Prozeßintegration in PRIME: Modelle, Architektur und Vorgehensweise

Klaus Pohl, Klaus Weidenhaupt, Ralf Dömges, Peter Haumer, Matthias Jarke
Lehrstuhl Informatik V, RWTH Aachen; email: pohl@informatik.rwth-aachen.de

Zusammenfassung

Die Forschung im Bereich *prozeßzentrierter* Entwicklungsumgebungen hat sich bisher hauptsächlich mit der Entwicklung geeigneter Prozeßmodellierungsssprachen und Prozeßmodellausführungsmechanismen befaßt. Die Auswirkungen der zunehmenden Prozeßorientierung auf die interaktiven Werkzeuge der Entwicklungsumgebungen wurden bisher nur wenig betrachtet.

In dieser Arbeit stellen wir einen modellbasierten Ansatz zur *Prozeßintegration interaktiver Werkzeuge* in prozeßzentrierten Umgebungen, genannt *PRIME* (PRocess-Integrated Modelling Environments), vor. Im Gegensatz zu anderen Ansätzen sind PRIME-basierte Werkzeuge in der Lage, ihr Verhalten gemäß den Prozeßdefinitionen sowie der aktuellen Prozeßsituation anzupassen. Die Prozeßintegration der Werkzeuge wird erreicht durch (1) die Definition von *Werkzeug- und Prozeßmodellen* sowie deren Integration; (2) die *Interpretation der integrierten Modelle* durch die Werkzeuge, die Prozeßmaschine und den Kontrollintegrationsmechanismus; (3) die Synchronisation der Prozeßmaschine und der Werkzeuge basierend auf einem *Interaktionsprotokoll*.

Neben dem *Modellierungsansatz* beschreiben wir die resultierende *Architektur für prozeßintegrierte Werkzeuge* sowie eine *sechsstufige Vorgehensweise* zur Erstellung von prozeßintegrierten Modellierungsumgebungen. Die Vorgehensweise sowie der allgemeine Ansatz wurden durch die Entwicklung von zwei prozeßintegrierten Umgebungen validiert.

1 Einleitung

Seit Mitte der 80er Jahre verlagert sich bei der rechnerbasierten Unterstützung von Modellierungstätigkeiten der Schwerpunkt weg von rein *produkt*zentrierten CASE-Tools hin zu *prozeß*zentrierten Umgebungen, die die Unterstützung explizit definierter Vorgehensweisen und Methodiken in den Vordergrund stellen. Um die Anpaßbarkeit an projekt- und organisationsspezifische Bedürfnisse und an die vielfältigen Änderungen, denen Modellierungsprozesse laufend unterworfen sind, zu gewährleisten, steuern in prozeßzentrierten Umgebungen sogenannte *Prozeßmaschinen* die Prozeßausführung durch die Interpretation explizit definierter Prozeßmodelle [17], [18]

Damit sind zusammen mit den traditionellen Werkzeugen am Arbeitsplatz und der Modellverwaltung drei Domänen und deren Interaktionen zu unterscheiden [12], [32]. In der *Modellierungsdomäne* werden Mechanismen und Konzepte zur expliziten Definition und persistenten Verwaltung von Prozeßmodellen zur Verfügung gestellt. Neben den Prozeßmodellen erfolgt auch die Verwaltung der entstandenen Produkte in der Modellierungsdomäne. Die *Leitdomäne* umfaßt alle Aktivitäten und Mechanismen zur dynamischen Interpretation instantiierter Prozeßmodelle, mittels derer die Prozeßausführung in der Ausführungsdomäne angeleitet, kontrolliert oder automatisiert wird (Prozeßsteuerung und -überwachung). In der *Ausführungsdomäne* werden einerseits die im Prozeßmodell definierten Aktivitäten von Menschen und/oder Werkzeugen ausgeführt. Andererseits wird mittels Rückmeldungen an die Leitdomäne die Prozeßmodellinterpretation beeinflußt. Sowohl die auszuführenden Aktivitäten als auch die für die Durchführung der Aktivitäten zur Verfügung stehende Unterstützung werden von der Leitdomäne vorgegeben.

1.1 Projektmanagement vs. feingranulare Arbeitsplatzunterstützung

Wie die Forschung über Workflowsysteme für strukturierte Standardaufgaben konzentrierte sich die Forschung über PCEE (Process Centered Engineering Environment) bisher weitgehend auf die Definition geeigneter Prozeßmodellierungssprachen und Prozeßmaschinen [24]. Es entstanden eine Vielzahl von Sprachen und prototypischen Systemen, die auf unterschiedlichen Paradigmen basieren (für einen Überblick siehe [17], [9]): netzbasierte (SPADE /SLANG [2], MELMAC [23], Income/Star [25], DYNAMITE [22]), regelbasierte (Marvel [4]), imperative (APPL/A [36]) und objektorientierte (EPOS [8]) Prozeßmodellierungssprachen sowie Erweiterungen aktiver Datenbanken (ADELE [5]).

Allen Ansätzen gemeinsam ist, daß Prozesse und deren Anpaßbarkeit und Erweiterbarkeit nur auf der relativ grobgranularen Ebene des Projektmanagements betrachtet werden. Prozeßzentrierte Umgebungen sind daher für die Unterstützung gut verstandener und somit vollständig definierbarer Prozesse gut geeignet.

Die Unterstützung von kreativen Prozessen bedingt jedoch eine feingranulare methodische Unterstützung des Entwicklers am Arbeitsplatz, die sich in wesentlichen Punkten von der grobgranularen Ebene des Projektmanagements unterscheidet (siehe [31] für eine detaillierte Betrachtung der Unterschiede):

- Arbeitsplatzunterstützung vs. Koordination der Zusammenarbeit zwischen Entwicklern;
- Betrachtung feingranularer Produktstrukturen vs. grobgranularer Dokumente;
- Benutzerinteraktion mit Leitdomäne über interaktive Modellierungswerkzeuge vs. Interaktion über Agendamanager, Task-Monitore o.ä.

Die mit der feingranularen, methodischen Unterstützung einhergehende *Prozeßintegration* der verwendeten Modellierungswerkzeuge wird von prozeßzentrierten Umgebungen bislang nicht unterstützt [32].

1.2 Anforderungen an feingranulare Prozeßintegration

Für die feingranulare Prozeßintegration lassen sich eine Reihe von Integrationsanforderungen identifizieren, die nicht nur die von Wasserman [39] begründeten Integrationsfacetten Plattform-, Benutzeroberflächen-, Kontroll- und Datenintegration in wesentlichen Punkten erweitern, sondern auch die in [12], [11], [13], [15], [37] genannten Aspekte der Prozeßintegration präzisieren.

Die unten erläuterten Anforderungen[1] beruhen im wesentlichen auf der Beobachtung, daß bei der Integration von Leit- und Ausführungsdomäne drei Arten von Diensten unterschieden werden können, die wechselseitig von der Prozeßmaschine und den Werkzeugen in Anspruch genommen werden. *Elementare Dienste* sind atomare Funktionalitäten, die von einem Werkzeug zur Automatisierung eines Prozeßschrittes zur Verfügung gestellt werden (z.B. das Erzeugen eines Modellierungsobjekts). *Auswahldienste* werden ebenfalls von Werkzeugen angeboten. Entsprechend der aktuellen Prozeßsituation sowie der Prozeßdefinition wird dem Benutzer eine Teilmenge von Prozeßschritten zur Auswahl angeboten. *Prozeßfragmente* sind definierte Abfolgen von Prozeßschritten, die von der Prozeßmaschine interpretiert und in ihrer Ausführung kontrolliert werden.

(1) Dienstintegration: Während der Interpretation der Prozeßfragmente stützt sich die Prozeßmaschine auf die von den Werkzeugen angebotenen elementaren Dienste und Auswahldienste ab. Um zu gewährleisten, daß die in den Prozeßfragmenten referenzierten Werkzeugdienste tatsächlich von der Werkzeugumgebung zur Verfügung gestellt werden, ist eine explizite Definition dieser Dienste auf der gleichen konzeptuellen Ebene wie die der Prozeßmodelle erforderlich.

(2) Werkzeugunterstützter Aufruf von Prozeßfragmenten: Die Modellierungswerkzeuge sollen den Entwickler beim Abgleich der aktuell vorliegenden Prozeßsituation mit den Prozeßdefinitionen unterstützen und ggf. den Aufruf von Prozeßfragmenten erlauben. Aus Sicht des Benutzers bedeutet dies, daß für ihn kein Unterschied zwischen dem Aufruf werkzeugeigener Dienste und Prozeßfragmenten (werkzeugfremder Dienste) bestehen sollte.

(3) Prozeßsensitive Anpassung der Benutzerinteraktionsmöglichkeiten: Abhängig vom aktuellen Prozeßzustand muß ein Modellierungswerkzeug in der Lage sein, an der Benutzeroberfläche den Zugriff auf Modellierungsobjekte und darauf definierten Aktionen dynamisch anzupassen, indem beispielsweise nur definierte Objekte (Produkte, Menüoptionen, Icons, etc.) angezeigt bzw. deren Selektierbarkeit eingeschränkt wird.

(4) Prozeßkonforme Rückmeldungen: Um eine Angleichung der Prozeßmodellinterpretation an die aktuelle Prozeßausführung zu gewährleisten, sind aussagekräftige Rückmeldungen der Modellierungswerkzeuge an die Leitdomäne erforderlich, z.B. über Benutzerauswahlen bei Auswahldiensten oder das Resultat einer Dienstausführung.

