

Eine Anwendungsarchitektur für adaptive Systeme (Kurzfassung)

Langfristiges Generalziel mit Sicht auf die nächsten 10 Jahre ist es, intelligente, selbstlernende und selbstregulierende Anwendungssysteme zu schaffen. Derartige Anwendungssysteme können als adaptive Systeme bezeichnet werden. Adaptive Systeme bestehen aus einer Vielzahl lose miteinander gekoppelten Einheiten mit jeweils eigener Intelligenz, die unabhängig voneinander entwickelt werden können und die Fähigkeit zur ad-hoc-Kooperation, auch über Unternehmensgrenzen hinweg, aufweisen. Adaptive Systeme können einige der Probleme heutiger Anwendungssysteme, die fast ausschließlich als Systeme mit zentralisierter Steuerung realisiert sind, lösen. Bei zentralisierter Steuerung ist immer eine bidirektionale Verbindung zu sämtlichen Objekten bzw. Komponenten erforderlich. Im Ergebnis wird die Steuerung sehr komplex und ist nur noch mit Mühe durchschaubar.

Für Anwenderunternehmen, die sich derzeit mit einem Redesign beschäftigen, besteht die Chance, grundsätzliche Gedanken über die Anwendungsarchitektur anzustellen. Kapazitäts- und Bandbreitenüberlegungen spielen zwar heute noch eine sehr wichtige Rolle, jedoch in 10 Jahren mit Sicherheit nicht mehr. Schon heute sind Übertragungsraten über dezidierte Weitverkehrsverbindungen im Gigabit- und sogar Terabit-Bereich möglich. Auch die Prozessorgeschwindigkeiten werden die heutigen um mehr als Faktor 10 übersteigen. Eine moderne Anwendungsarchitektur kann sich somit nicht an den heute gültigen Rahmenbedingungen orientieren, muß andererseits jedoch gleichzeitig auch die Aufwärtskompatibilität der bisherigen Anwendungen sicherstellen.

Eine Anwendungsarchitektur, die adaptiven Systemen zugrunde liegt, wird hochgradige Verteilbarkeit bei gleichzeitiger hoher Verfügbarkeit anstreben. Anwendungsbausteine sind flexibel verteilbar, um eine dynamische Lastverteilung im Systemverbund zu ermöglichen. Die hohe Verfügbarkeit ist gewährleistet, da Anwendungsbausteine portabel und lokationsunabhängig sind. In technischer Sicht möglicherweise erforderliche Abstriche aufgrund unterschiedlicher Basistechnologien spielen zunächst noch keine Rolle.

Da sich die Geschäftsprozesse auf Branchenebene im allgemeinen nur wenig unterscheiden, kann eine generische Anwendungsarchitektur branchenspezifisch konkretisiert werden. Die Unternehmen der Branche müssen auf dieser Basis nur noch unternehmensspezifische Elemente entwickeln. Die zur Entwicklung neuer Anwendungen benötigte Zeit wird, begünstigt durch die bereits vorhandene Basistechnologie, wesentlich verkürzt. Andererseits sind Geschäftsprozesse einer der differenzierenden Faktoren, über die sich das individuelle Unternehmen einen Wettbewerbsvorteil gegenüber seinen Mitbewerbern verschaffen kann. Deshalb muß eine Branchenarchitektur genügend Freiraum lassen und darf eine Vereinheitlichung der Geschäftsprozesse nicht erzwingen. Der implizite Zwang zur Anpassung von Geschäftsprozessen an die Software ist einer der schmerzlichen Nachteile heutiger Standardanwendungs-Software.

Jede Anwendungsarchitektur sollte sich so weitgehend wie nur möglich an Vorgängen und Objekten der realen Welt orientieren. Auf den ersten Blick scheint keine direkte Beziehung zwischen Anwendungsarchitektur und dem genannten Ziel zu bestehen. Andererseits wird die Blickrichtung bereits von Beginn an auf Eigenschaften wie Abgeschlossenheit, Verteilbarkeit, Flexibilität, Skalierbarkeit usw. gelenkt.

Das konzeptionelle Modell

In der heutigen betrieblichen Praxis wirken Objekte (Menschen, Maschinen, Software-Programme usw.) aufgabenorientiert in vielfältigen Kombinationsformen zusammen. Beispiele sind die Maschine-Maschine-Kommunikation im Fertigungsprozeß, die Programm-zu-Programm-Kommunikation, die Mensch-zu-Mensch-Kommunikation und die Mensch-Programm-Kommunikation. Grundlage des Zusammenwirkens sind strukturierte Geschäftsprozesse, denen wiederum definierte Regeln zugrundeliegen. Das Bild wird durch informelle, unstrukturierte ad-hoc-Prozesse, die in "offizielle" Geschäftsprozesse eingreifen, aber auch neben diesen ablaufen, vervollständigt.

Das Bild ist geprägt von einer Vielzahl selbständiger, miteinander interagierender Entitäten, die von unterschiedlichen Herstellern hergestellt werden und miteinander interagieren. In Analogie hierzu muß es auch das Generalziel einer modernen Anwendungsarchitektur sein, selbständige Entitäten zu fördern und maximale Entkopplung vorzusehen. Darüber hinaus muß im Interesse einer Höherentwicklung ein weiteres Ziel sein, die Lernfähigkeit und Selbstregulierungsfähigkeit der Entitäten zu entwickeln. Selbständige Entitäten sind durch folgende Eigenschaften charakterisiert:

- sie verfügen über bestimmte Kenntnisse, Fähigkeiten und Erfahrungen. Diese Eigenschaften sind nicht statisch, sondern werden sich im Zeitverlauf weiterentwickeln. Kenntnisse werden erweitert, Fähigkeiten werden weiter-

