Software Component Models

Kung-Kiu Lau
School of Computer Science
The University of Manchester, Manchester M13 9PL, UK
kung-kiu@cs.man.ac.uk

ABSTRACT

Component-based Development (CBD) is an important emerging topic in Software Engineering, promising long sought after benefits like increased reuse and reduced time-to-market (and hence software production cost). However, there are at present many obstacles to overcome before CBD can succeed. For one thing, CBD success is predicated on a standardised market place for software components, which does not yet exist. In fact currently CBD even lacks a universally accepted terminology. Existing component models adopt different component definitions and composition operators. Therefore much research remains to be done. We believe that the starting point for this endeavour should be a thorough study of current component models, identifying their key characteristics and comparing their strengths and weaknesses. A desirable side-effect would be clarifying and unifying the CBD terminology. In this tutorial, we present a clear and concise exposition of all the current major software component models, including a taxonomy. The purpose is to distill and present knowledge of current software component models, as well as to present an analysis of their properties with respect to commonly accepted criteria for CBD. The taxonomy also provides a starting point for a unified terminology.

Categories and Subject Descriptors

D.2.8 [**Software Engineering**]: [component-based software engineering]

General Terms

Software Component Models

Keywords

Software Components, Composition

1. INTRODUCTION

The cornerstone of any CBD methodology [16] is its underlying *component model*, which defines what components are, how they can be constructed and represented, how they can be composed or assembled, how they can be deployed and how to reason about all these operations on components.

In this tutorial, we aim to present distilled knowledge of current software component models, as well as an analysis of their properties with respect to commonly accepted criteria for CBD.

2. AN ABSTRACT MODEL

We introduce an abstract component model to act as a reference framework for current component models, which defines and explains terms of reference that we will use. The abstract model can be used to explain the semantics and syntax of components and their composition, at an abstract level, independent of any specific model.

3. COMPONENT LIFE CYCLE

An idealised life cycle of software components consists of the following phases: *design*, *deployment* and *run-time*.

In the design phase, components have to be constructed, catalogued and stored in a repository in such a way that they can be retrieved later, as and when needed. Components in the repository can be both source and binary code. They can be retrieved and composed to a composite that is deposited into the repository in binary code.

In the deployment phase, components have to be retrieved from the repository, and compiled to binary code. These binary components can be composed to a system that is ready for execution.

In the run-time phase, there is no new composition, but components of a system are instantiated with data and then executed.

The idealised component life cycle is based on the commonly accepted desiderata of CBD:

- components are pre-existing reusable software units;
- components can be produced and used by independent parties;
- components can be copied and instantiated;
- components can be composed into composite components which in turn can be composed with (composite) components into even larger composites (or subsystems), and so on.

All current component models reflect these criteria, to greater or lesser degrees.

The idealised component life cycle provides a basis for a taxonomy of software component models.

ICSE'06, May 20–28, 2006, Shanghai, China.
ACM 1-59593-085-X/06/0005.

4. A TAXONOMY

Based on component composition in the idealised component life cycle, current software component models can be classified into 4 categories [7, 8]:

- In **Category 1**, in the design phase, new components can be deposited in a repository, but cannot be retrieved from it. Composition is not possible in the design phase, i.e. no composites can be formed, and so no composites can be deposited in the repository. In the deployment phase, components can be retrieved from the repository, and their instances formed and composed.

  There is only one member in this category, viz. JavaBeans [15].

- In **Category 2**, in the design phase, new components can be deposited in a repository, but cannot be retrieved from it. Composition is possible, i.e. composites can be formed, but composites cannot be deposited in (and hence retrieved) from the repository. In the deployment phase, no new composition is possible; the composition of the component instances is the same as that of the components in the design phase.

  Enterprise JavaBeans [11], CORBA Component Model [13], COM [3], .Net [10] and Web Services [1] belong to this category.

