Proc. 25. GI-Jahrestagung und 13. Schweizer Informatikertag GISI 95, Zürich, September 1995

ADEPT: Ein integrierender Ansatz zur Entwicklung flexibler, zuverlässiger kooperierender Assistenzsysteme in klinischen Anwendungsumgebungen*)

P. Dadam[1], K. Kuhn[2], M. Reichert[1], T. Beuter[1], M. Nathe[1]
[1] Fakultät für Informatik, Universität Ulm
[2] Medizinische Universitätsklinik, Universität Ulm

Zusammenfassung

Die Entwicklung flexibler, kooperierender Assistenzsysteme mit der für den klinischen Bereich unerläßlichen Zuverlässigkeit ist auf Basis der heutigen Softwaretechnologie immer noch ein äußerst schwieriges und aufwendiges Unterfangen. In diesem Beitrag wird ein neuer Ansatz vorgestellt, der durch strikte Trennung und Kapselung von Ablauflogik sowie Ausnahme- und Fehlerbehandlung vom eigentlichen Anwendungsprogramm die Komplexität für den Anwendungsentwickler erheblich zu reduzieren vermag.

1. Einleitung

Durch die starke Verbreitung von PCs im beruflichen und privaten Bereich hat sich der Umgang mit dem Rechner in den letzten Jahren sehr stark gewandelt. Die Anwender sind mündiger geworden, und sie sind in zunehmendem Maße in der Lage, die Gestaltung ihres persönlichen DV-Umfeldes selbst in die Hand zu nehmen. Entsprechend dieser Entwicklung erwarten die Anwender heute Anwendungs- und Informationssysteme, die unterstützend wirken (z.B. durch Hinweis auf mögliche Fehler oder auf drohende Terminüberschreitungen), den Benutzer dabei aber nicht gängeln, sondern ihm letztlich die Entscheidung überlassen. Wir wollen Systeme dieser Art im folgenden als „Assistenzsysteme" bezeichnen. Man braucht kein Prophet zu sein, um vorherzusagen, daß sich hierdurch der heute schon erkennbare Trend zur dezentral orientierten Informationsverarbeitung in Zukunft eher noch verstärken wird. Personaleinstellungen im DV-Bereich werden zukünftig überwiegend „vor Ort", also in den Anwendungsbereichen - d.h. in den einzelnen Abteilungen, Kliniken oder Labors im klinischen Umfeld - stattfinden. Diese Bereiche werden hierbei primär anwendungsbezogene Informatikkenntnisse (und weniger systemnahe Kenntnisse) nachfragen.

Dieser Trend ist so lange relativ unproblematisch, wie es um die Entwicklung rein lokaler Anwendungen geht. Nun liegt jedoch die Herausforderung der Zukunft - insbesondere auch im Gesundheitswesen - nicht im Aufbau von lokalen "Informationsinseln", sondern in der Realisierung durchgängiger Gesamtkonzepte, in die sich die lokalen Anwendungen harmonisch einfügen. Unglücklicherweise aber ist die Entwicklung zuverlässiger, kooperierender Anwendungen durch die zusätzlichen Fehlermöglichkeiten in verteilten Systemen erheblich komplexer als die Entwicklung lokaler Anwendungen. Für ihre Realisierung bedarf es - insbesondere auf dem heutigen Stand der Softwaretechnik - sehr systemnaher Kenntnisse, etwa im Bereich der Rechnerkommunikation, der Prozeßsynchronisation, sowie des Logging und Recovery, um wirklich verläßliche verteilte Anwendungen realisieren zu können. Erschwerend kommt hinzu, daß ein durch den Rechner erzwungenes, starres schematisches Vorgehen bei der täglichen Arbeit heutzutage von den Benutzern immer weniger akzeptiert wird. In einigen Anwendungsbereichen - insbesondere im klinischen

---

*) ADEPT steht für **A**pplication **DE**velopment Based on **P**re-Modeled Activity **T**emplates. Die Forschungsarbeiten wurden im Rahmen des Projekts „**O**ffenes **k**linisches Datenbank- und **I**nformationssystem zur Integration autonomer **S**ubsysteme" (OKIS) als Forschungsschwerpunkt durch das Land Baden-Württemberg gefördert.

Umfeld mit dem „Unsicherheitsfaktor Patient" - ist dies schlichtweg nicht praktikabel. Die Entwicklung von Programmen, die in sehr flexibler Form Ausnahmen vom vorgeplanten Ablauf zulassen, ist programmtechnisch jedoch ebenfalls erheblich aufwendiger (und damit fehleranfälliger) als die Realisierung konventioneller Lösungen. Angesichts der oben beschriebenen Entwicklung wird es unseres Erachtens deshalb nur dann einen „Quantensprung" in der Entwicklung (verläßlicher) kooperierender, verteilter Anwendungen geben, wenn es gelingt, die Anwendungsentwicklung weitgehend von diesen systemnahen Aspekten sowie der Behandlung von Fehler- und Ausnahmefällen zu entlasten.

