Universität Ulm
Fakultät für Informatik

Diplomarbeit

Workflow-Anforderungen und -Konzepte für das Konfigurationsmanagement eingebetteter Systeme im Automotive-Bereich

vorgelegt von

Ulrich Bestfleisch

April 2005

Betreuer und Gutachter

Dr. Joachim Herbst (DaimlerChrysler, Forschungszentrum Ulm)

Prof. Dr. Manfred Reichert (Universität Twente, ehem. Universität Ulm, Abteilung DBIS)

in Kooperation mit

DaimlerChrysler

Research and Technology – Data and Process Management (REI/ID)

Vorwort

Mit Abgabe der vorliegenden Arbeit endet gleichzeitig auch mein Studium. Dies nehme ich zum Anlass, all denjenigen meinen Dank auszusprechen, die an meinem erfolgreichen Studium Anteil haben. Dies sind zuallererst meine Eltern, durch deren großzügige Unterstützung ein zügiges Studium erst möglich gewesen ist. Mein Dank gilt jedoch auch den mit mir befreundeten Kommilitonen, ohne die meine Studienzeit um einige reizvolle Facetten ärmer wäre.

Von meinen Betreuern Joachim Herbst und Manfred Reichert habe ich Unterstützung bei meiner Arbeit bekommen, wie sie bei Weitem nicht selbstverständlich ist. Ebenso zu Dank verpflichtet bin ich auch meiner Freundin Sabrina, deren kritisches Auge bei der Korrektur äußerst hilfreich gewesen ist.

Ulm im April 2005

Ulrich Bestfleisch

Inhaltsverzeichnis

1 Einleitung

1.1 Problembeschreibung

In einem modernen Fahrzeug sind heutzutage zunehmend mehr elektrische und elektronische Komponenten enthalten. Bis zu 70 elektronische Steuergeräte tauschen dabei über kilometerlange Kabelstränge tausende von Nachrichten miteinander aus [KnSc04]. Der Bereich *Elektrik/Elektronik* (E/E) trägt wesentlich zu Innovationen und damit zur Wertschöpfung im Automobilbereich bei, doch mit zunehmender Komplexität der *eingebetteten Systeme*, kürzeren Entwicklungszyklen und der steigenden Variantenvielfalt sind Qualitätsprobleme entstanden.

Abbildung 1.1: Steuergerätenetzwerk eines Mercedes-Benz-Fahrzeugs [Gr05]

Das Problem ist von der Automobilindustrie erkannt worden, und es werden große Anstrengungen unternommen, Entwicklungsprozesse zu verbessern [KnSc04]. Ziel ist es, die komplexen und nicht immer offensichtlichen Abhängigkeiten zwischen den elektrischen und elektronischen Komponenten (E/E-Komponenten) in der Entwicklung besser zu beherrschen und die Entwicklungszyklen zu verkürzen.

Eine zentrale Rolle für die Qualität spielen Integration, Test und Freigabe von E/E-Komponenten und das damit verbundene Management von Konfigurationen.

Handlungsbedarf besteht aus Sicht der Informationstechnologie u.a. in zwei zentralen Themenbereichen:

- *Integration von Produktdatenbanken für das Konfigurationsmanagement:* Die Produktdaten der E/E-Komponenten sind in einem heterogenen Verbund von IT-Systemen verteilt. Diese müssen integriert werden, um Konfigurationen abbilden zu können.
- *Workflow-Unterstützung der Freigabeprozesse:* Durch Workflow-Management-Technologie sollen die Prozessschritte bei Test und Freigabe von Konfigurationen aktiv unterstützt werden. Damit sollen die korrekte Ausführung der Prozesse gewährleistet, die Qualität erhöht und die Prozesszyklen verkürzt werden.

1.2 Aufgabenstellung

Die Aufgabe dieser Diplomarbeit ist es, die zentralen Anforderungen an solche IT-Systeme zu ermitteln, die parallele Freigabeprozesse basierend auf Workflow-Management-Technologie unterstützen sollen. Auf Grundlage dieser Anforderungen soll deren Umsetzbarkeit mit Ansätzen aus der Wissenschaft und dem Stand der Technik geklärt werden. Anschließend sollen für konzeptuell noch nicht adäquat umsetzbare Anforderungen eigene Lösungskonzepte entwickelt werden.

1.3 Gliederung der Arbeit

Die Arbeit gliedert sich folgendermaßen: nach Erläuterung der notwendigen Hintergrundinformation in Kapitel 2 wird in Kapitel 3 der Ist- und Soll-Zustand bei Test und Freigabe eingebetteter Systeme im Elektrik/Elektronik-Bereich dargestellt. Kapitel 4 fasst die Grundlagen des Workflow-Management zusammen. In Kapitel 5 werden die Ergebnisse der Anforderungsanalyse bezüglich einer Workflow-Unterstützung von Freigabeprozessen ausführlich beschrieben. Im daran anschließenden Kapitel 6 wird der Stand der Wissenschaft für wichtige und durch gängige Workflow-Technologie nicht umsetzbare Anforderungen diskutiert, bevor in Kapitel 7 die in dieser Arbeit entstandenen Lösungskonzepte vorgestellt werden. Die Arbeit schließt in Kapitel 8 mit der Zusammenfassung der wichtigsten Ergebnisse und einem Ausblick auf noch offene Fragestellungen.

2 Hintergrund

2.1 Fahrzeugentwicklungsprozess

Als Beispiel für einen typischen Fahrzeugentwicklungsprozess in der Automobilindustrie soll der Produktentstehungsprozess bei DaimlerChrysler [EP04, Ma04] herangezogen werden. Dieser standardisierte Prozess gliedert sich in vier aufeinander folgende Hauptphasen.

In der ersten Phase - der Strategiephase – werden die grundlegend strategischen Entscheidungen für eine Fahrzeugbaureihe gefällt. Das Ergebnis dieser Phase ist ein unabgestimmtes Konzeptheft, in dem alle wichtigen Produktanforderungen und zu erreichenden Ziele beschrieben sind.

In der darauf folgenden Produktkonzeptionsphase werden die Ziele detailliert definiert, sowie die Anforderungen abgestimmt und auf Konsistenz geprüft, so dass am Ende ein widerspruchsfreies, abgestimmtes Konzeptheft für eine Fahrzeugbaureihe vorliegt.

Dieses Dokument liefert die Grundlage für die folgende, aufwändigste Phase in der Produktentwicklung, in der das Fahrzeug Schritt für Schritt bis zum Erreichen der Serienreife entwickelt wird (Fahrzeugphase). Nach Abschluss dieser Phase beginnt mit der Serienphase die Produktion der Kundenfahrzeuge.

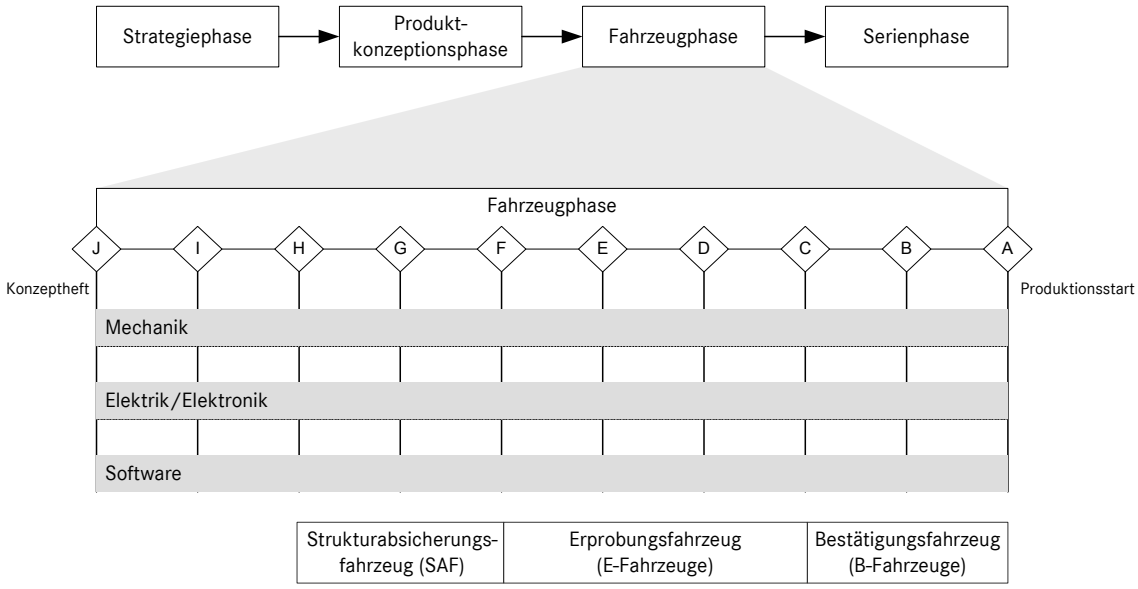

Abbildung 2.1: Der Fahrzeugentwicklungsprozess im Überblick

An der Entwicklung eines Fahrzeugs sind folgende Disziplinen beteiligt:

- In der Disziplin Mechanik werden alle mechanischen Teile wie Rohbau, Türen oder Achsen entwickelt. Die Mechanik ist die klassische Disziplin im Fahrzeugbau.
- In der Elektrik/Elektronik werden die elektrischen und elektronischen Komponenten wie Steuergeräte oder Sensoren entwickelt.
- Eng verzahnt mit der Disziplin Elektrik/Elektronik ist die Software-Entwicklung für die elektronischen Steuergeräte.

Während der Fahrzeugphase arbeiten die einzelnen bei der Fahrzeugentwicklung beteiligten Disziplinen Mechanik, Elektrik/Elektronik (E/E) und Software-Entwicklung im Prinzip parallel und werden an definierten Meilensteinen im Prozess, den sog. „Quality-Gates“, synchronisiert. An einem Quality-Gate wird der bis dahin erreichte Produktreifegrad zur Kontrolle systematisch bewertet und über eine Fortsetzung des Projekts entschieden.

Wenn alle Ist- und Soll-Werte der Messgrößen an einem Quality-Gate übereinstimmen, gilt das Quality-Gate als passiert und der Produktentwicklungsprozess wird weitergeführt. Ansonsten wird bei Vorliegen entsprechender Maßnahmen für Abweichungen das Quality-Gate mit Auflagen passiert. Wenn für eine Abweichung keine Maßnahmen definiert sind, wird das Qualiy-Gate nicht passiert und das Projekt entweder abgebrochen oder mit veränderten Inhalten weitergeführt [EP04].

Während der Fahrzeugphase entstehen Prototypen verschiedener Reifegrade, um das Gesamtfahrzeug realitätsnah testen zu können. Das Strukturabsicherungsfahrzeug (SAF) dient vor allem physischen Überprüfungen (z.B. Crash-Tests). Elektrik und Elektronik wird hier nur in ausgewählten Umfängen eingebaut. Erprobungsfahrzeuge (E-Fahrzeuge) sind hingegen „echte“ Fahrzeuge mit einem für den jeweiligen Zeitpunkt definierten Funktionsumfang. Die Bestätigungsfahrzeuge (B-Fahrzeuge) werden schon mit den Werkzeugen für die Serienproduktion gebaut, dort ist üblicherweise der Funktionsumfang der Elektrik/Elektronik schon komplett [DoKö04].

Typisch für den gesamten Fahrzeugentwicklungsprozess ist, dass dieser nicht nur beim Automobilhersteller alleine, sondern in enger Zusammenarbeit mit Zulieferern stattfindet.

2.2 Produktdatenmanagement

Während des ganzen Entwicklungsprozesses entsteht eine große Menge an Produktdaten, wie CAD-Modelle einzelner Fahrzeugteile, Lastenhefte von Steuergeräten oder elektrische Schaltpläne. Eine zentrale Herausforderung für jeden Automobilhersteller ist die adäquate Verwaltung dieser Daten in IT-Systemen. In diesem Umfeld spricht man von Produktdatenmanagement (PDM).

Im Produktdatenmanagement werden Produktdaten gemäß eines Produzenten-Konsumenten-Modells [Sc02] betrachtet (vgl. Abbildung 2.2): Produzenten von Produktdaten stehen Konsumenten gegenüber, die Daten der Produzenten für ihre Tätigkeiten im Entwicklungsprozess benötigen. Konsumenten produzieren Daten als Ergebnis ihrer Tätigkeit und treten dann wieder selbst für andere als Produzenten von Produktdaten auf. Dabei beschreibt ein Vertrag zwischen Produzent und Konsument die Art und Qualität der vom Konsumenten benötigten Produktdaten.

Doch welcher Zusammenhang besteht nun zwischen einem Produkt und dessen Produktdaten? In [Sc02] wird zunächst zwischen Produkten und Produktkomponenten unterschieden, wobei deutlich gemacht wird, dass es von der Perspektive abhängt, ob etwas als Produkt oder Produkt*komponente* betrachtet wird. So ist beispielsweise ein Fahrzeug aus Sicht des Automobilherstellers das Produkt. Aus Sicht eines Zulieferers, der für eine bestimmte Komponente des Fahrzeugs zuständig ist, stellt jedoch die Produktkomponente das Produkt dar.

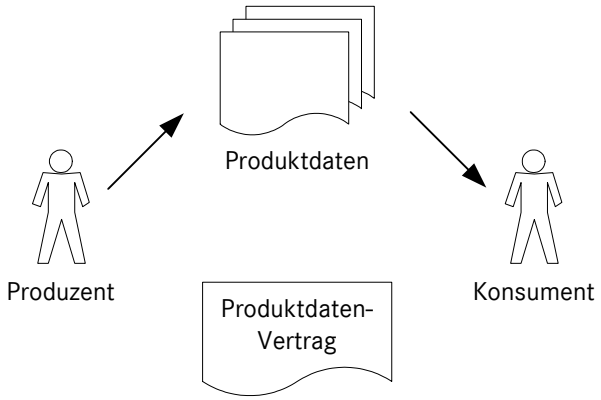

Abbildung 2.2: Produzenten-Konsumenten-Modell für Produktdatenmanagement nach [Sc02]

Wichtig ist auch die Unterscheidung zwischen Produkt bzw. Produktkomponente und beschreibenden Dokumenten [Sc02]. D.h. es gibt auf der einen Seite Daten, die ein Produkt oder eine Produktkomponente quasi als physisches Teil repräsentieren und auf der anderen Seite Dokumente, die dieses Teil näher beschreiben.

