Stability for Component Integration Assessment

Alejandra Cechich
Departamento de Ciencias de la Computación, Universidad Nacional del Comahue, Neuquén, Argentina
Email: acechich@uncoma.edu.ar

Mario Piattini
Escuela Superior de Informática, Universidad de Castilla-La Mancha, Ciudad Real, España
Email: Mario.Piattini@uclm.es

Abstract. Component-Based Software Development is focused on assembling previously existing components (COTS and other non-developmental items) into larger systems, and migrating existing systems toward component approaches. Ideally, most of the application developer's time is spent integrating components. We present an approach that can be used in the process of establishing component integration's quality as an important field to resolving CBS quality problems – problems ranging from CBS quality definition, measurement, analysis, and improvement to tools, methods and processes. In this paper, we introduce an important property we called *system's stability* as part of a cycle for assessing and improving component-based systems. This property is the basis for determining the impact of incorporating COTS components into a stable system.

1. Introduction

Ideally, most of the application developer's time is spent integrating components. In essence, using components to build systems reduces complexity because composers do not need to know how a component works internally. Components are plugged into a software architecture that connects participating components and enforces interaction rules. Architectures mediate and regulate component interaction. For example, in (Wallnau et al., 2001), there is a strong condition to integrate architectural reasoning and component technology: that the architectural structure used to reason about the quality attributes of a system is very similar to the structure of a deployed component assembly that implements the system. As another example, the method presented in (Bose, 1999), focuses on the role of the connectors to provide causal connection between the components and coordinate the component behaviours to satisfy scenario specific ordering constraints.

Interactions might potentially be characterised along many dimensions, depending on the architecture they are immerse. Interactions, as part of a component model, require an accepted component vocabulary and a set of design standards (Sparling, 2000). In general, the application composer should be able to search for patterns of interaction that reveal possible adaptation problems and allow calculating interaction effort. For example, the proposal in (Yakimovich et al., 1999), presents a general classification of possible types of mismatches between COTS products and software systems, which includes architectural, functional, non-functional, and other issues. As another example, the work presented in (Davis and Gamble, 2002), combines model-based development (e.g. architectural modelling) with component-based development (e.g. COTS and legacy systems), and shows how their mismatch-detection capabilities complement one another to provide a more comprehensive coverage of development risks.

As important as determining architectural mismatching is calculating integration effort. Decisions on CBS investments are strongly influenced by technological diversity: today technology is diverse and brings with it thousands of choices on components, with their opportunities and risks.

Architectural frameworks such as CORBA can help in determining the effort; however, estimation is not straightforward. BASIS (Ballurio et al., 2002), for example, combines several factors in order to estimate the effort required to integrate each potential component into an existing architecture – architectural mismatching, complexity of an identified mismatch, and mismatch resolution.

Finally, architectures used to build composite applications provide a design perspective for addressing interaction problems, although little attention is paid to the evolvability of these architectures. For example, the proposal in (Davis and Gamble, 2002), uses the history of interoperability conflicts and resolution decisions as a basis for understanding the design, thus evolution is possible with minimal changes to the integration solution.

In this paper, we briefly present an important property we called *system's stability*. This property is the basis for determining the impact of incorporating COTS components into a stable system. We discuss future work and conclusion afterwards.

2. Defining System's Stability

The characteristics of the system's stability are defined at two levels. At the highest level, the stability is conceptualized in terms of its functionalities for system's users. The users include client representatives, managerial staff, corporate lawyers, marketing experts, etc. Their perceptions of what constitutes a stable system need to be captured and reconciled. Hence, users must communicate, collaborate, and coordinate. At the lower level, one would identify the architecture's basic units and components and their relationships. In practice, often it is necessary to group units together. Thus, a careful management of the relationships between basic and compound units, and the corresponding processes that perform the mappings are critical because of their architectural impacts.

With the characteristics of the product specified, the next step is to identify its quality requirements considering the perspectives of developers, consumers, and managers. To introduce our approach, a number of concepts must be defined. We first define the notion of *constraint scope (CS)* of a system as the set of all aspects that influence and/or constraint system's requirements. Typically, the constraint scope will include aspects such as goals, schedule, cost, context, and domain – we consider as domain constraints those in which the application domain has been the cause of changes on the system's architecture, in contrast to context constraints, which have been caused by execution environment conditions. The notion of *system's functionality* is associated with a set of requirements, as distinct from the *system's quality properties* more related to non-functional features. Finally, the notion of *traced architecture* is defined as a set of basic or compound architectural units suitable of being connected at least to one requirement.