In den letzten Jahren sind erfreulicherweise eine Reihe von Technologien entwickelt worden, die, geschickt kombiniert, helfen können, auf diesem Weg einen großen Schritt voran zu kommen. Im Bereich der sog. *Workflow-Management-Systeme* [DeGr92, LeAl94, Rein93] ist das Konzept entwickelt worden, Daten- und Kontrollfluß durch das (verteilte) System separat vom eigentlichen Anwendungscode zu modellieren und durch Visualisierung und Animation zu überprüfen. Im Bereich *erweiterter Transaktionskonzepte* (siehe [Elma92] für einen Überblick) wurde vorgeschlagen, neben dem klassischen Rollback auch "Kompensation" (z.B. in Form einer "Stornobuchung") systemseitig zu unterstützen. Im Bereich der *Programmentwicklung* gibt es schon lange den Traum wiederverwendbarer Programmbausteine (siehe [Krue92]). Erst kürzlich ist die Idee wieder aufgegriffen worden, Programme durch Zusammenfügen vorgefertigter Bausteine zu entwickeln [MKSD90,NGT92,NTMS91,CKK93]; es gibt sogar erste Ankündigungen von Produkten, denen eine ähnliche Philosophie zugrundeliegt [Baum94].

In das von uns im folgenden vorgestellte Konzept sind Aspekte aus allen diesen Bereichen eingeflossen. Im Zentrum unseres Vorgehens steht das sog. „*Activity Template*", das - ähnlich wie bei einem Workflow - einen gesamten Ablauf bzw. Prozeß (wie z.B. Anordnen und Durchführung einer Untersuchung mit allen vor- und nachbereitenden Maßnahmen) beschreibt. In einem solchen Template werden ggf. auch bereits erwartete Ausnahmen vom normalen Ablauf vormodelliert. Hierbei wird festgelegt, welche Kompensationsschritte im Fehlerfall ausgeführt und welche Teilschritte durch das System zeitlich überwacht werden sollen. Die Teilschritte eines Templates sind Platzhalter für die eigentlichen Programmbausteine („Services"). Bei der Modellierung des Templates werden lediglich die Typen der zulässigen Services (z.B. „Termin vereinbaren", „Patient abrufen", etc.) und die Mindestanforderungen an ihre Ein- und Ausgabeparameter festgelegt. Ein ablauffähiges Programm („Aktivitätenimplementierung") erhält man dann durch Einsetzen von Services in das Template. Aus Sicht einer Endanwendung sind Aktivitätenimplementierungen vollständig gekapselt und können, unabhängig von ihrer konkreten Ausprägung, in objektorientierter Weise über einheitliche Methoden manipuliert werden.

Im folgenden Abschnitt wird anhand eines Anwendungsbeispiels illustriert, welche Anforderungen an ein klinisches Assistenzsystem zur Unterstützung medizinisch-organisatorischer Maßnahmen gestellt werden. In Abschnitt 3 wird dann auf die verschiedenen Phasen der Programmentwicklung im ADEPT-Ansatz und in Abschnitt 4 auf die erforderliche Laufzeitunterstützung eingegangen. In der abschließenden Diskussion in Abschnitt 5 gehen wir auf „related work" und in Abschnitt 6 auf den Stand der Implementierung sowie unsere weiteren Planungen ein.

2. Anwendungsbeispiel und Motivation

Die Komplexität klinischer Aktivitäten läßt sich am besten anhand des Ablaufs einer Untersuchung illustrieren. Typischerweise kooperieren hier verschiedene, organisatorisch weitgehend unabhängige Einheiten (Station, Radiologie, Labor etc.) sowie verschiedene Personalgruppen (Arzt, Pflege) in teilweise sehr komplexer Weise.

Nachdem vom Arzt auf einer Station oder in einer Ambulanz eine Untersuchung angeordnet wurde, ist häufig eine Terminvereinbarung mit der untersuchenden Stelle erforderlich. Von seiten der anfordernden Stelle können für einen Patienten mehrere Untersuchungen in einer bestimmten Reihenfolge gewünscht sein. Auf der Seite der Untersuchungsstelle kann die Terminvergabe für Standarduntersuchungen unkompliziert und damit auch leicht automatisierbar sein. Es gibt aber auch den Fall, daß für aufwendige und invasive Untersuchungen spezielle Geräte, erfahrene Untersucher und sogar Ärzteteams bereitgestellt werden müssen. Die untersuchende Stelle benötigt deshalb patientenbezogene Daten, i.a. zumindest den Grund für die Untersuchung (die Indikation); häufig ist sogar die Terminvergabe von dieser Indikation und ihrer Dringlichkeit abhängig. Hieraus können sich komplizierte Interaktionen, Nachfragen und Verschiebungen ergeben, die heute mit Hilfe langwieriger Telefonate ablaufen. Eine Studie zu den Informationsschwachstellen einer Medizinischen Klinik hat gezeigt, daß Probleme mit Terminvereinbarungen, Terminreihenfolgen und Terminverschiebungen von Ärzten und Schwestern als besonders zeitraubend eingestuft werden [KRKK94]. Die Situation wird noch wesentlich komplizierter, wenn für bestimmte Untersuchungen Vorbedingungen erfüllt sein müssen, die ihrerseits Untersuchungen erforderlich machen.

Am Untersuchungstag wird der Patient evtl. noch auf der Station vorbereitet, nach einiger Wartezeit in die Funktionseinheit transportiert, und schließlich (möglicherweise nach erneuter Wartezeit) untersucht. Nach seiner Rückkehr auf die Station erwartet diese einen Befund oder wenigstens einen Kurzbefund, von dem das weitere Vorgehen abhängt. Der Arzt muß hier aus einer Flut von rücklaufenden Befunden eine Zuordnung zu den Problemen des Patienten treffen und weitere Anordnungen vornehmen. Schwierigkeiten sind vorprogrammiert: Bei Nichteintreffen eines Befundes erhält der Arzt primär keinerlei Information. Erinnert er sich an seinen Auftrag und fragt nach, so kann es recht aufwendig sein, festzustellen, ob die Untersuchung gar nicht durchgeführt, nur kein Befund geschrieben oder dieser verloren bzw. an einen falschen Bestimmungsort gegangen ist. Diese Mischung aus organisatorischen Problemen und ständig neu eintreffenden medizinischen Daten bringt den Arzt in eine Situation der informationellen Überbelastung, zwingt ihn zu ständigen „Context Switches" und erhöht letztlich die Gefahr von Fehlentscheidungen.