Grundsätzlich können alle Produktdaten – Daten von Produkt und Produktkomponenten sowie deren beschreibende Dokumente – im Laufe des Produktentstehungsprozesses versioniert werden. Eine Version ist ein bestimmter Änderungsstand eines Objekts zu einem bestimmten Zeitpunkt. Eine Version zeichnet sich nach [Sc02] vor allem dadurch aus, dass sie jederzeit wiederherstellbar und nicht mehr veränderbar ist.

2.3 Domäne Elektrik/Elektronik und Software im Automobilbereich

Dieser Abschnitt gibt eine Einführung in die Domäne Elektrik/Elektronik im Automobilbereich (einschließlich Software) und einen groben Überblick über den prinzipiellen Entwicklungsprozess.

2.3.1 Grundlegende Begrifflichkeiten

Ein Fahrzeug ist ein komplexes System mit hochgradig voneinander abhängigen Elementen aus der Mechanik, Elektrik, Elektronik und Software. In einem modernen Fahrzeug arbeiten bis zu 70 vernetzte Steuergeräte mit Sensoren und Aktuatoren für die unterschiedlichsten Zwecke zusammen. Beispiele für größtenteils elektronisch umgesetzte Funktionen sind die Steuerung des Motors, die Regelung der Klimaanlage, die Verstellung der Spiegel oder die Zentralverriegelung.

Ein Steuergerät besteht zum einen aus der Steuergerätehardware (Mikrochip, Platine, Stecker und Gehäuse) und zum anderen aus der Steuergerätesoftware (Betriebssystem, Programm und Konfigurationsparameter) [We00].

Als *E/E-Gesamtsystem* eines Fahrzeugs soll im Folgenden die Gesamtheit aller Steuergeräte, Sensoren und Aktuatoren verstanden werden. Steuergeräte, Sensoren und Aktuatoren werden verallgemeinert auch als *E/E-Komponenten* bezeichnet.

Das E/E-Gesamtsystem eines Fahrzeugs besteht aus Sicht der E/E-Entwicklung aus einzelnen *E/E-Systemen*. Unter einem E/E-System versteht man alle Steuergeräte, Sensoren und Aktuatoren, die zu einer bestimmten, „kundenerlebbaren“ Funktion beitragen. Zu einem solchen System können mehrere Steuergeräte gehören, da die Realisierung einer Funktion auf mehrere Steuergeräte verteilt sein kann. Umgekehrt kann ein Steuergerät zu mehreren E/E-Sys-

temen gehören, da auf einem Steuergerät die Funktionalität mehrerer Systeme realisiert sein kann. Aus Gründen der Lesbarkeit wird im Weiteren der Begriff „System“ als Synonym für E/E-System bzw. „Gesamtsystem“ für E/E-Gesamtsystem verwendet.

Zur Illustration der Begriffe mögen die folgenden, stark vereinfachten Beispiele dienen: Betrachtet werden die zwei Systeme „Spiegelverstellung“ und „Außenlicht“.

Die wesentliche Funktion des Systems *„Spiegelverstellung“* (vgl. Abbildung 2.3) besteht darin, dem Fahrer eine elektrische Verstellung der linken und rechten Außenspiegel zu ermöglichen. Sensoren sind hier der Schalter zur Wahl des zu verstellenden Spiegels und der Schalter zum Verstellen des Spiegels. Aktuatoren sind die Motoren zur Spiegelverstellung auf beiden Seiten. Beteiligte Steuergeräte sind das Türsteuergerät der linken Tür (TSL) zur Spiegelmotorsteuerung auf der linken Seite, das Türsteuergerät auf der rechten Seite (TSR) zur Spiegelmotorsteuerung auf der rechten Seite und ein spezielles Steuergerät zur Signalerfassung der Schalter im vorderen Bereich des Fahrzeugs (SAMV). Die Steuergeräte sind untereinander über einen Standard-Kommunikationsbus vernetzt, über den sie Nachrichten austauschen, die der Realisierung des Systems „Spiegelverstellung“ dienen.

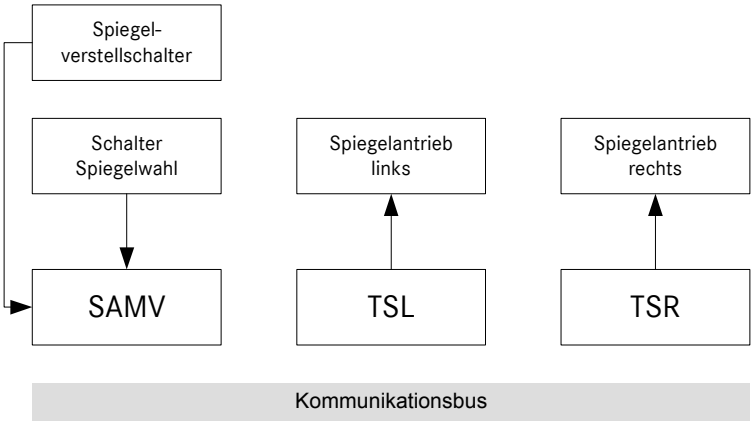

Abbildung 2.3: System Spiegelverstellung

Im System *„Außenlicht“* sind alle Funktionen gebündelt, die das Steuern aller nach außen sichtbaren Beleuchtungseinrichtungen betreffen. Dazu gehören Funktionen wie „Rückfahrlicht“, „Fernlicht“ oder „Richtungsblinken“.

Betrachtet werden sollen hier die beteiligten E/E-Komponenten für die Funktion „Richtungsblinken“ (vgl. Abbildung 2.4): Nach Wahl der Blinkrichtung mit einem Wählhebel sind auf der entsprechenden Seite die Blinker im Seitenspiegel bzw. am Heck aktiv. Sensor ist in diesem Fall der Wählhebel für die Blinkrichtung. Aktuatoren sind die Blinker an den Spiegeln und im Heck, sowie die Blinkrichtungsanzeige im Cockpit. Beteiligt an der Realisierung des Systems sind folgende Steuergeräte: Das Mantelrohrmodul (MRM) ist mit dem Blinkrichtungswählhebel verbunden und gibt den anderen Steuergeräten das entsprechende Blinksignal. Involviert sind außerdem beide Türsteuergeräte zur Ansteuerung der Blinker in den Seitenspiegeln, das Fondsteuergerät (FSG) zur Ansteuerung der Blinker im Heck und das Kombiinstrument im Cockpit zur Anzeige der Blinkrichtung.

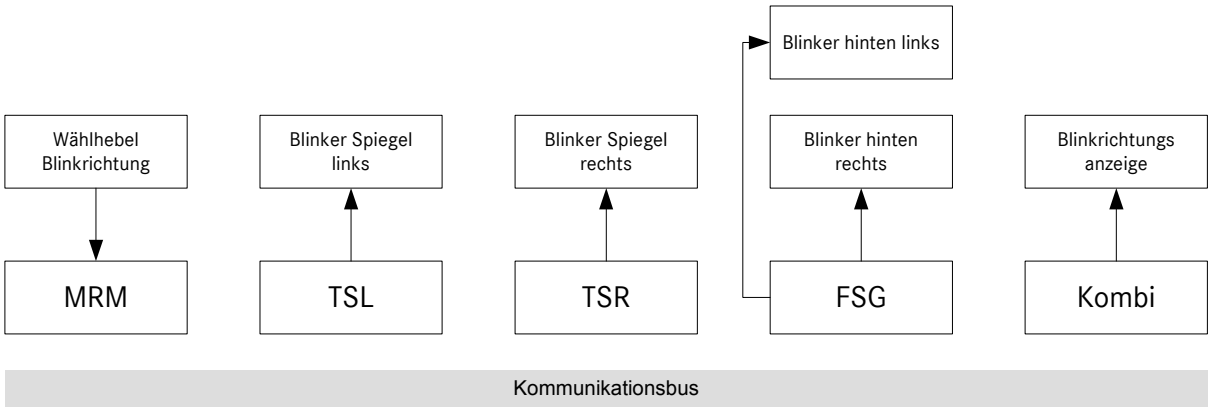

Abbildung 2.4: System Außenlicht (Richtungsblinken)

Wenn man für die beiden einfachen Beispiele die Zerlegung des E/E-Gesamtsystems in E/E-Systeme und die an deren Realisierung beteiligten Steuergeräte betrachtet, ergibt sich folgende azyklische, hierarchische Graphstruktur.

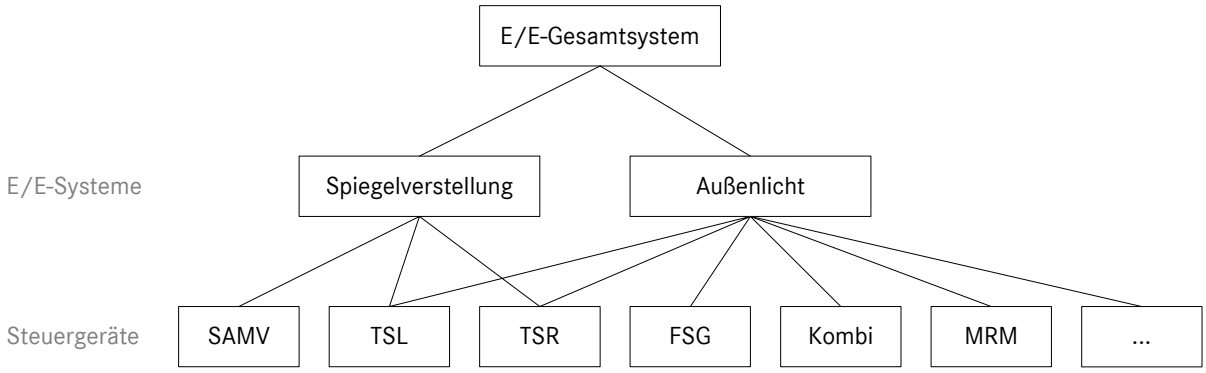

Abbildung 2.5: Struktur des E/E-Gesamtsystems

2.3.2 E/E-Entwicklungsprozess

Nachdem die Grundbegriffe der Domäne Elektrik/Elektronik vorgestellt wurden, wird jetzt der E/E-Entwicklungsprozess eines Fahrzeugs in allgemeiner und stark vereinfachter Form betrachtet (hauptsächlich nach [We00]).

Nach Analyse der Anforderungen (auf Grundlage des abgestimmten Konzepthefts) wird das Gesamtsystem zunächst als Funktionsnetzwerk konzipiert, in dem die Funktionen und die zwischen den Funktionseinheiten ausgetauschten Signale beschrieben werden. Im zweiten Schritt – der sog. *Partitionierung* – werden daraus die Aufteilung der Funktionen auf die Steuergeräte abgeleitet (Abbildung 2.7) und die Vorgaben für die ausgetauschten Signale zwischen den Steuergeräten festgelegt. In einem dritten Schritt wird der Einbauort der Steuergeräte unter Berücksichtigung zahlreicher elektrischer und physikalischer Randbedingungen (z.B. elektromagnetische Verträglichkeit oder Wartungsfreundlichkeit) festgelegt. Die eben beschriebenen Schritte der Konzeptionsphase werden meist iterativ solange durchlaufen, bis ein Optimum des E/E-Gesamtsystems in Bezug auf Anforderungen wie Kosten, Gewicht und Wartbarkeit erreicht wird.

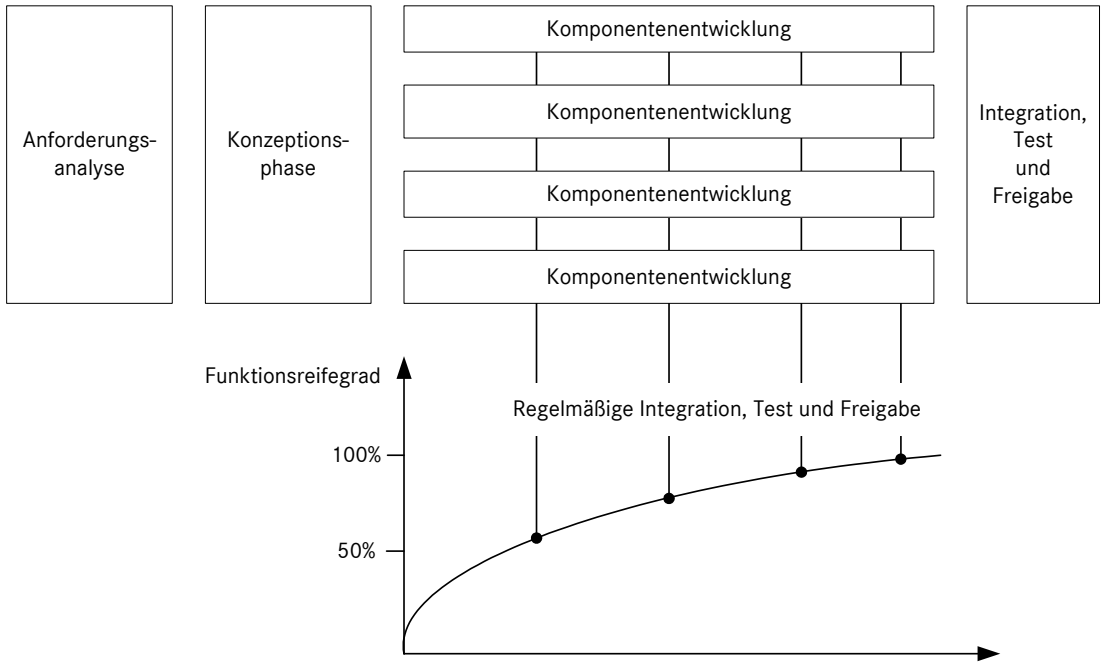

Abbildung 2.6 Schematischer E/E-Entwicklungsprozess nach [We00]

Ergebnis der Konzeptionsphase ist die Spezifikation der einzelnen Komponenten sowohl bezüglich der Hardware als auch der Software. Damit kann die parallele Entwicklung der einzelnen Komponenten beginnen. Diese wird zum Großteil nicht vom Automobilhersteller selbst, sondern von Zulieferern durchgeführt. Die Entwicklung und Produktion der Steuergeräte-Hardware liegt in der Regel in einer Hand. Die Software für ein Steuergerät kann jedoch durchaus von unterschiedlichen Zulieferern oder – bei als wettbewerbsdifferenzierend betrachteten Teilelementen – auch beim Automobilhersteller selbst entwickelt werden.

