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 diese 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. 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 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 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

*) ADEPT steht für **A**pplication **D**evelopment Based on **E**ncapsulated **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.

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]; 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. „*Encapsulated Pre-Modeled Activity Template*" (EPAT), 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 diesem EPAT 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 im EPAT sind Platzhalter für die eigentlichen Programmbausteine („Services"). Bei der Modellierung des EPAT 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 erhält man dann durch Einsetzen von implementierten Services in das EPAT. Aus Sicht einer Endanwendung sind Aktivitäten vollständig gekapselt und können, unabhängig von ihrer konkreten Ausprägung, in objekt-orientierter 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 selbst 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 von Anwenderseite getrieben werden muß, um die geforderte Zuverlässigkeit und Flexibilität zu erzielen. Wie in der Einleitung erwähnt, besteht unser Ansatz darin, ein Anwendungsprogramm in mehrere Bestandteile zu zerlegen, die separat entwickelt und getestet werden.

Dem Anwendungsprogrammierer werden vorgefertigte, wiederverwendbare Templates (EPATs) bereitgestellt, welche die komplexe Ablauflogik medizinisch-organisatorischer Abläufe beschreiben, und aus denen durch Konfiguration und Einsetzen von implementierten Services fertige Aktivitäten-Implementierungen resultieren. Auf diese Weise können Anwendungen realisiert werden, die zum einen dem Endbenutzer die geforderte flexible Unterstützung anbieten, zum anderen den Anwendungsprogrammierer „vor Ort“ aber nicht mit den systemnahen Details einer Aktivitäten-Implementierung belasten. Die Fortschritte in der Bearbeitung einer Aktivität, das Monitoring einzelner Teilschritte sowie sinnvolle Reaktionen auf auftretende Fehler- und Ausnahmefälle sind weitgehend durch die Laufzeitumgebung auf Basis des zugrundeliegenden EPATs zu bewerkstelligen.

Da Aktivitäten-Implementierungen vollständige (Teil-)Programme sind (einschl. evtl. erforderlicher Ein- und Ausgabeoperationen), ändert sich natürlich die Entwicklung einer Endanwendung, wie z.B. die Implementierung eines Stationsarbeitsplatzsystems, in signifikanter Weise. Wenn die Aktivitäten-Implementierungen, gemäß den anzubietenden Funktionen, konfiguriert und adaptiert worden sind, so beschränkt sich die Implementierung der Endanwendung auf das Erzeugen (create activity) und Verwalten von Aktivitäts-Instanzen („Aktivitätenobjekte“), auf das Erfragen der aktuell angebotenen Schritte eines Aktivitätenobjektes (next steps?), das Anstoßen eines Schrittes (perform step) etc., auf die Entgegennahme von Nachrichten über Statusänderungen in Aktivitätsobjekten (einschl. Alarmen) sowie auf die geeignete Präsentation dieser Verwaltungs- und Statusinformationen auf dem Bildschirm. Somit reduziert sich die Entwicklung einer Endanwendung auf die Implementierung einer Anwendungs-„Schale“ (*application shell*), die sich nur noch mit der Verwaltung von Aktivitätenobjekten sowie der Behandlung eintreffender Meldungen über Zustandsveränderungen oder aufgetretene Fehler befaßt. Alles andere wird in den Aktivitätenobjekten selbst bzw. durch das zugrundeliegende Laufzeitsystem durchgeführt. - Das Zusammenwirken von Anwendungsschale und Aktivitätenobjekten ist in Abb. 1 illustriert.

Alle Aktivitätenobjekte verhalten sich, unabhängig vom zugrundeliegenden EPAT-Typ, der Anwendungsschale gegenüber völlig gleich. Alle sind mit denselben Methoden zu manipulieren und verhalten sich auch hinsichtlich Status- und sonstiger Meldungen an die Anwendungsschale identisch. Dasselbe gilt auch für die einzelnen Teilschritte innerhalb der Aktivitätenobjekte.