Hier kann ein Assistenzsystem wesentliche Hilfe leisten, indem es von der Überwachung organisatorischer Abläufe befreit und die Terminvergabe unterstützt, daneben können auch Warnhinweise bei häufigen und typischen medizinischen Problemen erstellt werden. Bei einem solchen System ist nicht nur absolute Zuverlässigkeit, sondern vor allem auch hohe Flexibilität gefordert. So muß es jederzeit möglich sein, vom üblichen Ablauf abzuweichen, und beispielsweise einen Patienten ohne elektronische Anmeldung direkt und notfallmäßig einer Untersuchungseinheit zuzuführen. Häufige Ausnahmen sind Überspringen einzelner Schritte im Ablauf, Abbruch und Wiederaufsetzen.

3. Programmieren im ADEPT-Modell

Die Umsetzung des im vorangegangenen Abschnitt motivierten Assistenzgedankens in praxistaugliche Lösungen hängt entscheidend davon ab, welcher Entwicklungsaufwand auf seiten der Anwendungsprogrammierung getrieben werden muß, um die geforderte Zuverlässigkeit und Flexibilität zu erzielen. Der Ansatzpunkt des ADEPT-Ansatzes ist, die Komplexität der Gesamtaufgabe durch Zerlegung in Teilaufgaben, die sich jeweils nur auf einen bestimmten Teilaspekt konzentrieren und damit überschaubar bleiben, zu reduzieren. Die Programmentwicklung im ADEPT-Modell verläuft typischerweise in den folgenden Phasen, die aus Platzgründen hier leider nur skizziert werden können:

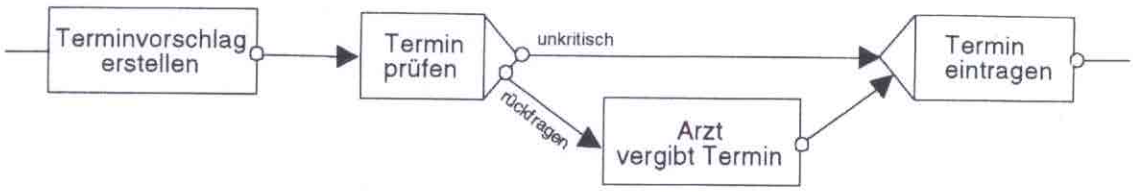


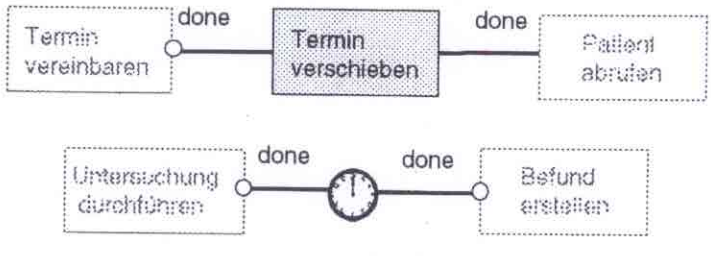
Abb. 1: Beispiel für eine „1 aus 2"- Auswahl bei einer halbautomatischen Terminvereinbarung

1. Zunächst wird ein geeignetes Ablaufmodell („Activity Template") erstellt bzw. aus dem Repository ausgewählt. Dieses Activity Template beschreibt - ähnlich wie bei prozeßorientierten Workflow-Management-Systemen, wie z.B. FlowMark [LeAl94] oder K/2r [Adom92], - den geplanten Ablauf (Kontroll- und Datenfluß) durch Aktivierungskanten, Verzweigungs- und Teilschrittknoten (siehe Abb. 1). Teilschritte (die durch die Teilschrittknoten repräsentiert werden) werden in dieser Phase nur durch ihre Schnittstellen beschrieben.

2. Im nächsten Schritt wird das Activity Template durch Angabe von bedingt wählbaren (Ausnahme-)Aktionen (siehe Abb. 2.a), von Zeitrestriktionen (siehe Abb. 2.b), von vorgeplanten Ausnahmen, wie Überspringen von Teilschritten (shortcuts) und Rücksprünge zur Wiederholung von Teilabläufen, ergänzt (siehe Abb. 3). - Nach diesem Schritt liegen der gesamte Standardablauf, die vormodellierten Ausnahmen, die bedingt wählbaren (Ausnahme-)Aktionen sowie die Zeitvorgaben für die Ausführungsdauer (für die Selbstüberwachung des Systems) fest.

3. Die im zweiten Schritt vormodellierten Rücksprünge haben, falls sie zur Laufzeit ausgewählt werden, zur Folge, daß der laufende Teilschritt abgebrochen und dieser sowie alle vorangegangenen Teilschritte - bis zum Rücksprungknoten - systemseitig wieder rückgängig gemacht werden (durch „Kompensation", d.h. etwa im Sinne einer „Stornierung"). Um dem System anzuzeigen, bis zu welchem Teilschritt diese (einfache) Kompensation noch möglich ist und ab wann nicht mehr, werden im dritten Schritt sog. „Kompensationssphären" eingeführt (siehe Abb. 3).