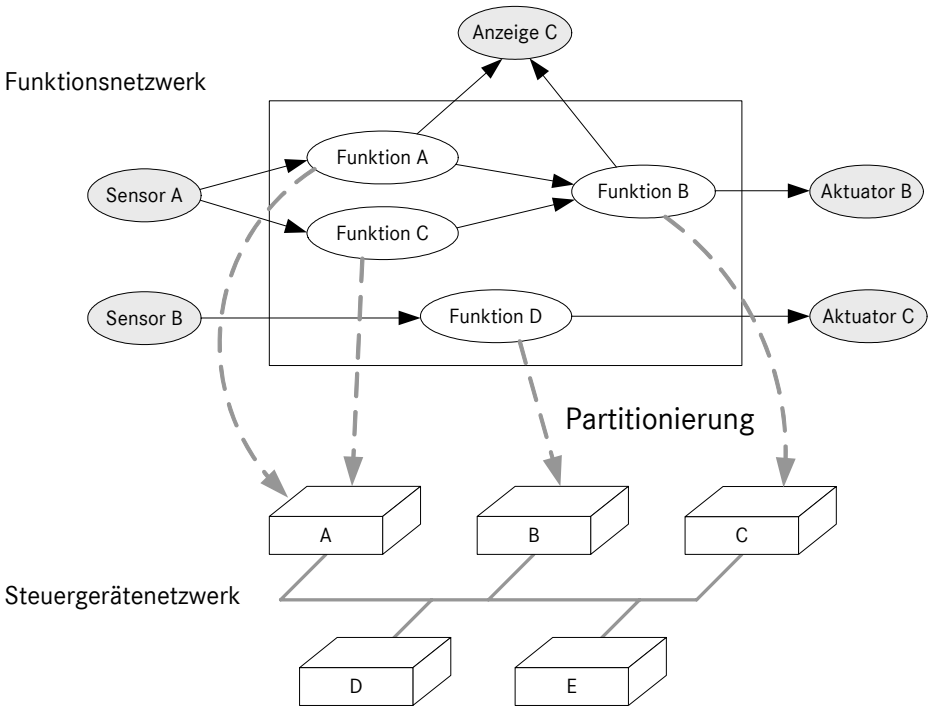

Abbildung 2.7: Partitionierung in der E/E-Entwicklung nach [We00]

Abbildung 2.8 zeigt einen komplexen Fall, bei dem die Entwicklung der Steuergeräte-Software sowohl vom Automobilhersteller als auch von einem Zulieferer übernommen und die Hardware von einem weiteren Zulieferer entwickelt wird.

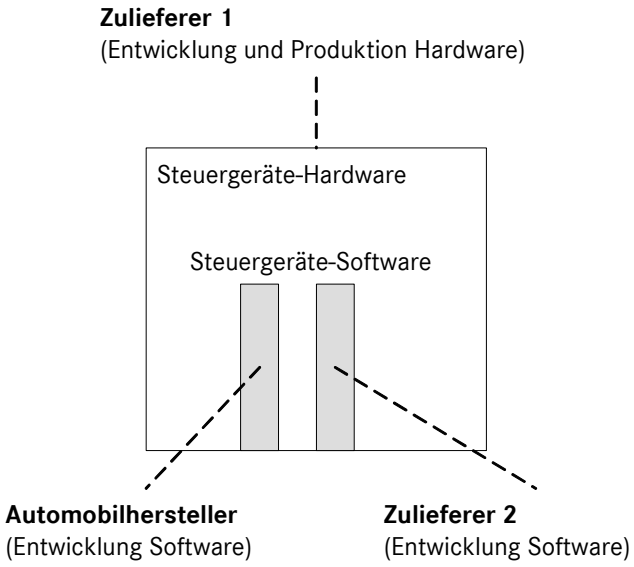

Abbildung 2.8 Entwicklungsbeteiligte bei einem Steuergerät nach [We00]

Die Komponenten werden nicht isoliert voneinander entwickelt und erst zum Schluss integriert, sondern die Entwicklung wird *inkrementell* durchgeführt. In einem sog. *Feature-Rollout-Plan* wird festgelegt, zu welchem Zeitpunkt welche Funktionen realisiert sein sollen. In regelmäßigen Abständen werden die Entwicklungsaktivitäten synchronisiert, indem ein bestimmter Entwicklungsstand aller Komponenten als Konfiguration schrittweise zum Gesamtsystem integriert, getestet und freigegeben wird. Dies ist Aufgabe des Automobilherstellers, der dazu von jedem Zulieferer einen bestimmten Stand der jeweiligen Komponenten geliefert bekommt. Beim Test gefundene Fehler werden im Normalfall in der weiteren Komponentenentwicklung behoben, bei erfolgreichem Test erfolgt eine Freigabe für die weitere Verwendung des E/E-Gesamtsystems in den Erprobungs- bzw. Bestätigungsfahrzeugen sowie für die weiteren Entwicklungsaktivitäten.

Abbildung 2.6 zeigt, neben den beschriebenen Entwicklungsaktivitäten, den über die Zeit zunehmenden Produktreifegrad. Dieser wächst zum einen dadurch, dass mit zunehmender Entwicklungszeit mehr Funktionen realisiert werden und zum anderen dadurch, dass mit der Synchronisation durch Integration und Test Fehler gefunden und in der weiteren Entwicklung behoben werden.

3 Test und Freigabe von E/E-Konfigurationen

Eine zentrale Rolle für das Erzielen von Qualität im Bereich E/E spielen der Test und die Freigabe von E/E-Komponenten und das damit verbundene Management von Konfigurationen. Sowohl aus Sicht der betrieblichen Prozesse als auch aus dem Blickwinkel der Informationstechnologie gibt es in diesem Bereich Verbesserungspotential.

Nach grundsätzlichen Begriffsklärungen in Abschnitt 3.1 wird in Abschnitt 3.2 die momentane Ist-Situation beschrieben. Abschnitt 3.3 skizziert einen Soll-Zustand – sowohl was die grundlegenden betrieblichen Abläufe als auch die informationstechnische Unterstützung angeht.

3.1 Begriffsklärungen

3.1.1 Der Konfigurationsbegriff und E/E-Konfigurationen

Integration, Test und Freigabe finden in der E/E-Entwicklung zu regelmäßigen Zeitpunkten statt (siehe Abschnitt 2.3.2). Getestet und freigegeben werden sog. *E/E-Konfigurationen*. Der Konfigurationsbegriff ist für die weiteren Betrachtungen zentral und selbst in der Fahrzeugentwicklung mit sehr unterschiedlichen Bedeutungen belegt. Deshalb soll geklärt werden, was mit *E/E-Konfigurationen* gemeint ist.

Während der Auseinandersetzung mit Konfigurationsmanagement im Automobilbau im Allgemeinen und im Bereich Elektrik/Elektronik im Speziellen ist sowohl in Gesprächen mit DaimlerChrysler-Mitarbeitern als auch durch einschlägige Literaturrecherche [ISO 10007, DoKö04, Sc02] deutlich geworden, dass mit dem Begriff „Konfiguration“ zwei grundsätzlich unterschiedliche Bedeutungsaspekte verknüpft sein können.

Mit dem einen Bedeutungsaspekt, der in [Sc02] auch als *Produktkonfiguration* bezeichnet wird, sind verschiedene Ausführungsvarianten eines Produkts gemeint. Beispielsweise ist eine „C-Klasse Limousine mit Klimaautomatik, Navigationssystem und Automatikgetriebe“ eine Ausführungsvariante des Produkts „C-Klasse“ und damit in diesem Sinne eine Konfiguration. Der zweite Bedeutungsaspekt einer Konfiguration betrifft die *Kompatibilität* und *Zusammengehörigkeit* einzelner Versionen bestimmter Produktkomponenten. Mit *Version* ist ein bestimmter Stand der Entwicklung einer Produktkomponente gemeint (vgl. Abschnitt 2.2). Konfigurationen im Software-Engineering [Pa04] entsprechen ebenfalls dieser Bedeutung.

Mit E/E-Konfigurationen ist hauptsächlich der letztgenannte Aspekt gemeint. Eine E/E-Konfiguration drückt aus, dass die in ihr gebündelten Versionen der Produktkomponenten miteinander verträglich sind *und* den relevanten Umfängen der Spezifikation entsprechend fehlerfrei interagieren.

Ein einfaches Beispiel für eine E/E-Konfiguration ist die in Abbildung 3.1 dargestellte Konfiguration für ein Steuergerät. Im Laufe der Entwicklung entstehen zunächst unabhängig voneinander Versionen sowohl der Hardware als auch der Software eines Steuergeräts. Die vereinfachte Konfiguration des linken Türsteuergeräts (TSL) besteht aus einer Version der Türsteuergeräte-Hardware und einer Version der Türsteuergeräte-Software. Dadurch wird ausgedrückt, dass Hardware und Software miteinander verträglich sind und fehlerfrei zusammenspielen.

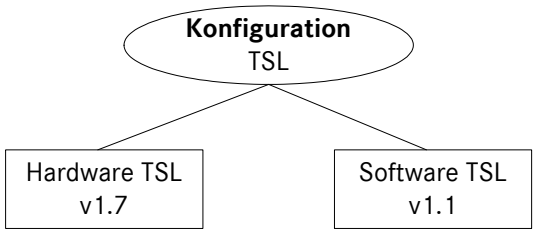

Abbildung 3.1: Beispielkonfiguration für ein Türsteuergerät

3.1.2 Test und Freigabe von E/E-Konfiguration

Eine E/E-Konfiguration bündelt Versionen bestimmter E/E-Komponenten. Die Abhängigkeiten zwischen den verschiedenen Versionen von E/E-Komponenten sind sehr komplex. Eine Konfiguration kann deshalb nicht am Schreibtisch oder gar automatisch gebildet und dann mit Sicherheit die Aussage getroffen werden, dass diese Versionen der Komponenten fehlerfrei zusammenspielen. Es ist vielmehr so, dass in diesem Umfeld eine Konfiguration zuerst „auf Verdacht“ gebildet und anschließend durch Tests abgesichert wird. Es muss also unterschieden werden, zwischen einer

- *nicht abgesicherten* Konfiguration, die eine *Soll-Kompatibilität* ausdrückt und einer
- *abgesicherten* Konfiguration, die eine *Ist-Kompatibilität* ausdrückt.

Allerdings ist die Ist-Kompatibilität theoretischer Natur. Denn wie in der Software-Entwicklung gilt, dass durch Prüfmaßnahmen niemals eine 100%-ige Fehlerfreiheit garantiert werden kann. Hinzu kommt, dass sich die Kompatibilität immer nur auf den jeweils betrachteten Kontext bezieht. Eine erfolgreich abgesicherte Konfiguration eines einzelnen Steuergeräts kann im Zusammenspiel in einem System mit anderen Steuergeräten trotzdem zu Fehlern führen.

Freigabeprozess

Als Freigabeprozess sollen in diesem Zusammenhang alle Maßnahmen verstanden werden, die durchgeführt werden, um eine noch nicht abgesicherte Konfiguration abzusichern. D.h. erst wenn alle Absicherungsmaßnahmen *erfolgreich* durchgeführt worden sind, gilt die Konfiguration als abgesichert. Sobald der Freigabeprozess für eine Konfiguration begonnen worden ist, darf diese nicht mehr verändert werden, da das Zusammenspiel einzelner Versionen abgesichert werden soll. Wenn einzelne Versionen während des Freigabeprozesses (und danach) verändert werden, sind Kompatibilität und Korrektheit nicht mehr gewährleistet.

Absicherungsmaßnahmen können neben dynamischen Tests auch rein formal sein, wie beispielsweise die offizielle Genehmigung der Konfiguration durch ein Gremium. Beispiele für Freigabeprozesse und Absicherungsmaßnahmen werden in Abschnitt 3.3.3 vorgestellt.

Freigabezustand

Eine Konfiguration kann mehrere Freigabezustände annehmen, die ausdrücken, ob und wie diese Konfiguration einen Freigabeprozess durchlaufen hat.

- *Noch nicht freigegeben:* Es wurden noch nicht alle Absicherungsmaßnahmen mit der Konfiguration durchgeführt. Bis jetzt ist keine der durchgeführten Absicherungsmaßnahmen fehlgeschlagen.
- *Nicht freigegeben:* Während des Freigabeprozesses ist mindestens eine Absicherungsmaßnahme fehlgeschlagen, d.h. in der Konfiguration wurde ein Fehler gefunden. Die Konfiguration ist daher fehlerhaft.
- *Freigegeben:* Alle Absicherungsmaßnahmen eines Freigabeprozesses wurden erfolgreich durchgeführt und die Konfiguration ist freigegeben.

Den Zusammenhang zwischen Freigabezustand und Freigabeprozess kann man folgendermaßen zusammenfassen (vgl. Abbildung 3.2): Nachdem eine Konfiguration erzeugt wurde, befindet sie sich im Zustand „Noch nicht freigegeben“. Nach erfolgreicher Durchführung eines Freigabeprozesses befindet sie sich im Zustand „Freigegeben“. Wenn mindestens eine Absicherungsmaßnahme fehlgeschlagen ist, ist der Zustand „Nicht freigegeben“.