(5) Synchronisation der Ausführungs- und Leitdomäne: Der Zustandsabgleich zwischen Ausführungsdomäne und Leitdomäne erfordert einen Nachrichtenaustausch, der durch ein Interaktionsprotokoll geregelt werden muß. Das Protokoll muß u.a. zwischen normaler Prozeßausführung, automatisierter Prozeßausführung, Benutzerentscheidungen und möglichen Prozeßabweichungen unterscheiden.

Der Rest des Artikels ist wie folgt gegliedert: In Kapitel 2 stellen wir unseren prinzipiellen Lösungsansatz (PRIME) für die obigen Anforderungen vor und vergleichen diesen mit anderen Ansätzen. In Kapitel 3 präsentieren wir unser integriertes Metamodell für die Prozeß- und Werkzeugmodellierung. In Kapitel 4 leiten wir daraus eine generische Architektur für prozeßintegrierte Werkzeuge ab und skizzieren deren Implementierung. In Kapitel 5 erläutern wir eine sechsstufige Vorgehensweise zur Realisierung prozeßintegrierter Umgebungen unter Verwendung der vorgestellten Modelle und Architekturen. In Kapitel 5 berichten wir über Erfahrungen mit der Entwicklung zweier prozeßintegrierter Umgebungen und geben einen Ausblick auf weitere Arbeiten.

2 Der PRIME Ansatz

2.1 Überblick

Abb. 1 gibt einen Überblick über PRIME (PRocess Integrated Modelling Environments), einen generischen Ansatz für prozeßintegrierte Modellierungsumgebungen, der Lösungen für die oben genannten Anforderungen bereitstellt. PRIME stützt sich auf drei wesentliche Grundpfeiler ab:

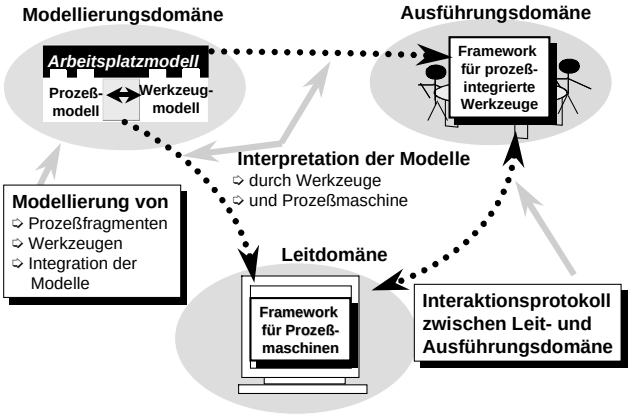

Abb.1: Prinzipielle Lösungsansätze in PRIME

(1) Integrierte Prozeß- und Werkzeugmetamodelle: In PRIME werden Prozesse und Werkzeuge gleichberechtigt auf einer konzeptuellen Ebene modelliert. Die Integration von Prozeß- und Werkzeugmodellen in sogenannten *Arbeitsplatzmodellen* bildet die Grundlage für:

- Datenintegration durch Definition gemeinsamer Produktdatenschemata für Werkzeuge und Prozeßmaschine;

[1] Eine detaillierte Herleitung und Erläuterung der Anforderungen wird in [31] gegeben.

- Dienstintegration durch uniforme Repräsentation von elementaren Diensten, Auswahldiensten und Prozeßfragmenten;
- Werkzeugunterstützter Aufruf von Prozeßfragmenten;
- Dynamische Anpassung der in den Werkzeugen zugreifbaren Objekte und Dienste gemäß aktuellem Prozeßzustand und der Prozeßdefinitionen;
- explizite Definition der erwarteten Rückmeldungen an die Leitdomäne für alle Dienstarten.

Die Metamodelle werden in Kapitel 3 erläutert.

***(2) Interaktionsprotokoll**:* Das Interaktionsprotokoll definiert das Verhalten der Werkzeuge und der Prozeßmaschine sowie deren Interaktionen. Es bildet die Basis für die Synchronisation beider Domänen und wurde mit State-Charts [21] spezifiziert (siehe [32]).

***(3) Frameworks für prozeßintegrierte Werkzeuge und Leitmechanismen**:* Basierend auf dem integrierten Arbeitsplatzmodell und dem Interaktionsprotokoll haben wir generische Architekturen für prozeßintegrierte Werkzeuge und Leitmechanismen entworfen. Beide Architekturen wurden in Form objektorientierter Implementierungsframeworks realisiert. Die Architektur für prozeßintegrierte Werkzeuge sowie das resultierende Implementierungsframework werden in Kapitel 4 erläutert.

Zusätzlich zu den oben genannten Modellen, Architekturen und Implementierungsframeworks haben wir für PRIME eine Vorgehensweise für die Realisierung einer prozeßintegrierten Modellierungsumgebung entwickelt (Kapitel 5)

2.2 Vergleich mit anderen Ansätzen

Im Vergleich mit anderen Integrationsansätzen in prozeßzentrierten Umgebungen erzielt PRIME eine wesentlich engere und feingranularere Integration der in der Ausführungsdomäne verwendeten Werkzeuge mit der Leit- und Modellierungsdomäne. Im Gegensatz zu der von PRIME angebotenen Prozeßintegration der Werkzeuge wird die Integration von Werkzeugen in prozeßzentrierten Umgebungen zumeist vernachlässigt. Die Notwendigkeit und das Problem der Integration existierender CASE Werkzeuge wird jedoch als wichtig erkannt (vgl. z.B. [18], [1],[15]).

Kontrollorientierte Werkzeugintegrationsansätze wie z.B. FIELD [33], die kommerziellen Derivate (wie z.B. HP's BMS [7] und Sun's ToolTalk [35]) und objektorientierte Ansätze wie beispielsweise CORBA [27] oder OLE/COM [6] verwalten Beschreibungen von Werkzeugen und deren Dienste in eigenen, internen Repositorien. In vielen prozeßzentrierten Umgebungen (z.B. SPADE [3], EPOS [8] Merlin [23], ProcessWeaver [16]) erfolgt der Aufruf von Werkzeugdiensten unter Verwendung eines Kontrollintegrationsmechanismus. Eine geeignete Integration von Prozeß- und Dienstdefinitionen wird jedoch nicht angeboten. Als Konsequenz koexisitieren Prozeß- und Dienstmodelle in Prozeß- und Kontrollintegrationsrepositorien ohne ausreichende Unterstützung für die Konsistenzsicherung beider Modelle.

Darüber hinaus beschränkt sich die Werkzeugintegration auf den Aufruf elementarer Dienste. Die Anleitung des Werkzeugbenutzers durch die dynamische Anpassung von angezeigten und selektierbaren Objekten und Werkzeugoperationen (Menüfunktionen) wird in keinem der Ansätze unterstützt.

In vielen prozeßzentrierten Umgebungen werden Werkzeuge daher als „black boxes" betrachtet, d.h. die Prozeßmaschine hat lediglich die Möglichkeit, das Starten und eventuell die Terminierung eines Werkzeugs bzw. Werkzeugdienstes anzustoßen. Dies erschwert jedoch die Synchronisation der Leit- und Ausführungsdomäne, da sich Prozeßmaschine und Werkzeuge nicht über die Ausführung und die Resultate feingranularer Dienste verständigen können. Der Versuch einer engen Integration existierender Werkzeuge mit der Leitdomäne (Prozeßmaschine) hat sich oft als sehr schwierig und teilweise unmöglich herausgestellt (z.B. [3]). Eine Ausnahme ist der im GOODSTEP Projekt [20] entwickelte GTSL Ansatz [14]. Durch die Generierung von werkzeugspezifischen Diensten und Konsistenzsicherungen, basierend auf einer mit der Prozeßdefinition integrierten Werkzeugdefinition, unterstützt dieser Ansatz hauptsächlich die Dienstintegration und Datenintegration. Der Aufruf von Prozeßfragmenten sowie die dynamische Anpaßbarkeit der Werkzeuge gemäß der Prozeßdefinition und der aktuellen Prozeßsituation wird jedoch nicht unterstützt.

Im Vergleich zu allen existierenden Ansätzen werden die Dienst- und Prozeßdefinitionen in PRIME in den Arbeitsplatzmodellen integriert. Dadurch wird die Dienst- und Datenintegration gewährleistet. Die Interpretation dieser Modelle durch die Werkzeuge, die Prozeßmaschinen sowie den Kontrollintegrationsmechanismus ermöglicht die prozeßsensitive Benutzerführung, die Unterstützung beim Aufruf von Prozeßfragmenten sowie die korrekte und interpretierbare Rückmeldung der Dienstergebnisse an die Leitdomäne.

3 Modellierung von Prozessen und Werkzeugen: Ein integriertes Metamodell

Abb. 2 zeigt in OMT-Notation das in PRIME zur Modellierung von Prozessen und Werkzeuge verwendete Metamodell. Während im oberen Teil von Abb. 2 Konzepte und Beziehung zur Prozeßmodellierung dargestellt sind (grau umrandet), dient der untere Teil des Metamodells (schwarz umrandet) der Repräsentation von Werkzeugen. Durch überlappende Konzepte (grau dargestellt) und eine kleine Menge von zusätzlichen Assoziationen (gestrichelt) werden Prozeßfragmente auf Werkzeugfunktionalitäten abgebildet und ein integriertes *Arbeitsplatzmetamodell* etabliert.

