Problembeschreibung

Die Programmierung von Beispielanwendungen im Agentenumfeld folgt bislang bekannten Konzepten. Nach der Analyse eines Problems werden Strukturen identifiziert, welche zum Einsatz kommen sollen, was schließlich im Entwurf einer oder mehrerer Komponenten mündet. Dieses Vorgehen wird für jedes Projekt im Einzelnen durchgeführt, womit jede Anwendung für sich isoliert entworfen und in Betrieb genommen wird. Leider findet die so genannte *Wiederverwendung* von Code schon im Umfeld der objektorientierten Programmierung in nur begrenztem Ausmaße statt, was sich durch die Tatsache, dass die Agentenprogrammierung sich heutzutage der Objektorientierung bedient, auf diese überträgt. Die Fähigkeit von Agenten, zur Erfüllung ihres Auftrags eigenständig Entscheidungen zu treffen, die möglicherweise auch die Kommunikation mit zunächst unbekannten Partnern beinhaltet, wird durch die starre Festschreibung der Komponenten konterkariert.

Multiagentenanwendungen sind auf die Kooperation der Agenten untereinander angewiesen: Jeder angesprochene Agent muss die ihm übermittelten Daten eines anderen Agenten zu verarbeiten wissen. Bisher vertraute man darauf, dass der Entwurf einer agentenbasierten Anwendung diese Probleme umgeht, da der Entwurf *abgeschlossen* ist, es also stets eine genau definierte Menge von möglichen, interagierenden Komponenten gibt. Es stellt sich die Frage, ob durch eine solche Strategie nicht ein großes Potenzial des Einsatzes von Agenten verschenkt wird. Gerade im Hinblick auf künftige Implementierungen elektronischer Märkte erscheint es sinnvoll anzunehmen, dass Multiagentenanwendungen *offen* sind, dass Agenten fortwährend mit bislang unbekannten Partnern kooperieren müssen.

Um diese Beschränkungen zu lockern und den Weg zu offenen Anwendungen zu ebnen, kann das Konzept der *Typen* hilfreich sein. Die Vorteile der Verwendung typisierter Sprachen liegen auf der Hand: Zum einen kann der Compiler bereits vor der Ausführung eine Prüfung des Programms durchführen. Dazu wird sichergestellt, dass den Variablen nur typkonforme Instanzen zugewiesen sowie nur legale Operationen auf diesen Variablen ausgeführt werden. Zum anderen erlauben Klassen, welche die Rolle von Typen bei komplexen Datenstrukturen spielen, eine bequeme Wiederverwendung geschriebenen Codes durch das so genannte *Ableiten*, was dem Implementierer die Möglichkeit bietet, Anwendungen erweiterbar zu gestalten. Während des Ablaufs einer Anwendung wird die Verfügbarkeit eines Objekts einer bestimmten Klasse erwartet; an deren Stelle darf eine Erweiterung in Form einer abgeleiteten Klasse treten. Es ist somit möglich, durch Aggregation von Funktionalitäten eine Wiederverwendung des Codes zu erreichen; das Objekt präsentiert sich den unterschiedlichen Anfragern stets mit der von ihnen erwarteten Menge von Funktionalitäten.

Jedoch ist eine bloße Übertragung des Klassenkonzepts auf die Agentenwelt nicht uneingeschränkt sinnvoll: Einerseits können Agenteninteraktionen sehr viel komplexer aufgebaut sein als solche zwischen Objekten; die traditionellen Klient-/Serverrollen wechseln beständig, da in der Regel jeder der autonomen Agenten seinen eigenen Auf-