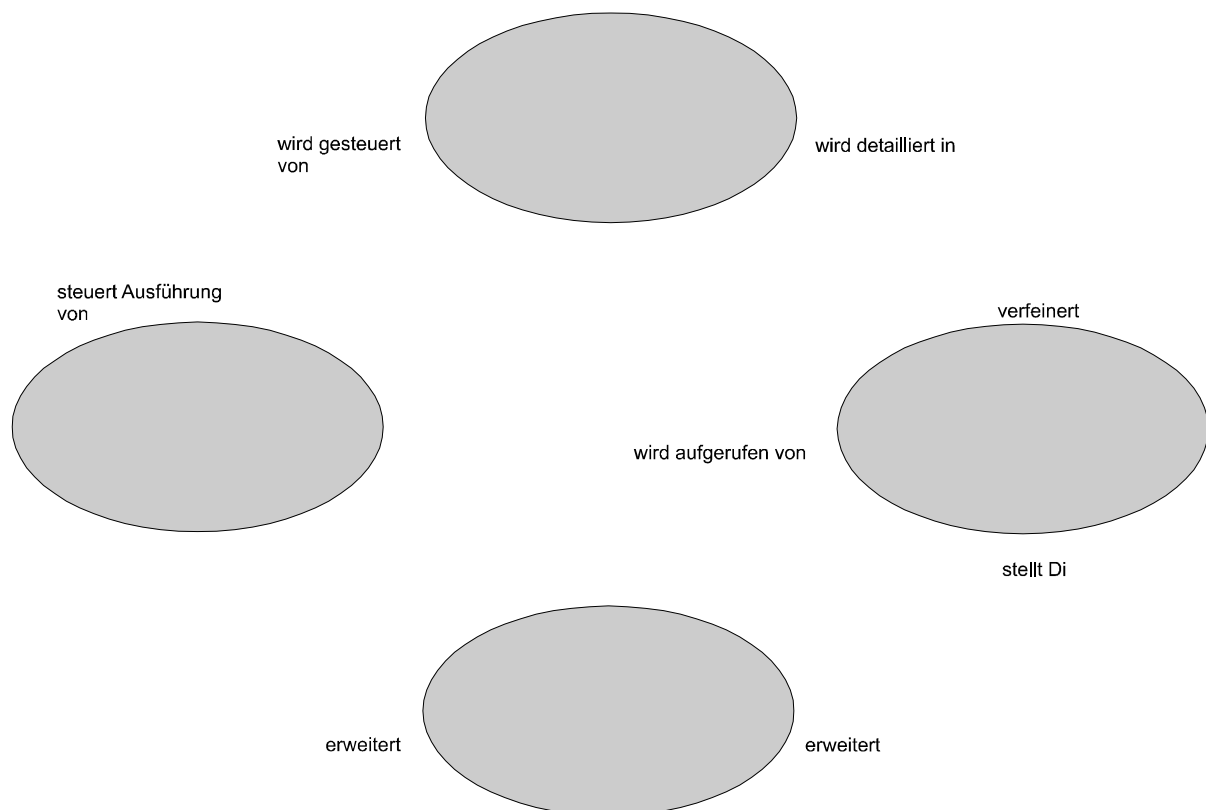
entwickelt und der Erfahrungsschatz wird ständig wachsen. Selbständige Entitäten sind lernfähig und in der Lage, ihr Verhalten zu ändern;

- sie können mit anderen Entitäten zusammenwirken. Sie können sich aufgrund ihres Wissens einen Partner (eine andere Entität) suchen, mit dem gemeinsam ein bestimmtes Ergebnis erzielbar ist. Jede Entität verfügt über ein begrenztes Wissen, jedoch kann das Ergebnis die Summe des Einzelwissens übersteigen;
- sie sind nicht an eine bestimmte Lokation gebunden. Selbständige Entitäten können selbst entscheiden, an welchem Ort sie ihre Dienste unter welchen Umständen am besten erbringen können.

Für die technische Ebene ergeben sich eine ganze Reihe von Konsequenzen. Die wichtigsten sind:

- Eine zentrale Steuerung von Geschäftsprozessen ist nicht erforderlich. In der realen Welt ist diese ebenfalls nicht vorhanden (die Abfolge von Subprozessen und Prozeßschritten und Ausführungsbedingungen sind im Geschäftsprozeß definiert und sind den beteiligten Entitäten bekannt (gehören zu deren Wissen)). Die Steuerung kann durch intelligente Entitäten in Absprache miteinander erfolgen;
- Kommunikationsfähigkeiten zwischen selbständigen Entitäten spielen eine sehr wichtige Rolle. Global vereinbarte Mechanismen zur Signalisierung von Ereignissen und zum Nachrichtenaustausch sind zwingend erforderlich;
- Plattformunabhängigkeit der Entitäten ist erforderlich. Jede Entität kann auf jedem Computersystem residieren und kann ohne Einschränkungen im Systemverbund verlagert werden. Die Technologien für verteilte Systeme müssen Plattformunabhängigkeiten zulassen. Lock-in-Strategien von Herstellern wirken kontraproduktiv.

Der Versuch, die angedeutete Vision adaptiver Systeme zu einem konzeptionellen Modell zu formen, bringt vier Entitäten hervor, die in enger Beziehung zueinander stehen: Geschäftsprozesse, Geschäftsobjekte, Agenten und Anwendungen. Geschäftsprozesse (z. B. Kundenauftrag ausführen) sind statisch und geben vor, welche Prozeßschritte in welcher Folge und unter welchen Bedingungen auszuführen sind. Geschäftsobjekte bilden Objekte und Konzepte der realen Welt ab, so z. B. Kunde, Vertrag und Bestellung. Agenten sind Stellvertreter für betriebliche Ressourcen im weitesten Sinne. Dazu zählen beispielsweise Personen und Maschinen. Anwendungen schließlich sind Hilfsmittel, die Agenten und Geschäftsobjekte erweitern und mit externen Entitäten (z. B. Anwender, Datenspeicher, Geräte usw.) interagieren.