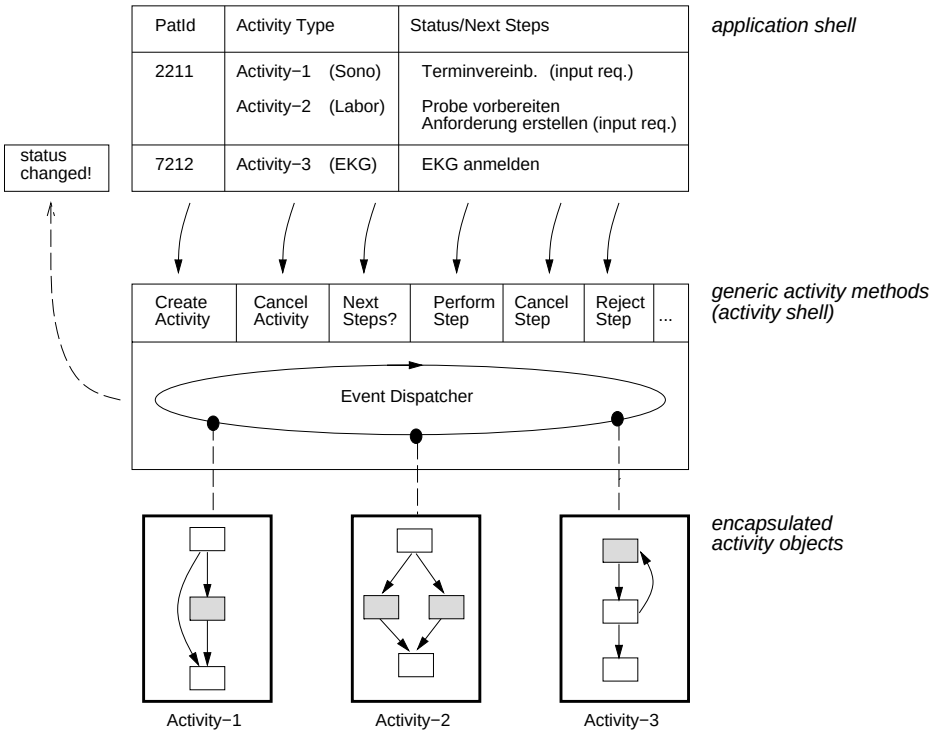


Abb. 1: Zusammenspiel von Anwendungsschale und Aktivitätenobjekten (Prinzip)

Eine Anwendungsschale muß prinzipiell die Möglichkeit besitzen, auf Statusänderungen eines Aktivitätenobjekts, wie das Vorliegen neuer Teilschritte oder das Auftreten einer Zeitüberschreitung, zu reagieren. Typischerweise jedoch erfolgt der Zugriff auf Teilschritte erst dann, wenn sie dem Benutzer kontextbezogen am Bildschirm angezeigt werden sollen. Die Anwendungsschale wird daher bei Auftreten einer Statusänderung zwar sofort über ein durch das Aktivitätenobjekt generiertes Ereignis benachrichtigt, kann aber selbst entscheiden, ob und wann sie dies in der Anwendung (d.h. dem Benutzer gegenüber) sichtbar werden läßt.

Die einer Anwendungsschale angebotenen Teilschritte benötigen für ihre Bearbeitung Daten. Diese müssen entweder aus dem Datenkontext eines Aktivitätenobjekts bereitgestellt werden oder sind durch den Benutzer bei Aufruf eines Teilschritts explizit einzugeben. Hierfür kann der Anwendungsentwickler entweder Standardmethoden für die Ein- und Ausgabe verwenden, die zu jedem interaktiven Baustein vorliegen, oder selbst entsprechende Methoden bereitstellen. Um die Unabhängigkeit einer Anwendungsschale von konkreten Aktivitäten-Implementierungen beizubehalten, sind die Ein- und Ausgabe-Methoden der einzelnen Teilschritte wiederum Bestandteil des gekapselten Aktivitätenobjekts. Es ist somit kein Austausch von anwendungsbezogenen Datenstrukturen zwischen der Anwendungsschale und dem Aktivitätenobjekt erforderlich. Der Anwendungsschale wird bei Zuteilung eines Schritts lediglich angezeigt, ob der Teilschritt Eingaben verlangt. Es besteht somit die Möglichkeit, dem Benutzer zu signalisieren, welche Teilschritte auf eine Eingabe warten.