3.1 Das Prozeßmetamodell

Die Repräsentation der in einer prozeßintegrierten Modellierungsumgebung vorkommenden drei Dienste basiert auf dem im ESPRIT-Projekt NATURE entwickelten kontextbasierten Prozeßmetamodell zur Modellierung *kreativer* Prozesse. Dieses Modell vereinigt entscheidungsbasierte, produktbasierte und KI-basierte Prozeßmodellierungskonzepte (vgl. [29], [34], [32]).

Das Metamodell repräsentiert Dienste mittels des zentralen Konzepts *Kontext.* Die Kontextabhängkeit von Prozeßschritten basiert auf der expliziten Verknüpfung von Pro-

zeß*situationen* mit den in diesen Situationen unterstützten Benutzer*intentionen*. Eine *Situation* wird definiert als eine bestimmte Konstellation von Modellierungsprodukten. Eine *Intention* reflektiert ein mögliches Ziel, das in einer gegebenen Situation zu erreichen ist. Mit Hilfe der Spezialisierungen *Automatisierungskontext*, *Entscheidungskontext* und *Plankontext* werden Prozeßschritte auf die in einer prozeßintegrierten Umgebung auftretenden elementaren Dienste, Auswahldienste und Prozeßfragmente abgebildet:

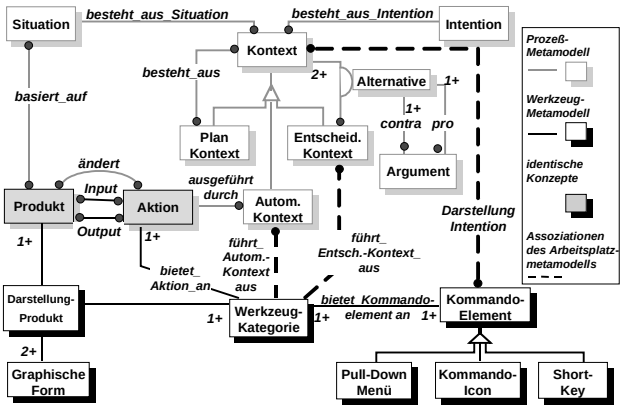

Abb. 2: Prozeß- und Werkzeugmetamodell (OMT-Notation)

Automatisierungskontexte repräsentieren die Teile einer Prozeßdefinition, die strikt erzwungen oder sogar automatisiert werden können. Ein Automatisierungskontext wird durch die Ausführung eines elementaren Werkzeugdienstes operationalisiert. Durch die Ausführung einer Aktion werden i.a. die Modellierungsprodukte verändert, so daß neue Situationen entstehen.

Entscheidungskontexte modellieren Prozeßzustände, in denen eine explizite Benutzerentscheidung über den weiteren Ablauf notwendig ist. Ein Entscheidungskontext schlägt wenigstens zwei alternative Vorgehensweisen vor, die ebenfalls wieder als Automatisierungs-, Entscheidungs- oder Plankontext modelliert werden. Für jede Alternative können Pro- und Kontraargumente zur Benutzerführung angegeben werden. Entscheidungskontexte korrespondieren mit Auswahldiensten, mit deren Hilfe ein Werkzeug den Benutzer auf die zu einem Zeitpunkt erlaubten und relevanten Möglichkeiten fokussiert.

Plankontexte definieren eine Strategie zur Erfüllung einer Intention, indem sie einen Kontrollfluß über eine bestimmte Teilmenge von Kontexten legen. Die Ausführung eines Plankontexts gibt dem Benutzer eine Reihenfolge vor, in der er die Subkontexte des Plankontexts ausführen soll. Plankontexte entsprechen somit Prozeßfragmenten, die von der Prozeßmaschine ausgeführt werden.

Im Vergleich zu anderen Prozeßmodellierungsansätzen besteht ein wesentlicher Vorteil in der freien Kombinierbarkeit von automatisierenden (Automatisierungskontexte), anleitenden (Entscheidungskontexte) und vorschreibenden (Plankontexte) Diensten. Weiterhin ermöglicht die kontextbasierte Beschreibung von Prozeßschritten die Definition von modularen Prozeßfragmenten und vermeidet somit die sonst übliche, aber gerade für *kreative* Spezifikationsprozesse völlig inadäquate starre, monolithische Modellierung des Gesamtprozesses.

3.2 Das Werkzeugmetamodell

Die Repräsentation von Prozessen und Werkzeugen auf einer gemeinsamen konzeptuellen Ebene schafft die Voraussetzung für den Vergleich zwischen den in den Prozeßmodellen definierten und den von der Werkzeugumgebung zur Verfügung gestellten Diensten. Im folgenden skizzieren wir eine Erweiterung des kontextbasierten Prozeßmodells um Konzepte zur Werkzeugmodellierung, welche die Abbildung von Prozeßschritten auf Werkzeugfunktionalitäten und Benutzerinteraktionsmöglichkeiten ermöglicht (vgl. [31]).

Im Zentrum des Werkzeugmodells steht das Konzept der *Werkzeugkategorie*. Durch Instanziierung dieses Konzepts lassen sich die in einer Werkzeugumgebung verfügbaren Werkzeuge definieren. Elementare Werkzeugdienste werden durch das Konzept *Aktion* definiert, das aus Werkzeugsicht durch eine Signatur von Eingabe- und Ausgabeprodukten charakterisiert ist (Assoziationen *Input* und *Output*). Die graphische Präsentation eines Produkt(teil)s wird durch Angabe seiner *Graphischen Form* definiert, wobei diese spezifisch für jede Werkzeugkategorie festgelegt werden kann. Neben den Darstellungsmöglichkeiten für ein Produkt spielen für die Kontextaktivierung die vom Werkzeug angebotenen Interaktionsmöglichkeiten eine wichtige Rolle. Für Produkte nehmen wir an, daß jedes Werkzeug die Selektion und Deselektion einzelner Produktteile erlaubt, so daß wir hier auf eine explizite Modellierung verzichten. Im Gegensatz dazu müssen die Elemente zur Kommandoaktivierung explizit modelliert werden. Zur Zeit beschränken wir uns auf drei Arten von Kommandoelementen: *Pull-Down-Menüs*, *Short-Keys* und *Kommandoicons.*

3.3 Das Arbeitsplatzmetamodell

Das Prozeßmetamodell definiert die Konzepte zur Modellierung der in einer Modellierungsumgebung angebotenen Prozeßunterstützung. Das Werkzeugmetamodell definiert die Konzepte zur Beschreibung der eingesetzten Werkzeuge. Durch die Integration dieser Metamodelle wird ein sogenanntes *Arbeitsplatzmetamodell* etabliert. Das Arbeitsplatzmetamodell legt u.a. fest, welches Werkzeug welche Automatisierungs- und Entscheidungskontexte anbietet bzw. ausführt.

Die Integration der beiden Metamodelle wird durch die unten näher erläuterten Assoziationstypen erzielt.

führt_Automatisierungskontext_aus: Jeder im Prozeßmodell definierte Automatisierungskontext muß einer Werkzeugkategorie als elementarer Dienst zugeordnet werden. Dies wird durch eine Assoziation vom Typ *führt_Automatisierungskontext_aus* zwischen einer Werkzeugkategorie und dem Automatisierungskontext ausgedrückt. Gegeben ein Automatisierungskontext E, eine damit assoziierte Aktion A und eine Menge W von Werkzeugkategorien $W_1,...,W_n$. Dann können bei der Zuweisung von E zu $W_i \in W$ drei Fälle auftreten:

- *Automatische Zuweisung:* Wenn genau eine Werkzeugkategorie W_i die erforderliche Aktion A anbietet, kann W_i automatisch mit E assoziiert werden.

- *Auswahl der Werkzeugkategorie:* Wenn zwei oder mehr Werkzeugkategorien $W_{i_1},...,W_{i_m}$ die erforderliche Aktion *A* anbieten, muß der Methodenmodellierer eine Auswahl unter den Werkzeugkategorien treffen.
- *Fehlende Werkzeugunterstützung:* Wenn keine Werkzeugkategorie W_i die erforderliche Aktion *A* anbietet, muß entweder die fehlende Aktion implementiert und im Werkzeugmodell modelliert, oder das Prozeßmodell so geändert werden, daß der Automatisierungskontext nicht benötigt wird.

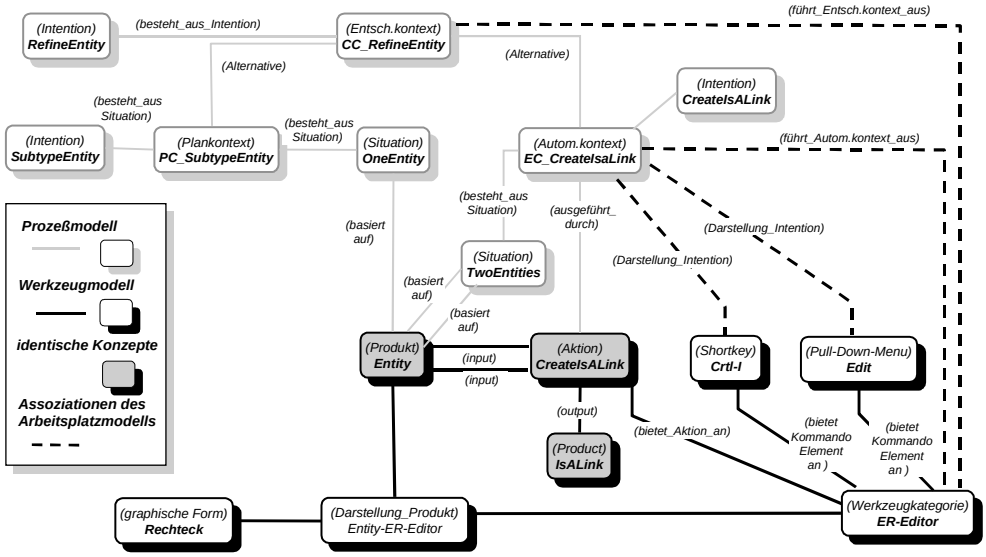

Abb. 3: Vereinfachtes Beispiel für ein Arbeitsplatzmodell

Es lassen sich Integritätsbedingungen formulieren, mit denen sichergestellt werden kann, daß die Signatur der Aktionen im Werkzeugmodell mit den Definitionen im Prozeßmodell korrespondiert (siehe [31]).

führt_Entscheidungskontext_aus: Jeder im Prozeßmodell definierte Entscheidungskontext muß einer Werkzeugkategorie zugeordnet werden. Dies geschieht durch Instanziierung der Assoziation *führt_Entscheidungskontext_aus* zwischen einem Entscheidungskontext und einer Werkzeugkategorie. Durch diese Zuweisung wird ein neuer Auswahldienst für die Werkzeugkategorie definiert.