Geschäftsprozeß

Ein Geschäftsprozeß ist eine Abfolge von Aufgaben (z. B. Kundendaten prüfen, Kundenbonität prüfen, Auftrag erfassen), die in einer definierten Reihenfolge unter Beobachtung definierter Regeln (z. B. sofern das Kreditlimit des Kunden überschritten wird, kann der Auftrag nicht angenommen werden) ausgeführt werden. Ausgelöst wird ein Geschäftsprozeß stets durch ein definiertes Ereignis (z. B. Auftragsformular eingegangen). An einem Geschäftsprozeß sind in der Regel mehrere Organisationseinheiten (z. B. Kunde selbst, Vertrieb) beteiligt. Im Geschäftsprozeß ist ebenfalls festgelegt, welche Organisationseinheiten welche Aufgaben ausführen.

Jede Aufgabe (z. B. Bonität prüfen) entspricht einem Prozeßschritt. Jeder Prozeßschritt wiederum produziert und konsumiert definierte Leistungen und trägt damit zur Gesamtleistung des Prozesses bei. Ein Geschäftsprozeß kann der besseren Übersichtlichkeit wegen durchaus in Subprozesse gegliedert sein. Subprozesse können sequentiell hintereinander oder auch parallel ablaufen. Als Merkmal eines Geschäftsprozesses läßt sich seine Abgeschlossenheit festhalten; ein sinnvolles Ergebnis wird nur dann erzielt, wenn der Geschäftsprozeß vollständig durchgeführt wird.

Der Geschäftsprozeß gibt eine allgemeine Handlungsanweisung vor. Jede konkrete Anwendung eines Geschäftsprozesses führt zu einem Geschäftsvorgang (z. B. die Verbuchung eines Zahlungseingangs des Kunden Müller über DM 123,45 zur Begleichung der Rechnung Nr. 987654). Geschäftsvorgänge sind prinzipiell langlaufende Transaktionen und können sich potentiell über mehrere Stunden, Tage oder gar Monate erstrecken. Sie werden in Subtransaktionen gegliedert, die jeweils einen konsistenten Verarbeitungszustand definieren. Subtransaktionen entsprechen in der Regel einem Prozeßschritt.

Geschäftsprozesse werden keinesfalls in Anwendungslogik abgebildet ("verdrahtet"), sondern werden anwendungsextern verwaltet. Workflow Management-Systeme bieten sich für strukturierte und unstrukturierte Geschäftsprozesse (ad hoc-Prozesse) an. Für hochgradig strukturierte Geschäftsprozesse können Geschäftsprozeß-Definitionen auch in Parameterdaten abgelegt werden (s. u.).

Geschäftsobjekt

Ein Geschäftsobjekt (Business Object) repräsentiert, der Definition der OMG Business Object Domain Task Force (BODTF) entsprechend, einen realen Gegenstand oder ein Konzept der Geschäftswelt. Es umfaßt einen Namen, unter dem es in der Geschäftswelt bekannt ist, eine Definition (Bedeutung und Zweck), Eigenschaften, Verhalten, Beziehungen zu anderen Geschäftsobjekten, Regeln und Einschränkungen (constraints). Geschäftsobjekte sind somit Objekte, die eine Bedeutung im Kontext der Geschäftstätigkeit des Unternehmens haben. Jedes Geschäftsobjekt ist, genauso wie die Entsprechung in der realen Welt, eindeutig identifizierbar und in sich abgeschlossen. Da Geschäftsobjekte in sich abgeschlossen sind, können sie auch unabhängig voneinander entwickelt werden. Beispiele für Geschäftsobjekte sind: Angestellter, Produkt, Bestellung, Lieferung, Rechnung und Zahlung.

Es ist jedoch durchaus nicht immer einfach zu entscheiden, welche Funktionen (Methoden) einem Geschäftsobjekt bzw. welchem Geschäftsobjekt welche Funktionen zugeordnet werden. So kann sich beispielsweise zu Beginn der Analyse die Frage stellen, ob die Funktion "Bonität prüfen" zum Geschäftsobjekt "Kunde", zum Geschäftsobjekt "Auftrag" oder gar zu beiden Geschäftsobjekten gehört. Die Zuordnung zum Geschäftsobjekt "Kunde" ist schlüssig, da es sich bei Bonität um eine Eigenschaft von Kunden handelt. Unter dem Kontextgesichtspunkt ist die Zuordnung zum Geschäftsobjekt "Auftrag" vertretbar, sofern die Kundenbonität immer nur im Kontext der Auftragsbearbeitung geprüft wird. Es wird in aller Regel sinnvoll sein, eine Funktion dem Geschäftsobjekt zuzuordnen, auf dessen Eigenschaft(en) es sich bezieht. Einen Anhaltspunkt für die Entscheidung gibt auch das Datenmodell. Dort wird, bezogen auf das Beispiel, die Bonität ein Attribut des Entitätentyps "Kunde" sein.