Aus der Kapselung von Aktivitäten-Instanzen ergeben sich wesentliche Vereinfachungen im Hinblick auf die Verwendung von Aktivitäten-Implementierungen. Die Möglichkeit, Aktivitäten und Teilschritte unabhängig von ihrer Ablauflogik bzw. ihrem Typ über dieselben Methoden manipulieren zu können, vereinfacht die Anpassung von EPATs und Aktivitäten-Implementierungen wesentlich, da sich Änderungen in der Ablauflogik eines EPATs nicht auf die Anwendungsschale auswirken. Die persistente Verwaltung von Aktivitätenobjekten durch die Laufzeitumgebung ermöglicht ein Wiederaufsetzen im Falle von Programm- oder Knotenabstürzen, so daß eine Anwendungsschale die von ihr aktuell bearbeiteten Schritte nicht explizit zu sichern braucht.

In den nachfolgenden Abschnitten werden die Rahmenbedingungen dieses Programmiermodells erörtert. Abschnitt 3.1 gibt einen Überblick über die Modellierung von *Aktivitäten-Templates* (EPATs). Die

Konfiguration eines EPATs durch Einfügen implementierter Softwarebausteine (*Plug-In*) sowie die Adaptionen an die lokale Umgebung sind Gegenstand von Abschnitt 3.2.

3.1 Encapsulated Pre-Modeled Activity Templates (EPATs)

Entscheidend für die Anwendbarkeit des skizzierten Konzeptes ist, daß es gelingt, das komplexe Verhalten, insbesondere das Verhalten bei Auftreten von Ausnahmesituationen (exceptions) und „echten" Fehlerfällen, in einer abstrakten, für den Benutzer verständlichen Notation zu beschreiben. Eine graphische, vorgangsbezogene Notation, welche sich an der Anwendung bzw. deren Teilschritten orientiert, scheint uns für die erforderliche „Verifikation" durch den Anwender am besten geeignet. Wie bei Workflow-Systemen üblich, läßt sich damit, basierend auf einer dem Graph zugrundeliegenden Semantik, das zukünftige Verhalten einer Aktivitäten-Implementierung mit Hilfe eines geeigneten Interpreters bereits während der Modellierungsphase visualisieren bzw. animieren. Wie üblich, müssen geeignete Konstrukte für sequentielle Ausführung, parallele Ausführung (und deren Zusammenführung), sowie für die Auswahl alternativer Ablaufzweige angeboten werden (für ein Anwendungsbeispiel siehe Abb. 2). Neu hinzu kommen in unserem Kontext jedoch die Formulierung von vorgeplanten Ausnahmefällen, wie z.B. Terminverschiebung, Zeitschranken (für die systemseitige Ausführungsüberwachung) sowie das konkrete Festlegen des Verhaltens im Fehlerfall (Systemfehler oder Abbruch einer Maßnahme durch den Benutzer).