Mit den Schritten 1 bis 3 ist jetzt die gesamte Ablauflogik festgelegt. Außerdem wurden die Voraussetzungen dafür geschaffen, daß die Ausführung der Aktivität (z.B. die Einhaltung von Zeitschranken) systemseitig überwacht sowie die systemseitige Behandlung von Ausnahmefällen ermöglicht wird.



a) bedingt wählbare (Ausnahme-)Aktion: "Termin verschieben" ist nach Abschluß von "Termin vereinbaren" und vor Eintritt von "Patient abrufen" wählbar

b) Zeitdauer-Überwachung: Maximale Zeitdauer, die zwischen der Bearbeitung von "Untersuchung durchführen" und "Befund erstellen" liegen darf

Abb. 2: Angabe prädikativer Bedingungen

4. Im nächsten Schritt werden nun die einzelnen Teilschritte als eigenständige Programme oder als Schnittstellen zu existierenden Programmmoduln gemäß den durch das Activity Template vorgegebenen Schnittstellen (Input-/Output- „Parameter") implementiert. Diese Programmbausteine („Services") sind - bis auf die Parameterversorgung - vollständig

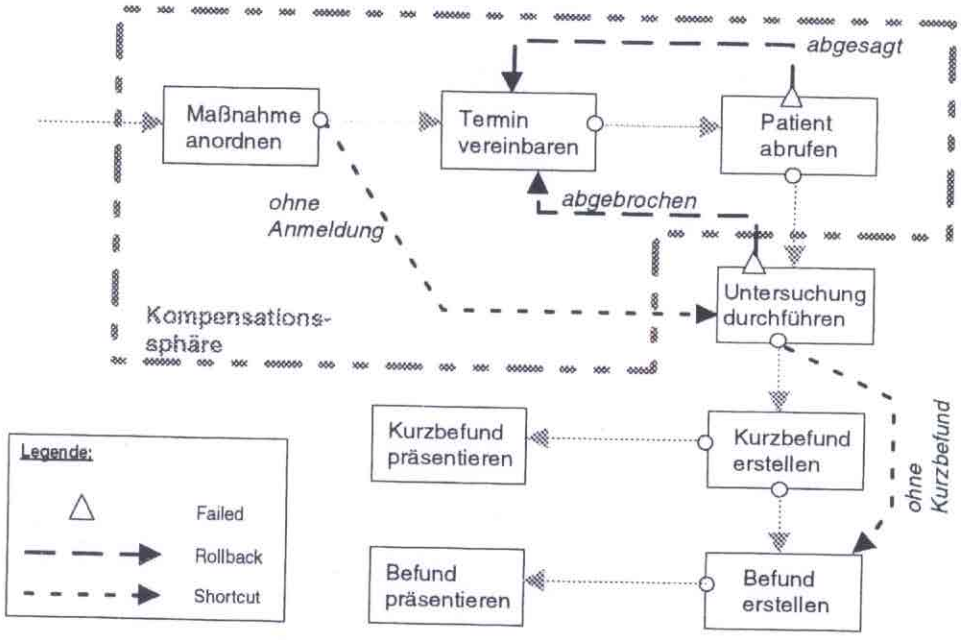


Abb. 3: Shortcuts, Rücksprünge und Kompensationssphären (vereinfacht) — Die Kompensationssphäre besagt, daß nach Beginn des Teilschritts „Untersuchung durchführen" (aus rechtlichen Gründen) keiner der vorangegangenen Teilschritte mehr kompensiert werden darf.

eigenständig ausführbar, d.h. machen auch ihre eigene Ein- und Ausgabe (ggf. unter Verwendung von bereitgestellten Methoden). Dies erleichtert das isolierte Testen in ganz erheblichem Umfang. — Zu jedem Service muß zwingend angegeben werden, welcher Service mit welcher Parameterversorgung im Kompensationsfall gerufen werden soll. Dies kann der Service selbst (mit z.B. den „inversen" Parametern), ein anderer Service oder auch die „Null-Kompensation" (falls keine Kompensation des Services möglich oder sinnvoll ist) sein.

5. Sind alle Bausteine implementiert, werden diese in das Activity Template „eingesetzt" (wodurch das Template zu einer „Aktivitätenimplementierung" wird) sowie die Input- und Output-Parameter geeignet verbunden. Um nicht zu viele Templates zu bekommen, die sich nur durch jeweils geringfügige Abweichungen in den Schnittstellenbeschreibungen unterscheiden, streben wir an, in den Activity Templates nur eine Art von Minimalvorgabe hinsichtlich der Input-/Output-Parameter zu machen. Die aktuell eingesetzten Services können dann ggf. weitere Parameter aufweisen. Die Verknüpfung der Parameter untereinander - im Sinne des Kontroll- und Datenflusses - soll dabei durch geeignete Entwicklungswerkzeuge graphisch unterstützt werden. (Durch Verwendung eines kontrollierten Vokabulars für die Schnittstellenbeschreibung soll dieser Prozeß dabei zudem teilweise automatisiert werden.)

Sind alle Programmbausteine in das Template eingesetzt und alle Input-/Output-Parameter geeignet miteinander verbunden, so ist die Aktivitätenimplementierung - von außen gesehen - ein ablauffähgies Programm, das mittels einer Standardmethode mit den erforderlichen Startparameterwerten beim Aufruf versorgt werden muß und das seine Ergebnisse (sofern eine Werterückgabe erfolgen soll) mittels einer anderen Standardmethode wieder an die Aufrufumgebung abliefert.