Auch die Frage, in welchem Umfang ein Geschäftsobjekt Regeln verkapselt, die andere Geschäftsobjekte betreffen, muß beantwortet werden. So könnte beispielsweise das Geschäftsobjekt "Auftrag" die Geschäftsregel verkapseln, daß bei jedem Auftrag oberhalb eines bestimmten Auftragswerts eine Bonitätsprüfung zu erfolgen hat. Das Geschäftsobjekt "Auftrag" würde sodann selbständig beim Geschäftsobjekt "Kunde" eine Bonitätsprüfung anfordern und damit selbst einen Geschäfts-Subprozeß verkapseln. Als Alternative wären Geschäftsregeln in den Geschäftsprozeß zu verlagern. Für die erstgenannte Möglichkeit sprechen folgende Gründe:

- Geschäftsregeln sind im Geschäftsobjekt verkapselt und nicht im Geschäftsprozeß (selbstverständlich kann das Geschäftsobjekt absolute Werte bzw. Wertebereiche auch zur Ausführungszeit aus Parameterdaten lesen),
- sofern sich eine Geschäftsregel ändert, muß lediglich an einer Stelle geändert werden (im Geschäftsobjekt) und nicht in möglicherweise mehreren Geschäftsprozessen,
- das Geschäftsobjekt hat vollständige Kontrolle über seine eigene Integrität,
- der Geschäftsprozeß kann sich auf die Steuerung auf hoher Ebene beschränken.

In nicht-technischer Sicht ist ein Geschäftsobjekt eine Abstraktion, die die darunterliegende Technologie vollständig verbirgt. Es ist z. B. unerheblich, auf welcher Betriebssystem-Plattform das Geschäftsobjekt residieren kann oder welches Datenhaltungssystem persistente Zustände speichert. Es ist lokationsunabhängig und kann dynamisch zwischen Computersystemen verschoben werden. Ferner kann ein Geschäftsobjekt deaktiviert und aktiviert werden.

Geschäftsobjekte sind normalerweise persistent (speichern ihren Zustand auf einem nicht-flüchtigen Speicher) und deshalb wiederherstellbar. Die Frage, wie dies konkret geschieht und welche Art von Datenhaltungssystem eingesetzt wird, ob objektorientierte Datenbank, relationale Datenbank oder gar ein konventionelles Dateisystem, spielt aus Sicht des Geschäftsobjekts keine Rolle. Es muß lediglich sichergestellt sein, daß der Zustand wiederherstellbar ist.

Operationen auf Geschäftsobjekte erfolgen grundsätzlich im Transaktionskontext, d. h. ein Geschäftsobjekt hat stets einen definierten Zustand. Entweder werden die im Transaktionskontext durchzuführenden Operationen vollständig durchgeführt und der dann erreichte Zustand wird festgeschrieben (*Commit*) oder der zu Bearbeitungsbeginn gültige Ausgangszustand wird wiederhergestellt (*Rollback*). Außerdem wird sichergestellt, daß im konkurrierenden Mehrbenutzerbetrieb geeignete Mechanismen (z. B. Sperrkonzepte) angewandt werden, um die Isolation der Transaktionen voneinander zu gewährleisten.

Die Frage, ob eine interaktive Bediener-Schnittstelle existiert und wie diese beschaffen ist, ist ebenfalls nicht von Bedeutung. Die Bediener-Schnittstelle eröffnet lediglich den Zugang zu ohnehin implementierten Operationen, d. h. die Funktionalität wird durch die Bediener-Schnittstelle nicht direkt berührt. Das Geschäftsobjekt "weiß" nicht, ob es über eine Dialog-Oberfläche verfügt.

Agent

Selbständige Entitäten der realen Welt können als Software-Agenten repräsentiert werden. Software-Agenten bilden Verhalten und Eigenschaften ihrer Gegenüber der realen Welt weitestgehend ab. Bei weiter Auslegung des Begriffs sind Software-Agenten:

- autonom: Ein Agent ist eine unabhängige Entität, die selbständig mit ihrer Umgebung interagieren kann. Er ist in sich abgeschlossen, steuert das eigene Verhalten und den Status, weiß wann die Integrität verletzt ist und kann einen konsistenten Zustand wiederherstellen;
- kommunikativ: kann sich mit anderen Entitäten verständigen (z. B. Menschen, Maschinen, Software-Agenten),
- mobil: Agenten sind in der Lage, sich selbst in eine andere Umgebung (z. B. eine andere System-Plattform) zu transportieren,
- reaktiv: Agenten erkennen definierte Veränderungen in der Umgebung und reagieren selbständig auf diese. Agenten können sich als Empfänger von Ereignissen registrieren, können jedoch aber auch selbst aktiv den Zustand ihrer Umgebung "scannen";
- ständig aktions- und reaktionsfähig: Agenten werden als separater, ständig aktiver Prozeß ausgeführt,
- flexibel: dem Agenten werden von außen keine bestimmten Aktionen vorgegeben (z. B. keine Skripts). Der Agent entscheidet selbständig über die durchzuführenden Aktionen;
- lernfähig: Jeder Software-Agent verfügt über eine eigene Wissensbasis und ist lernfähig. Agenten sind in der Lage, auf Basis gewonnener Erfahrungen ihr Verhalten zu ändern. Auf diese Weise können sich die Tätigkeitsprofile von Software-Agenten im Zeitverlauf ändern, ebenso wie die realer Personen.

Eine konkrete Agentenimplementierung muß nicht alle der genannten Eigenschaften aufweisen. Ein wichtiges differenzierendes Merkmal ist die Lernfähigkeit. In grober Betrachtung können zwei Generationen von Agenten unterschieden werden:

- Generation 1: reaktive Agenten, die in einer vordefinierten Weise auf bekannte Ereignisse reagieren (WHEN Ereignis, IF Bedingung, THEN Aktion).
- Generation 2: intelligente Agenten, die sich selbst anpassen können (lernfähige Agenten).

Kandidaten für Software-Agenten sind prinzipiell sämtliche Rollen, die in der realen Welt vorkommen. Beispiele sind Auftragserfasser, Antragsbearbeiter und Debitoren-Buchhalter. Jeder Software-Agent kann beliebig oft instanziiert werden. In der realen Welt würde dies der beliebigen Vermehrbarkeit von Personen und Maschinen entsprechen, wobei jede Instanz durch identisches Verhalten und identische Eigenschaften gekennzeichnet ist.

