulation of metal vapour lasers. This work is connected with creating Java wrappers of existing legacy physics software. These codes are modules written in the C, C++ and FORTRAN languages. Next, some of the wrapped modules were converted into web services and orchestrated by a Business Process Execution Language (BPEL) process of simulation.

The modernization process presented in [6] and [13] is connected with the creation of the WebPlasimo prototype which provides new interfaces to the Plasimo framework for modelling low-temperature plasma sources [16]. The aim of this work was: 1) to create a web interface to the Plasimo framework; 2) to expose certain Plasimo functionalities as web services for use by other scientific groups. Both tasks involve the creation of Java wrappers of basic Plasimo functionality. In our work wrapping was chosen as a preferred modernization strategy because this is the only approach that does not entail performing code changes to the legacy system and because this technique allows for reusing from one side, and application of modern technologies from the other.

4. Conclusions

Legacy applications are increasingly becoming a problem for all kinds of companies and organizations. Modernizing approaches and techniques allow for lowering the cost and complexity of legacy systems. In this process it is very important to choose the appropriate strategy according to the defined levels of usage of existing application assets and movement toward better technology environments.

5. Acknowledgements

This work was partially supported by the RS-2009-M13 project of the Scientific Fund of the University of Plovdiv "Paisii Hilendarski", Bulgaria.

References

[1] Almonaies, A., J. Cordy , T. Dean, Legacy System Evolution Towards Service-Oriented Architecture, Proc. SOAME 2010, Int. Workshop on SOA Migration and Evolution, Madrid, Spain, March 2010, pp. 53-62.

[2] Bhattacharya, S., Integrate legacy systems into your SOA initiative, 2007, http://www.ibm.com/developerworks/webservices/library/ws-soa-legacyapps/index.html.

[3] Chikofsky, E., J. Cross II, Reverse engineering and design recovery: A taxonomy, Software Reengineering, IEEE Computer Society Press, 1992, p.54–58.

[4] Comella-Dorda, S., K. Wallnau, R. Seacord, J. Robert, A Survey of Black-Box Modernization Approaches for Information Systems, Proc. of the Int. Conf. on Software Maintenance, 2000, ICSM. IEEE Computer Society, Washington, p. 173.

[5] Demeyer, S., S. Ducasse, O. Nierstrasz, Object-Oriented Reengineering Patterns, Square Bracket Associates, Switzerland, 2009.

[6] Dijk, J. van, A. Malinova, V. Yordanov, Mullen J.J.A.M. van der, New Interfaces for the Plasimo Framework, 6th Int. Conf. on Atomic and Molecular Data and Their Applications, Beijing, China, 27 - 31 Oct. 2008, AIP Conf. Proc., Vol. 1125, 2009, pp. 176-187.

[7] Fowler, M., K. Beck, J. Brant, W. Opdyke, D. Roberts, Refactoring: Improving the Design of Existing Code, AddisonWesley, 1999.

[8] Laszewski, T., J. Williamson, Oracle Modernization Solutions, Packt Publishing, 2008.

[9] Lehman, M. M., Ramil, J. F., Wernick, P. D., Perry, D. E., Turski, W. M, Metrics and Laws of Software Evolution - The Nineties View, Proc. of the 4th Int. Symp. on Software Metrics, METRICS'97, IEEE Computer Society, Washington, 1997, p. 20.

[10] Malinova, A. A., S. G. Gocheva-Ilieva, Application of the Business Process Execution Language for building scientific processes for simulation of metal vapor lasers, Proc. of the 3rd Balkan Conf. in Informatics, Sofia, Bulgaria, 27-29 Sept., 2007, Volume 2, pp.75-86.

[11] Malinova, A. A., S. G. Gocheva-Ilieva, I. P., Iliev, Web services – based simulation of metal vapor lasers, Proc. of the IX Int. Conf. on Laser & Laser Inf. Techn. & V Int. Symp. on Laser Techn. & Lasers ILLA/LTL '2006, Smolyan, Bulgaria, October 4-7, 2006, pp. 315-321.