Um verschiedene Aktivitätenimplementierungen (die ggf. dann auch sehr verschiedene Dinge tun) in einheitlicher Form ansprechen und in ein darüberliegendes Anwendungsprogramm einbinden zu können, das dann in einem gewissem Sinne nur noch eine „Anwendungsschale" (application shell) ist, werden alle Aktivitätenimplementierungen mit

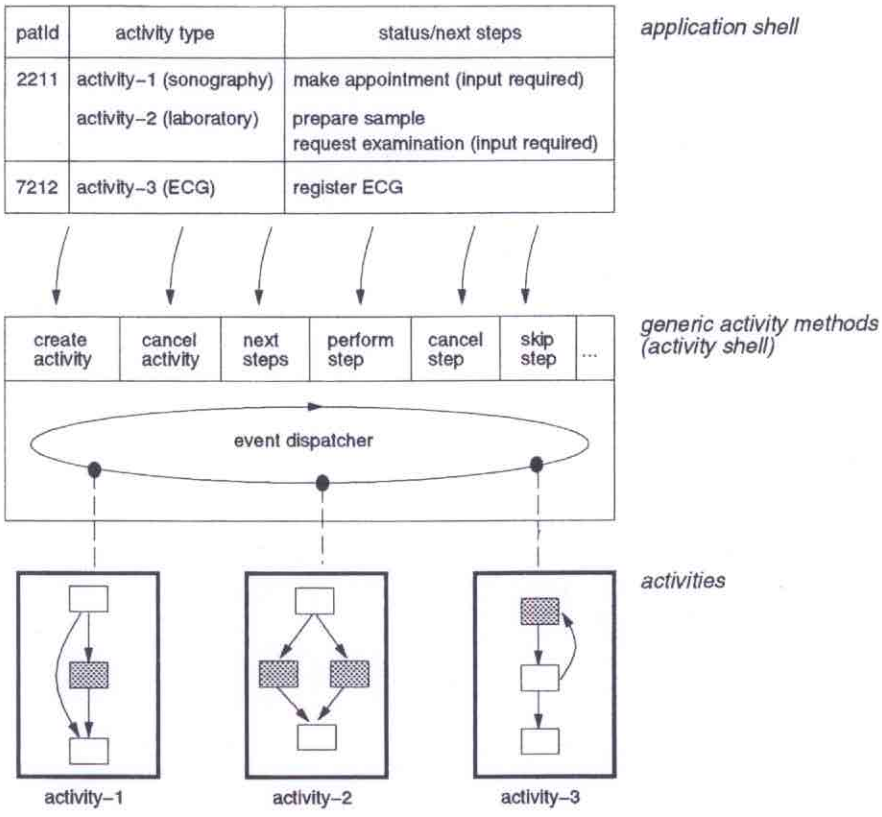


Abb. 4: Zusammenspiel von Anwendungsschale und Aktivitäten

den gleichen generischen Methoden angesprochen. Diese Methoden dienen z.B. zum Erzeugen einer neuen Aktivitäteninstanz, zum Abfragen des aktuellen Status, zum Erfragen der wählbaren Schritte, zum Anstoßen oder Abbrechen eines Teilschritts. Die Hauptaufgabe der Anwendungsschale besteht im wesentlichen im Verwalten und geeigneten Reagieren auf Ereignisse, wie z.B. „Teilschritt beendet", „Teilschritt wartet auf Eingabe" oder „Zeitüberschreitung" (siehe Abb. 4).

4. Laufzeitumgebung

Die Ausführung der in den Aktivitätenimplementierungen gekapselten Services, unter Beachtung der durch das Activity Template vorgegebenen Reihenfolgen, erfolgt durch die in Abb. 5 dargestellte Laufzeitumgebung, die Parallelen zu anderen Ansätzen [Schw93,MWFF92,Sher93] aufweist. An jedem Knoten ist ein *Agent* für die Durchführung von Aktivitäten zuständig. Da eine Aktivität in der Regel über mehrere verschiedene Rechner verteilt bearbeitet wird, überwachen und koordinieren die Agenten der beteiligten Rechner den Ablauf der Aktivität gemeinsam. Der jeweils lokale Agent dient dabei der Anwendung als Ansprechpartner. Ein Agent umfaßt Komponenten für die Steuerung des Kontroll- und Datenflusses, für das Monitoring sowie für die Recovery. Für das Ausführen von Services bedient sich der Agent des lokalen *Task Managers*. Im nicht-lokalen Fall leitet dieser den Auftrag an den Task Manager des betroffenen Knotens weiter [vgl. Kuhn94].

Der Ablaufgraph und der gegenwärtige Zustand einer Aktivität wird von den *Cooperation Managern* verwaltet. Diese bieten ihren jeweiligen Anwendungen die gerade aktivierbaren Schritte an und führen sie in Abstimmung mit den anderen Cooperation Managern durch, sofern dies konform zum Ablaufgraphen ist.

Die persistente Verwaltung des Datenkontexts einer Aktivität und die Bereitstellung der notwendigen Aufrufparameter für einen Service erfolgt durch die *Data Manager*. Stellt ein Data Manager (z.B. infolge eines vom Benutzer erzwungenen, nicht vormodellierten