Sowohl der individuelle Anwender als auch die Anwendergruppe werden von Agenten repräsentiert. Der "individuelle Agent" handelt im Auftrag des Anwenders, während der "Gruppenagent" gewissermaßen die Interessen der Gruppe vertritt und die Gruppenmitglieder führt und koordiniert (er übernimmt die Rolle eines Manager).

Prinzipiell können mehrere Agenten dieselbe Leistung anbieten. Auch hier findet sich eine Entsprechung zur realen Welt. Personen können temporär nicht verfügbar sein, z. B. bedingt durch Urlaub oder Krankheit. Auch Software-Agenten können temporär nicht verfügbar sein. Anlässe sind hier beispielsweise die abnormale Beendigung des Prozesses oder eine Störung der Kommunikationsverbindung.

Die Prinzipien der Objektorientierung finden sich vollständig wieder. In technischer Betrachtung sind Agenten Objekte, ergänzt um aktives Verhalten und um eine Agenten-Kommunikationssprache (Agent Communication Language). Das aktive Verhalten wird durch einen Interpreter konstituiert, der Nachrichten und Ereignisse auswertet und entsprechende Aktionen ausführt. Agenten können mehrere Schnittstellen zum Anwender unterstützen, so z. B. eine Dialog-Schnittstelle und/oder eine Sprach-Schnittstelle. "Lernt" der Software-Agent (die Objektklasse(n)), lernen alle Agenteninstanzen (Objekte) gleichermaßen.

Software-Agenten verständigen sich untereinander über ein definiertes Protokoll, das international standardisiert sein sollte. Auf diese Weise lassen sich auch unternehmensexterne Agenten einbinden. Am sinnvollsten ist ein logisches Vier-Schritte-Protokoll, das die folgenden Aktivitäten umfaßt:

- Anfrage: spezifiziert die angefragte Leistung,
- Zusage: die Zusage, daß die angefragte Leistung erbracht werden kann,
- Lieferung: das Erbringen der zugesagten Leistung,
- Akzeptanz: die Akzeptanz der Leistung durch den Anfrager.

Jeder Software-Agent greift auf seine eigene Datenbasis zu. Auf diese Weise wird neben der logischen auch die physische Unabhängigkeit sichergestellt. Eine gemeinsame Datenbasis würde die Unabhängigkeit gefährden. Wäre beispielsweise die Datenbank nicht verfügbar, wären mehrere Software-Agenten betroffen. Agentenspezifische Datenbanken vermindern das Risiko, können es jedoch möglicherweise nicht ganz ausschließen (z. B. wenn das Datenbanksystem nicht verfügbar ist, sind auch alle von diesem verwalteten Datenbanken nicht verfügbar. Mögliche Konstellationen müssen produktspezifisch geprüft werden). Die Vielzahl von Datenbanken kann hingenommen werden. Möglicherweise können jedoch technische Restriktionen des DBMS dies verhindern (durch Beschränkung der Anzahl gleichzeitig offener Datenbanken). Konventionelle Dateien eignen sich ebenso.

Durch die vollständige Unabhängigkeit der Software-Agenten voneinander ist es leichter möglich, die einzelnen Agenten individuell weiterzuentwickeln. Ungewollte Nebeneffekte, die auf andere Agenten durchschlagen, können nicht auftreten. Davon abgesehen, wird auch das Konfigurations-Management vereinfacht.

Die individuelle Identifizierbarkeit von Software-Agenten ist erforderlich, auch wenn sämtliche Agenteninstanzen hinsichtlich ihrer Fähigkeiten und Eigenschaften identisch sind. Jeder Agent muß seinen Status in den Objektattributen festhalten. Im Hinblick auf die Implementierung stehen alle Möglichkeiten offen. Jeder Agent kann als separater Prozeß oder als Thread realisiert werden. Zweifellos ist die letztgenannte Lösung unter Performanzgesichtspunkten vorteilhaft. Agenten müssen dann grundsätzlich über die Multithreading-Fähigkeit verfügen.

Jeder Agent verwaltet seinen Status selbst und unabhängig von anderen Agenteninstanzen. Er kennt Integritätsregeln, kann erkennen, ob eine Integritätsregel verletzt ist und kann selbst wieder einen integren Zustand herstellen. Auch hier wird wieder der Charakter eines Agenten als abgeschlossene Einheit deutlich.

Bei der Abarbeitung eines Geschäftsvorgangs ändert sich der Zustand des Geschäftsvorgangs sehr häufig. Die Zustandsinformation muß unter Steuerung der beteiligten Agenten verwaltet werden. Dies muß in einem separaten, nicht-flüchtigen Speicher erfolgen, auf den sämtliche Agenten zugreifen können. Dieser Vorgangsspeicher wird mit Beginn eines Geschäftsvorgangs eingerichtet und mit dessen Ende wieder freigegeben.

Anwendung

Der Anwendungsbegriff verliert im Kontext von Software-Agenten und Geschäftsobjekten an Schärfe. Eine Anwendung wird bisher als Programm verstanden, die eine bestimmte Geschäftsfunktion ausführt. Diese Abgrenzung ist nicht mehr zutreffend, da auch Geschäftsobjekte an der Ausführung von Geschäftsfunktionen beteiligt sind. Das Verständnis einer Anwendung wird sich deshalb weiter wandeln.

Geschäftsobjekte bilden als Repräsentanten von Objekten der realen Welt deren Verhalten und Eigenschaften vollständig ab. Ihnen fehlt jedoch die Bediener-Oberfläche und damit die Möglichkeit, mit Anwendern zu kommunizieren und/oder Daten in bestimmter Weise aufzubereiten. In der Konsequenz erweitert eine Anwendung Geschäftsobjekte um den konkreten Ausführungskontext und stellt die Verbindung zum konkreten Ausführungsumfeld her. Anwendungen können als Komponenten realisiert werden.

