

Specifications of COTS-based systems

Sofia Guerra and Anthony Finkelstein

Department of Computer Science

University College London

Gower Street

London WC1E 6BT, UK

{S.Guerra,A.Finkelstein}@cs.ucl.ac.uk

1 INTRODUCTION

Using ‘Commercial Off-The-Shelf’ (COTS) software components to build systems has been proposed as a means of developing software with reduced risk and cost while increasing functionality and capability of the system. Building a system based on COTS components involves buying a set of pre-existing, proven components, building extensions to satisfy local requirements, and gluing the components together. The advantages claimed are that the COTS components are honed in the competitive marketplace resulting in increased capability, reliability, and functionality for the end user over what would be available from customer built components. COTS software components from different vendors are expected to be integrated easily, work in a range of environments, and support extensions and tailoring to local requirements.

The reality of the situation is quite different. The development of COTS-based systems, and the migration of legacy systems towards COTS practices are complex. New products and technologies constantly emerge into the marketplace. The vendors seek to distinguish their products from those of their competitors. This leads to a marketplace characterised by several different products with different capabilities and claims, and many product incompatibilities, even when the vendors claim compatibility. COTS-based systems, as they stand, present a number of challenges. It is difficult to understand the actual functionality and capability of a product, and to select the best among the available packages according to the customers requirements. Once a package is selected, it is difficult to foresee and resolve mismatches between the package and the system into which it is inserted.

As noted elsewhere, building systems from off-the-shelf components is a type of software re-use [10], where components are bought from third party developers and then integrated into the system. Inserting a component into a system also resembles the ‘feature addition’ to software systems, a problem well known in the telecommunication systems community. Our research program explores some of the ideas developed for spec-

ification of software re-use and particularly feature addition, and explores their suitability to software specification of COTS-based systems.

2 MOTIVATION

Requirements elicitation

The acquisition of requirements for acquiring COTS-based systems is interactive and incremental [4]. The initial requirements based on the customer’s understanding of the domain and operational environment constraints are revised on the basis of advertisements, descriptions and demonstrations provided by suppliers. These requirements are prioritised according to several factors. For example, some requirements might be irrelevant for some customers, while others are essential for the addition of the component to the system in question. It is important to consider specific constraints of the operating environment into which the package is going to be inserted. These have to be combined with the necessity for flexibility to accommodate the fluctuations of the commercial marketplace and to provide the possibility to use the same package in different environments.

Understanding and evaluating components

A component which is worth buying is likely to be a complex piece of software. In order to use such a component effectively it is necessary to understand it at quite a deep level. Customers have to understand whether packages satisfy their requirements. Equally, it is important to understand the consequences of the integration of a package into the initial system, as well as the interaction between several packages that might be already integrated. Extensive evaluation of the COTS component will be required to ensure not only that the component has the functionality to perform the required tasks within the system, but also that the additional functionality inherent within the component does not interfere with the system.

Evolution of software

Systems and COTS components within systems are constantly changing. Organisations implementing COTS-based systems strategies for new or legacy systems must consider not only immediate system requirements but also the unceasing evolution of computing and software

technology [9]. Upgrading a COTS-based software system means that as new releases of the commercial components are made available by the various vendors, the system will have to incorporate them. However, multiple component upgrades can result in numerous unforeseen problems. New releases of the COTS components will probably entail changes in the behaviour or assumptions; specialisation and extensions to the older version of the COTS-component must be integrated into the new component. Whenever a new version of a component is purchased, it is necessary to remove the older version and add the new one.

3 RESEARCH AGENDA

Hierarchic (default) specifications

When purchasing a software package, the customer begins with a set of requirements (organised by priorities), and revision of these requirements are introduced at a later stage. Therefore, it would be desirable to be able to revise a specification en route. Defaults can be seen as a high-level method of revising specifications; if a way of handling defaults is available, it is possible to say that we want as much as possible of a specification, given that we also want certain property. Thus defaults can be used to specify requirements of software packages and their revisions.

In order to have several levels of priorities among (possible contradictory) requirements, it is desirable to assign different levels to the defaults. The same idea has been used in specifications of object-oriented systems [1]. Prioritised defaults set up a way of dealing with inheritance hierarchies in object-oriented specifications: defaults for general objects are implicitly inherited by more specific objects; but defaults applying to more specific objects override those applying to less specific when there is a conflict. This idea is formalised by the concept of *hierarchic specifications*, where defaults are organised by different priority levels.

