Issues in Holistic System Design

Julia L. Lawall
DIKU
University of Copenhagen
Universitetsparken 1
2100 Copenhagen Ø
julia@diku.dk

Christian W. Probst
IMM
Technical University of Denmark
Richard Petersens Plads
2800 Kongens Lyngby
probst@imm.dtu.dk

Ulrik Pagh Schultz
Maersk Institute
University of Southern Denmark
Campusvej 55
5230 Odense M
ups@mip.sdu.dk

Abstract

The coordination of layers in computer and software systems is one of the main challenges in designing such systems today. In this paper we consider *Holistic System Design* as a way of integrating requirements and facilities of different system layers. We also discuss some of the challenges that this kind of system design poses for computer science in general as well as programming languages and operating systems in particular.

Categories and Subject Descriptors D.2.11 [*Software Engineering*]: Software Architectures; C.4 [*Computer Systems Organization*]: Performance of Systems

General Terms Design, Languages, Reliability, Verification

Keywords Software Architecture, Domain-specific Languages, Adaptive Software

1. Introduction

Recent years have seen a rapid growth in the diversity and complexity of computer system components, ranging from new kinds of hardware to advanced applications. This rapid growth has brought with it the potential for significant improvements in the features provided to the end user, to satisfy an increasing demand for highly customizable, multifunctional systems. Nevertheless, these benefits have been slow to appear, for reasons ranging from programmability to reliability. Indeed, existing programming mechanisms are increasingly inadequate to coordinate the many different components contributing to the overall system behavior. Furthermore, the diversity and complexity of components make it difficult to reason about global system properties. Consequently, development costs and times are exploding, software is unable to make optimal use of the available resources, and adaption to specific user needs is difficult.

We contend that a major source of this problem is the stratification of computing systems into layers: the user layer, the operating system (OS) layer, the hardware layer, etc. While the user layer is highly customizable, as we move to the lower layers, they become increasingly rigid and unconfigurable. This stands in contrast to, *e.g.*, biological systems, where the different layers are able to adapt individually and together in response to changing needs. Analogously, techniques in computing systems are needed to allow interaction and feedback between layers, and gradual integration and adaptation of new sub-systems, as found in nature.

To address these problems, we find it promising to take a *holistic* approach to system design. Rather than enforcing barriers between the various layers of a computer system, the goal of Holistic System Design is to integrate requirements and functionalities of components scattered across layers, while taking into account the complexity of today's systems. Two key issues are to enrich the interfaces between components to allow tighter interaction between them, and to develop techniques to exploit the semantic information provided by these enriched interfaces to reason about the system behavior. Holistic System Design should lead to a new kind of system that is easier to develop, more robust and better adaptable to user demands.

While many different research areas can contribute to holistic system design, we see the main challenge in the areas of programming languages and operating systems. Domain-specific languages will be needed for expressing and specifying properties and requirements of different layers, and to allow reasoning about these properties. Operating systems, or the functionality pro-

vided by operating systems, will be needed to provide the infrastructure for holistic system and to support dynamic re-configuration of these systems.

2. The Research Challenges

We consider some of the significant research challenges that must be addressed to achieve a holistic approach to system design.

Understanding the behavior and requirements of components at various layers. To be able to tune a system to the behaviors and requirements of a given component, we need to be able to characterize these properties. For example, while we may normally think of a text editor as an application that manipulates and displays text and a video player as an application that manipulates and displays video, to get the most appropriate treatment for these applications from a process scheduler or power manager, it may be more useful to characterize the text editor as a user-driven application and a video player as a periodic one. Other characteristics may be useful when considering the interaction with other system components.

Expressing information about component behaviors and requirements. Behaviors and requirements can change over time, in complex ways. For example, although the aforementioned video player normally exhibits periodic behavior, when the user is selecting the next video the player's behavior is determined by user interaction, like that of a text editor. A component interface model is thus needed that enables safe, efficient, and expressive intercommunication of this kind of complex, changing information.

Exploiting this information in all layers. To be able to coordinate the behaviors and requirements of components in all layers, systems must be flexible and reconfigurable. For example, in the context of a battery-powered laptop or mobile phone, power consumption should be reduced to the minimum required to achieve system goals. This, however, requires that the hardware provides a wide range of power levels, that the OS provides efficient mechanisms for changing between them, and that there is an infrastructure for mediating their interaction. In general, these extra capabilities can increase the burden on system developers. Thus, research is needed into programming-language and software-engineering techniques to provide a good development model at all layers.