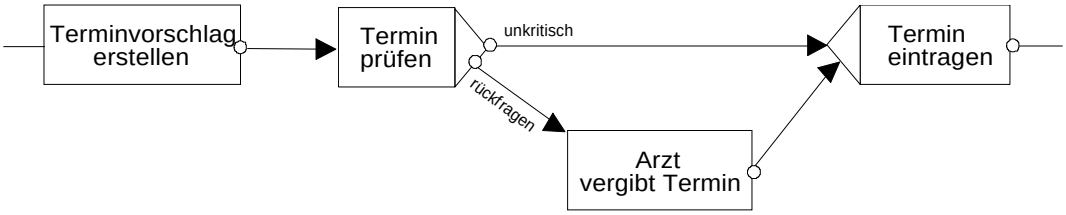
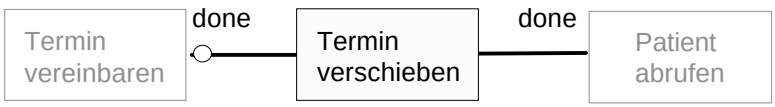
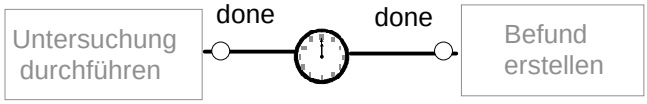


Abb. 2: Beispiel für eine „1 aus 2"- Auswahl bei der halbautomatischen Terminvereinbarung

Um die graphische Repräsentation möglichst anschaulich zu halten, sollte die Modellierung des Normalablaufs von der Ausnahme- und Fehlerbehandlung getrennt werden. Zum einen wird hierzu ein Schichtenmodell mit mehreren Ebenen entwickelt, das die Modellierung verschiedener Aspekte erlaubt, wobei einzelne Ebenen je nach Bedarf ein- oder ausgeblendet werden können (vgl. heutige Graphik-Editoren). Zum anderen werden wir die Modellierung von Ausnahmebedingungen durch eine prädikative, graphische Notation unterstützen. Ein Beispiel hierfür ist in Abb. 3 angeben. Abb. 3.a besagt, daß die (Ausnahme-)Aktion „Termin verschieben" nach Abschluß der Aktion „Termin vereinbaren" und vor Eintritt in die Aktion „Patient abrufen" wählbar ist. Abb. 3.b besagt in analoger Weise, daß bei Überschreiten einer vorgegebenen Zeitdauer zwischen dem Abschluß der Aktionen „Untersuchung durchführen" und „Befund erstellen" das Aktivitätenobjekt eine entsprechende Meldung an die Anwendungsschale geben soll.



a) bedingt wählbare Aktion



b) Zeitdauer-Überwachung

Abb. 3: Angabe prädikativer Bedingungen

Wie bereits oben erwähnt, ist es ein besonderes Anliegen des ADEPT-Ansatzes, dem Anwendungsprogrammierer die systemnahen Aspekte der Ausnahme- und Fehlerbehandlung abzunehmen und ihn zum anderen auch bei der „Flexibilisierung" der Abläufe zu unterstützen. Hierzu ist vorgesehen, daß auf

EPAT-Ebene - wiederum auf separaten Modellierebenen - auch Vorwärtssprünge (im folgenden „Shortcuts“ genannt) und Rückwärtssprünge (z.B. als Folge nicht durchgeführter bzw. abgebrochener Teilschritte) modelliert werden können. Die exemplarische Verwendung dieser Konstrukte zeigt Abb. 4.

Bei der späteren Implementierung der für die Teilschritte einzusetzenden Services (z.B. „Termin vereinbaren“) ist für jeden Service ein „Kompensationsservice“ (z.B. „Termin stornieren“) anzugeben, um das systemseitige Zurücksetzen von Aktivitäten zu ermöglichen. Das Zurücksetzen einer Aktivität wird hierbei auf das Zurücksetzen (Kompensieren) ihrer Teilschritte zurückgeführt (vgl. [GaSa87]). Soll die Zurücksetzbarkeit einzelner Teilschritte nach Erreichen eines bestimmten Bearbeitungszustandes ausgeschlossen werden, so kann dies durch eine entsprechende *Kompensationssphäre* modelliert werden. Im Beispiel in Abb. 4 kann nach begonnener Untersuchung (aus rechtlichen Gründen) weder die Terminvereinbarung noch die Anordnung der Maßnahme zurückgenommen werden.