Introducing a software component may refine the set of possible behaviours of a specification, but more likely it will *revise* it. This is an important distinction. When refining a set of behaviours we get a subset of the original set (taking account of the new constraints). Revising a set of behaviours, on the other hand, introduces new behaviours which do not correspond to any old ones. This is desirable when the new constraints are inconsistent with the old specification. Introducing a software package might lead to non-monotonic extensions of the initial behaviour. Therefore, adding a new component does not simply involve the addition of further axioms. It involves removing axioms, or preferably, it involves some way of generating non-monotonic extensions. Defaults have been suggested as a way of dealing with similar problems in feature addition [6, 8], and their applicability to the specification of COTS-

components is promising. In this case, hierarchic specifications would be considered, where the defaults of the COTS-components are assigned higher priority than the ones of the system where they are being inserted.

COTS-components as specification units

The need to add, remove or respecify software components requires us to build specifications in a way that enables those components to be handled easily. In the same way specifications can be described in terms of their features, we proposed a coarse framework where specifications are described in terms of their packages.

The specification of a software package should be easy to add (possibly into several different specifications) and, as importantly, easy to remove. However, typically a software component is widely spread over the whole system. In order to promote the evolution of COTS-based systems, the specification of a component should be as minimally dependent on the underlying specification as possible. Thus, the idea of providing a ‘COTS-oriented’ approach to software specification is natural.

Another issue from the description of features that seems important to the specification of COTS-based systems is related to the minimal assumptions the feature makes about the specification to which it is being inserted. Although we might want to apply the same component to more than one specification, a given software package cannot be applied to any system. In order to be applicable to a specification, a component will require the system to have a certain form or satisfy some conditions. If these elements are not present, then the component cannot be added. If the elements are present, then adding the component will transform the specification by extending it to include the new behaviour of the component added.

If components and systems are hierarchic specifications, we can get the extended specification if we compose the specifications. Much formal work have been devoted to the composition of specifications [5]. Formal work has also addressed composition of default specifications, which would provide the ‘glue’ to integrate components and a framework to understand the consequences of integration. Similar ideas have been proposed for the specification of feature addition.

4 CONCLUSION

In this paper we review some of the problems of COTS-based systems specification and compare them with related work. We propose a research agenda based on ideas suggested for the specification of features. This agenda pursue a theoretical approach to the specification of COTS-based systems.

REFERENCES

- [1] S. Brass and U. Lipeck. Semantics of inheri-

- tance in logical object specifications. In C. Delobel, M. Kifer, and Y. Masunaga, editors, *2nd International Conference on Deductive and Object-Oriented Databases (DOOD'91)*, volume 566 of *Lecture Notes in Computer Science*, pages 411–430. Springer Verlag, 1991.
- [2] K. E. Cheng and T. Ohta, editors. *Third International Workshop of Feature Interactions in Telecommunications Systems*, Tokyo, Japan, October 1995. IOS Press.
- [3] J. Fiadeiro and P.-Y. Schobbens, editors. *ModelAge'96: Proceedings of Second Workshop of the ModelAge Project, January 15-17*, Sesimbra, Portugal, 1996.
- [4] A. Finkelstein, M. Ryan, and G. Spanoudakis. Software package requirements and procurement. In *Proceedings 8th International Workshop on Software Specification and Design IWWSSD-8*, pages 141–146. IEEE Cs Press, 1996.
- [5] J. A. Goguen and R. M. Burstall. Institutions: Abstract model theory for specification and programming. *Journal of the Association for Computing Machinery*, 39(1):95–146, January 1992.
- [6] S. Guerra and M. Ryan. Towards feature-oriented specifications. In Fiadeiro and Schobbens [3], pages 125–136.
- [7] G. Saake and C. Tücker, editors. *FIREworks'98: Proceedings of the 1st FIREworks Workshop, May 15-16*, Magdeburg, Germany, 1998. Fakultät für Informatik, Universität Magdeburg.
- [8] H. Velthuijsen. Issues of non-monotonicity in feature-interaction detection. In Cheng and Ohta [2], pages 31–42.
- [9] M. R. Vigder and J. C. Dean. Managing long-lived COTS based systems. In *Software Engineering Standards Workshop*, Monterey, California, August 1998.
- [10] M. R. Vigder, W. M. Gentleman, and J. C. Dean. COTS software integration: State of the art. Technical report, National Research Council Canada, 1996.