15. Zusammenfassung

In dieser Arbeit wurde der modular erweiterbare Rahmen ECL vorgestellt, der die Erstellung einer Koordinationssprache mit einem zugehörigen Laufzeitsystem ermöglicht. Die Kombination von Koordinationssprache und Laufzeitsystem bildet ein Koordinationssystem. Dieser Rahmen eignet sich als Grundlage, um Koordinationssysteme für die Erstellung von Applikationen für verteilte Systeme zu realisieren. Er enthält keine Beschränkungen bzgl. der unterstützbaren Zielsysteme und Koordinationsparadigmen und eignet sich damit als Basissystem für beliebige Applikationsklassen.

Die Freiheit von Beschränkungen ergibt sich daraus, dass ECL kein allgemeines semantisches Modell besitzt, auf das alle Erweiterungen abgebildet werden müssen. Die einzige Einschränkung ist syntaktisch und besteht aus dem Einsatz einer Architekturbeschreibungssprache, so dass eine verteilte Applikation stets als statische Software-Architektur darstellbar ist.

Der Rahmen wird durch die Kombination mit Erweiterungsmodulen zu einem funktionsfähigen Koordinationssystem, mit dem ein Applikationsentwickler verteilte Applikationen unter Einsatz eines grafischen Editors implementieren kann. Ein Erweiterungsmodul kann sowohl bestimmte Zielsysteme als auch bestimmte Koordinationsparadigmen unterstützen. Da es keinen semantischen Beschränkungen unterworfen ist, kann ein Entwickler ein Erweiterungsmodul so entwerfen, dass alle Eigenschaften des speziellen Zielsystems dem Applikationsentwickler komfortabel zur Verfügung stehen und dass das spezielle Koordinationsparadigma möglichst adäquat benutzt werden kann.

Hierfür enthält ein Erweiterungsmodul sowohl syntaktische als auch semantische Erweiterungen, die in den Rahmen integriert werden. Die syntaktischen Erweiterungen bestehen aus neuen Typen für Architekturelemente und aus Regeln in Form logischer Ausdrücke, wie diese Typen miteinander verknüpft werden können. Die semantischen Erweiterungen liegen in Form von Java-Klassen vor. Diese werden benutzt, um eine Applikationsarchitektur zu einem Geflecht von Java-Objekten zu übersetzen, welche dann zur Laufzeit die Koordination der einzelnen Applikationskomponenten durchführen.

Erweiterungsmodule können voneinander abhängen, d.h. ein Erweiterungsmodul *A* kann Architekturelementtypen und Java-Klassen eines Erweiterungsmoduls *B* selbst benutzen, um z.B. abgeleitete Elemente dem Applikationsentwickler anzubieten. Diese Wiederverwendung verringert den Entwicklungsaufwand für neue Module. Sie ermöglicht auch, dass ein Koordinationsparadigma in unterschiedlichen Zielsystemen eingesetzt werden kann, ohne dieses jeweils komplett neu zu implementieren.

Da dieser Ansatz auf keinem allgemeinen semantischen Modell basiert, ist die Kompatibilität zwischen zwei Modulen in dem Sinne, dass die Elemente

miteinander interagieren können, i.A. nicht gewährleistet. Aber dadurch, dass zwei Erweiterungsmodule A_1 und A_2 die gleichen Elemente des Moduls B einsetzen, kann man bei einem sorgfältigen Entwurf der drei Module gewährleisten, dass Elemente aus A_1 mit Elementen aus A_2 interagieren können und somit miteinander kompatibel sind. So sind in dem System Applikationen realisierbar, die gleichzeitig unterschiedliche Zielsysteme und unterschiedliche Koordinationsparadigmen einsetzen.

Um zu überprüfen, ob die Wiederverwendbarkeit bei der Erstellung neuer Module und deren Interkompatibilität in der Praxis wirklich gegeben ist, wurden mehrere voneinander abhängige Erweiterungsmodule implementiert. Dabei wurden sowohl unterschiedliche Koordinationsparadigmen, wie Task-Graphen, Datenflussarchitekturen und deklarative Spezifikation von Abhängigkeiten, untersucht als auch zwei unterschiedliche Zielsysteme, die Middleware CORBA und eine prototypische dienst-orientierte Infrastruktur für verteiltes Rechnen. Als Ergebnis zeigte sich, dass die beiden oben gesetzten Ziele bei einem sorgfältigen Entwurf zu erreichen sind. Die während der Untersuchungen entstandenen Erfahrungswerte wurden als Entwurfsrichtlinien formuliert.

Die allgemeine Praxistauglichkeit des System für den Anwendungsentwickler wurde anhand mehrerer Fallstudien evaluiert. Diese stammen aus zwei Applikationsklassen:

- verteilte Informations- und Kontrollsysteme,
- rechenintensive numerische Applikationen.

Es konnte demonstriert werden, dass sich aufgrund der komfortablen grafischen Entwicklungsumgebung der Entwicklungsaufwand für den Applikationsentwickler deutlich reduziert, da er sich nur wenig mit technischen Implementierungsdetails beschäftigen muss. Der durch den Einsatz des Systems zusätzlich auftretende Rechen- und Kommunikationsaufwand war bei allen Fallstudien vernachlässigbar.

Insbesondere bei den verteilten Informations- und Kontrollsystemen zeigte sich, dass sich Applikationen mit einer einfacher Struktur auch einfach und schnell entwickeln lassen. Komplexere Applikationen, die spezifische Fähigkeiten des Zielsystems ausnutzen, benötigen zwar ein gutes technisches Verständnis des Applikationsentwicklers, die aufwendige Einarbeitung in die komplexen Programmierschnittstellen ist aber nicht notwendig. Dieser Einarbeitungsaufwand ist auch deshalb so relevant, da er zwischen zwei Einsätzen der Technologie dann neu zu erbringen ist, wenn die Pause zwischen den Einsätzen so groß war, dass der Entwickler die einzelnen Details wieder vergessen hat. Das kann bei komplizierten Schnittstellen schon nach wenigen Wochen der Fall sein.

Zusammengefasst lässt sich formulieren, dass der hier vorgestellte Rahmen für Koordinationssysteme in mehreren Fallstudien erfolgreich eingesetzt wurde. Er bietet damit eine hervorragende Grundlage, um die Entwicklung von Applikationen für verteilte heterogene Systeme zu vereinfachen.