14. Diskussion und offene Punkte

In dieser Arbeit wurde die Tauglichkeit des ECL-Systems als Grundlage für Koordinationsprachen zur Entwicklung verteilter Applikationen erfolgreich demonstriert. Dabei wurden unterschiedliche Applikationsbereiche und Zielsysteme durch verschiedene Erweiterungsmodule unterstützt und diese Module mit unterschiedlichen Fallstudien evaluiert. Die folgenden Koordinationsparadigmen wurden untersucht:

- nebenläufige Ausführung von Operationen (Taskgraphen),
- Aufrufparadigma für Datenobjekte,
- Datenflussparadigma,
- deklarative Spezifikation von Abhängigkeiten zwischen Klienten und entfernten Dienstanbietern.

Es wurde gezeigt, dass das ECL-System gut geeignet ist, alle diese unterschiedlichen Paradigmen so zu unterstützen, dass sie dem Applikationsentwickler in einer komfortablen Entwicklungsumgebung als intuitiv benutzbare grafische Architekturelemente zur Verfügung stehen. Dies entspricht den Ergebnissen anderer Arbeiten über Entwicklungsumgebungen, die es ermöglichen, eine Applikation aus komplexen Komponenten grafisch zu erstellen [LK02, LF99, BA96, AS98]. Die Wahl einer Architekturbeschreibungssprache als Grundlage hat sich dabei als besonders vorteilhaft erwiesen, da sich solche Sprachen auf natürliche Weise für eine Visualisierung eignen.

Es konnte ebenfalls erfolgreich demonstriert werden, dass der modulare Ansatz des ECL-Systems die Wiederverwendung von Elementen von Erweiterungsmodulen bei der Erstellung neuer Erweiterungsmodule ermöglicht. Zusätzlich bietet der modulare Ansatz die Möglichkeit, dass die Architekturelemente unterschiedlicher Erweiterungsmodule über ein gemeinsames Basiserweiterungsmodul ohne weitere Anpassungen des Sprachentwicklers miteinander kompatibel sind. Diesen Vorteil teilen alle erweiterbaren Systeme, die auf einem semantischen Basismodell für die Koordination basieren, wie z.B. [IFC96, GW01, PA00] und diverse andere.

Werden in einer Applikation unterschiedliche Koordinationsparadigmen eingesetzt, so sind Probleme bei der Interaktion der Elemente prinzipiell nicht vermeidbar. Dies wird z.B. in [GAO95] am Beispiel eines großen Softwaresystems demonstriert. Dessen inadäquater Ressourcenbedarf und der hohe Implementierungsaufwand konnte darauf zurückgeführt werden, dass die bei

der Implementierung benutzten Werkzeuge und Bibliotheken nicht miteinander kompatibel waren. Diese Unverträglichkeiten waren aber nicht offensichtlich, sondern ergaben sich durch unterschiedliche Annahmen über die Systemarchitektur. Insbesondere ergaben sich, aus der Architektursicht betrachtet, Inkompatibilitäten bei den eingesetzten Konnektoren, die leicht unterschiedliche Koordinationsprotokolle einsetzten.

In [Löh03, Löh02] wird die Problematik untersucht, wie Komponenten, die gänzlich unterschiedliche Koordinationsparadigmen unterstützen, miteinander kombiniert werden können. Es werden erste Vorschläge und Teillösungen für eine automatische Anpassung von Komponenten vorgestellt, trotzdem bleiben wichtige Fragen noch ungeklärt.

In den Fallstudien dieser Arbeit konnten diese aus Paradigmenbrüche resultierenden Probleme alle mit ad hoc-Lösungen zufriedenstellend überwunden werden (s. z.B. Kapitel 10.3). Da die Qualität der Lösung damit aber von den Fähigkeiten des Sprachentwicklers abhängt, wäre es vorteilhaft, wenn Architekturelemente mit inkompatiblem Interaktionsverhalten automatisch aneinander angepasst würden. Um dieses zu ermöglichen, muss das ECL-System um ein allgemeines Interaktionsmodell erweitert werden. Dieses erlaubt dann die Spezifikation des Interaktionsverhaltens der Architekturelemente, welche eine Vorbedingung darstellt, um eine automatische Anpassung zu ermöglichen. Das ist allerdings Gegenstand zukünftiger Forschung.

Da das ECL-System ein Rahmen (Framework) für Koordinationssysteme darstellt, muss die konkrete Funktionalität in den Erweiterungsmodulen bereit gestellt werden. Die Qualität des ECL-Systems zeigt sich daher auch in dem Aufwand, der für den Sprachentwickler notwendig ist, um Erweiterungsmodule zu implementieren. Die in dieser Arbeit entwickelten Module, haben gezeigt, dass sehr viele Freiheitsgrade bei dem Entwurf des Architekturvokabulars, der Elementübersetzer, der Applikationsobjekte und der Laufzeitunterstützungen gibt. Der Aufwand für die Erstellung eines neuen Erweiterungsmoduls hängt allerdings davon ab, wieviele Elemente anderer Erweiterungsmodule wiederverwendet werden können. Die Wiederverwendbarkeit hängt wiederum davon ab, wie gut die einzelnen Erweiterungsmodule entworfen wurden.