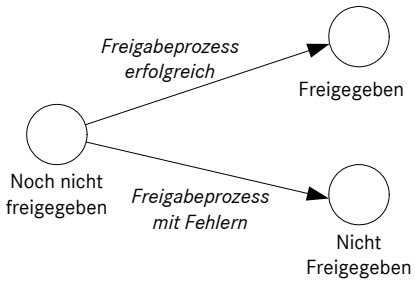

Abbildung 3.2: Zusammenhang zwischen Freigabeprozess und Freigabezustand

Freigabe

Eine Konfiguration besitzt eine Freigabe, wenn sie den Freigabezustand „Freigegeben“ hat. Eine Freigabe gilt immer für einen bestimmten Zweck, der beispielsweise davon abhängt, in welcher Phase der Produktentwicklung integriert und freigegeben wird. In den frühen Phasen der Entwicklung ist eine Freigabe wichtig für weitere Entwicklungsschritte, da auf den Ergebnissen der vorhergehenden Entwicklung aufgebaut wird. In späteren Phasen geht es darum, Konfigurationen für die Produktion freizugeben.

3.2 Ist-Zustand

Im Moment werden E/E-Konfigurationen ausschließlich auf Gesamtsystemebene gebildet. Eine solche flache Konfiguration enthält jeweils eine Version der Software und Hardware aller Steuergeräte eines Fahrzeugs [HRKH04].

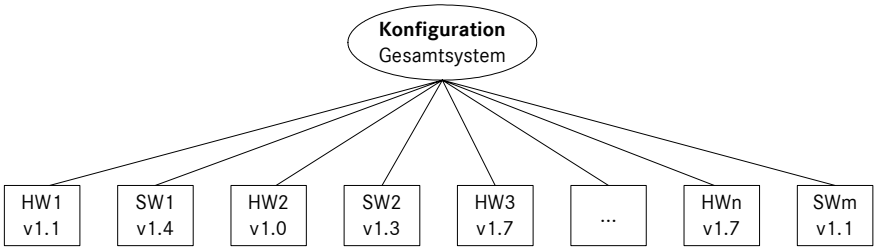

Abbildung 3.3: Flache Konfiguration auf Gesamtsystemebene

Für das Erzeugen und Testen solcher Konfigurationen existiert ein Prozess, dieser ist jedoch von der Freigabe entkoppelt. Freigegeben werden momentan ausschließlich *einzelne* Komponenten ohne Berücksichtigung des Zusammenspiels. Da Komponenten nicht unabhängig voneinander sind und Inkompatibilitäten zwischen verschiedenen Versionen der Komponenten möglich sind, können dadurch Fehler entstehen. Es gibt auch Fälle, in denen Komponenten trotz des Fehlschlags einzelner Absicherungsmaßnahmen freigegeben werden, da zu diesem Zeitpunkt der Entwicklung keine andere Komponente verfügbar ist.

Weitere Qualitätsprobleme und hohe Kosten entstehen durch nicht „gelebte“ und teilweise nicht definierte Prozesse. So kann es passieren, dass während eines Tests festgestellt wird, dass die getesteten Steuergeräte auf unterschiedlichen Versionen der Spezifikation basieren und somit gar nicht zusammenspielen können. Wenn dieser Sachverhalt in vorhergehenden Prozessschritten geprüft wird, können Zeitaufwand und Kosten für Vorbereitung solcher unnötiger Tests vermieden werden.

Diese Probleme auf Seite der betrieblichen Abläufe werden durch eine historisch gewachsene und damit äußerst heterogene IT-Systemlandschaft noch verschärft. Produktdaten sind über viele Systeme verteilt und es gibt damit keine einheitliche, integrierte Sicht auf die Daten von E/E-Komponenten. Freigaben sind nicht in allen Systemen dokumentierbar und falls doch, dann nur auf der Ebene von Einzelkomponenten. Die Bildung von Konfigurationen über mehrere Einzelkomponenten und die Dokumentation von deren Freigabe ist nicht einheitlich für alle Beteiligten im Entwicklungsprozess möglich.

3.3 Soll-Zustand

Die im Ist-Zustand beschriebenen Probleme sollen sowohl durch Änderung der betrieblichen Abläufe als auch durch Anpassung der IT-Landschaft angegangen werden. Im Folgenden wird diesbezüglich der Soll-Zustand beschrieben.

3.3.1 Regelmäßige Bildung hierarchischer Konfigurationen

Eine zentrale Lösungsidee aus der Perspektive der betrieblichen Abläufe ist die Bildung hierarchischer Konfigurationen. Im Gegensatz zum bisherigen Vorgehen sollen Konfigurationen schrittweise aufeinander aufgebaut werden. Dabei orientiert man sich an der Struktur des E/E-Gesamtsystems.

Zuerst werden Konfigurationen auf Ebene der einzelnen E/E-Komponenten gebildet. So enthält beispielsweise eine Konfiguration für ein Steuergerät eine Version der Hardware und eine Version der Software dieses Steuergeräts. Aus diesen Konfigurationen auf Komponentenebene werden im nächsten Schritt auf höherer Ebene Konfigurationen auf E/E-Systemebene zusammengestellt. Dies wird bis zur Ebene des Gesamtsystems fortgeführt. Eine mögliche Ausprägung einer stark vereinfachten Konfigurationshierarchie für Teile der in Abschnitt 2.3 vorgestellten Beispielsysteme zeigt Abbildung 3.4.

Wenn eine hierarchisch strukturierte Konfiguration den Freigabezustand „Freigegeben“ besitzt, bedeutet dies, dass die Kompatibilität und Korrektheit *aller* Teilkonfigurationen im Zusammenspiel der in den Teilkonfigurationen enthaltenen Produktkomponenten abgesichert ist.

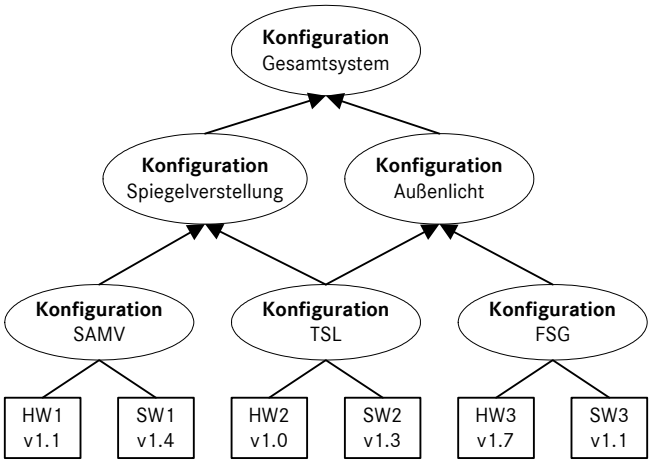

Abbildung 3.4: Beispiel für eine Konfigurationshierarchie

Eine Konfiguration kann im Laufe der Zeit Teilkonfiguration mehrerer Konfigurationen sein. Wie viele dies sein werden steht zu Beginn im Allgemeinen *nicht* fest. So kann in unserem Beispiel im Laufe der Produktentwicklung die dargestellte Konfiguration für das Fondsteuergerät (FSG) von einer neuen Konfiguration des Außenlichts verwendet werden, die jedoch für das Türsteuergerät links (TSL) eine neue Konfiguration verwendet (siehe Abbildung 3.5).

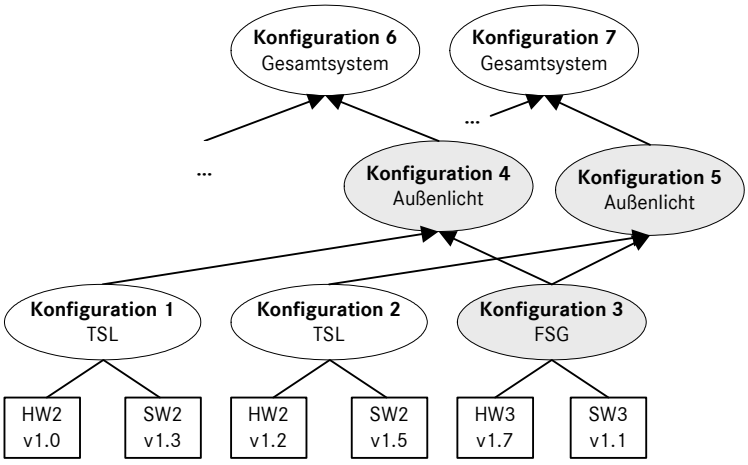

Abbildung 3.5: Mehrfache Verwendung einer Konfiguration als Teilkonfiguration

3.3.2 Integrierte Freigabeprozesse

Anstatt Freigaben wie bisher auf Komponentenebene zu erteilen und entkoppelt vom Konfigurationsmanagement zu sehen, sollen Freigaben auf allen Ebenen der beschriebenen Konfigurationshierarchie vergeben werden. Die Freigabe soll hierbei in den Testprozess integriert werden, so dass es integrierte Test- und Freigabeprozesse gibt, die bereits in Abschnitt 3.1.2 verallgemeinernd als *Freigabeprozesse* bezeichnet wurden.

Konfigurationen sollen jeweils für sich diese Freigabeprozesse durchlaufen. Im Gegensatz zum bisherigen Zustand sollen alle Prozesse klar definiert sein und ausnahmslos *jede* Konfiguration soll die entsprechenden Prozesse vor ihrer Freigabe durchlaufen haben. Aus prozessorientiertem Blickwinkel ergibt sich nun das in Abbildung 3.6 dargestellte Bild, in dem jede Konfiguration mit einem entsprechenden Freigabeprozess assoziiert ist.

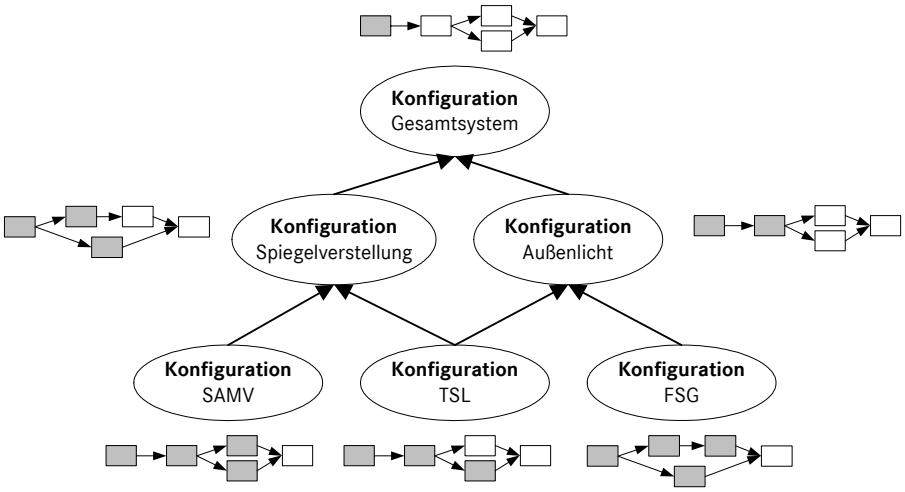

Abbildung 3.6: Konfigurationen und assoziierte Freigabeprozesse

Durch dieses Vorgehen verspricht man sich einfachere und transparentere Freigabeprozesse, da nicht mehr alle Produktkomponenten auf einmal freigegeben werden müssen, sondern dies schrittweise geschieht.

Denkbar wäre nun, dass Konfigurationen auf unterer Ebene zuerst freigeben sein müssen, bevor Konfigurationen auf der darüber liegenden Ebene zusammengestellt und deren Freigabeprozess durchgeführt werden kann. Dieses strikt sequenzielle Vorgehen würde jedoch zu sehr langen Gesamtprozessdauern führen, was den Gesamtfreigabeprozess der Gesamtkonfiguration betrifft: angefangen vom Beginn der Freigabe ihrer Teilkonfigurationen auf unterster Ebene bis zum Abschluss des Freigabeprozesses der Gesamtkonfiguration.

Aus diesem Grund strebt man eine teilweise parallele Ausführung der Freigabeprozesse an. Dabei sind die folgenden Randbedingungen einzuhalten:

- Um die Semantik der Konfigurationsfreigaben zu erhalten (abgesicherte Kompatibilität und Korrektheit), darf eine Konfiguration erst dann freigegeben werden, wenn alle ihre Teilkonfigurationen freigegeben worden sind, d.h. den Freigabezustand „Freigegeben“ besitzen.
- Die Schritte der parallelen Freigabeprozesse dürfen sich nicht beliebig überlappen: Bestimmte Schritte eines Freigabeprozesses einer Konfiguration dürfen erst ausgeführt werden, nachdem bestimmte Schritte in den Freigabeprozessen der Teilkonfi-

gurationen bereits erfolgreich durchgeführt worden sind. Diese Abhängigkeiten sind darin begründet, dass Testschritte für Konfigurationen der darüber liegenden Ebene aufwändiger und damit teurer sind. Vor der Ausführung eines solchen Tests möchte man Sicherheit über einen gewissen Reifegrad der Teilkonfigurationen haben, um den Test überhaupt sinnvoll ausführen zu können.

