

Position Summary. DiPS: A Unifying Approach for Developing System Software

Sam Michiels, Frank Matthijs, Dirk Walravens, Pierre Verbaeten
DistriNet, Department of Computer Science, K.U.Leuven
Celestijnenlaan 200A
B-3001 Leuven
Sam.Michiels@cs.kuleuven.ac.be

Abstract

In this position paper we unify three essential features for flexible system software: a component oriented approach, self-adaptation and separation of concerns. We propose DiPS (DistriNet Protocol Stack) [4], a component framework, which offers components, an anonymous interaction model and connectors to handle non-functional aspects such as concurrency.

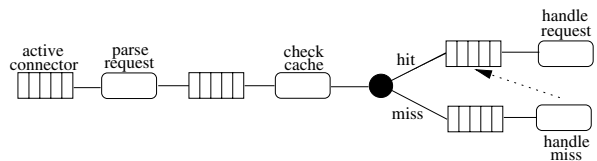
This position statement explains why component framework technology is needed for flexible system software and how we are using DiPS (DistriNet Protocol Stack) [4], a component framework, to build complex adaptable system software such as protocol stacks and device drivers. We state that there are three essential features for flexible system software (a component oriented approach, self-adaptation and separation of concerns) which are unified in DiPS. Several systems and paradigms exist which rarely offer each of these features.

Operating system research, such as [1] or [2], shows the advantage of providing reusable system components as basic building blocks. DiPS offers functional components such as a request parser or a component that looks for a page in the cache (see the figure).

Other system research focuses on the flexibility to adapt the system to application domain changes [5]. System software must allow run-time adaptations and, in certain cases, it must be intelligent enough to adapt itself based on application and system level triggers. E.g., database i/o access patterns are likely to be random, while video streaming i/o access is more sequential. File system performance is likely to increase when the prefetching strategy is adapted to the file access pattern and the physical disk block distribution. To allow components (such as prefetching) to be transparently added, removed or replaced (at run-time), DiPS components are anonymously connected to each other.

Strict separation of functional and non-functional code (such as concurrency code) has proven to be an essen-

tial feature to become adaptable, maintainable and reusable software [3]. For example, popular websites handle more than 100 million page views a day which results in many thousands of concurrent i/o requests. DiPS therefore allows fine-grained per-component concurrency control by its active connectors (AC) (see figure). An AC encapsulates a scheduler thread (or a thread pool) and a buffer to store pending requests. E.g., it can collect several requests to minimize the start-up cost of a component and therefore to increase the overall performance. Separating concurrency control from functional components allows to apply different concurrency models in a flexible way.



References

- [1] B. Ford, K. V. Maren, J. Lepreau, and e.a. The flux os toolkit: Reusable components for OS implementation. *In Proceedings of the Sixth IEEE Workshop on Hot Topics in Operating Systems*, May 1997.
- [2] E. Gabber, C. Small, J. Bruno, J. Brustoloni, and A. Silberschatz. The pebble component-based operating system. *In Proceedings of the USENIX 1999 Annual Technical Conference*, June 1999.
- [3] J. Itoh, Y. Yokote, and M. Tokoro. Scone: Using concurrent objects for low-level operating system programming. Technical report, Department of Computer Science, Keio University, 1995.
- [4] F. Matthijs. *Component Framework Technology for Protocol Stacks*. PhD thesis, Katholieke Universiteit Leuven, Dec 1999. Available at <http://www.cs.kuleuven.ac.be/samm/netwg/dips/index.html>.
- [5] M. Seltzer and C. Small. Self-monitoring and self-adapting operating systems. *In Proceedings of the Sixth IEEE Workshop on Hot Topics in Operating Systems*, May 1997.