Bei der Entwicklung der in dieser Arbeit entstandenen Erweiterungsmodule kam es immer wieder zu Entwurfsänderungen, die eine durchaus komplexe Anpassung verschiedener Module notwendig machte. Die hohe Flexibilität des ECL-Systems stellt damit ein potenzielles Problem dar. Allerdings sind aus den hier gewonnenen Erfahrungen Entwurfsrichtlinien entstanden, die bei der Entwicklung neuer Module beachtet werden sollten, um derartige spätere Entwurfsänderungen zu vermeiden:

- *Autonomie der Architekturelemente*
  Die Syntax und Semantik einzelner Architekturelemente sollte möglichst nur anhand der Eigenschaften der Elemente selber und nicht anhand anderer Elemente der Applikationsarchitektur definiert sein.
  Als konkretes Beispiel für eine autonome Spezifikation der Semantik seien hier Datenobjekte und Rechenkomponenten angeführt. Um zu

bestimmen, auf welchen Datenobjekten Berechnungen durchzuführen sind, sollten sich die Parameter der Rechenkomponente nur auf die Anschlüsse der Komponente, bzw. auf die über Konnektoren mit den Anschlüssen verbundenen Datenobjekte beziehen. In den Parametern sollte kein direkter Bezug auf ein Datenobjekt, z.B. über dessen Name enthalten sein.

- *Basismodelle in abstrakten Erweiterungsmodulen*
  Jedes Erweiterungsmodul X für ein bestimmtes Zielsystem setzt auf einem bestimmten Koordinationsparadigma auf. Dieses Paradigma sollte von einem eigenen Erweiterungsmodul A mit abstrakten Architekturelementtypen unterstützt werden. Das Modul X kann dann auf A aufsetzen und konkrete spezialisierte Architekturelementtypen implementieren.
  Jedes andere Zielsystem, das mit dem gleichen Koordinationsparadigma angesprochen werden soll, kann mit einem Erweiterungsmodul Y unterstützt werden, das ebenfalls auf A aufsetzt. Neben einem verringerten Implementierungsaufwand, kann man mit einem sorgfältigen Entwurf auch gewährleisten, dass die Architekturelemente aus X mit denen aus Y kompatibel sind.

- *Hohe Kohäsion der Erweiterungsmodule*
  Ein Erweiterungsmodul sollte nur Elemente enthalten, die eindeutig zusammengehören. Daher besteht das System Amica aus sieben unterschiedlichen Modulen. Damit wird unnötigen Abhängigkeiten in dem Code entgegengewirkt, die evtl. später zusätzliche Änderungen im Code bedingen.

- *Vollständige Darstellung der Abhängigkeiten*
  Das Architekturvokabular eines Erweiterungsmoduls sollte von Anfang an daraufhin angelegt sein, dass alle Abhängigkeiten zwischen den Komponenten in der Applikationsarchitektur ersichtlich sind. Versteckte Abhängigkeiten führen in der Regel zu nachträglichen Korrekturen oder Anpassungen der Applikationsarchitektur oder des Applikationscodes, der in der Architektur referenziert wird.

Jede einzelne Fallstudie lässt sich prinzipiell auch mit einem anderen Koordinationssystem in ähnlicher Form wie in dieser Arbeit vorgestellt realisieren, wobei durchaus Änderungen und Erweiterungen notwendig sein können. Für jedes hier eingesetzte Koordinationsparadigma existiert ein System, das dieses unterstützt. Eine Mischung unterschiedlicher Paradigmen ist aber i.a. nicht möglich.

Das ECL-System bietet den Vorteil, dass es keine Beschränkungen bzgl. der Paradigmen enthält und damit die Möglichkeit bietet zu untersuchen, wie die einzelnen Paradigmen miteinander interagieren können. Auch erlaubt es zu erforschen, wie ein Koordinationssystem zu strukturieren ist, das sich einfach um die Unterstützung zusätzlicher Zielsysteme erweitern lässt. Für die beiden Fragestellungen wurden in dieser Arbeit Ergebnisse geliefert.

Damit bietet das ECL-System eine hervorragende Grundlage, um neue Applikationsbereiche und Zielsysteme zu untersuchen. So wäre es z.B.

interessant, moderne Web-basierte Datenbankapplikationen zu untersuchen [GM01]. Eine genaue Analyse typischer Strukturen für die Web-Schnittstelle, die Applikationslogik und die persistente Datenhaltung ergibt ein Vokabular von Architekturelemente. Diese lassen sich dann anhand konkreter Applikationen überprüfen.

Ein weiteres höchst interessantes Gebiet sind Systeme und Anwendungen, die auf spontaner Vernetzung basieren. Als aktuelle Forschungsbereiche sind hier Ubiquitous Computing (Allgegenwart des Rechners) und das Pervasive Computing (Verschwinden von Rechnern) zu nennen [BCL+03]. Hier ist es aktuell noch sehr unklar, wie Applikationen gut zu strukturieren sind. Mit ECL lassen sich unterschiedliche Strukturierungsansätze konzipieren und experimentell evaluieren.