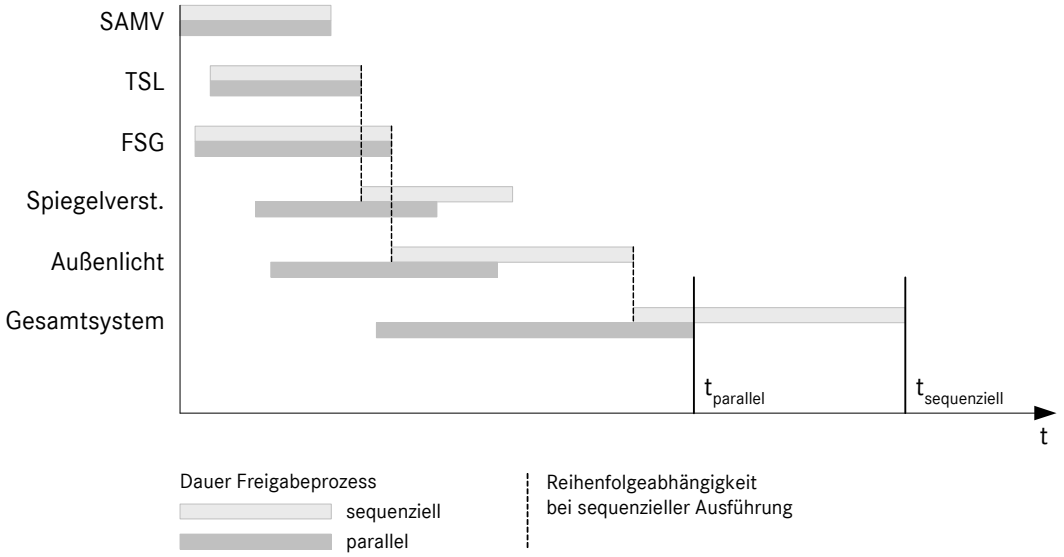

Abbildung 3.7: Sequenzielle und parallele Ausführung von Freigabeprozessen im Vergleich

Abbildung 3.7 zeigt die durch parallele Ausführung der Freigabeprozesse zu erreichende qualitative Zeitersparnis. Dauer und Beginn der Freigabeprozesse der Einzelkonfigurationen werden dort durch Balken repräsentiert. Es werden zwei Fälle dargestellt: zum einen eine strikt sequenzielle Ausführung der Freigabeprozesse, bei der der Freigabeprozess einer Konfiguration erst beginnen kann, wenn die Freigabeprozesse aller Teilkonfigurationen beendet worden sind und zum anderen eine mögliche parallele Überlappung der Freigabeprozesse auf allen Ebenen. Eine gestrichelte Linie kennzeichnet im sequenziellen Fall die Reihenfolgeabhängigkeit zwischen dem Ende des am längsten dauernden Freigabeprozesses der Teilkonfigurationen und dem Beginn des Freigabeprozesses der entsprechenden Oberkonfiguration.

3.3.3 Beispielprozesse

Zur Illustration der Freigabeprozesse sollen zwei stark vereinfachte Prozesse dienen: zum einen ein Freigabeprozess auf Ebene der Steuergeräte und zum anderen ein Freigabeprozess auf der darüber liegenden Systemebene. Zwei Tests, die näherer Erklärung bedürfen, spielen in beiden Prozessen eine Rolle: Beim *Bretttest* werden eine oder mehrere Komponenten *manuell* in Betrieb genommen und getestet. Manuell bedeutet, dass beispielsweise die Eingangssignale für ein Steuergerät, die im Fahrzeug durch Sensoren geliefert werden, durch den Tester persönlich erzeugt werden. Im Gegensatz dazu steht der sehr aufwändige *Hardware-in-the-Loop-Test (HiL)*: Ein Simulationsrechner liefert hier in Echtzeit Eingangssignale für Steuergeräte und verwendet die Ausgangssignale des Steuergeräts wieder zur Berechnung neuer Eingangssignale. Durch diesen Regelkreis wird die Interaktion eines Steuergeräts mit der Realwelt simuliert (Abbildung 3.8).

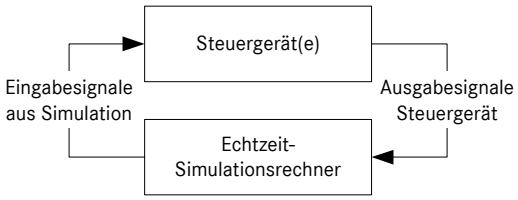

Abbildung 3.8: Regelkreis bei Hardware-in-the-Loop-Tests

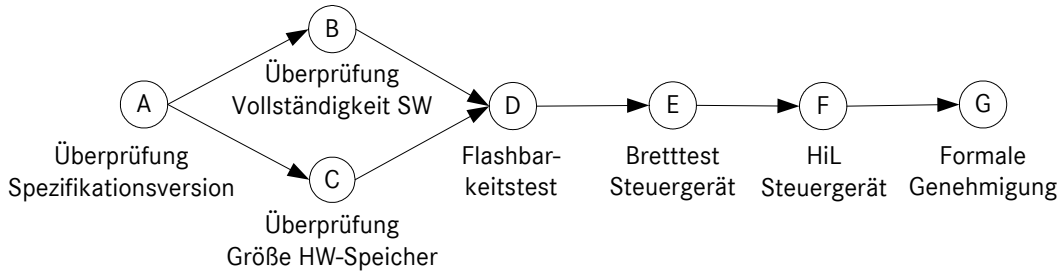

Abbildung 3.9: Beispielfreigabeprozess für die Konfiguration eines Steuergerät

Der Freigabeprozess für die Konfiguration eines Steuergeräts *könnte* vereinfacht folgendermaßen ablaufen (vgl. Abbildung 3.9): Nachdem überprüft wurde, ob alle Komponenten des Steuergeräts in der abzusichernden Konfiguration auf Basis derselben Spezifikation entstanden sind (Prüfschritt A), können parallel folgende Maßnahmen durchgeführt werden: Zum einen wird überprüft, ob die in der Konfiguration enthaltenen Software-Komponenten vollständig sind (Prüfschritt B), und zum anderen, ob der Hardwarespeicher ausreicht, um die Software darauf laden zu können (Prüfschritt C). Nach diesen Schritten wird im *Flashbarkeitstest* (Prüfschritt D) überprüft, ob auf die Steuergeräte-Hardware gemäß den Richtlinien des Unternehmens die Software aufgespielt (im Fachjargon „geflasht“) werden kann. Bevor die Konfiguration des Steuergeräts im aufwändigen HiL-Test auf seine Spezifikation überprüft wird (Prüfschritt F), werden erste funktionale Tests mittels eines Brettests durchgeführt (Prüfschritt E). Eine formale Genehmigung, in der ein Gremium die Steuergerätekonfiguration offiziell freigibt, bildet den Abschluss des Freigabeprozesses (Prüfschritt G).

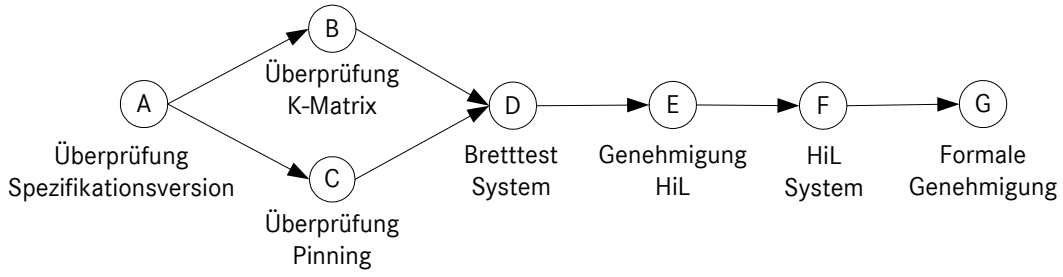

Abbildung 3.10: Beispielfreigabeprozess für Konfiguration eines E/E-Systems

Auf der Ebene der E/E-Systeme ist folgender Freigabeprozess für eine Konfiguration denkbar (vgl. Abbildung 3.10): Zuerst wird in Prüfschritt A überprüft, ob alle abzusichernden Produktkomponenten auf derselben Spezifikation beruhen. Anschließend wird in Prüfschritt B überprüft, ob die zu Grunde liegende *Kommunikationsmatrix (K-Matrix*), die die auszutauschenden Signale zwischen Steuergeräten beschreibt, bei allen Steuergeräten dieselbe ist. Parallel dazu wird überprüft, ob die Steckanschlüsse der Steuergeräte (das sog. *Pinning*) so-

wohl geometrisch als auch elektrisch zueinander passen (Prüfschritt C). Anschließend werden in Prüfschritt D mit einem Bretttest Funktionen gegen ihre Spezifikation getestet. Bevor der HiL-Test des gesamten Systems durchgeführt werden kann (Prüfschritt F), muss ein Gremium diesen erst genehmigen, da dieser Test sehr aufwändig und kostenintensiv ist. Den Abschluss des Freigabeprozesses auf Systemebene bildet wiederum die formale Genehmigung der Konfiguration.

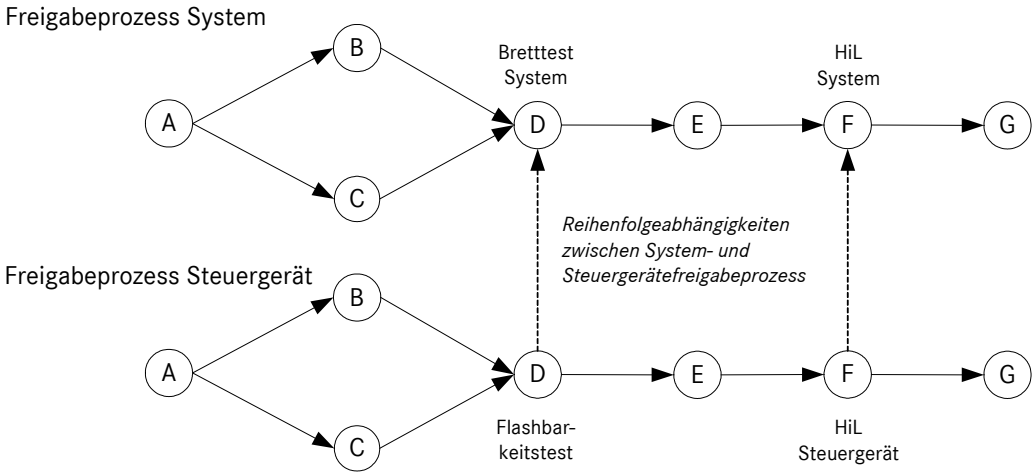

Abbildung 3.11: Hierarchische Reihenfolgeabhängigkeiten zwischen Freigabeprozessen

Bei der angestrebten, parallelen Durchführung der Freigabeprozesse auf mehreren Ebenen sind folgende Abhängigkeiten denkbar (vgl. Abbildung 3.11): Bevor der Bretttest auf Systemebene durchgeführt werden kann, soll für *alle* Teilkonfigurationen (d.h. in diesem Fall Steuergerätekonfigurationen) der Flashbarkeitstest durchgeführt worden sein: auf nicht flashbare Steuergeräte kann keine Software aufgespielt werden und damit auch kein Bretttest durchgeführt werden. Eine weitere Abhängigkeit besteht zwischen dem Prüfschritt F (HiL-Test) für Steuergerätekonfigurationen und dem Prüfschritt F (ebenfalls HiL-Test) für Systemkonfigurationen: der HiL-Test auf Ebene der Systeme ist sehr teuer und soll erst dann durchgeführt werden, wenn die funktionalen HiL-Tests für die einzelnen Steuergeräte wirklich erfolgreich waren.

3.3.4 IT-Unterstützung

Um den skizzierten Soll-Zustand aus organisatorischer Sicht zu erreichen, ist aufgrund der hohen Komplexität in diesem Umfeld und der Vielzahl beteiligter Nutzer eine adäquate IT-Unterstützung unabdingbar. Dabei lassen sich zwei Problembereiche identifizieren, in denen unbedingter Handlungsbedarf besteht: zum einen die Integration der bestehenden, heterogenen Produktdatenbanken und zum anderen die aktive Unterstützung bei der Durchführung der Freigabeprozesse durch Workflow-Management-Systeme.

Für beide Problembereiche werden im Rahmen dieser Arbeit Anforderungen analysiert. Der Schwerpunkt dieser Arbeit liegt im Bereich Workflow-Management für Freigabeprozesse. Vor der ausführlichen Darstellung entsprechender Anforderungen in Kapitel 4.8 sollen die Analyseergebnisse bezüglich der Integration von Produktdatenbanken für das Konfigurationsmanagement kurz zusammengefasst werden.

Wie in Abschnitt 3.2 angedeutet, ist die Systemlandschaft der E/E-Entwicklung äußerst heterogen. Die Produktdaten, die bei der Entwicklung anfallen, sind auf unterschiedlichste Sys-

teme verteilt. Dadurch ist keine einheitliche Sicht auf die E/E-Komponenten und ihre Daten möglich. Insbesondere ist es nicht möglich, systemübergreifend Konfigurationen zu bilden. Aus diesem Grund sollen die bestehenden Systeme durch eine *Integrationsschicht* so gekoppelt werden, dass hierarchische Konfigurationen (wie in Abschnitt 3.3.1 beschrieben) über Systemgrenzen hinweg abgebildet werden können. Die zu integrierenden Quellsysteme sind äußerst heterogen: von Webanwendungen, über einfache Dateiablagen, bis hin zu hochkomplexen PDM-Systemen.

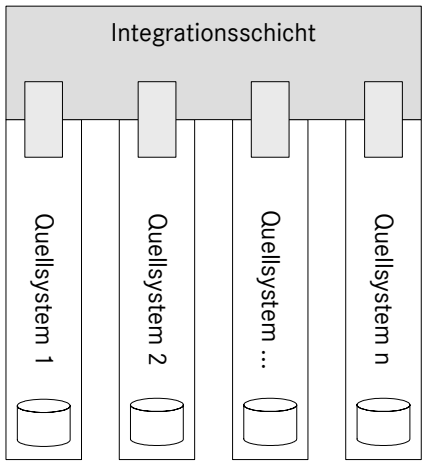

Abbildung 3.12: Integrationsschicht integriert Quellsysteme

Zu E/E-Komponenten liegen Produktdaten in den unterschiedlichsten Systemen vor. Die Integrationsschicht soll es ermöglichen, hierarchische Konfigurationen über verschiedene Versionen von E/E-Komponenten abzubilden und dazu einen globalen Freigabestatus zu dokumentieren. Den Nutzern soll es auf einfache Art und Weise ermöglicht werden, auf die Produktdaten einer Konfiguration zuzugreifen. Da jeder Nutzer unterschiedliche Produktdaten für seine Aufgaben benötigt, sollen die irrelevanten Produktdaten durch ein Sichtenkonzept ausgeblendet werden können.