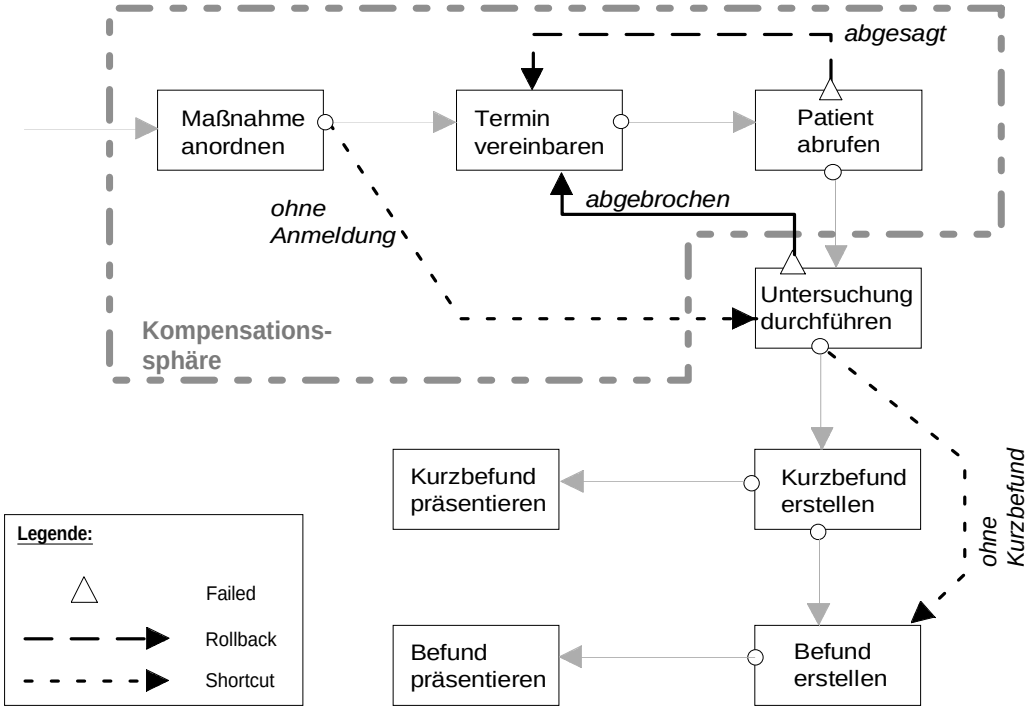


Abb. 4: Modellierung von Shortcuts, Rücksprüngen und Kompensationssphären (vereinfacht)

3.2 Konfiguration und Adaption von EPATs bzw. von Aktivitäten

Um von einem EPAT zu einer ausführbaren Aktivitäten-Implementierung zu gelangen, müssen in einem Konfigurationsschritt für die einzelnen Teilschritte konkrete Implementierungen (Services) festgelegt werden. In vielen Workflow-Management-Systemen ist es nicht möglich, daß ein einzusetzender Service (einer entsprechenden Kategorie) gegenüber der Schnittstelle des Teilschritts zusätzliche Parameter aufweist. Geringfügige Abweichungen in den Parameterleisten von Services machen es dann erforderlich, daß entweder in der Schnittstelle eines Teilschritts alle möglichen Varianten berücksichtigt werden müssen (was zu umfangreichen Parameterleisten führen kann) oder aber, daß unterschiedliche Service-Typen jeweils in einem anderen Workflow modelliert werden müssen. Im ADEPT-Ansatz ist es deshalb vorgesehen, daß Erweiterungen gegenüber den „EPAT-Schnittstellen“ in einem gewissen Umfang toleriert werden. Somit kann für den Anwendungsprogrammierer die Anzahl verschiedener EPATs in einer überschaubaren Größenordnung gehalten werden.