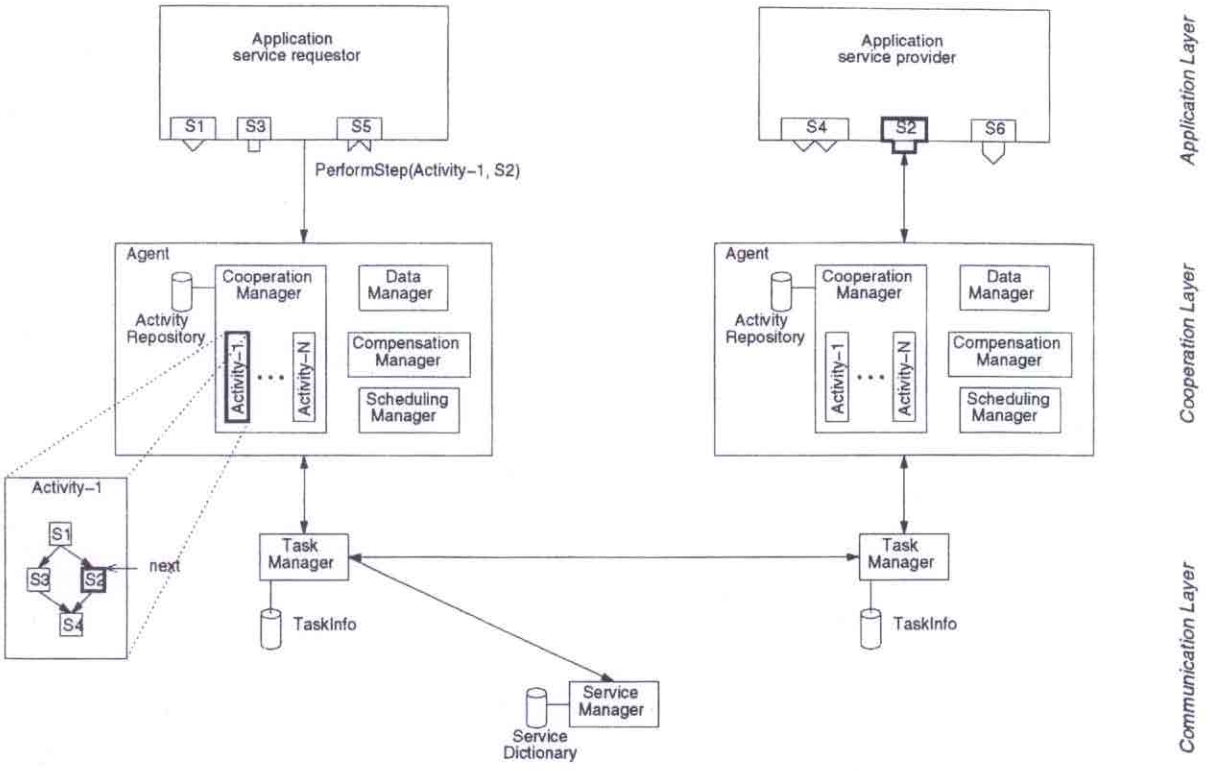


Abb. 5: Komponenten der Laufzeitumgebung

Zustandsübergangs) fest, daß noch zwingend erforderliche Eingabeparameter unversorgt sind, so fordert er diese vom Anwender selbständig nach.

Die *Scheduling Manager* verwalten die Zeitrestriktionen von Aktivitäten und melden Zeitüberschreitungen an den dafür zuständigen Cooperation Manager, der dann ein entsprechendes Ereignis für die Anwendungsschale erzeugt. Für das (teilweise) Zurücksetzen einer Aktivität (Backward/Forward-Recovery) sind die *Compensation Manager* an den jeweils betroffenen Knoten zuständig. Die Compensation Manager protokollieren für alle Aktivitäten jeweils deren aktuellen Weg durch ihren Ablaufgraphen, um im Kompensationsfall die erforderlichen Kompensationsschritte bestimmen zu können.

5. Diskussion

Wie schon in der Einleitung ausgeführt, flossen in unser Modell Konzepte aus verschiedenen Technologien ein. Die Idee, die komplexe Ablauflogik einer Anwendung von dem eigentlichen Anwendungs-Code zu trennen und auf einer abstrakten Ebene zu beschreiben, haben wir aus dem Bereich des Workflow-Managements übernommen [DeGr92,GaWu93,LeAl94,Rein93] und hinsichtlich einer systemseitigen Unterstützung von Ausnahmen und Fehlern adaptiert. Dazu wurde bereits in einigen Arbeiten vorgeschlagen, erweiterte Transaktionskonzepte in Workflow-Konzepte zu integrieren [Brei93,DEB93, Leym95,VoEr92]. Durch die Einführung von Kompensationssphären geht unser Ansatz dabei bzgl. Recovery-Aspekten über die bisher vorgeschlagenen erweiterten Transaktionskonzepte [Elma92] hinaus. Diese Sphären sind notwendig, weil sich die Kompensierfähigkeit von Teilschritten im Laufe einer Aktivität ändern kann. Ein vergleichbarer Ansatz wurde jüngst in [Leym95] für den Bereich der Geschäftsprozesse vorgeschlagen.

Ein im Bereich der Programmentwicklung verbreiteter Ansatz ist die Entwicklung einer Anwendung durch Verknüpfung vorgefertigter Software-Bausteine. Entsprechende Ansätze wurden z.B. in den Projekten REX [MKSD90] und ITHACA [NTMS91,NGT92] unter-

nommen. Die Idee verknüpfbarer Software-Bausteine wurde von uns im Zusammenhang mit der Konfiguration von Activity Templates aufgegriffen. Im Gegensatz zu den genannten Projekten, wo die Konfiguration einer Anwendung durch Verknüpfung von Bausteinen vollständig im Aufgabenbereich des Anwendungsprogrammierers liegt, versuchen wir den Programmierer durch Bereitstellung vormodellierter Templates sowie durch eine (weitgehend) systemseitige Verknüpfung der Ein- und Ausgabeparameter der eingesetzten Bausteine zu unterstützen.

