The State of COMPOSITE*

Jiguo Song, Qi Wang, Gabriel Parmer
The George Washington University
{jiguos,interwq,gparmer}@gwu.edu

I. THE TAO OF COMPOSITE.

COMPOSITE is a component-based operating system that has been under development since 2006 with design goals including configurability, predictability, and reliability. Unlike many previous component-based operating systems that focus on kernel-based configurability, COMPOSITE implements most system policies, mechanisms and abstractions as user-level, hardware-protected, fine-grained units of functionality that are harnessed through well-defined interfaces. COMPOSITE's structure is most similar to μ-kernels: "A concept is tolerated inside the microkernel only if moving it outside the kernel, i.e., permitting competing implementations, would prevent the implementation of the system's required functionality"[1]. COMPOSITE philosophically expands on this in two ways:

1) *Component-based policy definition.* We strive to eliminate policies from the kernel, thus including only mechanisms. This enables for both customized resource management, and for designers to trade between complexity and TCB size, for flexibility and capability. Though the line between policy and mechanism is not clean [2], functionality common to most modern microkernels including scheduling and structured memory mapping is moved to user-level components where it can be redefined. Unlike exokernels [3] we avoid distributed management of resources, instead centralizing the policy into specific manager components. To enable flexibility of resource management (diversity of policy), *resource managment abstraction* is enabled via inter-component protocols to hierarchically control scheduling, manage memory, or perform I/O [4]. This support enables concurrent execution of multiple virtual environments that trade between heightened isolation with customized resource management, and resource sharing.

2) *System behavior via composition of fine-grained components.* One of the most successful component-based systems is the UNIX command line, based on the composition of simple programs into pipelines of complex functionality. COMPOSITE emphasizes the composition of complex systems from fine-grained components. The structure of this composition is a general DAG, and the functional protocols between components are encoded in explicit interfaces. Though a pervasive *separation of concerns, and extensive interface-level polymorphism*, developers have significant leeway in programming a system all the way down to resource management policies *at the composition-level*.

Mutable Protection Domains enables protection boundaries between components to be dynamically raised and lowered [5] to trade protection and performance. Collections of components can be collapsed into the same protection domain to mimic the structure of μ-kernels, monolithic systems, or exokernels. This fine-grained control over protection domains enables the generic study of system structure.

This material is based upon work supported by the National Science Foundation under Grants No. CNS 1137973, CNS 1149675, and CNS 1117243. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

COMPOSITE's focus. COMPOSITE provides unique opportunities due to its component-based structure. These include:

- *Configurability.* Supports concurrent execution of divergent virtual environments ranging from a separation-kernel environment emphasizing strong barriers between high- and low-criticality tasks, to a simple web-server that serves both static and dynamic content composed of 25 components. In the server, different communication protocols, altering the data source, or even changing the interrupt scheduling to more aggressively avoid livelock, can all be done by changing the composition of components. Though extensively decoupled, this web-server performs at least as well as Apache.
- *Predictability.* All aspects of the system are designed around the goal of bounded-latency. Notably, COMPOSITE places an emphasis on the end-to-end bounded latency of invocations across a possibly long chain of components. This solves by design the dependency problem that complicates scheduling in many component-coordination systems. This end-to-end predictability is currently being extended to multi-core systems.
- *Reliability.* By pervasively memory isolating components, fault propagation is significantly constrained in COMPOSITE. COMPOSITE enables even the lowest-level system components to fail, and will predictably reconstitute their state with overhead on the order of 10s of μ-seconds.

II. CURRENT STATE OF COMPOSITE

COMPOSITE is a research OS, and current goals do not include executing existing applications. Including external libraries, COMPOSITE is 160K lines of code (LOC) including a 7K LOC kernel, and 30K in components (minus third-party libraries). The system includes some POSIX support, some scripting language support (via LUA), and networking via LWIP.

Who should use COMPOSITE? COMPOSITE is in a state of constant development, and is not yet appropriate for production environments. Researchers investigating some combination of OS structure, resource management, parallelism, and real-time execution could benefit from the system. Developers interested in expanding the corpus of components are always welcome.

Online presence. The development mailing list, and more information can be found at http://composite.seas.gwu.edu/. The source is available at https://github.com/gparmer/Composite.

REFERENCES

[1] J. Liedtke, "On micro-kernel construction," in *Proceedings of the 15th ACM Symposium on Operating System Principles.* ACM, December 1995.
[2] R. Levin, E. Cohen, W. Corwin, F. Pollack, and W. Wulf, "Policy/mechanism separation in hydra," in *Proceedings of SOSP*, 1975.
[3] D. R. Engler, F. Kaashoek, and J. O'Toole, "Exokernel: An operating system architecture for application-level resource management," in *Proceedings of SOSP*, 1995.
[4] G. Parmer and R. West, "HiRes: A system for predictable hierarchical resource management," in *Proceedings of RTAS*, 2011.
[5] ——, "Mutable protection domains: Adapting system fault isolation for reliability and efficiency," in *ACM Transactions on Software Engineering (TSE)*, July/August 2012.