Analysing the Impact of Change in COTS-Based Systems

Gerald Kotonya and John Hutchinson

Computing Department, Lancaster University, Lancaster, LA1 4YR, UK
{gerald, hutchinj}@comp.lancs.ac.uk

Abstract. Commercial off-the-shelf (COTS) software components promise benefits in terms of greater productivity, reduced time to market and reliability. However, their blackbox nature poses significant challenges assessing and managing the impact of change. We propose an approach to help developers to understand the impact of change. It relies on the use of a COTS component-oriented development process and an architecture description language (ADL) for documenting component system architectures; both elements contributing to create a combined approach to impact analysis in COTS-based system.

1 Introduction

Component-based software engineering (CBSE) promises to revolutionize the way software is developed with the re-use of stable software components giving more functionality for less effort along with benefits in terms of time to market and reliability. Components-based development (CBD) should make it possible for developers to buy in "expertise" from the market place in a form that is tested and reliable, pluggable and cost effective. Pluggable in this case means in a form that can be incorporated in the intended system with minor or no modification. This view is consistent with both the application software market for PCs and the prevailing situation in other engineering disciplines where it would be inconceivable to think about developing each and every component from scratch.

Unfortunately, the potential benefits of COTS components come at a price. Their blackbox nature presents novel maintenance challenges while their commercial nature leaves their users the dilemma of choosing between enforced upgrade and obsolescence; change is imposed for what are essentially arbitrary reasons [12, 14].

2 Code as Documentation

The fundamental problem associated with blackbox components is that they can never be fully documented. In a traditionally developed system, the final level of documentation that can be used to resolve all questions about the system is the code. This is not to suggest that program code forms a straightforward and easy to understand documentation of the system, but ultimately, the question "what will happen if I change X?" can be answered by an experienced engineer examining the code to assess the consequences of the proposed change. If a software component is

X. Franch and D. Port (Eds.): ICCBSS 2005, LNCS 3412, pp. 212–222, 2005.

provided as a blackbox component, then its code is not available for inspection, and thus an engineer can assess the consequences of a change only by examining the component documentation, or by carrying out blackbox tests.

The documentation supplied with a component can only ever be prepared to satisfy the foreseen needs of users. Where unforeseen needs coincide with a component's undocumented design assumptions, the user will have a system that is potentially almost impossible to maintain. Underpinning the task of impact analysis (IA) in COTS component-based system is the notion of "process". It is necessary to document the process by which a system is developed, *as it is developed*, if it is going to be possible to analyse the potential impacts of future changes.

3 A Component-Based Development Process

We have developed a method called COMPOSE – COMPonent-Oriented Software Engineering [7] – for CBD. A significant element of the method, with respect to later IA, is COREx – Component-Oriented Requirements Expression [6]. The process by which requirements are elicited and manipulated is vitally important in CBD [1].

There has been little work on the problem of how to derive requirements for component-based systems. Vigder [10] rightly points out that flexibility is vital and proposes that system requirements should be defined according to what is available in the marketplace. However, some requirements are specific in nature and flexibility is not always an option. Another approach integrates the requirements process with COTS product selection (the Procurement-Oriented Requirements Engineering method - PORE [8]). Whilst this sheds light on the product evaluation process, it reduces the scope for requirements negotiation.

The COMPOSE method, illustrated in Fig. 1, embodies a cyclical development process that integrates verification into every part of the process to ensure that there is an acceptable match between components and the system being built. It also includes "negotiation" in each cycle as an explicit recognition of the need to trade-off and accept compromise in the successful development of component-based systems. This ensures that even the earliest stages of system development are carried out in a context of COTS component availability, system requirements and critical architectural concerns.

We use an intermediate modelling layer, comprising services and constraints, to map from requirements to components [6, 11]. Services represent expressions of functionality expressed in a way which shows how available components satisfy requirements. Constraints may represent non-functional requirements such as a component cost, certification, memory and platform restrictions, or dependability requirements such as security, performance and availability [13]. They may also represent elements of interdependence that are introduced to allow services to meet certain architectural considerations (e.g. Service X and Service Y may not reside in the same COTS component).