Auf Seiten der klinischen Anwender ist seit einigen Jahren ein zunehmendes Interesse an einer Integration verteilter Anwendungen [Flec95,EJD92,MuTi94] sowie an einer computerbasierten Unterstützung medizinisch-organisatorischer Abläufe zu beobachten [GaWu93, Fran94,FFW94, PSS94]. Im Projekt RICHE [Fran94,RICH92] wurden typische Anforderungen im Zusammenhang mit der Unterstützung solcher Abläufe (*Act Management*) festgelegt. Das Projekt EDITH [FFW94] greift diese Arbeiten auf und schlägt eine Architektur für eine rechnerbasierte Ablaufunterstützung im Krankenhaus vor, deren Kern das *Distributed Hospital Environment* (DHE) bildet. DHE umfaßt u.a. Services für das (zentrale) Management von Aktivitäten und den Informationsaustausch zwischen Anwendungen verschiedener Hersteller. RICHE und EDITH machen jedoch bisher keine Aussagen darüber, wie die zu unterstützenden Abläufe formal beschrieben werden und wie Anforderungen hinsichtlich Flexibilität und Zuverlässigkeit technisch umgesetzt werden sollen.

Im Projekt Helios [EJD92] wurde eine integrierte Umgebung für die Entwicklung medizinischer Anwendungen entworfen. Ein Anwendungsprogrammierer kann dabei auf Services zurückgreifen, die von anderen Stellen über einen Software-Bus angeboten werden. Während sich Helios auf Aspekte der Wiederverwendbarkeit und auf die Unterstützung des kompletten Softwarelebenszyklus konzentriert, liegt unser Schwerpunkt mehr darin, den Anwendungsprogrammierer von systemnahen Aspekten bei der Entwicklung verteilter Anwendungen zu entlasten und die Fehler- und Ausnahmebehandlung weitgehend dem Laufzeitsystem zu überlassen. Im Projekt HERMES [MuTi94] wird ebenfalls ein Integrationsansatz unternommen, dessen Schwerpunkt in der Kapselung von *Legacy Systems* und der Einbindung kommerzieller Software-Komponenten liegt.

Der von uns verfolgte Ansatz stellt somit eine Ergänzung zu den genannten Projekten dar, insbesondere im Hinblick auf die zu behandelnden technischen Probleme, die sich aus der dezentralen Entwicklung verteilter, kooperativer Anwendungen ergeben. Die in den verschiedenen Arbeiten genannten Anforderungen im Hinblick auf die Flexibilität und Zuverlässigkeit solcher Systeme sind in unseren Ansatz mit eingeflossen.

6. Zusammenfassung und Ausblick

Die vorliegende Arbeit entstand im Rahmen des vom Land Baden-Württemberg geförderten Forschungsschwerpunkts OKIS [Kuhn94,Kuhn94a]. Neben den oben angesprochenen Themen werden, im Zusammenhang mit der Entwicklung verteilter, kooperativer Anwendungen, in diesem Projekt auch Aspekte der Autorisierung und Authentifizierung in offenen, verteilten Umgebungen sowie das Problem der Softwaresicherheit bei dezentraler Entwicklung von Software-Komponenten behandelt. Einen weiteren Schwerpunkt bildet die Integration von Untersuchungsrichtlinien in ärztliche Arbeitsplatzsysteme [HKD94].

Ein erster, noch sehr rudimentärer Prototyp wurde auf der Basis von OSF/DCE realisiert, um das Zusammenspiel der wesentlichen Komponenten des Laufzeitsystems besser verstehen zu lernen. Auf Anwendungsseite wurden hierzu Demonstratoren für den Bereich "Klinischer Workflow" (basierend auf dem Kernel 2/R System [DeGr92, Adom92]), für ein integriertes (wissensbasiertes) Modul zur Vermittlung klinischer Richtlinien [HKD94], sowie für Endbenutzerschnittstellen [Dein94] erstellt. Derzeit findet eine Übertragung der Architek-

turkonzepte des Laufzeitsystems (Agenten, Task Manager, Service Manager) auf das Werkzeug FlowMark [LeAl94] mit dem Ziel einer Überprüfung des dynamischen Verhaltens statt. Außerdem werden sukzessive einzelne Komponenten implementiert. Weiterführende Forschungsarbeiten sind in Richtung Verbundaktivitäten (d.h. Aktivitäten, die einzelne Teilschritte gemeinsam haben, oder Aktivitäten, für die Reihenfolgen beachtet werden müssen) sowie für den Bereich dynamisch erzeugter Aktivitäten geplant.

Literatur

Adom92 Adomeit, R.; Deiters, W.; Holtkamp, B.; Schülke, F.; Weber, H.: K/2R: A Kernel for the ESF Software Factory Support Environment. Proc. 2nd Int'l Conf on System Integration, Morristown, NJ, June 1992, 325-336

Baum94 Baumeister, J.: AppWareBus und Visual AppBuilder. PC Magazin, 43, Okt 1994, 42-45

Brei93 Breitbart, Y.; Deacon, A.; Schek, H.-J.; Sheth, A.; Weikum, G.: Merging Application-centric and Data-centric Approaches to Support Transaction-oriented Multi-system Workflows. ACM SIGMOD Record, 22(3), 1993, 23-30

CKK93 Convent, B.; Kohaut, T.; Kulins, S.: The ReUse Class Browser, IBM Wissenschaftliches Zentrum Heidelberg, Technical Report TR 75.93.03, July 1993

DEB93 Data Engineering Bulletin, Special Issue on Workflow and Extended Transaction Systems, 16 (2), 1993

DeGr92 Deiters, W.; Gruhn, V.: The FUNSOFT Net Appoach to Software Process Management. FhG/ISST Dortmund, Bericht 2/92, November 1992

Dein94 Deininger, R.: Konzepte und Werkzeuge für eine flexible Präsentation medizinischer Daten, Diplomarbeit, Universität Ulm, Abt. Datenbanken und Informationssysteme, 1994

EJD92 Engelmann, U.; Jean, F.C.; Degoulet, P. (eds.): The Helios Software Engineering Environment. Computer Methods and Programs in Biomedicine, 45, Supplement, 1994

Elma92 Elmagarmid, A.K. (ed.):. Database Transaction Models for Advanced Applications. Morgan Kaufmann Publishers, San Mateo, CA, 1992.

