Modularity as a systems design principle

DISS. ETH NO. 19610

Modularity as a Systems Design Principle

A dissertation submitted to
ETH ZURICH

for the degree of
Doctor of Sciences

presented by
JAN S. RELLERMEYER
MSc in Computer Science, ETH Zurich
born February 9, 1978
citizen of Germany

accepted on the recommendation of

Prof. Dr. Gustavo Alonso, examiner
Prof. Dr. Timothy Roscoe, co-examiner
Prof. Dr. Harald Gall, co-examiner

2011

Abstract

Computing systems are increasingly becoming dynamic. One example is cloud computing and its property of elasticity. On cloud platforms, resources in the form of additional nodes can be added and removed at any time. Software systems expected to run in such environments, on the other hand, are not nearly as elastic and flexible. Another example is the clearly visible trend towards incorporating an increasing number of processor cores into modern computer systems. In the future, it is likely that not all of these cores will have a uniform instruction set anymore but specialized units are used to accelerate certain tasks. At the same time, however, the power envelope of computer systems is increasingly becoming an issue so that probably not all cores can run at the same time anymore. Software written for such systems is thereby required to adapt to a changing pool of resources, a situation that today's software is hardly prepared for.

This thesis contributes towards understanding how to build software systems that are able to reflect such a degree of flexibility. The fundamental observation underlying this work is that in order to respond to the emerging dynamism in the platforms, software has to become equally flexible in its design. This requires a segregation of the software into smaller units. In the early days of computer science similar challenges in the development process of complex software have catalyzed the concept of software modularity. The premise of this thesis is that the same kind of modularization—when not only applied on a logical level to the source code but also in a physical form and preserved until runtime— results in the required degrees of freedom in the design of software systems. In combination with a smart runtime system, this approach turns software into flexible, fluid entities able to adapt to a dynamic environment.

Three systems are presented in this thesis which are based upon the OSGi standard for dynamic modules in the Java language. They enhance the standard with different degrees of flexibility. The Juggle system co-manages software modules and the binaries for reprogramming an FPGA device so that applications can be selectively accelerated on demand and without interrupting running operations. The CPU/FPGA board serves as an example of a computer system that is already dynamic and reprogrammable today.

It requires software systems to adapt to and actively manage the resources. R-OSGi turns modularity into a programming model for distributed systems by transparently turning OSGi services into remote services. At the same time, failures arising from network unreliabilities are mapped to module and service unload events which applications written for OSGi are already prepared for to handle gracefully. This permits developers to focus on the functional decomposition of the application into modules and defer the decision where to put the boundaries of distribution to deploy-time or even runtime. The Cirrostratus module runtime system for the cloud drives the idea of location-transparency for software modules even further by providing a single system image on top of a variable set of machines in the cloud. Modules installed on one machine are uniformly visible and accessible on all machines. The system is able to provide the communication facilities to invoke services crossing machine boundaries and to create replicas of services on demand while keeping the internal state of modules consistent across the entire system. Without requiring adaptations to the code, modular software running on Cirrostratus has become completely fluid within the constraints of the modular decomposition.

The three contributions indicate how software modularity—not only applied to the source code as traditionally but understood as a runtime concern—is the system design principle required to cope with the challenges of increasingly dynamic environments. Most importantly, it does so in a generic way and beyond ad-hoc solutions.

Zusammenfassung

Computersysteme werden zunehmend dynamischer. Ein Beispiel dafür ist Cloud-Computing und die Eigenschaft der Elastizität. Auf Cloud-Plattformen können Ressourcen in Form von zusätzlichen Rechnerknoten zu jeder Zeit hinzugefügt oder entfernt werden. Softwaresysteme für solche Umgebung sind hingegen nicht annähernd so dynamisch und flexibel. Ein weiteres Beispiel für Dynamik ist der eindeutig erkennbare Trend, eine immer grössere Anzahl an Prozessorkernen in moderne Computer einzubauen. In der Zukunft ist es wahrscheinlich, dass nicht mehr alle Prozessoren den selben Instruktionssatz verarbeiten, sondern spezialisierte Einheiten bestimmte Aufgaben beschleunigen. Gleichzeitig rückt die Gesamtleistungsaufnahme von Computersystemem zunehmend in den Mittelpunkt, so dass sehr wahrscheinlich nicht mehr alle Kerne zur gleichen Zeit laufen können. Der für solche Systeme entwickelten Software wird dadurch Anpassungfähigkeit an den veränderlichen Ressourcenpool abverlangt, eine Situation für die heutige Software unzureichend vorbereitet ist.