Ein Geschäftsobjekt kann indirekt über beliebig viele unterschiedliche Bediener- bzw. Darstellungs-Oberflächen verfügen. Die Verbindung zwischen Geschäftsobjekt und Darstellungsoberfläche wird über Anwendungen hergestellt. Ein anderes Beispiel neben der Dialog-Oberfläche sind Berichte (*Reports*), wo die Anwendung die aufgabenspezifische Aufbereitung und Darstellung übernimmt. Im wesentlichen ist es Aufgabe von Anwendungen, die Interaktion zwischen Geschäftsobjekt und Dialog-Manager (s. u.) einerseits und dem Daten-Manager (s. u.) andererseits zu ermöglichen.

Eine Anwendung erfüllt eine spezifische Aufgabe, in der Regel eine bestimmte Geschäftsfunktion (z. B. das Erfassen eines Auftrags). Die Geschäftsfunktion kann ihrerseits wiederum aus mehreren Funktionen bestehen (z. B. Bonität prüfen, Auftragskopf erfassen, Auftragsposition erfassen usw.) und damit einen vollständigen Geschäfts-Subprozeß abdecken. Im Funktionenmodell ist dies die nächste Hierarchiestufe bzw. Detaillierungsebene.

Anwendungen werden unter Steuerung von Agenten und Geschäftsobjekten aufgerufen. Die Steuerung kehrt nach Beendigung der Anwendung wieder zum Aufrufer zurück. Die Wahrscheinlichkeit, daß ein und dieselbe Anwendung sowohl von Agenten als auch von Geschäftsobjekten aufgerufen wird, ist sehr gering. Agenten rufen lediglich Anwendungen auf, die in enger Beziehung zur Geschäftsprozeßsteuerung stehen. Ein Beispiel ist die Behandlung einer entdeckten Inkonsistenz, die von Geschäftsobjekten nicht erkannt werden konnte.

Die Architektur

Agenten, Geschäftsobjekte und Anwendungen werden im Kontext eigenständiger Prozesse ausgeführt (Agenteninstanzen und Geschäftsobjektinstanzen können in Threads ausgeführt werden). Sie sind eingebettet in die anwendungs- und die systemorientierte Infrastruktur. Den gängigen Design-Grundsätzen entsprechend ist eine Schichtenarchitektur das geeignete Mittel, um Aufgaben besser abgrenzen zu können und die Komplexität in Grenzen zu halten.

Die Ausführungssteuerung von Geschäftsprozessen obliegt Agenten, die diese Aufgabe gemeinschaftlich wahrnehmen. Agenten wirken auf Geschäftsobjekte und erteilen diesen Aufträge entsprechend dem im Geschäftsprozeß definierten Ausführungsfluß. Einem Agenten muß bekannt sein, welches Geschäftsobjekt eine bestimmte Aufgabe repräsentiert. Das Geschäftsobjekt "Auftrag" wird beispielsweise Methoden beinhalten, die die Aufgabe "Auftrag erfassen" ausführen.

Prinzipiell können Geschäftsobjekte auch Geschäfts-Subprozesse verkapseln. Dies würde nahelegen, eine kooperative Steuerung bei Beteiligung von Agenten und Geschäftsobjekten zu gestalten. Dies ist durchaus sinnvoll. Die folgenden Rahmenbedingungen spielen bei der Abgrenzung, wie die Steuerung aufzuteilen ist, eine Rolle:

- ein Geschäftsprozeß tangiert in aller Regel mehrere Geschäftsobjekte (z. B. bei der Bearbeitung eines Auftrags sind neben dem Geschäftsobjekt "Auftrag" auch weitere Geschäftsobjekte beteiligt (z. B. "Kunde", "Artikel", "Lieferschein", usw.);
- Geschäftsobjekte können lediglich ihre eigene Konsistenz und Integrität überwachen;
- Geschäftsobjekte haben keinerlei Kenntnis, welche Organisationseinheiten in welchen Rollen am Geschäftsprozeß beteiligt sind.

In der Konsequenz bietet sich als Lösung an, Software-Agenten die allgemeine Steuerung des Ausführungsflusses von Geschäftsprozessen zu übertragen. Wie bereits erwähnt, kooperieren in der Regel mehrere Agenten während der Ausführung eines bestimmten Geschäftsprozesses miteinander und kommunizieren miteinander über eine

Agenten-Kommunikationssprache. Geschäftsobjekte können Geschäfts-Subprozesse steuern. Maßgebendes Kriterium für die Reichweite der Steuerung sind Konsistenz und Integrität der Geschäftsobjekte. Transaktionsgrenzen spielen als Leitlinie eine wichtige Rolle.