Darstellung_Intention: Gegeben ein Entscheidungskontext *C* mit den Alternativen $C_1,...,C_n$. Um die Aktivierung der Alternativen $C_1,...,C_n$ in den Werkzeugkategorien zu ermöglichen, muß die Darstellung aller Intentionen von $C_1,...,C_n$ durch Kommandoelemente festgelegt werden. Da eine Intention (z.B. *lösche*) mit mehreren Kontexten (z.B. lösche_Entität und lösche_Attribut) assoziiert sein kann, ist eine kontextabhängige Zuordnung der Intention zu den Kommandoelementen erforderlich. Dies wird erreicht, indem man jeden alternativen Kontext C_i durch eine Assoziation vom Typ *Darstellung_Intention* mit einem oder mehreren Kommandoelementen verknüpft.

Eine korrekte Zuordnung von Werkzeugkategorien mit Entscheidungskontexten kann durch entsprechende Integritätsbedingungen unterstützt werden (siehe [31]).

3.4 Beispiel für ein Arbeitsplatzmodell

Abb. 3 illustriert anhand eines vereinfachten Beispiels die Zuweisung der im Prozeßmodell definierten Automatisierungs- und Entscheidungskontexte zu den im Werkzeugmodell definierten Werkzeugressourcen. Zur Übersichtlichkeit sind die jeweiligen Klassen der Instanzen in Klammern und kursiv dargestellt.

Der obere Teil von Abb. 3 stellt einen stark vereinfachten Ausschnitt aus dem Prozeßmodell dar, in dem für den Entscheidungskontext *CC-RefineEntity* zwei Alternativen, nämlich der Automatisierungskontext *EC-CreateIsALink* und der Plankontext *PC-SubtypeEntity* definiert werden. Weiterhin sind die Situationen (*OneEntity, TwoEntities*) und die Intentionen (*CreateIsALink, SubtypeEntity*) der beiden alternativen Kontexte sowie das Produkt (*Entity*) und die mit dem Automatisierungskontext verbundene Aktion (*CreateIsALink*) im Prozeßmodell dargestellt.

Der untere Teil von Abb. 3 zeigt einen Ausschnitt aus dem Werkzeugmodell. Hier ist die Werkzeugkategorie *ER-Editor* zusammen mit den unterstützten Kommandoelementen (der Shortkey *Crtl-I* und das Pulldown-Menü *Edit*) dargestellt. Weiterhin steht die Werkzeugkategorie *ER-Editor* mit gemeinsamen Konzepten aus dem Prozeßmodell in Beziehung (mit der Aktion *CreateIsALink* und dem Produkt *Entity)*.

Zusätzlich sind die Assoziationen des integrierten Arbeitsplatzmodell dargestellt. Die Assoziation vom Typ *führt-Entscheidungskontext-aus* definiert den Entscheidungskontext *CC-RefineEntity* als Auswahldienst des *ER-Editors*. Bei der Ausführung dieses Entscheidungskontextes muß der ER-Editor die beiden Alternativen *EC-CreateIsALink* und *PC-SpecializeEntity* zur Auswahl anbieten. Dies erfordert, daß der *ER-Editor* die Intentionen (hier: *CreateIsALink* und *SubtypeEntity*) und Produktteile (hier: Entity) der Alternativen anbietet. Die Assoziation *Entity-ER-Editor* vom Typ *Darstellung-Produkt* legt die Visualisierung eines *Entity* im *ER-Editor* in Form eines *Rechteckes* fest. Die Assoziationen vom Typ *Darstellung-Intention* ermöglichen die Aktivierung der Intention *CreateIsALink* mittels Shortkey *Crtl-I* und dem Pulldown-Menü *Edit*. In ähnlicher Weise erfolgt die Abbildung der Intention *Subtype-Entity* auf die entsprechenden Kommandoelemente (nicht in Abb. 3 dargestellt). Die Assoziation vom Typ *führt-Automatisierungskontext-aus* legt fest, daß der Automatisierungskontext *EC-CreateIsALink* von der Werkzeugkategorie *ER-Editor* ausgeführt wird. Voraussetzung für die Definition dieser Assoziation ist, daß der *ER-Editor* mit der zum Automatisierungskontext assoziierten Aktion *CreateIsALink* durch eine durch Assoziation vom Typ *bietet-an* verbunden ist.

3.5 Zusammenfassung

Die Definition von Werkzeug- und Prozeßmodellen, sowie deren geeignete Integration bildet die Grundlage für

die Prozeßintegration von Werkzeugen in prozeßzentrierten Entwicklungsumgebungen. Der Methodeningenieur wird in der Zuordnung der im Prozeßmodell definierten Dienste zu den entsprechenden Werkzeugfunktionalitäten unterstützt. Die korrekte Zuordnung kann durch entsprechende Konsistenzbedingungen gewährleistet werden [31]. Dadurch werden sowohl *Daten-* als auch *Dienstintegration* erzielt

Darüber hinaus bildet das integrierte Arbeitsplatzmodell die Grundlage für die Unterstützung des Benutzers bei der Aktivierung von Prozeßfragmenten durch die Werkzeuge. Neben der Zuordnung von Automatisierungskontexten und Entscheidungskontexten zu den Werkzeugmodellen erlangt das Werkzeug Wissen über die Prozeßdefinitionen durch die Definition von Plankontexten als Alternativen von Entscheidungskontexten. Diese Assoziationen bilden zudem die Basis für die *prozeßsensitive Anpassung der Benutzerinteraktionsmöglichkeiten* (angezeigte und selektierbare Produkte, Menüoptionen, Icons, etc.) und die *prozeßkonforme Rückmeldung* der Dienstergebnisse (bei Entscheidungskontexten durch die vordefinierten möglichen Alternativkontexte; bei Automatisierungskontexten durch die definierten Output-Parameter). Zusätzlich versetzt das Arbeitsplatzmodell die Leitdomäne (Prozeßmaschine) in die Lage, das für die Ausführung eines Dienstes verantwortliche Werkzeug zu identifizieren und entsprechend zu aktivieren

Die Interpretation des Arbeitsplatzmodells zur Laufzeit durch die drei Hauptkomponenten des PRIME-Ansatzes (prozeßintegrierte Werkzeuge, Prozeßmaschine und Kontrollintegrationsmechanismus) gewährleistet die modellkonforme Prozeßintegration in einer PRIME-basierten Umgebung.

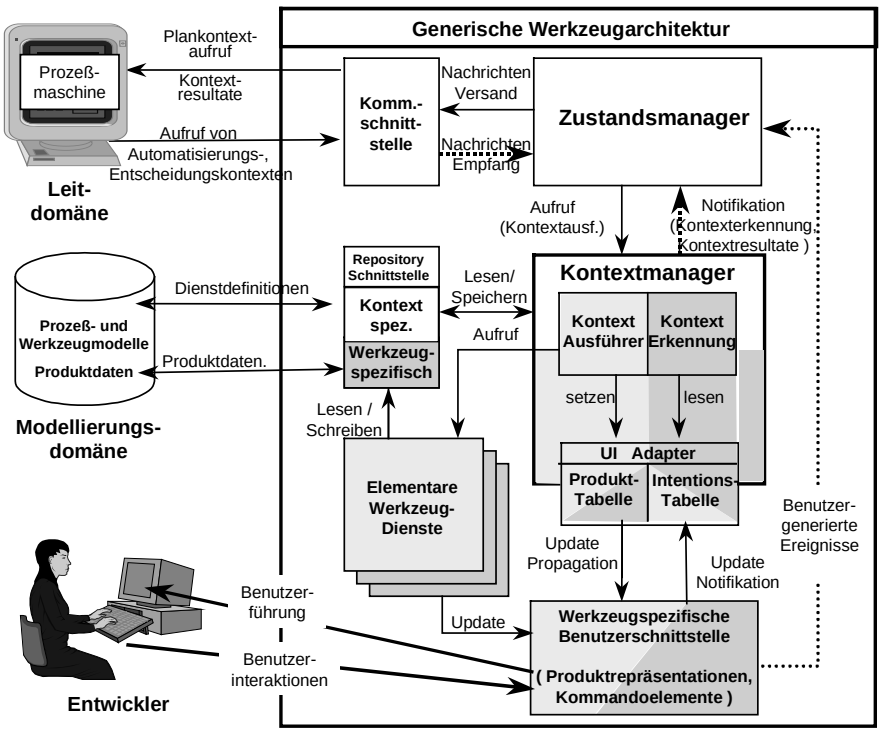

Abb.4 : Generische Werkzeugarchitektur

4 Architektur für prozeßintegrierte Werkzeuge

Aufbauend auf dem integrierten Prozeß- und Werkzeugmodell und der Definition eines Interaktionsprotokolls zwischen Ausführungs- und Leitdomäne haben wir in PRIME eine generische Architektur für prozeßintegrierte Werkzeuge (Abb. 4) entwickelt. Die vier Hauptaufgaben sind der Nachrichtenaustausch mit der Leitdomäne und Wartung des Werkzeugzustands, die Werkzeuginitialisierung gemäß dem Arbeitsplatzmodell, die Kontextausführung und die Kontexterkennung. Für diese Aufgaben sind zwei zentrale, generische Subsysteme zuständig: der Zustandsmanager und der Kontextmanager.