Diese Dissertation leistet einen Beitrag zum Verständnis, wie Softwaresysteme zu entwickeln sind, die in der Lage sind, einen solchen Grad an Flexibilität zu reflektieren. Die fundamentale Beobachtung, die dieser Arbeit zu Grunde liegt, ist, dass Software als Antwort auf die zunehmende Dynamik der Platformen selbst ein gleiches Mass an Flexibilität aufweisen muss. Dies erfordert eine Aufteilung der Software in kleinere Einheiten. In den frühen Tagen der Informatik katalysierte eine vergleichbare Herausforderung, der Entwicklungprozess von komplexen Softwaresystemen, das Konzept der Softwaremodularisierung. Die Prämisse dieser Dissertation ist, dass die gleiche Art von Modularisierung, wenn sie nicht nur auf der logischen Ebene des Quellcodes, sondern in einer physikalischen Form angewandt und bis zur Laufzeit beibehalten wird, in den nötigten Freiheitsgraden im Design von Softwaresystemen resultiert. In Kombination mit einem intelligenten Laufzeitsystem verwandelt dieser Ansatz Software in flexible, fluide Einheiten mit der Fähigkeit, sich an die dynamische Umgebung anzupassen.

Drei Systeme werden in dieser Dissertation präsentiert, die auf dem OSGi Standard für dynamische Module in der Java-Programmiersprache basieren. Sie erweit-

ern den Standard um verschiedene Freiheitsgrade. Das Juggle-System verwaltet Softwaremodule und die Binärdaten zum Reprogrammieren von FPGA Schaltkreisen zusammen, so dass Anwendungen selektiv und nach Bedarf beschleunigt werden können, ohne laufende Operationen zu beeinträchtigen. Das CPU/FPGA-Board dient als ein Beispiel für ein Computersystem, welches bereits heute dynamisch und reprogrammierbar ist und von der Software Anpassungsfähigkeit und eine aktive Verwaltung von Ressourcen verlangt. R-OSGi verwandelt Modularität in ein Programmiermodell für verteilte Systeme, indem es transparent OSGi-Dienste zu verteilten Diensten macht. Gleichzeitig werden die durch den Einsatz eines fehlerbehafteten Netzwerks entstehenden Kommunikationsfehler auf Verfügbarkeitsereignisse der Module und Dienste abgebildet, welche für OSGi entwickelte Anwendungen bereits zu verarbeiten in der Lage sind. Dies erlaubt es dem Entwickler, sich auf die funktionale Zerlegung der Anwendung in Module zu konzentrieren und die Entscheidung, an welchen Stellen die Grenzen für Verteilung zu setzen sind, zur Verteilzeit oder sogar erst zur Laufzeit zu treffen. Die Cirrostratus Cloud-Laufzeitumgebung führt die Idee der Ortsunabhängigkeit von Softwaremodulen weiter, indem sie der Anwendung das Bild eines einzelnen Systems bietet, aufbauend auf einer variablen Menge von Maschinen in der Cloud. Module, die auf einer Maschine installiert sind, werden uniform auf allen Maschinen sichtbar und benutzbar. Das System ist in der Lage, die Kommunikationsmittel zum Dienstaufruf über Maschinengrenzen hinweg bereitzustellen und replizierte Kopien von Diensten nach Bedarf zu erstellen, unter Beibehaltung der Konsistenz des internen Zustands der Module im gesamten System. Ohne Anpassungen am Programmcode der Module zu erfordern, wird modulare Software durch Cirrostratus vollständig fluide, innerhalb der durch die modulare Zerlegung vorgegebenen Nebenbedingungen.

Die drei Ergebnisse zeigen auf, wie Modularität von Software, wenn sie nicht nur, wie im traditionellen Gebrauch, auf den Programmcode angewender wird, sondern als Laufzeiteigenschaft verstanden wird, das Systemdesignprinzip darstellt, um sich den Herausforderungen von immer dynamischer werdenden Umgebungen zu stellen. Vor allen Dingen leistet sie dies in einer generalisierbaren Art und Weise, die über Ad-Hoc-Lösungen hinausgeht.