Im Falle solcher Abweichungen müssen dann, nach dem „Plug-in“ der Services, die Versorgung der Schnittstellen hinsichtlich der für den Aufruf erforderlichen Input-Parameter und die Übernahme der Werte der Output-Parameter („Parameter“ jeweils logisch gesehen) geregelt werden. Um diese Aufgabe möglichst einfach (und damit weniger fehleranfällig) zu gestalten, wird bei der Beschreibung dieser Schnittstellen mit einem kontrollierten Vokabular gearbeitet werden, so daß (hoffentlich ein Großteil) der

erforderlichen Verknüpfungen - unter Beachtung der durch das EPAT vorgegebenen Datenflüsse - automatisch durch das Konfigurations-Dienstprogramm vorgenommen werden kann. Nicht versorgte oder typunverträgliche Parameter werden durch die Analyse entdeckt und der Anwendungsentwickler kann mit Hilfe des Konfigurations-Dienstprogramms eine manuelle Zuordnung vornehmen.

Wie schon zu Beginn dieses Abschnitts erörtert, reduziert sich die Entwicklung einer Endanwendung im wesentlichen auf die Implementierung einer Anwendungsschale, die sich mit der Verwaltung von Aktivitätenobjekten befaßt. Für die Implementierung einer solchen Schale stehen dem Anwendungsprogrammierer generische Methoden bereit, die auf Aktivitätenobjekte eines beliebigen EPAT-Typs sowie auf beliebige Teilschritte anwendbar sind. Aufgrund der verteilten Bearbeitung einer Aktivität werden Teilschritte typischerweise durch mehrere Endanwendungen kontrolliert.

Die Verwendung neu hinzukommender oder geänderter Aktivitäten kann prinzipiell ohne Anpassung der jeweiligen Endanwendungen erfolgen. Um dem Anwendungsprogrammierer allerdings eine Adaption an seine Benutzerschnittstelle zu ermöglichen und damit der Forderung nach einer nahtlosen Integration von Abläufen in ein klinisches Arbeitsplatzsystem [PSS94] nachzukommen, können die für einzelne Teilschritte standardmäßig festgelegten Ein- und Ausgabemethoden (vgl. Abschnitt 2) durch eigene Implementierungen überlagert werden. Diese „Bildschirmadapter“ müssen wieder in das konfigurierte EPAT eingesetzt werden, wodurch die Unabhängigkeit der Anwendungsschale von konkreten Aktivitäten-Implementierungen beibehalten wird. Ein Anwendungsprogrammierer kann hierdurch ein konfiguriertes Aktivitätenobjekt isoliert testen, einschließlich der vom ihm festgelegten Ein- und Ausgabemethoden, ohne daß eine Anbindung an eine Endanwendung erforderlich wird. Für das Testen kann er dabei eine einfache Standard-Anwendungsschale verwenden, in die entsprechende Testtreiber eingebunden sind.

Die Steuerung des Fortschritts einer Aktivität, das Monitoring einzelner Teilschritte sowie die Behandlung von Fehler- und Ausnahmefällen erfolgt weitgehend durch die Laufzeitumgebung unter Verwendung der zum EPAT vorliegenden Beschreibung. Im folgenden Abschnitt werden die wesentlichen Komponenten dieser Umgebung und ihre Aufgaben kurz vorgestellt.

4. Laufzeitumgebung

Die Ausführung der in den Aktivitätenobjekten gekapselten Services, unter Beachtung der durch das EPAT 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 A<genten 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.

Der EPAT-Graph und der gegenwärtige Zustand eines Aktivitätenobjektes 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 EPAT-Graphen ist. Eine solche Abstimmung mit den anderen an der Aktivität beteiligten Cooperation Managern ist immer dann notwendig, wenn die verschiedenen Anwendungen parallel Services aktivieren können, die zu verschiedenen Verläufen in der Aktivität führen.