Reasoning about the resulting systems. Current approaches to reasoning about system behaviors and requirements are hampered by the stratification of systems into layers. For example, in safety-critical real-time systems, it is necessary to identify the worst-case execution time of each component, to ensure that the entire system can respond in a timely manner. Due to the inability of user-layer components to control components in lower layers, execution times can widely vary. Thus, such estimates are typically gross over-approximations, resulting in underutilization of computing resources. The challenge is to enable precise configuration of components at all layers, to make their behavior more predictable, making it easier to reason about the overall system behavior.

Balancing openness with encapsulation. Expressing and exploiting information across layers means opening up the implementation of each layer, but this should be done in a structured and disciplined way, to preserve internal invariants and avoid spurious implementation dependencies. Again, we find the same principles at work in biological systems. For example, the inner workings of a cell is protected by the cellular membrane but can nevertheless be affected both by neighboring cells and by hormones global to the living organism. This way, biological systems feature a rigid division into layers as we know it from today's systems, while at the same time providing means for interaction and adaption across layer boundaries.

Adapting legacy software and software engineering methodologies. A large body of software is available that is designed in terms of the standard layers. To benefit from this existing work, techniques are needed to evolve legacy software for use within a holistic system. Furthermore, well-tested software engineering methodologies have been developed to enable limited communication between layers. Where appropriate, these methodologies should be evolved to encompass the principles of holistic system design.

Validating holistic systems. Building complete systems is essential for experimentally assessing the system-wide benefits of Holistic System Design, such as improved responsiveness and reduced power consumption. Evaluation techniques will be needed to determine the individual contributions of system components in all layers and to understand how these contributions synergize into the whole system behavior.

3. Current Research Efforts

In this section, we describe some of the authors' current research efforts that cut across the traditional layers of a computer system, either by exploiting existing properties of lower layers from higher layers, or by redesigning lower layers to make more interaction with them possible. In each case, we consider what new

features or further research are needed to improve the holistic nature of the approach.

Interaction between the user layer and the hardware layer: Controlling the cache behavior of network servers. As memory sizes have grown in recent years, network servers have increasingly been able to serve requests entirely out of memory, without relying on slow access to disk storage. In this setting the data and instruction cache behavior becomes the limiting factor in the ability of a network server to handle large numbers of concurrent request [3]. Furthermore, when the network server is implemented in an event-based style, as is the case of many efficient network servers today, its memory requirements can be highly predictable. Based on these observations, Bhatia, Consel, and the first author have developed a memory allocator that allocates memory from a pool limited to a cache-sized memory region, implying that it should be possible to freely use the memory within the pool without cache conflicts [4]. The allocator furthermore constrains the concurrency of the network server, using its knowledge of the server's memory requirements, so that all active requests can be handled within the memory available in the pool. In our benchmarks of several network servers running on a Pentium IIIM, including the highly efficient server TUX, when bombarded with requests, we have found that our memory allocator drastically decreases the number of cache misses, and significantly increases the number of requests that can be treated concurrently [5].

Our approach connects the network server running in the user layer with the behavior of the cache in the hardware layer. Some other work has also found the value of tuning user-layer code to caching properties, including that of Chilimbi *et al.* on data structure layout [7, 8]. Although our benchmarks show that we have succeeded in drastically reducing the number of cache misses on the Pentium IIIM architecture, all of the work in this area could benefit from predictable mechanisms for controlling the cache alignment of critical memory locations.

Interaction between the user layer and the OS layer: Application-specific process scheduling. Process scheduling is the mechanism by which an OS decides which process has access to the CPU and at what time. Traditionally, general-purpose OSes such as Linux and Windows run each process for a given time slice in a best-effort, round-robin fashion, ensuring that each process eventually gets access to the CPU but providing no other guarantees. Real-time processes, however, have more specialized timing requirements. For example, a video player, which is often used in a general-purpose computing environment, must execute periodically to maintain its frame rate. Execution at a fixed rate cannot be guaranteed by a round-robin scheduler in the presence of a potentially unbounded number of processes.

For standard OSes, such as Linux and Windows, user-layer applications can only control the scheduling behavior via a fixed set of process priorities. To allow application programmers to specify more complex scheduling policies, Muller and the first author have developed the Bossa scheduling framework [12]. This framework provides an OS-independent domain-specific language for implementing scheduling policies, and an OS-specific run-time system that connects a Bossa scheduling policy to the target OS. The latter must be implemented just once for each OS. Verifications are provided that ensure statically that a Bossa scheduling policy respects the basic semantics of each scheduling-related operation. A Bossa scheduling policy can furthermore specify an interface that allows an application to interact with the scheduler in a policy-specific way. We have used Bossa to implement a number of scheduling policies, mostly targeted to multimedia applications. Bossa has also been used in teaching scheduling at several institutions. Our experiments show that Bossa is easy to use and incurs no noticeable overhead in the execution of real applications.

