Diss. ETH Nr. 12231

Integration aktiver Objekte in Oberon am Beispiel eines Serversystems

A B H A N D L U N G
zur Erlangung des Titels
DOKTOR DER TECHNISCHEN WISSENSCHAFTEN
der
EIDGENÖSSISCHEN TECHNISCHEN HOCHSCHULE ZÜRICH

vorgelegt von
Andreas Reto Disteli
Dipl. Informatik-Ing. ETHZ
geboren am 17. Oktober 1966
Bürger von Olten, Kanton Solothurn

Angenommen auf Antrag von
Prof. Dr. J. Gutknecht, Referent
PD Dr. M. Reiser, Korreferent

1997

Kurzfassung

In der heutigen Zeit ist der Begriff der Objektorientierung nicht mehr aus der Informatik wegzudenken. Obwohl die Auslegung dieses Begriff zum Teil sehr frei gehandhabt wird, hat die Grundidee, die dahintersteht, zweifellos grosse Vorteile. Objekte beeinhalten ihre eigenen Methoden und Daten, die sie gegen aussen zur Verfügung stellen können und die auch als vererbbare Basis für Erweiterungen dieser Objekte dienen können. Dies fördert die Lokalität der Daten und das Kapseln der wichtigen Information in kleine Einheiten. Normalerweise stehen aber solche Objekte in einem passiven Verhältnis zur aufrufenden Umgebung. In Oberon [Rei91] [WG92] steuert das Hauptprogramm die Aufrufe der Objekte und kontrolliert daher den gesamten Programmablauf. Die naheliegendste und sinnvollste Weiterentwicklung ist nun, die Objekte aufzuwerten und ihnen ihr eigenes Programm mitzugeben. Ein Objekt soll selbstständig handeln können, ohne dauernd auf einen Aufruf entweder vom Hauptprogramm oder von einem anderen Objekt warten zu müssen. Auch die Information über die Dauer einer Wartephase oder über eine Bedingung, die für den Programmablauf etabliert sein muss, soll ein Objekt selbst verwalten. Anders ausgedrückt soll es nebst seinen Daten auch noch seinen eigenen Ablaufplan (sein programmiertes Verhalten) beeinhalten. Derartige Objekte, die fähig sind, sich selbst zu verwalten, nennen wir *aktive Objekte*, im Gegensatz zu den herkömmlichen passiven Objekten.
Jeder dieser Ablaufpläne ist fest mit der Instanz eines aktiven Objektes verbunden. Da jedes Objekt nun selbständig ist, muss auch sein Plan eine eigene Einheit sein. Diese Eigenständigkeit lässt sich am Einfachsten als Prozess realisieren. Davon ausgehend, dass in einem realen System mehrere aktive Objekte gleichzeitig vorhanden sind, führen diese Überlegungen zu einem Mehrprozessystem. Der parallele Ablauf der Prozesse kann dazu führen, dass unter Umständen alle gleichzeitig zur Ausführung gelangen. Diese Parallelität muss auf einem Einprozessorsystem durch ein möglichst einfaches und effizientes Verfahren simuliert werden. Dem darunterliegenden Betriebssystem obliegt nun die Aufgabe, den Ablauf dieser Prozesse zu koordinieren.
In dieser Arbeit wollen wir zeigen, dass es mit geringem Softwareaufwand möglich ist, ein derartiges Mehrprozessystem zu implementieren, das mit einigen wenigen Ergänzungen in der dazugehörigen Programmiersprache in der Lage ist, aktive Objekte mit einfachen, aber mächtigen Kontrollstrukturen anzubieten.
Als Fallstudie dazu dient die Entwicklung eines Serversystems, welches auf diesen aktiven Objekten und auf dem dazugehörigen Betriebssystem basiert. Ein Serversystem deshalb, weil jeder darauf angebotene Dienst dazu prädestiniert ist, als aktives Objekt zu agieren (ein Dienst kann auch aus mehreren aktiven Objekten bestehen) und weil gleichzeitig mehrere, teilweise auch gleiche, Dienste ablaufen, die von einer unbestimmten Anzahl Klienten aufgerufen werden. Dies

bedeutet, dass pro Klient und Dienst gleichzeitig eine Vielzahl aktiver Objekte involviert sein können. Durch den modularen Aufbau des Servers ist es problemlos möglich, neue Dienste einzufügen, ohne dass der Rest des Systems tangiert wird. Der Server, das Netzwerk, sowie auch die Vielzahl von verschiedenen Diensten, die zur Verfügung stehen, beruhen allesamt auf aktiven Objekten.
Obwohl das Konzept der aktiven Objekte bewusst einfach gehalten ist, erfordert das Arbeiten mit ihnen vom Programmierer eine erhöhte Programmierdisziplin und ein abstraktes Denkvermögen, da nun mehrere Prozessebenen gleichzeitig Auswirkungen auf den globalen Systemzustand haben können. Dies kann als durchaus nützliche Eigenschaft eines Entwicklungssystems angesehen werden.

Abstract

The term of object orientation cannot be imagined as absent from today's computer science. Although the interpretation of this term is handled quite frankly to some extent, the basic idea has without doubt its great advantages. Objects containing their own methods and data can offer the public part of them in the programming interface and they can be used as an inheritable basis for extensions of these objects. This promotes the locality of data and the encapsulation of important information into small units. Normally such objects are in a passive relation to the calling environment. In Oberon, the enclosing main program controls the calls to the objects and therefore often decays to the function of a bare scheduler. The most obvious and sensible further development is now to increase the value of the objects and give them a sort of self-activity. An object should be capable to act independently, without having to wait for a call either from the main program or from another object. As well, the information when and how long an object has to take action should be managed by the object itself. In other words, besides the data an object should also contain its own control flow or programmed behaviour. Such objects, able to manage themselves, we call active objects in contrast to the conventional passive objects.

Each programmed behaviour is firmly connected to an instance of an active object. As each object acts independently its behaviour should also be an unit of its own. The easiest way to realize this self-reliance is the creation of a process. Starting from the principle that in a real system there are several active objects running simultaneously, these reflections point to a multitasking system. The parallel order of events may lead to the fact that all of them are scheduled to run at the same time. This parallelism must be simulated on a single processor machine using a technique as simple and efficient as possible. The coordination of these processes should apply to the underlying operating system.

In this thesis we want to show the possibility of implementing a multitasking system capable of offering active objects in an easy-to-program way with just a few additions to the original programming language and with only a small software expenditure.

Developing a server system using active objects and relying on the corresponding operating system is a non-trivial case study for testing our new concepts. We decided to implement a server system for the reason that every service is simply predestinated to act as an own active object (A service may also consist of several active objects). This means that for each client and service there is an undefined quantity of active objects involved. Furthermore, several partly equal services can run simultaneously, called by a unknown amount of clients. Due to the modular structure of the server it is unproblematic to insert more new services without touching the rest of the system. The server, the network as well as the amount of

different services being at the client's disposal are based on the principle of our active objects.
Although the concept of the active objects is intentionally kept simple and plain, working with them requires an increased discipline of programming and a abstract reasoning power, because of the multiple process levels that can take effect on global system states. This can absolutely be seen as a useful characteristic of a development system. Nevertheless the linear programming of the control flow of the active objects allows a big flexibility in the design of applications.