4.1 Der Zustandsmanager

Der Zustandsmanager stellt sicher, daß der Nachrichtenaustausch mit der Leitdomäne gemäß dem definierten Interaktionsprotokoll erfolgt und wartet den Werkzeugzustand in Abhängigkeit von Ereignissen, die innerhalb des Nachrichtenprotokolls definiert sind.

Es gibt zwei prinzipielle Arten von Ereignissen. *Externe* Ereignisse werden durch den Empfang einer Nachricht aus der Leitdomäne initiiert, z.B. die Anforderung einen als Entscheidungskontext definierten Auswahldienst auszuführen. Die für die Interaktion mit der Leitdomäne notwendigen Dienste für das Empfangen und Versenden verschiedener Nachrichtentypen werden in einer *Kommunikationsschnittstelle* gekapselt.

Interne Ereignisse werden entweder vom Kontextausführer (s.u.) generiert, um die Ergebnisse einer Kontextausführung (Aktionsresultate oder Benutzerauswahl) bekanntzugeben, oder vom Kontexterkenner nach der Identifikation eines Kontextes durch Benutzerinteraktionen erzeugt. Handelt es sich bei dem vom Kontexterkenner identifizierten Kontext um einen Automatisierungs- oder Entscheidungskontext, der vom Werkzeug selbst angeboten wird, wird der Kontextausführer des Werkzeugs aktiviert; ansonsten wird die Ausführung des Kontexts von der Leitdomäne angefordert.

4.2 Der Kontextmanager

Der Kontextmanager hat drei wesentliche Aufgaben: (1) Während der Startphase des Werkzeuges lädt er die für das Werkzeug relevanten Kontextinformationen aus dem Prozeßrepository und initialisiert das Werkzeug entsprechend. (2) Das Subsystem *Kontextausführer* koordiniert die Ausführung von Automatisierungs- und Entscheidungskontexten. (3) Der *Kontexterkenner* vergleicht Benutzerinteraktionen (Selektion von Produkten und Aktivierung von Kommandoelementen) mit den Kontextdefinitionen und meldet identifizierte Kontexte an den Zustandsmanager.

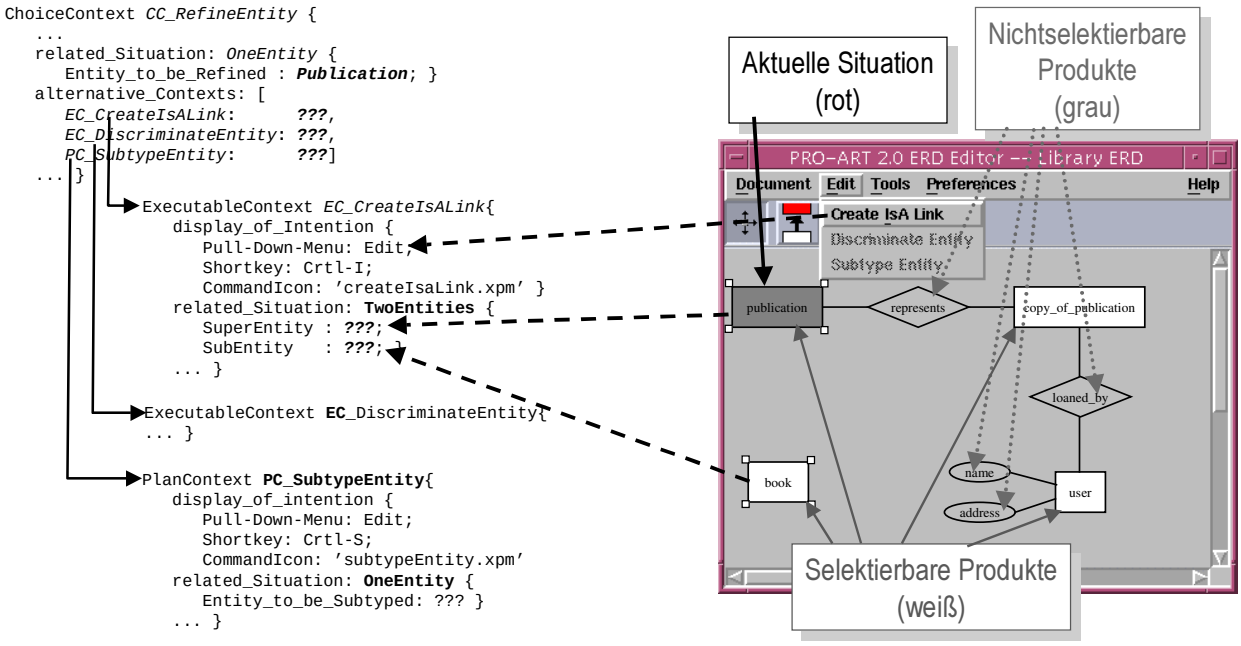

Abb. 5 : Ausführung eines Entscheidungskontexts und Auswahl einer Alternative.

***Kontextausführer**:* Der Kontextausführer steuert nach dem Aufruf durch den Zustandsmanager die Ausführung eines angeforderten Automatisierungs- oder Entscheidungskontexts. Zur Laufzeit bindet der Kontextausführer die im Werkzeugmodell assoziierte Werkzeugaktion an den Kontextaufruf. Die Situationsdaten des aktivierten Kontexts werden dabei auf die Parameter der Aktion abgebildet.

Bei der Ausführung eines Entscheidungskontexts paßt der Kontextausführer die Benutzeroberfläche des Werkzeugs an die Alternativen des Entscheidungskontexts und die aktuellen Situationsdaten an. In der Kommandoregion werden nur noch solche Menüoptionen und Kommandoicons dargestellt, die zu Intentionen der Alternativen des Entscheidungskontexts gehören. In der Produktregion werden alle Produktteile der aktuellen Situationsinstanz hervorgehoben dargestellt, um die Aufmerksamkeit des Benutzers auf diese Produktteile zu lenken. Aus den mit den Alternativen assoziierten Situationstypen lassen sich die Produkttypen ableiten, die zu einer gültigen Produktkonstellationen beitragen können. Instanzen solcher Produkttypen werden als selektierbar gesetzt, während die nicht in Frage kommenden Produkte als nicht selektierbar gekennzeichnet werden. Die Anpassung der Benutzeroberfläche geschieht über zwei Adapter-Komponenten (Produkttabelle und Intentionstabelle), um die Unabhängigkeit und leichte Austauschbarkeit von spezifischen Oberflächen-Toolkits zu gewährleisten.

***Kontexterkenner**:* Während der Ausführung eines Entscheidungskontexts selektiert und deselektiert der Benutzer Produktteile und aktiviert Kommandoelemente. Die Aufgabe des Kontexterkenners ist es, diese Aktionen mit den Aktivierungsbedingungen der alternativen Kontexte zu vergleichen, d.h., die selektierten Produktteile den Situationen und die Kommandoaktivierungen den Intentionen der möglichen Alternativen zuzuordnen. Sobald die selektierten Produktteile und das selektierte Kommandoelement mit der Definition eines Alternativkontexts übereinstimmen (d.h. ein Kontext ausgewählt wurde), meldet der Kontexterkenner den ausgewählten Kontext an den Zustandsmanager. Man beachte, daß die Alternativen nicht nur werkzeuginterne Automatisierungs- und Entscheidungskontexte, sondern auch von anderen Werkzeugen angebotene Kontexte sowie Plankontexte beinhalten können.

4.3 Ein Beispiel

Ausführung eines Auswahldienstes: Wir zeigen zunächst, wie ein mit Hilfe des Implementierungsframeworks entwickelter ER-Editor einen Auswahldienst ausführt, d.h. seine Benutzeroberfläche an die Prozeßdefinition eines Entscheidungskontexts anpaßt.

Abb. 5 zeigt auf der rechten Seite den ER-Editor bei der Ausführung des Auswahldiensts *CC_RefineEntity* bezüglich der selektierten Entität *Publication*. Auf der rechten Seite ist ein in C++-ähnlicher Syntax notierter Ausschnitt der Laufzeitstruktur des Kontextmanagers dargestellt. Für den aktuell aktiven Entscheidungskontext CC_RefineEntity gibt diese Laufzeitstruktur an, daß die aktuelle Situationsinstanz aus dem Produkt *Publication* besteht und daß es drei mögliche Alternativen gibt. Die Teile der Laufzeitstruktur, die erst nach Benutzerinteraktion vom Kontexterkenner an bestimmte Werte gebunden werden, sind mit Fragezeichen dargestellt.