More formally, information for defining system's stability, or s*ystem's basics*, can be summarised as the tuple $SB = (CS, R, QA, TA)$ where CS represents the constraints on the system, R is the set of requirements, QA is the set of quality attributes of the system, and TA is the software architecture, including the definition of every piece of software with its interfaces and its relationships with one or more requirements. To be measured, every quality attribute has an acceptance threshold associated. So, let Q a particular attribute, M a measure and TH its accepted threshold, a quality attribute set is expressed as the $QA = (Q \times (M \times TH)\text{-}set)\text{-}set$. Similarly, let P a piece of software, *I-set* its set of interfaces – provided and required – and L a relationship that direct or indirectly links a piece of software to a non-empty set of requirements, a traceable software architecture is expressed as the $TA = ((P \times I\text{-}set) \times L)\text{-}set$.

Then, we define *system's stability*, *SS*, as a state of a system or part of a system in which a set of requirements is satisfied by an architecture, and whose quality attribute's measures are acceptable according to given thresholds. Besides, every piece of software belonging to the architecture contributes to reach the requirements by providing a particular functionality identified by the

interfaces, and the contribution to the requirements is established by trace ability. In other words, we refer to system's stability as the property of a system to have a set of measurable quality attributes defined and committed by the users, which are satisfied by some of the system's outputs. Full details of these specifications can be found in (Cechich A. and Piattini M., 2003).

3. Conclusions

We have identified an important property we called *system's stability* when component compositions are assessed. This property is the basis for determining the impact of incorporating COTS components into a stable system.

However, this is only the beginning. After a system is altered by incorporating COTS components, the real work starts: measuring changes and possible perturbations on the system. Some works on this direction propose to measure the complexity of the interactions – and its changes – by using metrics derived from the information theory, such as the L-metric (Chapin, 2002) (Shereshevsky et al., 2001), although further research is currently needed on this area. A possible future work would be on identifying the difference of having direct and indirect measures of system's stability. Another aspect that needs further discussion is the possibility of establishing a set of measures that determine the stability level of a system – a system's stability scale ranging from completely stable through completely unstable.

4. References

Ballurio K., Scalzo B., and Rose L, 2002. Risk Reduction in COTS Software Selection with BASIS. In *Proceedings of the First International Conference on COTS-Based Software Systems*, ICCBSS 2002, Springer-Verlag Berlin Heidelberg New York, pp. 31-43.

Beck K. and Cunningham W., 1989. A Laboratory for Teaching Object-Oriented Thinking, *Proceedings of OOPSLA '89* , *ACM SIGPLAN Notices vol.24*(10), pp.1-6.

Bose P., 1999. Scenario-Driven Analysis of Component-Based Software Architecture Models, In *Proceedings of the First Working IFIP Conference on Software Architecture*, San Antonio, TX, USA, available at http://www.ece.utexas.edu/~perry/prof/wicsa1/final/bose.pdf

Cechich A. and Piattini M., Defining Stability for Component Integration Assessment, *ICEIS 2003 - 5th International Conference on Enterprise Information Systems*, Angers – France, 23-26 April, 2003 (to appear).

Chapin N., 2002. Entropy-Metric For Systems With COTS Software. In *Proceedings of the Eight IEEE Symposium on Software Metrics (METRICS'02*).

Davis L. and Gamble R, 2002. Identifying Evolvability for Integration. *Proceedings of the First International Conference on COTS-Based Software Systems*, ICCBSS 2002, Springer-Verlag Berlin Heidelberg New York, pp. 65-75

Deming E., 1986. *Out of the Crisis*. Center for Advanced Engineering Study, MIT, Cambridge, MA.

Egyed A., Medvidivic N., and Gacek C., 2000. Component-based perspective on software mismatch detection and resolution, *IEE Software Engineering, Vol. 14*(6), pp. 225-236.

Shereshevsky M. et al, 2001. Information Theoretic Metrics for Software Architectures. In *Proceedings of the 25th Annual International Computer Software and Applications Conference* (COMPSAC'01).

Sparling M, 2000. Lessons Learned Through Six Years of Component-Based Development. *Communications of the ACM, Vol. 43*(10) pp . 47-53.

Wallnau K, Stafford J., Hissam S., and Klein M., 2001. On the Relationship of Software Architecture to Software Component Technology, In *Proceedings of the 6th International Workshop on Component-Oriented Programming (WCOP6)*, in conjunction with *ECOOP'01,* Budapest, Hungary, 2001.

Yakimovich D., Bieman J., and Basili V., 1999. Software architecture classification for estimating the cost of COTS integration, In *Proceedings of ICSE 99*, pp. 296-302.