Die persistente Verwaltung des Datenkontexts eines Aktivitätenobjekts 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 Zustandsübergangs) fest, daß noch zwingend erforderliche Eingabeparameter unversorgt sind, so fordert er diese vom Anwender selbständig nach.

Die *Scheduling Manager* verwalten die Zeitrestriktionen eines Aktivitätenobjekts. Sie werden von den Cooperation Managern mit dem Starten der Uhren beauftragt und melden das Ablaufen einer Uhr an

diese zurück. Der Cooperation Manager kann dann ein entsprechendes Ereignis für die Anwendungsschale erzeugen. 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 Aktivitätenobjekte jeweils deren aktuellen Weg durch ihren EPAT-Graphen, um im Kompensationsfall die erforderlichen Kompensationsschritte bestimmen zu können.

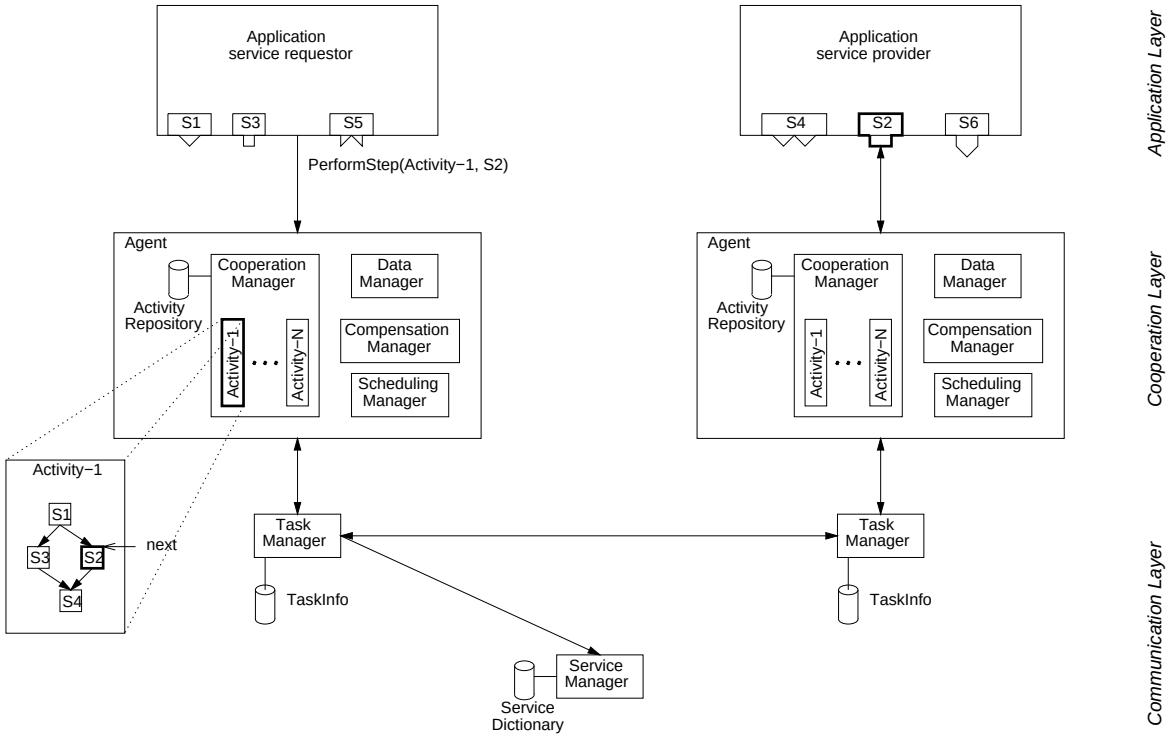


Abb. 5: Komponenten der Laufzeitumgebung

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 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] unternommen. Die Idee verknüpfbarer Software-Bausteine wurde von uns im Zusammenhang mit der Konfiguration von EPATs 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 EPATs 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 Unter-

stützung 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 Architekturkonzepte des Laufzeitsystems (Agenten, TaskMgr, ServiceMgr) 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

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