[12] Malinova, A. A., S. G. Gocheva-Ilieva, I. P., Iliev, Wrapping legacy codes for Numerical simulation applications, Proc. of the III International Bulgarian-Turkish Conf. Computer science, Istanbul, Turkey, October 12-15, 2006, Part II, pp. 202-207, 2007.

[13] Malinova, A., V. Yordanov, J. van Dijk, Leveraging existing plasma simulation codes, International Book Series "Information Science & Computing", Number 5, pp.136-142, Suppl. to the Int. J. "Information Technologies & Knowledge", Vol. 2/2008.

[14] Mens, T., S. Demeyer, Software Evolution, Springer, 2008.

[15] Norton, D., V. Decyk, Re-engineering legacy mission scientific software, Space 2001 Conference and Exposition Albuquerque, New Mexico, USA, 2001.

[16] Plasimo simulation software, http://plasimo.phys.tue.nl

[17] Seacord, R., D. Plakosh, G. Lewis, Modernizing Legacy Systems: Software Technologies, Engineering Processes, and Business Practices, Addison-Wesley, 2003.

[18] Sneed, H., Encapsulating Legacy Software for Reuse in Client/Server Systems, Proc. of WCRE-96, IEEE Press, Monterey, 1996.

[19] Sneed, H., Encapsulation of legacy software: A technique for reusing legacy software components, Ann. Softw. Eng, 9, 1-4 Jan. 2000, pp. 293-313.

[20] Venema, T., The Oracle IT Modernization Series: The Types of Modernization, An Oracle White Paper, http://www.oracle.com/technologies/modernization/docs/typesofmodernization.pdf, 2008.

[21] Venkatraghavan, N., Legacy Modernization: Modernize and Scale, InfoSys White Paper, http://www.infosys.com/microsoft/resource-center/Documents/legacy-modern.pdf, 2008.

[22] Weiderman, N., J. Bergey, D. Smith, Tilley, Scott R., Approaches to Legacy System Evolution (CMU/SEI-97-TR-014), Pittsburgh, Pa.: Software Engineering Institute, Carnegie Mellon University, 1997.

[23] Weiderman, N., L. Northrop, D. Smith, S. Tilley, K. Wallnau, Implications of Distributed Object Technology for Reengineering (CMU/SEI-97-TR-005 ADA326945). Pittsburgh, Pa.: Software Engineering Institute, Carnegie Mellon University, 1997.

[24] Zou, Y., K. Kontogiannis, Re-engineering Legacy Systems to Web-enabled Environments, In book "Managing Corporate Information Systems Evolution and Maintenance", Idea Group Publishing, Hershey, PA, USA, pp. 138-146, 2004.

Faculty of Mathematics and Informatics
University of Plovdiv
236 Bulgaria Blvd.,
4003 Plovdiv, Bulgaria
e-mail: malinova@uni-plovdiv.bg

Received 18 September 2010

ПОДХОДИ И ТЕХНИКИ ЗА МОДЕРНИЗИРАНЕ НА НАСЛЕДЕН СОФТУЕР

Анна Малинова

Резюме. Целта на статията е да се направи обзор на основните подходи за модернизиране на наследен софтуер. Дискутирани са техники, изискващи познания за вътрешната архитектура и имплементация на наследения код (white-box modernization), както и техники, фокусирани върху функционалните изисквания и интерфейса на системата (black-box modernization). Разгледана е и модернизацията на наследен софтуер в посока на архитектура, ориентирана към използването на услуги (Service-Oriented Architecture, SOA) и нейното реализиране чрез обвиване. В текста се правят препратки към основни резултати от модернизирането на наследени физични кодове в областта на физиката на плазмата и симулацията на лазери с метални пари.