FFW94 Ferrara, F.; Fossey, T.; van der Werff, A.: Technosynthesis in Hospital Information Systems. Proc. 12 Int'l Congr. Europ. Federation Med. Informatics (MIE'94), Lisbon, May 1994, 370- 373

Flec95 Fleck, E. (ed.): Open Systems in Medicine.Amsterdam: IOS Press, Amsterdam, 1995

Fran94 Frandji, B.; Schot, J.; Joubert, M.; Soady, I.; Kilsdonk, A.: The RICHE Reference Architecture. Med Inform, 19 (1), 1994, 1-11

GaSa87 Garcia-Molina, H.; Salem, K.: Sagas, Proc. ACM Sigmod Conf., San Francisco, May 1987, 249-259

GaWu93 Gangopadhyay, D.; Wu, P.Y.F.: An Object-Oriented Approach to Medical Process Automation. Proc 17th SCAMC, C. Safran (ed.), Washington, DC, Nov 93, McGraw-Hill, New York, 1993, 507-511

HKD94 Heinlein, C.; Kuhn, K.; Dadam, P.: Representation of Medical Guidelines Using a Classification-Based System. Proc. Third Int'l Conference on Information and Knowledge Management (CIKM), Gaithersburg, Maryland, Nov/Dec 1994, 415-422

KRKK94 Kuhn, K.; Reichert, M.; Käfer, R.; Köhler, C.O.: Situations- und Schwachstellenanalyse der Informationsverarbeitung in einer Universitätsklinik, 1994 (zur Veröffentlichung eingereicht)

Krue92 Krueger, Ch.W.: Software Reuse, ACM Comp. Surveys, 24 (2), 1992, 131-184

Kuhn94 Kuhn, K.; Reichert, M.; Nathe, M.; Beuter, T.; Dadam, P.: An Infrastructure for Cooperation and Communication in an Advanced Clinical Information System. Proc 18th SCAMC '94, Ozbolt, J. (ed.), JAMIA 1, 1994, Symposium Supplement, 519-523

Kuhn94a Kuhn, K; Reichert, M.; Nathe, M.; Beuter, T.; Heinlein, C.; Dadam, P.: A Conceptual Approach to an Open Hospital Information System. Proc. 12th Int'l Congr. Europ. Federation Med. Informatics (MIE'94), Lisbon, May 1994, 374-378

LeAl94 Leymann, F.; Altenhuber, W.: Managing Business Processes as an Information Resource. IBM Systems Journal, 33 (2), 1994, 326-348

Leym95 Leymann, F.: Supporting Business Transactions via Partial Backward Recovery in Workflow Management Systems. To appear: Proc. BTW 95, Dresden, Germany, March 1995

MKSD90 Magee, J.; Kramer, J.; Sloman, M.; Dulay, N.: An Overview of the REX Software Architecture. Proc. 2nd Workshop on Future Trends of Distributed Computing, Kairo, 1990, 396-402

MWFF92 Medina-Mora, R.; Winograd, T.; Flores, R., Flores, F.: The Action Workflow Approach To Workflow Management Technology, Proc. CSCW 92, Toronto, Canada, Oct./Nov. 1992, 281-288

MuTi94 van Mulligen, E.; Timmers, T.: Beyond Client and Servers. Proc. 18th SCAMC, Ozbolt, J. (ed.), JAMIA 1, 1994, Symposium Supplement, 546-550

NGT92 Nierstrasz, O.; Gibbs, S.; Tsichritzis, D.: Component-oriented Software Development, Comm. of the ACM, 35 (9), 1992, 160-165

NTMS91 Nierstrasz, O.; Tsichritzis, D.; de Mey, V.; Stadelmann, M.: Object + Scripts = Applications. Proc. Esprit 1991 Conf., Dordrecht NL, Kluwer Academic Publ, 1991, 534-552

PSS94 Patil, R.S.; Silva, J.S.; Swartout, W.R.: An Architecture for a Health Care Provider's Workstation. Int'l Journal of Bio-Medical Comp, 34, 1994, 285-299

Rein93 Reinwald, B.: Workflow-Management in verteilten Systemen. Stuttgart, Teubner, 1993

RICH92 The RICHE Consortium: RICHE Final Report. Louveciennes, France, 1992

Schw93 Schwenkreis, F.: APRICOTS - A Prototype Implementation of a ConTract System - Management of the Control Flow and the Communication System. Proc. 12th Symp on Reliable Distributed Systems, Princeton, NJ, 1993.

Sher93 Sherman, M.: Architecture of the Encina Distributed Transaction Processing Family. Proc. ACM SIGMOD Conf., Washington, DC, May 1993; SIGMOD Record, 22 (3), 1993, 460-463.

VoEr92 Vogel, P.; Erfle, R.: Backtracking Office Procedures. Proc. DEXA 92, Valencia, Spain, 1992, 506-511