Zu diesen funktionalen Anforderungen kommen weitere nicht-funktionale Anforderungen hinzu: Für Produktdaten gelten hohe Sicherheitsanforderungen, da diese das Ergebnis des Produktentstehungsprozesses darstellen und unter allen Umständen zu vermeiden ist, dass diese unkontrolliert nach außen gelangen. Deshalb hat die Integrationsschicht dafür zu sorgen, dass Produktdaten nur von dafür autorisierten Personen zugegriffen werden können. Da die Menge der Daten in den Quellsystemen beträchtlich ist, sollte die Integrationsschicht auch gut skalieren. Gleichzeitig soll die Verteilung der Daten aus Sicht der Nutzer möglichst transparent sein. Da Konzernstrategien einem Wandel unterworfen sind und dieser Wandel auch auf die IT-Systeme durchschlagen kann, soll die Architektur der Integrationsschicht so gestaltet sein, dass im Sinne von „Plug'n'Play" weitere Quellsysteme integriert werden können.

4 Grundlagen des Workflow-Management

Viele Unternehmensprozesse werden heutzutage durch IT-Systeme unterstützt. Diese Unterstützung erfolgt in vielen Fällen in Form von *daten- und funktionszentrierten Anwendungen* [DRK00]. Die Abläufe selbst existieren meist nur in den Köpfen der Mitarbeiter, was in der Praxis zu signifikanten Problemen führt [RDMK00]: Aufgaben werden übersehen und durch Nichtberücksichtigung von Abhängigkeiten zwischen Tätigkeiten Prozessschritte u.U. wiederholt ausgeführt. Die Mitarbeiter müssen außerdem die für eine Aufgabe benötigten Daten mühsam in den Anwendungssystemen suchen. Auch der Status eines Prozesses (z.B. der Stand eines Bestellvorgangs) kann nur schwer ermittelt werden. Es liegt daher nahe, für diese Zwecke *prozessorientierte Anwendungen* zu realisieren, die Benutzer aktiv bei der Durchführung ihrer Prozesse unterstützen, indem sie Daten und Anwendungen aufgabenbezogen bereitstellen. Wenn man mit konventionellen Programmiermethoden versucht, solche prozessorientierten Anwendungen zu realisieren, ist dies eine nicht triviale und sehr fehleranfällige Aufgabe [DRK00], da die gesamte Prozesslogik „hart verdrahtet" kodiert werden muss. Hinzu kommt, dass sich Prozesse in Unternehmen schnell ändern. Dies führt zu ständigen Änderungen des prozessorientierten Anwendungssystems und damit in der Folge zu hohen Wartungskosten und beträchtlichem Fehlerpotential.

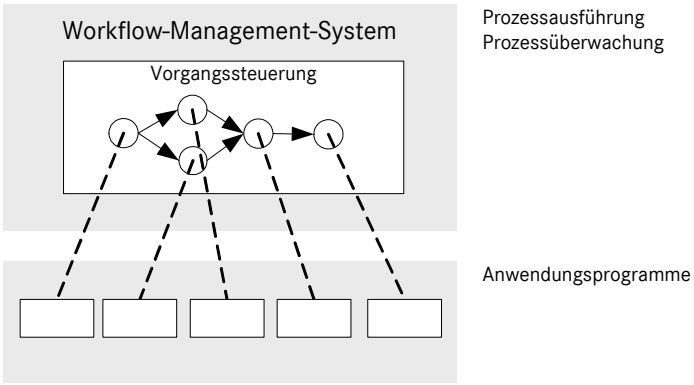

Abbildung 4.1: Trennung von Prozesslogik und Anwendungscode durch WfMS nach [Re03]

Workflow-Management-Systeme (WfMS) bieten hierzu eine vielversprechende Alternative [LeRo00, JBS97]: Durch die Trennung von Prozesslogik und Anwendungscode ist eine *explizite* Prozesssteuerung möglich, die die Ausführung von Prozessen aktiv überwacht und eine einfache Anpassung an neue Prozesse potenziell möglich macht (vgl. Abbildung 4.1). Ein WfMS ist für die Ausführung eines Prozesses verantwortlich und entlastet den Anwendungsentwickler zudem von systemnahen Aspekten, wie Synchronisation oder Fehlerbehandlung [Re00]. Dieser kümmert sich im Idealfall nur noch um die Implementierung der Anwendungskomponenten für die einzelnen Prozessschritte und um die (explizite) Modellierung des Prozesses.

4.1 Prozesse vs. Workflows

Im Bereich Workflow-Management ist zwischen Prozessen, Workflows und deren Modellen zu unterscheiden [LeRo00]. Ein *Prozessmodell* beschreibt, wie ein Prozess in der Realwelt ablaufen soll. Eine konkrete Ausprägung dieses Prozessmodells (z.B. der Kundenauftragsprozess von Herrn Müller) wird als *Prozess* (oder auch *Prozess-Instanz*) bezeichnet. Ein *Workflow-*

Modell (oder auch *Workflow-Schema*) beschreibt die Teile eines Prozessmodells, die mit Hilfe eines WfMS ausgeführt werden sollen. Ein *Workflow* (oder auch *Workflow-Instanz*) stellt demnach eine konkrete Instanz dieses Modells zur Laufzeit dar, das von einem WfMS ausgeführt wird. Abbildung 4.2 veranschaulicht diesen Sachverhalt.

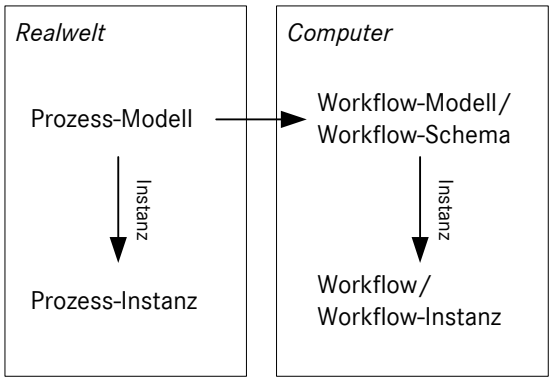

Abbildung 4.2: Prozesse und Workflows nach [LeRo00]

4.2 Architektur von WfMS

Typisch für WfMS ist die Trennung in Modellierungs- und Laufzeitkomponenten. Die Modellierungskomponente (oft auch *Buildtime-Komponente* genannt) gestattet es, alle für die automatische Ausführung eines Prozesses wichtigen Sachverhalte in einem Workflow-Modell abzubilden. Dieses Workflow-Schema wird dann von der Laufzeitkomponente (häufig als *Workflow-Engine* oder *Runtime-Komponente* bezeichnet) ausgeführt. Genauer gesagt, erlaubt die Laufzeitkomponente den Nutzern, Instanzen der Workflow-Schemata zu erzeugen und diese gemäß der definierten Prozesslogik auszuführen.

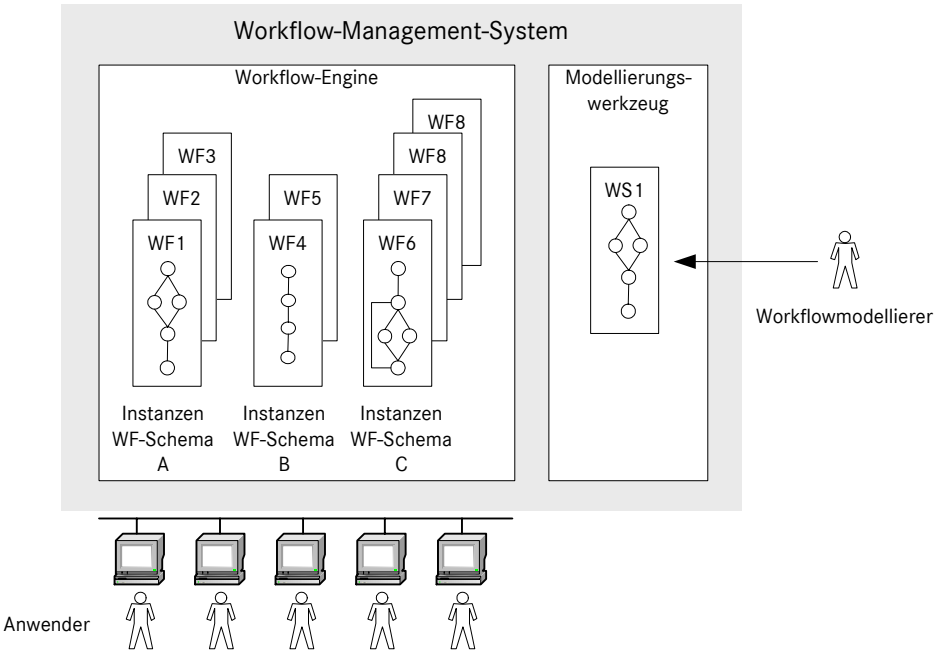

Abbildung 4.3: Konzeptuelle Architektur eines WfMS nach [Re03]

Abbildung 4.3 zeigt eine Workflow-Engine, die mehrere Instanzen unterschiedlicher Workflow-Schemata ausführt. Die Anwender interagieren mit dem WfMS über spezielle

Klientenprogramme auf ihren Arbeitsrechnern. Dazu wird meist eine Arbeitslisten-Metapher benutzt: In einer persönlichen Arbeitsliste werden alle auszuführenden Aktivitäten eines bestimmten Anwenders (auch unterschiedlicher Workflow-Schemata) zusammengefasst und diesem zur Ausführung angeboten. Wenn er sich dafür entscheidet, eine Aktivität auszuführen, wird das entsprechende Anwendungsprogramm aufgerufen, das er für diese Aufgabe benötigt. Details zur Ausführung eines Workflows und Interaktion mit Anwendern diskutiert Abschnitt 4.4.

Zum Zweck der Standardisierung wurde von der Workflow-Management-Coalition (WfMC) eine Referenzarchitektur [WfMC95] vorgeschlagen, die grundlegende Komponenten und Schnittstellen eines WfMS definiert (siehe Abbildung 4.4).

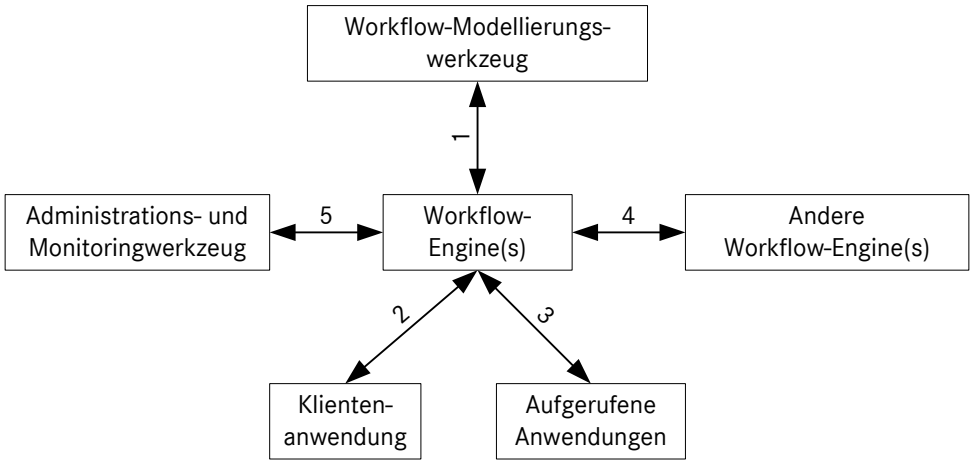

Abbildung 4.4: Referenzarchitektur der WfMC [WfMC95]

Das Referenzmodell der WfMC geht von einer (logisch) zentralen Ausführungseinheit aus, die Workflow-Instanzen ausführt und über standardisierte Schnittstellen mit anderen Komponenten interagiert.

- *Schnittstelle 1* legt ein Format zum Austausch von Workflow-Schemata zwischen Modellierungswerkzeug(en) und der ausführenden Workflow-Engine fest.
- *Schnittstelle 2* ist eine standardisierte Programmierschnittstelle, über die Klientenanwendungen mit der Workflow-Engine kommunizieren können. Angeboten werden von dieser Schnittstelle Operationen wie Starten eines Workflows, Starten einer Aktivität, Beenden einer Aktivität oder Abrufen der Arbeitsliste des jeweiligen Anwenders.
- Der Aufruf von externen Anwendungen wird durch *Schnittstelle 3* ermöglicht.
- Über die *Schnittstelle 4* ist die Kommunikation mit WfMS anderer Hersteller möglich.
- Die *Schnittstelle 5* gibt Administrations- und Monitoringwerkzeugen Zugriff auf Daten, die für die Überwachung und Kontrolle von Workflow-Instanzen relevant sind.

4.3 Modellierung von Workflows

Die wichtigsten Elemente einer Sprache zur Workflow-Modellierung können nach [JBS97] durch die folgenden Aspekte charakterisiert werden, die in ihrer *Gesamtheit* einen Workflow beschreiben und dadurch das Workflow-Modell oder Workflow-Schema bilden.

- Der *funktionale Aspekt* beschreibt die funktionalen Basiseinheiten, aus denen sich ein Workflow zusammensetzt. Diese werden als *Aktivitäten*, *Tasks* oder *Workflow-Schritte* bezeichnet und stellen die elementaren Bausteine eines Workflows dar.
- Der *verhaltensbezogene Aspekt* beschreibt den Kontrollfluss zwischen Aktivitäten, d.h. die Reihenfolge und Bedingungen für ihre Ausführung. Dies kann beispielsweise die Nacheinanderausführung oder die parallele Ausführung von Aktivitäten sein.
- Die zwischen Aktivitäten fließenden Daten werden im *datenbezogenen Aspekt* dargestellt.
- Im *Organisationsaspekt* werden die Organisationsstrukturen eines Unternehmens abgebildet. Darauf basierend wird in Bearbeiterzuordnungen beschrieben, *wer* die Aktivitäten eines Workflows ausführen darf.
- Der *operationale Aspekt* steuert die Einbindung von Applikationen, die für die Ausführung von Aktivitäten benötigt werden.

Abhängig vom Anwendungsgebiet können noch weitere Aspekte wichtig sein, wie beispielsweise temporale Abhängigkeiten zwischen Aktivitäten. Ein Beispiel für eine konkrete Sprache zur Workflow-Modellierung sind Aktivitätennetze (siehe Abschnitt 4.5).

