Components vs. Objects*

Luigia Petre
Turku Centre for Computer Science (TUCS)
FIN-20520 Turku, Finland
lpetre@abo.fi

Component-based software engineering [5, 6] means constructing new systems from already existing, service-providing *components*. Object-based software engineering [2, 3] means constructing a new system in terms of interacting, distinct units of information and services called *objects*. Both components and objects have encapsulation properties and are accessed via well-defined interfaces. Both components and objects are considered to improve the reuse of software and to alleviate its evolution phase. Finally, both notions are thought of being natural abstractions of real-world entities. Moreover, a real-world entity can be modelled or implemented using either notion. Their similar appearance in abstract modelling gives rise to confusion in distinguishing between these two notions [4]. In order to solve this, the main questions to be answered to are:

- What is the essential difference between components and objects?
- (Why) do we need them both?

We examine the conceptual difference between components and objects, and aim to clarifying it. We consider the components as an engineering approach to software and the objects as a mathematical modelling approach to it. More precisely, the components of a software system provide a partition of the service space, while the objects of a software system provide a partition of the state space. Using the diagrams from the Unified Modeling Language (UML) [1] we outline a software development method that makes use of both components and objects. In particular, this method emphasizes the importance of designing a software system in terms of components and implementing it in terms of objects.

In the analysis phase, we consider a software system as a set of services that have to be implemented. By conveniently grouping these services we can determine a set of components that are to provide them. These components have to be constructed, or, if they already exist, simply reused. Excluding the already existing components, we end up with a set of

*This is the abstract of the paper presented at the 12-th Nordic Workshop on Programming Theory, Bergen, Norway, 11-13 October 2000, http://www.ii.uib.no/ nwpt00

components to construct, for whom the services are well-documented. The design of each component basically consists in determining the services to be used from other components and the services to be implemented by that component. In order to implement the services, a suitable object-oriented decomposition providing the required implementation has to be found. At this stage, the object-orientation is not mandatory, but the experience shows that such an approach is more intuitive and stable, withstanding the evolution of the software system better [3].

Summarizing, components are important as software construction concepts based on the notion of *service*. Objects are also important, but in providing a construction solution to each component and component service. Objects are best in underlying a problem domain, while components are most suitable in describing a system functionality. By using components and objects together, as suggested above, we employ these notions at their real potential.

References

[1] G. Booch, J. Rumbaugh and I. Jacobson. *The Unified Modeling Language User Guide.* Addison-Wesley, 1998.

[2] M. Bouzeghoub, G. Gardarin, P. Valduriez. *Object Technology: Concepts and Methods.* International Thomson Computer Press, 1997.

[3] C. Horstmann. *Practical Object-Oriented Development in C++ and Java.* John Wiley & Sons, Inc., 1997.

[4] B. Henderson-Sellers, C. Szyperski, A. Taivalsaari, A. Wills. Are Components Objects? In *OOPSLA'99 Companion*, Panel Discussion, 1999.

[5] M. C. McIlroy. Mass Produced Software Components. In *Proc. of NATO Conference on Software Engineering*, Garmisch, Germany, 1968.

[6] C. Szyperski. *Component Software-Beyond OO Programming.* Addison-Wesley, 1997.