- In **Category 3**, in the design phase, new components can be deposited in a repository, and components can be retrieved from the repository. Composition is possible, and composites can be deposited in the repository. In the deployment phase, no new composition is possible; the composition of the component instances is the same as that of the components in the design phase.

  Koala [17], KobrA [2] and SOFA [14] belong to this category.

- In **Category 4**, in the design phase, there is no repository. Therefore components are all constructed from scratch. Composition is possible. In the deployment phase, no new composition is possible; the composition of the component instances is the same as that of the components in the design phase.

  Architecture Description Languages [9], UML2.0 [12], PECOS [5], Pin [6] and Fractal [4] belong to this category.

5. CONCLUSION

This survey of current software component models points out similarities and differences between the models. The similarities suggest a unifying terminology, or even a unified model. The differences point out possible improvements of the models.

In an ideal unified model, composition should be possible in both the design and the deployment phases, in order to maximise component reuse and design flexibility. This has not been achieved by current models, but we believe our survey provides a useful starting point.

6. REFERENCES

[1] G. Alonso, F. Casati, H. Kuno, and V. Machiraju. *Web Services: Concepts, Architectures and Applications*. Springer-Verlag, 2004.

[2] C. Atkinson, J. Bayer, C. Bunse, E. Kamsties, O. Laitenberger, R. Laqua, D. Muthig, B. Paech, J. Wüst, and J. Zettel. *Component-based Product Line Engineering with UML*. Addison-Wesley, 2001.

[3] D. Box. *Essential COM*. Addison-Wesley, 1998.

[4] E. Bruneton, T. Coupaye, and M. Leclercq. An open component model and its support in Java. In *Proceedings of 7th CBSE*, pages 7–22. Springer -Verlag, 2004.

[5] T. Genssler, A. Christoph, B. Schulz, M. Winter, C.M. Stich, C. Zeidler, P. Müller, A. Stelter, O. Nierstrasz, S. Ducasse, G. Arévalo, R. Wuyts, P. Liang, B. Schönhage, and R. van den Born. *PECOS in a Nutshell*. `http://www.pecos-project.org/`, September 2002.

[6] J. Ivers, N. Sinha, and K.C Wallnau. A Basis for Composition Language CL. Technical Report CMU/SEI-2002-TN-026, CMU SEI, 2002.

[7] K.-K. Lau and Z. Wang. A survey of software component models. Pre-print CSPP-30, School of Computer Science, The University of Manchester, April 2005. `http://www.cs.man.ac.uk/cspreprints/PrePrints/cspp30.pdf`.

[8] K.-K. Lau and Z. Wang. A taxonomy of software component models. In *Proc. 31st Euromicro Conference*, pages 88–95. IEEE Computer Society Press, 2005.

[9] N. Medvidovic and R. N. Taylor. A classification and comparison framework for software architecture description languages. *IEEE Transactions on Software Engineering*, 26(1):70–93, January 2000.

[10] Microsoft Corporation, `http://www.microsoft.com/net/`. *.NET documentation*, 2001.

[11] Sun Microsystems. Enterprise Java Beans Specification, Version 3.0, 2005.

[12] OMG. *UML 2.0 Superstructure Specification*. `http://www.omg.org/cgi-bin/doc?ptc/2003-08-02`.

[13] OMG. *CORBA Component Model, V3.0*, 2002. `http://www.omg.org/technology/documents/formal/components.htm`.

[14] F. Plasil, D. Balek, and R. Janecek. SOFA/DCUP: Architecture for component trading and dynamic updating. In *Proc. ICCDS98*, pages 43–52. IEEE Press, 1998.

[15] Sun Microsystems. *JavaBeans Specification*, 1997. `http://java.sun.com/products/javabeans/docs/spec.html`.

[16] C. Szyperski, D. Gruntz, and S. Murer. *Component Software: Beyond Object-Oriented Programming*. Addison-Wesley, second edition, 2002.

[17] R. van Ommering, F. van der Linden, J. Kramer, and J. Magee. The Koala component model for consumer electronics software. *IEEE Computer*, pages 78–85, March 2000.