Gemäß der Definition des Entscheidungskontexts (linke Seite von Abb. 5) stehen dem Benutzer für die Verfeinerung der Entität drei alternative Vorgehensweisen zur Auswahl: (1) Hinzufügen einer Spezialisierungsbeziehung zu einer existierenden Entität (`EC_CreateIsALink`), (2) Hinzufügen eines diskriminierenden Attributs (`EC_DiscriminateEntity`), und (3) Erzeugen neuer Entitätstypen als Subtypen von *Publication* (`PC_SubtypeEntity`). Hier-

bei entsprechen die beiden ersten Dienste elementaren Werkzeugdiensten (Automatisierungskontexte), während letzterer ein Prozeßfragment (Plankontext) darstellt. Gemäß dem integrierten Prozeß- und Werkzeugmodell werden vom Kontextmanager (über die Kommando-Tabelle) diese Alternativen auf das Edit-Menü sowie auf die Kommandoicons in der Iconleiste abgebildet, während alle anderen Menüoptionen ausgeschaltet werden. Als aktuelle Situationsinstanz wird die Entität *Publication* hervorgehoben (rot) dargestellt. Die mit den Alternativen assoziierten Situationstypen definieren die selektierbaren Produktkonstellationen. In diesem Fall kommen nur noch Entitätstypen (als der Bestandteile der möglichen Situationen *OneEntity* und *TwoEntities*) in Frage. Dagegen schaltet der Kontextausführer die Selektierbarkeit von Attributen und Beziehungstypen aus und stellt sie grau dar, da diese aufgrund der Prozeßdefinition nicht zu gültigen Situationen der aktuell möglichen Alternativkontexte beitragen können.

Erkennen einer Kontextaktivierung: Nachdem der Benutzer die Menüoption *Create IsA Link* ausgewählt hat, vergleicht der Kontexterkenner die ausgewählten Entitätstypen *Publication* und *Book* mit den Situationstypen und die aktivierte Menüoption mit den assoziierten Intentionen. In unserem Beispiel korrespondieren die Benutzerinteraktionen mit der Alternative *EC_CreateIsALink*. Da dieser Kontext einem werkzeuginternen elementarem Dienst zugeordnet wurde, fordert der Zustandsmanager dessen Ausführung über den Kontextausführer an. Hätte es sich bei dem erkannten Kontext um ein komplexes Prozeßfragment gehandelt, hätte der Zustandsmanager eine entsprechende Anforderungsnachricht an die Prozeßmaschine gesendet.

4.4 Implementierung

Die generischen Teile der Architektur wurden als objektorientierte Implementierungsframeworks in C++ auf zwei verschiedenen Plattformen implementiert (Sun Solaris Unix und Windows 95/NT).

Das *Prozeßrepository* dient der persistenten Verwaltung des Arbeitsplatzmodells, das alle Prozeß-, Werkzeug- und Produktmodelle umfaßt. Es wurde auf Basis eines relationalen DBMS (Sybase 11) realisiert. Das Arbeitsplatzmetamodell wurde in ein aus insgesamt 27 Tabellen bestehendes relationales Schema transformiert. *Metamodellierungswerkzeuge* erleichtern die Erstellung und Wartung von Produktmodellen, Prozeßfragmenten, Werkzeugmodellen und deren Beziehungen untereinander.

Das *generische Werkzeug-Implementierungsframework* stellt Bibliotheken bereit für die Kommunikationsschnittstelle, das Zustandsmanagement, das Kontextmanagement und die Anpassung der Benutzeroberfläche. Die Bibliotheken für das Zustands- und Kontextmanagement umfassen etwa 70.000 Zeilen C++-Code. Der Benutzeroberflächenadapter besteht insgesamt aus 15.000 Zeilen Code und bildet z.B. die im Arbeitsplatzmodell definierten Kommandoelemente auf die Menüklassen des von uns verwendeten Oberflächen-Toolkits ILOG Views ab. Wir haben auf eine sorgfältige Trennung der Oberflächenklassen von den Kontext- und Zustandsmanagement-Bibliotheken geachtet, um eine einfache Portierung an andere Oberflächen-Toolkits zu ermöglichen.

Das in diesem Artikel nicht näher erläuterte generische Implementierungsframework für Prozeßmaschinen stellt analog zum Werkzeug-Framework Bibliotheken für den Nachrichtenaustausch und das Zustandsmanagement zur Verfügung. Außerdem verfügt es über eine Komponente, die den nächsten auszuführenden Kontext bei der Plankontextausführung ermittelt. Dieses stützt sich auf den Interpreter für die spezifische Plankontextsprache ab. Die generischen Anteile der Implementierungsframeworks umfassen etwa 35.000 Zeilen Code, wobei insbesondere beim Zustandsmanagement die entsprechenden Basisklassen des Werkzeug-Frameworks wiederverwendet wurden.

Die *Kommunikationsinfrastruktur* wurde in zwei Versionen realisiert: (1) auf Basis von Sockets für Solaris und Windows 95/NT, und (2) mit Hilfe der komfortableren Dienste von Sun's ToolTalk [35] für Solaris. Oberhalb dieser Basismechanismen wurde ein prozeßsensitiver Nachrichtenvermittler realisiert, der Wissen über die laufenden Werkzeuginstanzen wartet und die Nachrichtenverteilung gemäß der Kontext-Werkzeug-Zuordnung im Arbeitsplatzmodell vornimmt.

5 Eine Vorgehensweise zur Entwicklung PRIME-basierter Umgebungen

Für die Entwicklung einer prozeßintegrierten Modellierungsumgebung auf der Basis der PRIME-Modellierungskonzepte und der Implementierungsframeworks haben wir eine aus sechs Schritten bestehende Vorgehensweise entwickelt. Wie in Tabelle 1 dargestellt, betreffen vier der sechs Schritte hauptsächlich die Modellierungsebene, während zwei Schritte Implementierungsarbeiten erfordern. Im folgenden erläutern wir die Schritte im Detail, die natürlich auch teilweise parallel sowie iterativ ausgeführt werden können.

	Aufgabe	Mod.ebene	Impl.ebene
1	Auswahl Sprache zur Plankontextdefinition	X	
2	Def. method.Unterstützung im Prozeßmodell	X	
3	Def. Werkzeugressourcen im WZ.-Modell	X	
4	Def. Arbeitsplatzmodell	X	
5	Impl. domänenspezifische WZ-Funktionalität		X
6	Integration des Prozeßspracheninterpreters		X

Tabelle 1: Vorgehensweise zur Entwicklung einer PRIME-basierten Umgebung

Schritt 1 – Auswahl der Sprache zur Plankontextdefinition: Plankontexte bestehen aus einer Menge von Kontexten beliebigen Typs und spezifizieren unter diesen eine Ausführungsreihenfolge. Da das Prozeßmetamodell die Wahl einer spezifischen Sprache zur Kontrollflußdefinition offenläßt, muß zunächst eine passende Sprache ausgewählt werden. Die Auswahl hängt von der erforderlichen Ausdrucksstärke ab. Während für hauptsächlich sequentielle Abläufe Zustandsübergangsdiagramme hinreichend sind, ist i.a. eine vollwertige imperative Programmiersprachen für Abläufe mit vielen konditionalen Verzweigungen und Schleifen geeigneter. Um die verschachtelte Definition von Kontexten (innerhalb von Plan- und Entscheidungskontexten) zu ermöglichen, müssen die drei Kon-

texttypen auf Templates innerhalb der ausgewählten Sprache abgebildet werden. Wir haben z.B. das kontextbasierte Prozeßmetamodell sowohl in C++ abgebildet als auch in die Petrinetz-basierte Prozeßmodellierungssprache SLANG [1] eingebettet, indem wir spezielle Plätze für Situationen und Intentionen eingeführt und Subnetz-Templates für die drei Kontextarten definiert haben (siehe [Pohl, 1996]).

Schritt 2 – Definition der methodischen Unterstützung im Prozeßmodell: Der Methodenmodellierer definiert die methodische Unterstützung unter Verwendung der Konzepte des kontextbasierten Prozeßmetamodells. Basierend auf einem Produktmodell werden relevante Produktkonstellationen identifiziert und als Situationen definiert. Durch Definition der in den jeweiligen Situationen zu verfolgenden Intentionen werden die zu unterstützenden Kontexte festgelegt. Als nächstes legt der Methodenmodellierer die Art der Unterstützung bei der Aktivierung eines Kontexts fest. Wenn der Kontext hinreichend feingranular ist, um durch eine elementare Aktion implementiert zu werden, wird er als Automatisierungskontext modelliert und zur betreffenden Aktion assoziiert. Entscheidungspunkte, die der Benutzerintervention bedürfen, werden durch Angabe der alternativen Kontexte als Entscheidungskontexte modelliert. Komplexe Methodenfragmente, die aus mehreren Teilschritten bestehen, werden als Plankontexte in der in Schritt 1 ausgewählten Sprache spezifiziert.

Schritt 3 – Definition der Werkzeugressourcen im Werkzeugmodell: Das Werkzeugmodell dient als high-level-Spezifikation der erforderlichen Werkzeugfunktionalitäten. Die benötigten Werkzeugkategorien sind hauptsächlich von der Struktur des zugrundeliegenden Produktmodells abhängig. Für jedes Teilmodell (Dokumenttyp) wird eine Werkzeugkategorie definiert, z.B. ER-Editor für die Bearbeitung von ER-Diagrammen. Die Werkzeugkategorie wird dann gemäß dem Prozeßmodell mit allen Produkt(teil)en des jeweiligen Dokumenttyps und den darauf operierenden Aktionen verknüpft. Weiterhin werden die möglichen Darstellungsarten eines Produkttypen sowie die vom Werkzeug angebotenen Kommandoelemente modelliert.