4.4 Ausführung von Workflows

Die Workflow-Engine führt Instanzen der vom Modellierer definierten Workflow-Schemata aus. Dabei geschieht in stark vereinfachter Darstellung Folgendes: Nach dem Start einer Workflow-Instanz werden alle Aktivitäten dieser Instanz ermittelt, die gestartet werden *können*. Für jede dieser Aktivitäten werden alle potenziellen Bearbeiter ermittelt. Dazu wird der Organisationsaspekt des Workflow-Modells verwendet. Daten für diese Aktivität werden bereitgestellt und der Start von Anwendungskomponenten vorbereitet. Jeder der ermittelten Bearbeiter wird darüber benachrichtigt, dass er eine Aktivität durchführen kann. Dies geschieht meist dadurch, dass in der persönlichen Arbeitsliste des Bearbeiters ein neuer Eintrag mit der Aktivität angezeigt wird [Re00].

Entscheidet sich der Bearbeiter für die Durchführung einer Aktivität, teilt er dies der Workflow-Engine über sein Klientenprogramm mit. Die Aktivität verschwindet dann von den Arbeitslisten aller anderer möglichen Bearbeiter und das mit der Aktivität verknüpfte Anwendungsprogramm wird gestartet und ggf. mit Daten versorgt. Der Bearbeiter führt die Aktivität mit Hilfe des Anwendungsprogramms durch und meldet dann die Beendigung der Aktivität an die Workflow-Engine zurück. Diese „schaltet“ in der Workflow-Instanz weiter, indem diese Aktivität als bearbeitet gekennzeichnet und von Neuem mit der Ermittlung der zu startenden Aktivitäten begonnen wird.

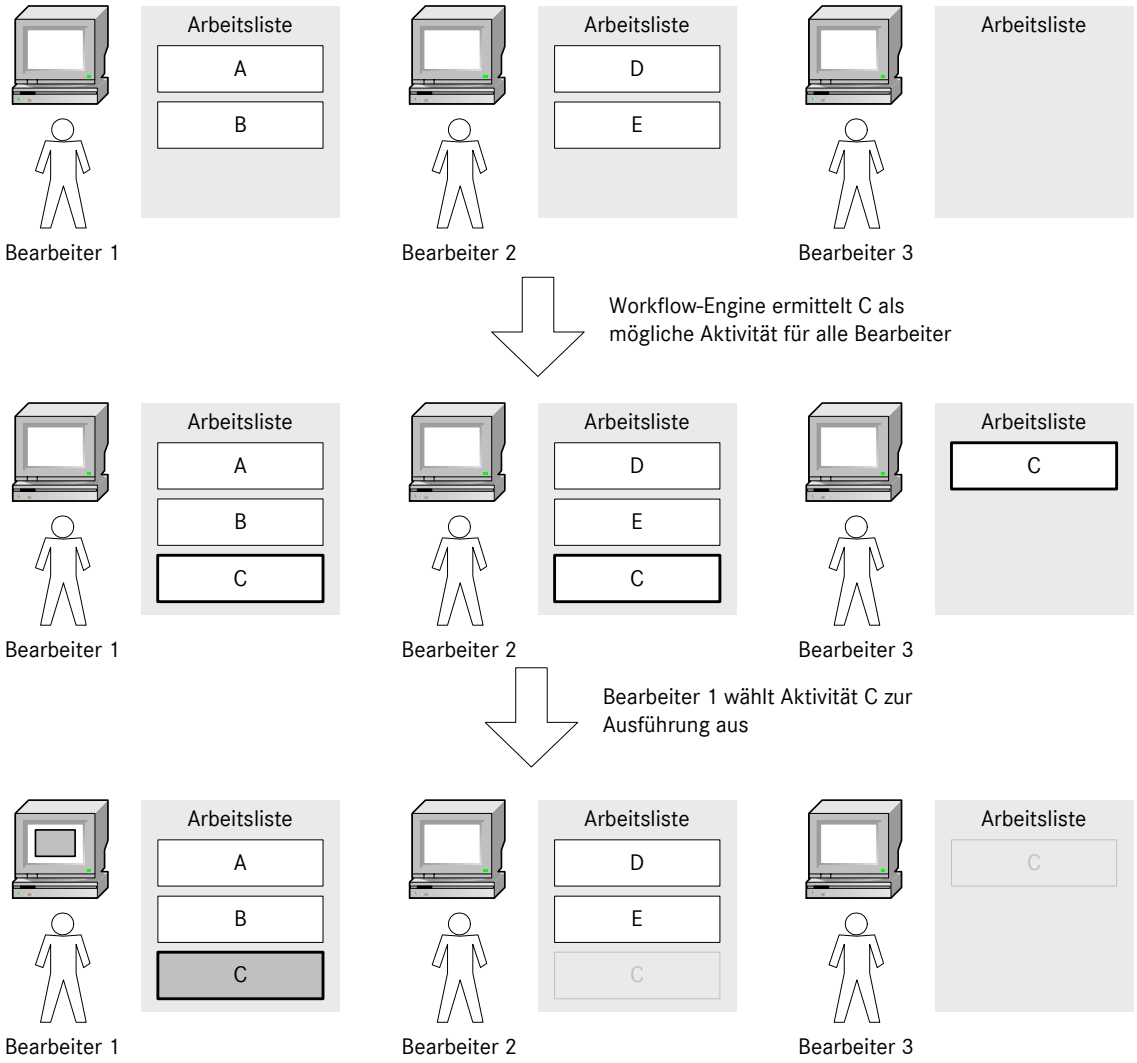

Abbildung 4.5: Ausführung eines Workflows aus Sicht der Bearbeiter

Abbildung 4.5 illustriert die Ausführung eines Workflows aus Sicht der Bearbeiter von Aktivitäten. In diesem Beispiel gibt es drei Bearbeiter mit entsprechend gefüllten Arbeitslisten (oben). Die Workflow-Engine ermittelt nun *C* als nächste zu bearbeitende Aktivität einer Workflow-Instanz und stellt fest, dass alle drei Bearbeiter in Frage kommen. Aktivität C wird nun zu allen Arbeitslisten der Bearbeiter hinzugefügt (Mitte). *Bearbeiter 1* entscheidet sich für die Bearbeitung der Aktivität C. Die entsprechende Anwendungskomponente wird gerufen und Aktivität C verschwindet aus den Arbeitslisten der Bearbeiter 2 und 3 (unten).

Aktivitäten durchlaufen bei Ausführung des Workflows logische Ausführungszustände. Der in Abbildung 4.6 abgebildete Zustandsautomat zeigt die wesentlichen Zustände und Zustandsübergänge gängiger WfMS. Geschlossene Pfeilspitzen bei Zustandsübergängen bedeuten, dass dieser Übergang direkt von Benutzern ausgelöst wird, offene Pfeilspitzen bedeuten, dass der Übergang automatisch durch das System erfolgt.

Die Bedeutung der Zustände im Einzelnen:

- *nicht aktiviert*: die Aktivität kann noch nicht ausgeführt werden und steht auf keiner Arbeitsliste zur Ausführung bereit.
- *nicht ausgeführt:* die Aktivität wurde bei der Ausführung ausgelassen.
- *aktiviert:* die Aktivität kann ausgeführt werden und wird allen möglichen Bearbeitern auf deren Arbeitsliste zur Ausführung angeboten.
- *laufend:* ein Bearbeiter hat sich für die Bearbeitung entschieden und die Aktivität wird gerade ausgeführt.
- *angehalten:* die Ausführung der Aktivität wurde unterbrochen.
- *abgebrochen:* die Ausführung einer Aktivität wurde abgebrochen.
- *beendet:* die Ausführung einer Aktivität wurde beendet.

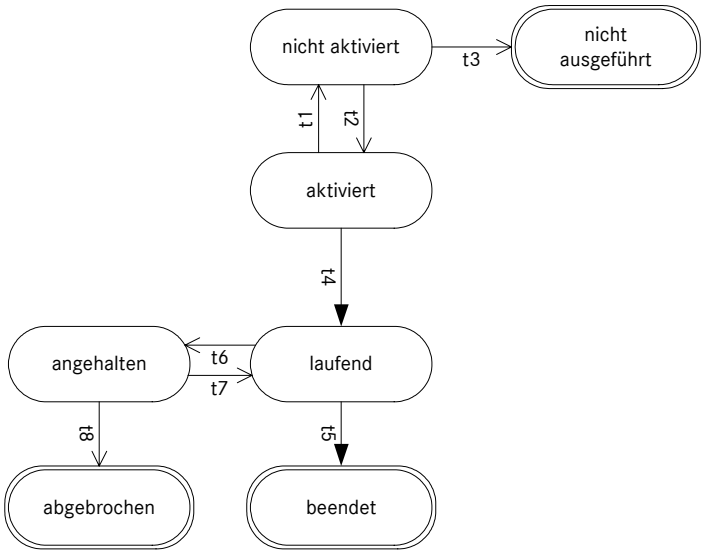

Abbildung 4.6: Logische Zustände einer Aktivität

Nicht nur Aktivitäten, sondern auch Workflow-Instanzen durchlaufen bei ihrer Ausführung verschiedene Zustände (Abbildung 4.7). Diese sind im Einzelnen:

- *initialisiert:* die Workflow-Instanz wurde technisch initialisiert, wird jedoch noch nicht ausgeführt.
- *laufend:* die Workflow-Instanz wird gerade ausgeführt.
- *beendet:* die Workflow-Instanz wurde beendet, d.h. keine Aktivität wartet mehr auf ihre Ausführung.

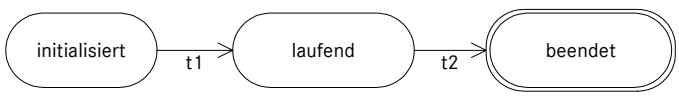

Abbildung 4.7: Logische Zustände einer Workflow-Instanz

4.5 Aktivitätennetze

Als Beispiel für einen typischen Vertreter einer Workflow-Sprache sollen Aktivitätennetze [Re03, IBM03] dienen, die das grundlegende Konzept zur Prozessmodellierung des Produkts *IBM Websphere MQWorkflow* sind. Aktivitätennetze wählen zur Modellierung eine getrennte Beschreibung von Daten- und Kontrollfluss. Aspekte der Organisationsmodellierung (z.B. Definition von Bearbeiterzuordnungen für Workflow-Aktivitäten) werden hier nicht näher betrachtet, da sie in diesem Kontext nicht von zentraler Bedeutung sind.

4.5.1 Aktivitäten

Die einzelnen Prozessschritte werden als Aktivitäten bezeichnet. Mit einer Aktivität kann ein Programm verknüpft sein, das bei der Ausführung der Aktivität durch den Benutzer ausgeführt wird. Eine Aktivität wird als Knoten eines Graphen repräsentiert.

4.5.2 Kontrollfluss

Aktivitäten werden durch gerichtete Kontrollflusskanten miteinander verbunden. Durch die gerichteten Kontrollflusskanten dürfen sich *keine Zyklen* ergeben, da sonst zur Laufzeit Verklemmungen entstehen können.

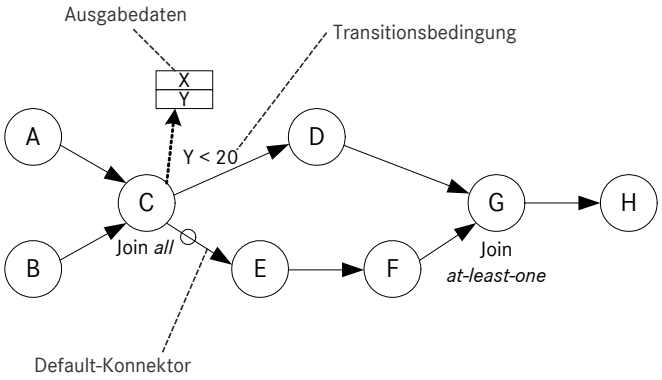

Abbildung 4.8: Beispiel eines Aktivitätennetzes

Mit jeder Kontrollflusskante kann eine Transitionsbedingung verknüpft werden, die auf die Ausgabedaten der Quellaktivität Bezug nimmt. Für jede Aktivität kann festgelegt werden, ob alle Transitionsbedingungen der eingehenden Kanten erfüllt sein müssen (Join-Semantik *all*), bevor die Aktivität zur Ausführung kommt, oder ob es genügt, wenn eine Transitionsbedingung zutrifft (Join-Semantik *at-least-one*). Ein sog. *Default-Konnektor* ist eine spezielle Kontrollflusskante *ohne explizite* Transitionsbedingung, deren implizite Transitionsbedingung darin besteht, dass sie dann wahr wird, wenn keine der anderen, vom selben Quellknoten ausgehenden, Transitionsbedingungen zutrifft. Ein Beispiel für ein Aktivitätennetz zeigt Abbildung 4.8.

Damit eine Aktivität X gestartet werden kann, müssen alle direkten Vorgänger von X beendet sein oder aber für diese muss feststehen, dass sie nicht mehr ausgeführt werden dürfen. Zusätzlich müssen die Transitionsbedingungen der eingehenden Kontrollflusskanten – entsprechend der definierten Join-Semantik – erfüllt sein. Die Ausführung eines Workflows beginnt mit den Aktivitäten, die keine einmündenden Kontrollflusskanten besitzen. Ist für eine Aktivität Y klar, dass sie nicht mehr zur Ausführung kommen kann, werden mit der sog. *Dead-*

Path-Elimination alle weiteren Aktivitäten bestimmt, die dadurch ebenfalls nicht mehr zu Ausführung kommen können.