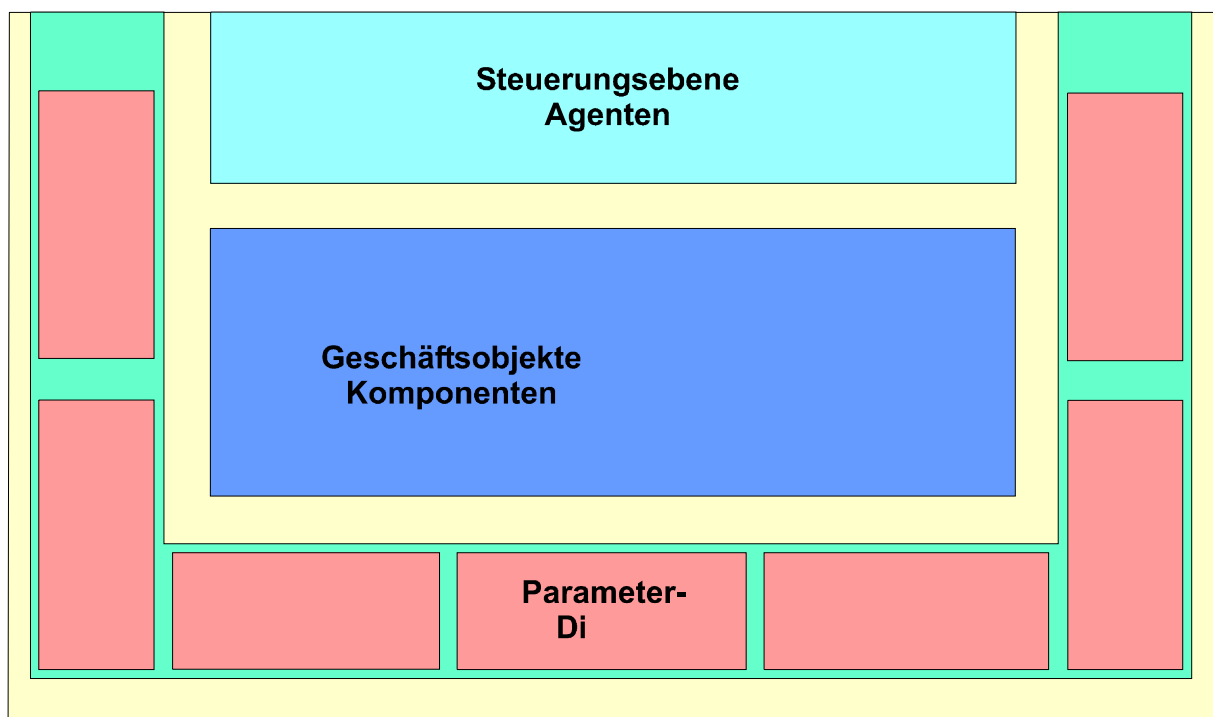
Agenten steuern den Ausführungsfluß somit auf hoher Ebene. Über Methodenaufrufe übergeben sie Direktiven an Geschäftsobjekte (z. B. "Auftrag erfassen"). Die Direktiven werden auch Informationen zum konkreten Ausführungskontext enthalten (z. B. eindeutige Identifikationen des Geschäftsvorgangs, der Organisationseinheit, des Anwenders, usw.). Wenn das Geschäftsobjekt die Steuerung an den Agenten zurückgibt, ist der Auftrag entweder erfaßt oder der Auftrag konnte nicht erfaßt werden. Als Nebeneffekt sind gleichzeitig sinnvolle Transaktionsgrenzen beachtet. Die Subtransaktion "Auftrag erfassen" ist ein sinnvoller Arbeitsschritt im Kontext der langlaufenden Transaktion (Geschäftsprozeß) "Auftrag bearbeiten".

Sowohl Agenten als auch Geschäftsobjekte können interagieren mit:

- Anwendern: dies geschieht über den Dialog-Manager, der als Isolationsschicht zum Präsentations-System (z. B. Windows GUI-System) fungiert,
- Datenobjekten: dies geschieht über den Daten-Manager, der als Isolationsschicht zum Datenhaltungssystem (z. B. ORACLE) fungiert,
- Dienstleistungsobjekten: ein Sammelbegriff für Dienste mit unterschiedlichen Aufgaben. Zu den Diensten zählen beispielsweise der Parameterbereitstellungs-Dienst und der Fehlerbehandlungs-Dienst).

Die folgende Grafik zeigt das Schichtenmodell einer Anwendungsarchitektur. Die oberste Ebene wird durch die Steuerungsebene (kooperative Steuerung) als gleichzeitig oberste Abstraktionsebene ausgefüllt. Geschäftsobjekte auf der nächsten Ebene verkapseln die gesamte Funktionalität sämtlicher Anwendungsdomänen. Die dritte Ebene wird durch die anwendungsorientierte Infrastruktur repräsentiert. Zur Infrastruktur zählen Dienste, die für die beiden übergeordneten Schichten erbracht werden. Das Modell ist nicht streng schichtenorientiert. Somit können auch Agenten direkt Infrastrukturdienste nutzen.

Um sichtbar zu machen, daß alle drei Schichten in die Systemumgebung eingebettet sind, sind die Dienste der systemorientierten Infrastruktur in der dritten Dimension dargestellt. Zur systemorientierten Infrastruktur zählen vor allem die Plattformen für verteilte Systeme (COM/DCOM, CORBA und Java-Plattform) und die Komponenten-Plattformen (COM/ActiveX, CORBA Components, JavaBeans und Enterprise JavaBeans), aber auch GUI-Systeme und Datenhaltungssysteme.



Zusammenfassung und Bewertung

Die skizzierte Anwendungsarchitektur eröffnet einen behutsamen ersten Einstieg in agentenbasierte Systeme und wird sicherlich nicht alle Bereiche abdecken. In erster Linie werden Geschäftsprozesse unterstützt, die das operationale Tagesgeschäft des Unternehmens ausmachen. Diese Production Workflows sind durch hohe Transaktionsraten gekennzeichnet. Überwiegend dispositive und taktische Aufgaben (z. B. das Erstellen einer Marktanalyse), für die oft keine Geschäftsprozesse spezifiziert sind, werden nicht unterstützt. Dies liegt jedoch nicht in den Fähigkeiten der Software-Agenten selbst begründet. Intelligente Agenten an sich können durchaus auch komplexe Sachverhalte, so z. B. komplexe Entscheidungsprozesse und Optimierungsprobleme lösen.