Bossa is an example of how lower layer services can be redesigned to be more configurable from upper layers, while retaining safety. Our approach relies critically on the use of a domain-specific language to encapsulate expertise and provide safety. Other projects that have used programming languages to both allow and constrain configuration of lower layers include the SPIN OS [2] and the Berkeley Packet Filter [11]. More work, however, is needed in language design strategies, both to lower the expertise barrier for programming such complex systems and to be able to specify and check relevant properties.

Interaction between low-level components in the user layer: Middleware for self-reconfigurable robots. A self-reconfigurable robot is a robot that can change its own shape. Self-reconfigurable robots are built from multiple identical modules that can manipulate each other to change the shape of the robot [6, 10, 13, 14, 16, 19]. Changing the physical shape of a robot allows it to adapt to its environment, for example by changing from a car configuration, best suited for flat terrain, to a snake configuration, best suited for uneven terrain. Programming self-reconfigurable robots is complicated by the need to distribute control across the modules that constitute the robot and furthermore to coordinate the actions of these modules. Algorithms for controlling the overall shape and locomotion of the robot have been investigated (e.g. [9, 17]), but the is-

sue of providing a high-level programming platform for developing controllers remains largely unexplored.

The primary challenges in programming modular, self-reconfigurable robots are coordinating transformations (and thus movements in general) during shape change and propagating information between individual modules. Given a control algorithm, coordinating transformations entails coordinating the movements of the individual modules to produce the required overall effect on the robot. Propagating information typically entails propagating sensor information to the parts of the robot that need to react to external events and propagating commands to the parts of the robot that need to perform actions. Shape change however complicates information propagation, since it not only causes the topology of the robot to change, but also can change the functionality (and hence the information requirements) of each module.

The complex communication requirements prompt the need for a middleware providing high-level communication primitives. This middleware layer must however be highly adaptable to support different sets of requirements for different shapes. The third author is currently investigating such a middleware for the ATRON reconfigurable robot, in collaboration with members of the Adaptronics group [1, 10, 15].

Moving lower layers up to higher layers, to facilitate inter-layer communication: Power management for portable devices. Recent years have seen a tremendous increase in the capabilities of mobile devices. Nevertheless, while hardware performance has increased, power supply technology has not kept pace, becoming a significant bottleneck. This has led to a large collection of techniques to reduce power consumption, the most popular being Dynamic Voltage Scaling (DVS). This technique, which can be controlled from the hardware, operating system or compiler levels, works by reducing a processor's clock frequency and voltage in lockstep. Because CPU power dissipation is quadratic with respect to supply voltage and linear with respect to clock frequency, DVS should, under ideal conditions, reduce a processor's instantaneous power dissipation cubically. However, when the processor is slowed down, program execution slows down as well. Thus DVS is only effective in code that can be slowed down without unacceptably increasing a program's total execution time (or a user's experience of it).

Multi-tasking induces further challenges for DVS. Changing the frequency of the processor affects the behavior of other systems, such as memory. Thus, the ideal frequency for one process might not be appropriate for other processes running concurrently. For this reason, in systems that offer multi-tasking, DVS is currently often done at the hardware or OS level, where a global view of the system behavior is available. Nevertheless, at this low level there is no mechanism available for applications to communicate their requirements to the DVS management system.

Venkatachalam, Franz, and the second author have therefore suggested a holistic system for mobile-code execution, that integrates hardware, operating system, and applications into a feedback loop, using a virtual machine [18]. In this framework, applications can easily communicate their resource needs to other system parts. Furthermore, the virtual machine layer can use its high-level information to re-compile (parts of) the application to, e.g., satisfy the needs of applications running in parallel, thereby connecting different applications directly to the DVS management system.

4. Conclusion

This paper describes holistic system design as a way of integrating requirements and facilities of different system layers. The goal of holistic system design is to fundamentally improve the way we design systems. We expect that it will be possible to more quickly develop complex systems that will make better use of available resources. The overall system behavior will furthermore become more predictable because of the ability to reason about the interaction between components. Finally, these features will facilitate research into new resource management strategies.

References

[1] Adaptronics group, Maersk Institute. `http://www.adaptronics.dk`.