Schritt 4 – Definition des Arbeitsplatzmodells: Aufbauend auf der Beziehung zwischen Werkzeugkategorien und den von diesen angebotenen Produkten und Aktionen, erfolgt die Zuweisung von Automatisierungskontexten und Entscheidungskontexten zu Werkzeugkategorien über die Assoziationstypen des Arbeitsplatzmodells. Die Modellierung von Werkzeugkategorien und die Verknüpfung mit Automatisierungs- und Entscheidungskontexten iteriert solange, bis es im Prozeßmodell keine Automatisierungs- und Entscheidungskontexte ohne Assoziation zu einer Werkzeugkategorie mehr gibt.

Schritt 5 – Implementierung der domänenspezifischen Werkzeugfunktionalität: Das generische Implementierungsframework reduziert den Realisierungsaufwand für die modellierten Werkzeuge signifikant. Die generischen Mechanismen erledigen den allgemeinen Kontrollfluß innerhalb eines Werkzeugs, den Nachrichtenaustausch mit der Prozeßmaschine und die Kontextausführung und Kontexterkennung. An definierten Stellen des Frameworks muß der Werkzeugbauer lediglich die spezifischen Werkzeugaktionen, die eventuell erforderlichen Oberflächenobjekte für die darzustellenden Produkte sowie die produktspezifischen Datenbankoperationen hinzufügen. Dies geschieht in den meisten Fällen durch Spezialisierung von Adapterklassen des PRIME-Frameworks. Eine typische Werkzeugaktion erfordert etwa 1-2 Seiten C++-Code. Der Aufwand für ein neues Oberflächenobjekt zur Produktdarstellung hängt stark vom benutzten Oberflächen-Toolkit, übersteigt aber nach unseren Erfahrungen selten zwei Seiten Code.

Schritt 6 – Integration des Prozeßspracheninterpreters: Für die in Schritt 1 ausgewählte Sprache zur Plankontextspezifikation muß ein entsprechender Interpreter in das generische Prozeßmaschinenframework eingebettet werden. Dies geschieht auf ähnliche Weise wie die Anreicherung des generischen Werkzeugframeworks um spezifische Funktionalität.

6 Erfahrungen

Die oben beschriebene Vorgehensweise und damit verbunden das Implementierungsframework des PRIME-Ansatzes wurde für die Entwicklung zweier prozeßintegrierter Modellierungsumgebungen verwendet: PRO-ART 2.0 im Bereich Requirements Engineering und TECHMOD für die Modellierung verfahrenstechnischer Prozesse. Die Umgebungen wurden im Rahmen kleinerer Studien (mit Studenten bzw. Verfahrenstechnik-Ingenieuren) validiert. Die dabei gewonnenen Erfahrungen ermöglichen eine Bewertung des Ansatzes aus drei verschiedenen Blickwinkeln:

Benutzer: Aus Benutzersicht wurde die Anpassung des Werkzeugverhaltens an die aktuellen Prozeßdefinitionen und –situationen als durchweg sehr hilfreich empfunden. Durch die automatische Anpassung sowie die in diesem Papier nicht näher erläuterten Hilfedienste wurden die Benutzer über die jeweils aktuellen Prozeßdefinitionen informiert und mußten diese nicht offline in beispielsweise projektspezifischen Vorgehensbeschreibungen nachlesen.

Die einheitliche Aktivierung von werkzeuginternen Diensten, Diensten von anderen Werkzeugen und Prozeßfragmenten in den Werkzeugen wurde als großer Fortschritt angesehen. Die Ausführung eines Dienstes von einem anderen Werkzeug wurde vom Benutzer oft nicht einmal bemerkt. Mehrere Benutzer erwähnten, daß sie gelegentlich ein ihnen unbekanntes Prozeßfragment aktivierten. Die Möglichkeit, sich über die Hilfedienste vorab die Prozeßfragmente erklären zu lassen und die angebotene Unterstützung ggf. ablehnen zu können, war hierbei sehr wichtig. Dieser Vorteil von prozeßintegrierten Werkzeugen erwies sich vor allem bei der Aufzeichnung von projektspezifischen Trace-Information als sehr vorteilhaft (siehe [28]).

Werkzeugentwickler: Die Realisierung der insgesamt zwölf Werkzeuge für die beiden Umgebungen PRO-ART 2.0 und TECHMOD zeigte, daß die Konzepte der Werkzeugmodelle und die Verknüpfung zu Automatisierungs- und Entscheidungskontexten zur Definition prozeßinte-

grierter Werkzeuge ausreichen. Die Definition komplexer Abläufe innerhalb des Prozeßmodells zwang die Werkzeugentwickler, Prozeßwissen explizit zu machen, und so das „Prozeß-im-Werkzeug"-Syndrom [26] zu vermeiden. Die Werkzeugimplementierung wurde durch das Implementationsframework signifikant erleichtert, da wesentliche Teile der Architektur schon als wiederverwendbare Komponenten vorlagen. Es zeigte sich zudem, daß die Entwickler die Funktionalität eines Werkzeugs erweitern konnten, ohne sich in die Struktur dieses Werkzeuge einzuarbeiten bzw. es selbst implementiert zu haben. Dies wurde vor allem durch die einheitliche Architektur und die vollständige Kapselung des Kontrollflusses ermöglicht. Die Architektur zwang die Entwickler zudem zur modularen Programmierung. Durch die Kontextgebundenheit von Benutzereingaben bei Entscheidungskontexten reduzierte sich der Programmieraufwand für Konsistenzchecks, da mögliche Fehleingaben des Benutzers dadurch stark reduziert werden.

***Methodenmodellierer**:* Die drei Kontexttypen unterstützten den Modellierer bei der Strukturierung von Prozeßmodellen, und zwar unabhängig von einer konkreten Prozeßmodellierungssprache wie beispielsweise Petri-Netze. Im Unterschied zu den meisten anderen Modellierungsansätzen wurde der Methodenmodellierer angehalten, Entscheidungen explizit zu definieren. Die Notwendigkeit, die Werkzeugfunktionalitäten in Modellen zu spezifizieren, veranlaßte die Modellierer, intensiver über die geeignete Granularität von Werkzeugfunktionalitäten nachzudenken. Darüber hinaus versetzten die Werkzeugmodelle den Modellierer in der Lage, bereits bei der Prozeßmodelldefinition die vorhandene Werkzeugfunktionalität berücksichtigen zu können. Die Definition von Werkzeug und Prozeßmodellen sowie die integrierten Arbeitsplatzmodelle ermöglichten eine leichte Anpassung der von der Umgebung angebotenen Unterstützung. Besonders in experimentellen Anwendungen und für wenig verstandene Prozesse, in denen ständig neues Wissen über bessere Prozeßdurchführung gewonnen wird, ist die Aufwandsminimierung für die Anpassung der Unterstützung essentiell.

7 Zusammenfassung und Ausblick

Ausgehend von der Notwendigkeit einer besseren Integration von interaktiven Werkzeuge in prozeßzentrierten Umgebungen und den daraus abgeleiteten Hauptanforderungen haben wir in diesem Aufsatz den Lösungsansatz PRIME vorgestellt. PRIME unterscheidet sich von prozeßzentrierten Umgebungen im wesentlichen durch die automatische Anpassung des Verhaltens der interaktiven Werkzeuge an die Prozeßdefinitionen und an die aktuelle Prozeßsituation sowie die einheitliche Aktivierung von werkzeuginternen und -externen Diensten (jeweils elementare und Auswahldienste) sowie von Prozeßfragmenten.

Wir haben argumentiert, daß für die Prozeßintegration von Werkzeugen die explizite Definition der interaktiven Werkzeugfähigkeiten und der Werkzeugdienste sowie die Integration von Werkzeug- und Prozeßmodellen unbedingt notwendig ist. In PRIME werden die Prozesse unter Verwendung des im ESPRIT BRA Projekt NATURE entwikkelten kontextbasierten Prozeßmodell definiert. Wie in der vorliegenden Arbeit vorgestellt, haben wir dieses Modell mit einem Werkzeugmodell integriert. Das integrierte Arbeitsplatzmodell bildet die Grundlage für die beschriebene PRIME-Architektur für prozeßintegrierte Werkzeuge, die u.a. das integrierte Modell interpretiert und dadurch die Anpassung der interaktiven Werkzeuge sowie die Aktivierung von werkzeugexternen Diensten ermöglicht. Die Architektur wurde durch die Realisierung des PRIME-Implementierungsframeworks, durch die Entwicklung von zwei prozeßintegrierten Umgebungen sowie deren Anwendung in kleinen Fallstudien validiert.

Neben den überwiegend positiven Rückmeldungen (siehe Kapitel 6) zeigten die durchgeführten Fallstudien auch Schwächen in dem derzeitigen Ansatz auf, die teilweise Gegenstand unserer zukünftigen Arbeiten sind. Dazu gehören die werkzeugübergreifende Kontexterkennung und die Sicherung von Datenkonsistenz bei Prozeßabbrüchen und -abweichungen. Darüberhinaus befassen wir uns mit der Integration von Generierungsansätzen, mit deren Hilfe sich Basisfunktionalitäten von Werkzeugen auf der Grundlage eines verfeinerten Werkzeugmodells automatisch erzeugen lassen, sowie mit der Einbindung von (partiell prozeßintegrierbaren) Fremdwerkzeugen durch geeignete „wrapping"-Techniken.