Die strukturierte Architektur mit klaren Aufgabenzuordnungen erleichtert die Spezifikation von Komponenten, die mit Hilfe einer standardisierten Beschreibungssprache erfolgen kann. Damit sind bereits wesentliche Voraussetzungen erfüllt, um sämtliche Elemente von Anwendungen (Dialog-Oberfläche, Anwendungsobjekte und Komponenten) aus der formalen Spezifikation generieren zu können.

Agenten-orientierte Systeme sind eine Evolution, keine Revolution. Um die Investitionen in bestehende Systeme zu schützen, verbietet sich ein revolutionärer Ansatz ohnehin. Neue Technologien müssen behutsam angegangen werden. Die Architektur sollte deshalb schrittweise eingeführt werden. Zunächst sollten ohnehin ohnehin nur wenige Agenten zusammenwirken. Einzelne Agenten können nach und nach entwickelt und in das Gesamtsystem eingebaut werden. Außerdem sollten zunächst kleine und einfache Agenten entwickelt werden. Kleine Agenten sind leichter zu erstellen und zu verstehen. Sie decken Routineaufgaben ab und bewältigen strukturierbare Entscheidungsprozesse mit ausschließlich bekannten Entscheidungsalternativen. Im Einklang mit den gewonnenen Erfahrungen ist eine schrittweise Evolution zu intelligenteren und mächtigeren Agenten sinnvoll, die dann auch komplexe Entscheidungsprozesse übernehmen können.

Mit agenten-orientierten Systemen können einige wichtige Vorteile realisiert werden, so vor allem:

- Keine zentrale Steuerung: Agenten steuern einen Geschäftsvorgang partizipativ. Je intelligenter die beteiligten Agenten sind, desto besser können auch Ausnahmefälle behandelt werden. Einzelne Agenten haben nur begrenztes Wissen. In der Gesamtheit können sie jedoch mehr leisten als nur die Summe der Agenten;
- Weitgehende Entkopplung: die an der Abarbeitung von Geschäftsvorgängen beteiligten Entitäten (Geschäftsprozesse, Agenten, Geschäftsobjekte und Anwendungen) können unabhängig voneinander weiterentwickelt werden. Fernwirkungen bleiben minimal;
- Weniger Engpässe: durch die Möglichkeit zur Verteilung von Software-Agenten auf mehrere Computersysteme und die Möglichkeit, sich selbst zu transferieren, wird die Gefahr von Überlastsituationen auf einzelnen Computersystemen deutlich vermindert. Eine hohe Bandbreite im Netzwerk ist jedoch Voraussetzung;
- Geringere Kosten: Software-Agenten und Geschäftsobjekte können unabhängig voneinander weiterentwickelt werden. Die Wahrscheinlichkeit, daß eine Weiterentwicklung eines Software-Agenten andere Software-Agenten beeinflusst, ist gering. Gleiches trifft sinngemäß auch für Geschäftsobjekte zu. Die logische Abgeschlossenheit der einzelnen Entitäten führt zu einer insgesamt geringeren Kostenbelastung.

Den Vorteilen stehen nur wenige Nachteile gegenüber. Zu nennen sind insbesondere:

- Two-Phase Commit wird erforderlich: Das Zusammenwirken physisch voneinander unabhängiger Einheiten auf potentiell unterschiedlichen Computersystemen erfordert ein Two-Phase Commit-Protokoll. Das Nachrichtenvolumen ist hoch. Die Möglichkeit, daß in Extremsituationen (z. B. physisch unwiederherstellbar zerstörtes Transaction Log auf einem Computersystem) ein systemübergreifend konsistenter und integrier Zustand nicht mehr wiederhergestellt werden kann, ist nicht ausschließbar. Das Risiko kann jedoch mit den heute verfügbaren Möglichkeiten (z. B. RAID-Subsysteme) weitestgehend ausgeschlossen werden;
- Hoher Bandbreitenbedarf: Bedingt durch das hohe Nachrichtenvolumen ist eine hohe Bandbreitenkapazität im Netzwerk erforderlich (möglichst 100 MBit/s.);
- lange Transaktionen sind die Regel, nicht die Ausnahme. Ein Problem ist das Zurückrollen langer Transaktionen. Das genannte Problem ist jedoch nicht spezifisch für das angesprochene Konzept. Nahezu jeder Geschäftsprozeß resultiert in "langen" Transaktionen.

Die vorgeschlagene Architektur verletzt das Prinzip der Kapselung von Daten und Funktionen. Dies ist jedoch aus pragmatischen Gründen vertretbar, da auf diese Weise nicht bekannt sein muß, welches Datenbank-Schema zugrunde liegt und mit welchem Datenhaltungssystem die persistent gespeicherten Daten verwaltet werden. Davon abgesehen, werden auch mit einigen objektorientierten Datenbanken Funktionen (Methoden) und Daten getrennt. Die Trennung von Funktionen und Daten betrifft sinn analog auch die Schnittstelle zum GUI-System.

Solange GUI- und Datenhaltungssysteme über keine einheitliche, standardisierte Schnittstelle angesprochen werden können, ist die Trennung von Funktionen und Daten vertretbar.

Die Abkehr von der zentralen Steuerung und die konsequente Entkopplung führen nicht zu höherer Komplexität, sondern im Gegenteil zu überschaubaren Strukturen und zu Redundanzfreiheit. Das Prinzip der Wiederverwendbarkeit von Design und Code wird nicht berührt und kann uneingeschränkt durchgesetzt werden. Die einzelnen Entitäten können weitestgehend unabhängig voneinander weiterentwickelt werden. Dadurch bedingt gestaltet sich auch das Konfigurations-Management weniger komplex. Im Ergebnis wird eine Kostensenkung im Design- und Entwicklungsprozeß realisierbar sein, deren Umfang jedoch derzeit noch nicht hinreichend abgeschätzt werden kann.