[2] B. N. Bershad, S. Savage, P. Pardyak, E. G. Sirer, M. E. Fiuczynski, D. Becker, C. Chambers, and S. J. Eggers. Extensibility, safety and performance in the SPIN operating system. In *Proceedings of the Fifteenth ACM Symposium on Operating System Principles*, pages 267–284, Copper Mountain Resort, CO, Dec. 1995.

[3] S. Bhatia. *Optimisations de compilateur pour les systèmes réseaux*. PhD thesis, University of Bordeaux, June 2006.

[4] S. Bhatia, C. Consel, and J. Lawall. Memory-manager/scheduler co-design: optimizing event-driven servers to improve cache behavior. In *Proceedings of the 2006 international symposium on Memory management*, pages 104–114, Ottawa, Canada, June 2006.

[5] S. Bhatia, C. Consel, and J. Lawall. Minimizing cache misses in an event-driven network server: A case study of TUX. In *The 31st IEEE Conference on Local Computer Networks (LCN)*, Tampa, FL, USA, Nov. 2006. To appear.

[6] A. Castano and P. Will. Autonomous and self-sufficient conro modules for reconfigurable robots. In *Proceedings of the 5th International Symposium on Distributed*

Autonomous Robotic Systems (DARS), pages 155–164, Knoxville, Texas, USA, 2000.

[7] T. M. Chilimbi, B. Davidson, and J. R. Larus. Cache-conscious structure definition. In *Proceedings of the ACM SIGPLAN'99 Conference on Programming Language Design and Implementation (PLDI'99)*, pages 13–24, May 1999.

[8] T. M. Chilimbi, M. D. Hill, and J. R. Larus. Cache-conscious structure layout. In *Proceedings of the ACM SIGPLAN'99 Conference on Programming Language Design and Implementation (PLDI'99)*, pages 1–12, May 1999.

[9] D. Christensen and K. Støy. Selecting a meta-module to shape-change the ATRON self-reconfigurable robot. In *Proceedings of IEEE International Conference on Robotics and Automations (ICRA)*, pages 2532–2538, Orlando, USA, May 2006.

[10] M. W. Jorgensen, E. H. Ostergaard, and H. H. Lund. Modular ATRON: Modules for a self-reconfigurable robot. In *Proceedings of IEEE/RSJ International Conference on Robots and Systems (IROS)*, pages 2068–2073, Sendai, Japan, Sept. 2004.

[11] S. McCanne and V. Jacobson. The BSD packet filter: A new architecture for user-level packet capture. In USENIX Association, editor, *Proceedings of the Winter 1993 USENIX Conference*, pages 259–269, San Diego, CA, Jan. 1993. USENIX.

[12] G. Muller, J. L. Lawall, and H. Duchesne. A framework for simplifying the development of kernel schedulers: Design and performance evaluation. In *HASE 2005 - High Assurance Systems Engineering Conference*, Heidelberg, Germany, Oct. 2005.

[13] S. Murata, E. Yoshida, K. Tomita, H. Kurokawa, A. Kamimura, and S. Kokaji. Hardware design of modular robotic system. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2210–2217, Takamatsu, Japan, 2000.

[14] D. Rus and M. Vona. Crystalline robots: Self-reconfiguration with compressible unit modules. *Journal of Autonomous Robots*, 10(1):107–124, 2001.

[15] U. Schultz, K. Støy, N. Dvinge, and D. Christensen. Sensor networks and self-reconfigurable robots. Technical Report 1, Maersk Institute, Aug. 2006. To be presented at the OOPSLA 2006 Workshop on Building Software for Sensor Networks (BSSN'06).

[16] W.-M. Shen, M. Krivokon, H. Chiu, J. Everist, M. Rubenstein, and J. Venkatesh. Multimode locomotion via superbot robots. In *Proceedings of the 2006 IEEE International Conference on Robotics and Automation*, pages 2552–2557, Orlando, FL, 2006.

[17] K. Støy. How to construct dense objects with self-reconfigurable robots. In *Proceedings of European Robotics Symposium (EUROS)*, pages 27–37, Palermo, Italy, May 2006.

[18] V. Venkatachalam, C. W. Probst, and M. Franz. A Multilevel Introspective Dynamic Optimization System For Holistic Power Aware Computing. Technical Report 04-08, University of California, Irvine, School of Information and Computer Science, April 2004.

[19] M. Yim, D. Duff, and K. Roufas. Polybot: A modular reconfigurable robot. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 514–520, San Francisco, CA, USA, 2000.