Danksagungen: Für die konstruktiven Diskussionen möchten wir uns an dieser Stelle ganz herzlich bei unseren Kollegen R. Klamma und S. Zlatintsis sowie bei den studentischen Hilfskräften S. Brandt, S. Ewald, M. Hoofe, T. Rötschke, K. Schreck und W. Thyen bedanken. Ohne das überdurchschnittliche Engagement der Hilfskräfte wäre die Realisierung des PRIME-Implementierungsframeworks sowie der beiden Modellierungsumgebungen TECHMOD und PROART 2.0 und die damit gewonnen Erfahrungen nicht möglich gewesen.
Die dieser Veröffentlichung zugrundeliegenden Arbeiten wurden innerhalb des DFG-Projektes 445-1 „Prozeßintegration von Modellierungsarbeitsplätzen" und des SFB 476 IMPROVE durchgeführt.

Literaturreferenzen

[1] M. Anderson und P. Griffiths. *The Nature of the Software Process Modelling Problem is Evolving.* In Proc. of the 3rd European Workshop on Software Process Technology, EWSPT '94, LNCS 772, pp. 31-34, 1994.

[2] Bandinelli, S.; Baresi, L.; Fuggetta, A.; Lavazza, L.; *Experiences in the Implementation of a Process-Centered Software* Engineering *Environment Using Object-Oriented Technology*; In: Theory and Practice of Object Systems, 1(2):115 – 131, 1995

[3] S. Bandinelli, E. Di Nitto und A. Fuggetta. *Supporting Cooperation in the SPADE-1 Environment.* IEEE Transactions on Software Engineering 12, 12 (1996), pp. 841-865.

[4] Barghouti, N.S.; Krishnamurty, B.; *An Open Environment for Process Modelling and Enactment*. In: W. Schäfer, Editor, Proc. of the 8th Intl. Software Process Workshop: State of the Practice in Process Technology, Seite(n) 33-36, Wadern, Deutschland, März 1993. IEEE Computer Society Press.

[5] Belkhatir, N.; Estublier, J.; Melo, W.L.; *Software Process Model and Workspace Control in the ADELE System*. In: L. Osterweil, Editor, Proc. of the 2nd Intl. Conference on the

Software Process, Seite(n) 2-11, Berlin, Deutschland, Februar 1993. IEEE Computer Society Press.

[6] K. Brockschmidt. *Inside OLE*, Second Edition. Microsoft Press, Redmond WA, 1995.

[7] M. Cagan. *The HP Softbench Environment: An Architecture for a New Generation of Software Tools*. Hewlett Packard Journal, 41(3):36-47, June, 1990.

[8] R. Conradi, M. Hagaseth, J.-O. Larsen, M. Nguyen, B. Munch, P. Westby, W. Zhu, M. Jaccheri and C. Liu. *EPOS: Object-Oriented Cooperative Process Modelling.* In A. Finkelstein, J. Kramer and B. Nuseibeh (Eds.),Software Process Modelling and Technology, pp. 33-70. RSP, London, 1994.

[9] Curtis, B.; Kellner, M.I.; Over, J.; *Process Modelling*. In: Communications of the ACM, 35(9):75 – 90, September 1992

[10] W. Deiters und V. Gruhn. *The FUNSOFT Net Approach to Software Process Management.* Intl. Journal of Software Engineering and Knowledge Engineering 4, 2 (1994).

[11] M. Dowson und C. Fernström. *Towards Requirements for Enactment Mechanisms.* In B. Warboys (Ed.), Proc. of the 3rd Europ. Workshop on Software Process Technology, No. 772, LNCS, pp. 90-106, Villard de Lans,Frankreich, February 1994. Springer-Verlag.

[12] M. Dowson. *Consistency Maintenance in Process Sensitive Environments.* In Proc. of the Process Sensitive Software Engineering Environments Architectures Workshop, Boulder, Colorado, USA, September 1992.

[13] ECMA-NIST. A Reference Model for Frameworks of Software Engineering Environments. No. TR/55 Version 3. ECMA & NIST, 1993.

[14] W. Emmerich. *Tool Construction for Process-Centred Software Development Environments based on Object Databases.* PhD thesis, University of Paderborn, Paderborn, Germany, 1995.

[15] C. Fernström und L. Ohlsson. *Integration Needs in Process-Enacted Environments.* In Proc. of the 1st Intl. Conf. on the Software Process, pp. 142-158, 1991.

[16] C. Fernström. *Process WEAVER: Adding Process Support to Unix.* In Proc. of the 2nd International Conference on Software Processes, pp. 12-26, Los Alamitos, CA, USA, 1993.

[17] Finkelstein, A.; Kramer, J.; Nuseibeih, B., Editoren; *Software Process Modelling and Technology*. Nummer 3 in Advanced Software Development Series. J. Wiley & Sons Inc., Chichester, West Sussex, England, 1994.

[18] A. Fuggetta und C. Ghezzi. *State of the Art and Open Issues in Process-Centered Software Engineering Environments.* Journal of Systems and Software 26 (1994), pp. 53-60.

[19] D. Garlan und E. Ilias. *Low-cost, Adaptable Tool Integration Policies for Integrated Environments.* In Proc. of the 4th ACM SIGSOFT Symposium on Software Development Environments, Volume 15, 1990.

[20] GOODSTEP-Team. *The GOODSTEP Project: General Object-Oriented Database for Software Engineering Processes.* In Proc. of the Asia-Pacific Software Engineering Conference, pp. 410-420, Tokyo, Japan, 1994.

[21] D. Harel. *StateCharts: A Visual Formalism for Complex Systems.* Science of Computer Programming, 8:231-274, 1987.

[22] P. Heimann, G. Joeris, C.-A. Krapp und B. Westfechtel. *DYNAMITE: Dynamic Task Nets for Software Process Management.* In Proc. of the 18th Int. Conf. on Software Engineering, pp. 331-341, 1996.

[23] G. Junkermann, B. Peuschel, W. Schäfer und S. Wolf. *MERLIN: Supporting Cooperation in Software Development Through a Knowledge-Based Environment.* In A. Finkelstein, J. Kramer and B. Nuseibeh(Eds.), Software Process Modelling and Technology, pp. 103-130. RSP, London, 1994.

[24] J. Lonchamp. *An Assessment Exercise.* In A. Finkelstein, J. Kramer und B. Nuseibeh (Eds.), Software Process Modelling and Technology, pp. 335-356. RSP, London, 1994.

[25] T. Mochel, A. Oberweis und V. Sänger. *Income/star: The petri net simulation concepts.* Journal of Mathematical Modelling and Simulation in Systems Analysis 13 (1993), pp. 21-36.

[26] C. Montangero. *The Process in the Tool Syndrome: is it becoming worse?* In Proc. of the 9th Intl. Software Process Workshop, pp. 53-56, Arlie, Virginia, USA, October 1994. IEEE Computer Society Press.

[27] OMG. *CORBA: Architecture and Specification.* Object Management Group, Inc., 1995.

[28] K. Pohl, R. Dömges: *Towards Modell-Driven Trace Capture*. In Proc. of the 9th Intl. Conference on Advanced Information Systems Engineering, Barcelone, June, Springer Verlang, LNCS, 1997.

[29] K. Pohl, R. Dömges und M. Jarke. *Decision Oriented Process Modelling.* In Proc. of the 9th Intl. Software Process Workshop, pp. 124-128, Arlie, Virginia, USA, October 1994. IEEE Computer Society Press.

[30] K. Pohl, R. Klamma, K. Weidenhaupt, R. Dömges, P. Haumer und M. Jarke. *A Framework for Process Integrated Tools.* Technical report, RWTH Aachen, 1996.

[31] K. Pohl, Weidenhaupt, K.*; A Contextual Approach for Process-Integrated Tools*. In: Proc. of the 6th ESEC/FSE Conference, Zürich, Springer-Verlag, 1997.

[32] K. Pohl. *Process Centered Requirements Engineering.* RSP marketed by J. Wiley & Sons Ltd., England, 1996.

[33] S. P. Reiss. *Connecting Tools Using Message Passing in the FIELD Environment.* IEEE Software 4, 7 (July 1990), pp. 57-67.

[34] C. Rolland und G. Grosz. *A General Framework for Describing the Requirements Engineering Process.* In Proc. of the Intl. Conf. on Systems, Man, and Cybernetics, San Antonio, Texas, USA, October 1994. IEEE Computer Society Press.

[35] SunSoft. *The ToolTalk Service (White Paper).* Technical report, SunSoft Inc., June 1991.

[36] S. Sutton und L. Osterweil. APPL/A: A Language for software process programming. *ACM Transactions on Software Engineering and Methodology*, 4(3):221-286, Juli 1995.

[37] I. Thomas and B. A. Nejmeh. *Definitions of Tool Integration for Environments.* IEEE Software 8, 2 (1992), pp. 29-35.

[38] X. Wang und P. Loucopoulos. *The Development of Phedias: a CASE Shell.* In Proc. 7th. Int. Workshop on CASE, Toronto, Canada, pp. 122 - 131. IEEE Computer Society Press, 1995.

[39] A. I. Wasserman. *Tool Integration in Software Engineering Environments.* In F. Long (Ed.), Proc. of the Intl. Workshop on Software Engineering Environments, pp. 137-149, Berlin, Germany, 1990. Springer-Verlag.