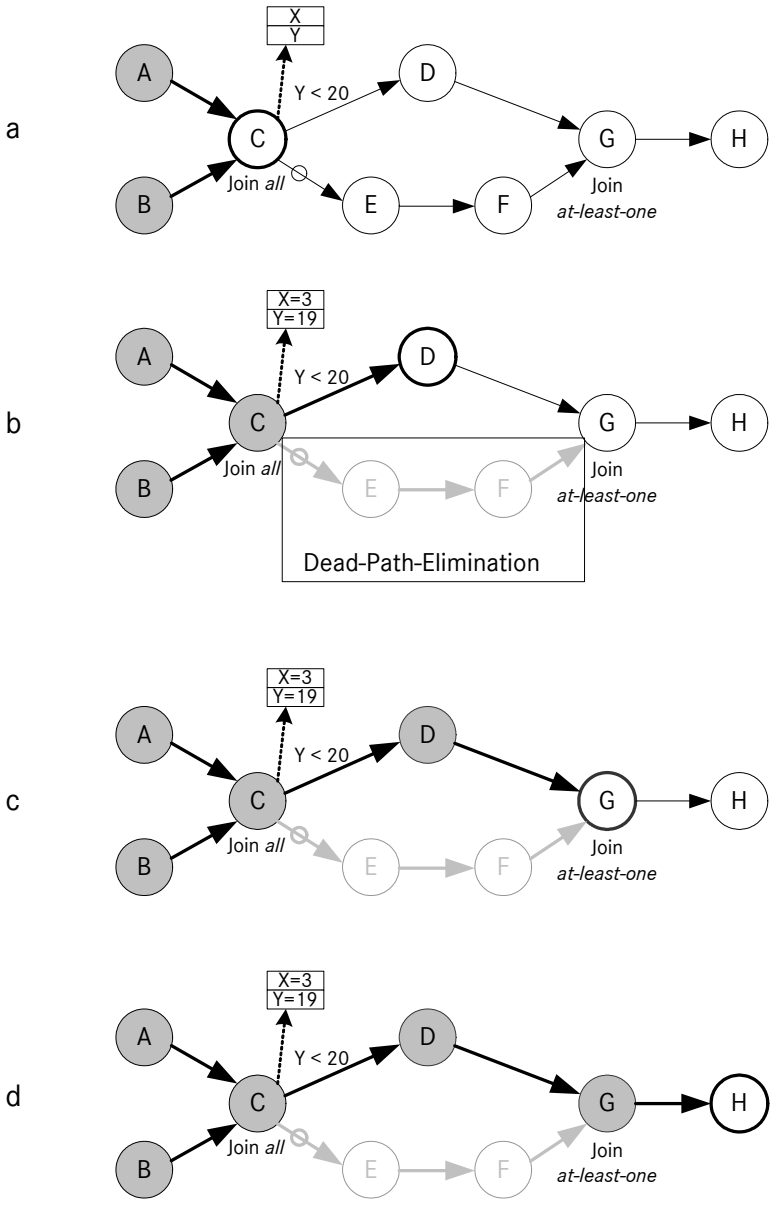

Abbildung 4.9: Ausführung eines Aktivitätennetzes

Eine mögliche Ausführung des Beispielnetzes zeigt Abbildung 4.9. Begonnen wird mit der Ausführung der Aktivitäten A und B, da beide keine eingehenden Kontrollflusskanten haben. Sind beide beendet, kann C zur Ausführung kommen (Abbildung 4.9 a). Nach Ausführung von C sind die Ausgabedatenfelder mit X=3 und Y=19 belegt. Die Transitionsbedingung Y<20 trifft zu, somit kann Aktivität D ausgeführt werden. Die Default-Transitionsbedingung nach C trifft nun nicht zu (grau dargestellt). Dadurch kann E nicht mehr ausgeführt werden, da dies die einzige eingehende Kontrollflusskante ist. Im Rahmen einer *Dead-Path-Elimination* wird daraufhin ermittelt, dass dann auch F nicht mehr zur Ausführung gelangen kann. Die Dead-Path-Elimination stoppt bei Aktivität G, da G immer noch durch den oberen Zweig aktiviert werden kann (Abbildung 4.9 b). Nach Ausführung von D kann G (Abbildung 4.9 c) und anschließend Aktivität H (Abbildung 4.9 d) ausgeführt werden.

4.5.3 Datenfluss

Jedes Workflow-Schema besitzt jeweils einen Container für Eingabedaten und einen für Ausgabedaten des Gesamtworkflows. Jede Aktivität besitzt selbst wiederum einen Ein- und Ausgabedatencontainer, durch die hinterlegte Programme mit Daten versorgt werden können bzw. in denen die Ergebnisse von Programmaufrufen abgelegt werden. Ein Datencontainer enthält ein oder mehrere Felder eines Typs, der elementar oder komplex strukturiert sein kann.

Der Daten*fluss* wird mit Datenflusskanten modelliert. Eine Datenflusskante verbindet immer einen Ausgabedatencontainer mit einem Eingabedatencontainer, es kann sich dabei sowohl um die Datencontainer des Gesamtworkflows als auch um Datencontainer einzelner Aktivitäten handeln. Verbindet eine Datenflusskante eine Aktivität A mit einer Aktivität B, werden die Felder der Eingabedatencontainer mittels eines *Mapping* auf die des Ausgabedatencontainers abgebildet. Zur Laufzeit heißt dies, dass die Daten, die von einer bestimmten Aktivität produziert werden, nach deren Beendigung einer anderen Aktivität zur Verfügung gestellt werden. Der Modellierer muss darauf achten, dass der Datenfluss immer in Richtung des Kontrollflusses verläuft, um sicherzustellen, dass zur Laufzeit alle Aktivitäten mit ihren Eingabedaten versorgt werden können. Abbildung 4.10 zeigt ein Beispiel für Datenfluss in Aktivitätennetzen.

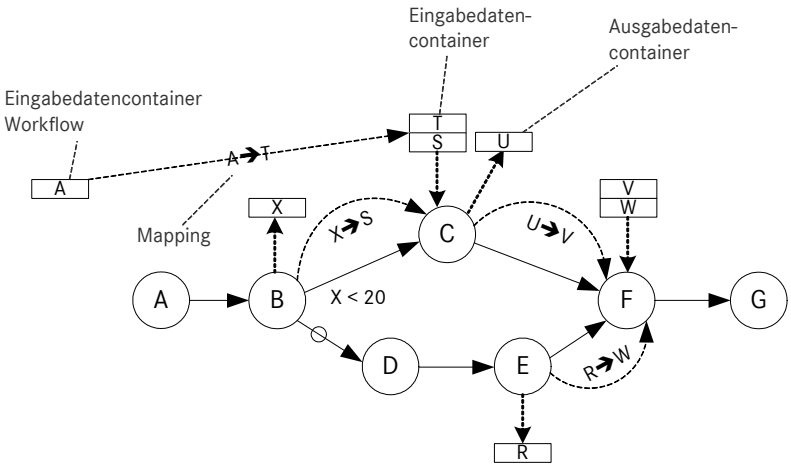

Abbildung 4.10: Datenfluss in Aktivitätennetzen

4.6 Klassifikation von Workflows

Workflows werden in [LeRo00] durch drei orthogonale Dimensionen klassifiziert (siehe Abbildung 4.11). Die Dimension der *Geschäftswertigkeit* beschreibt die Bedeutung eines Workflows für den Geschäftserfolg eines Unternehmens. Die *Wiederholung* misst, wie oft ein bestimmter Workflow demselben Schema folgend ausgeführt wird. Bezüglich dieser beiden Dimensionen lassen sich nun vier Klassen von Workflows unterscheiden:

- *Ad-hoc-Workflows* sind Workflows von niedrigem Wert für den Geschäftserfolg, die sich selten in derselben Art wiederholen. Typischerweise wird der nächste Schritt in einem Workflow erst nach Abschluss eines zuvor beendeten Schrittes bestimmt.
- *Administrative Workflows* sind Workflows, die sich zwar häufig wiederholen, jedoch zum Erfolg eines Unternehmens einen eher geringen Beitrag leisten. Beispiel hierfür ist ein Workflow zur Spesenabrechnung.

- *Kollaborative Workflows* sind durch einen hohen Beitrag zum Unternehmenserfolg charakterisiert, werden jedoch selten in derselben Art wiederholt. Entwicklungsworkflows sind beispielsweise in diese Kategorie einzuordnen.
- *Produktionsworkflows* haben sowohl eine hohe Wiederholungsrate als auch einen hohen Wert für das Unternehmen. Die effiziente Ausführung dieser Workflows bietet Unternehmen Vorteile gegenüber Wettbewerbern.

Als weiteres orthogonales Unterscheidungsmerkmal wird in [LeRo00] der *Automatisierbarkeitsgrad* herangezogen. Dieser gibt die Unabhängigkeit des Workflows von menschlicher Beteiligung an. Die Spanne reicht dabei von vollständig von Menschen ausgeführten bis hin zu vollautomatisierten Workflows.

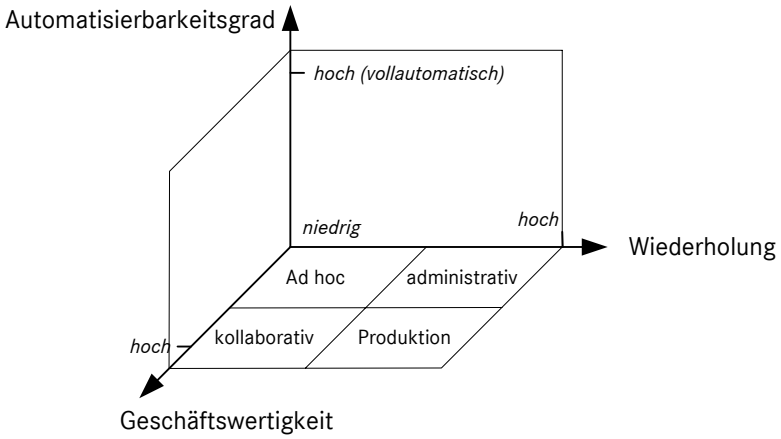

Abbildung 4.11: Kategorien von Workflows nach [LeRo00]

4.7 Kategorien von WfMS

So wie es unterschiedliche Arten von Workflows gibt, existieren auch unterschiedliche Typen von WfMS. Eine wesentliche – jedoch nicht vollständige – Unterscheidung ist diejenige zwischen *prozessorientierten* und *dokumentenorientierten* WfMS [Re00, Re03].

Prozessorientierte WfMS basieren auf der Vorstellung, dass ein Prozess durch ein Workflow-Schema explizit modelliert und logisch zentral ausgeführt wird. Wesentlich ist hier das explizite Bekanntmachen der Prozesslogik gegenüber dem WfMS und die Trennung von Prozesslogik und Anwendungscode. Somit entsprechen prozessorientierte WfMS im Wesentlichen dem, was in dieser Arbeit bis jetzt als WfMS bezeichnet wurde. Sie werden meist für *Produktionsworkflows* eingesetzt.

Bei *dokumentenorientierten WfMS* stehen die in einem Workflow bearbeiteten Dokumente im Mittelpunkt. Ein Workflow ist hier eine Menge von Dokumenten, die von Bearbeiter zu Bearbeiter weitergereicht werden [Re00]. Hier wird der Weg einer Menge von Dokumenten als Workflow modelliert. Typisch ist dabei auch, dass dem WfMS selbst der Inhalt der Dokumente nicht bekannt ist. Dokumentenorientierte WfMS werden bei Ad-hoc-Workflows oder administrativen Workflows eingesetzt.

Orthogonal zu dieser Unterscheidung kann zwischen *autonomen* und *eingebetteten* WfMS unterschieden werden [MuAl00]. *Autonome WfMS* sind eigenständige Software, die Workflow-Funktionalität anbietet. Sie besitzen eine eigene Benutzeroberfläche und können unterschiedliche externe Anwendungen prozessorientiert miteinander verknüpfen. Ein *eingebette-*

tes WfMS funktioniert nur innerhalb des Anwendungssystems, in das es integriert ist. Die eingebetteten Workflow-Komponenten werden dazu benutzt, Ausführungsreihenfolgen von Anwendungsfunktionen zu steuern. Dies ist häufig bei PDM-Systemen, ERP-Anwendungen oder Dokumentenmanagement-Lösungen der Fall.

4.8 Zusammenfassung

WfMS stellen eine viel versprechende Technologie zur Entwicklung flexibler, prozessorientierter Anwendungen dar. Auch für Freigabeprozesse erwartet man sich durch den Einsatz von Workflow-Technologie Vorteile. Das nächste Kapitel beschäftigt sich daher mit den Gründen für eine Workflow-Management-Unterstützung von Freigabeprozessen und mit Anforderungen in diesem Zusammenhang.

5 Anforderungen bezüglich Workflow-Unterstützung von Freigabeprozessen

Wie in Kapitel 3 angedeutet, ist eine Unterstützung der Freigabeprozesse durch Workflow-Technologie dringend erforderlich. Dieses Kapitel diskutiert die in diesem Zusammenhang bestehenden Anforderungen: In Abschnitt 5.1 wird dargestellt, warum für Freigabeprozesse eine Unterstützung durch Workflow-Management-Technologie notwendig ist. In Abschnitt 5.2 werden die funktionalen Anforderungen an ein IT-System diskutiert, das auf Basis eines WfMS Freigabeprozesse unterstützen soll. Abschnitt 5.3 skizziert, wie diese Anforderungen mit den Konzepten eines gängigen WfMS umgesetzt werden könnten. In diesem Zusammenhang wird gezeigt, dass konventionelle WfMS nicht alle Anforderungen abdecken. Daraus lassen sich *weitergehende* Anforderungen an WfMS selbst ableiten, die in Abschnitt 5.4 zusammengefasst sind. Eine kompakte Zusammenstellung der Anforderungen ist auch in [BHR05] zu finden.

5.1 Warum wird Workflow-Management-Unterstützung gebraucht?

In einem modernen Fahrzeug befinden sich bis zu 70 Steuergeräte. Dies bedeutet, dass während der Entwicklung hunderte hierarchische Konfigurationen entstehen, da für die in Konfigurationen gebündelten Produktkomponenten zahlreiche Versionen existieren. Zu jeder einzelnen Konfiguration gehört ein Freigabeprozess mit entsprechenden Abhängigkeiten von einer Vielzahl anderer Prozesse. Abbildung 3.6 (S. 15) zeichnet ein sehr idealisiertes Bild dieses Sachverhalts, da in der Realität eine Teilkonfiguration zu mehreren Konfigurationen gehören kann. Ein realistischeres Bild zeigt Abbildung 5.1, in der die Reihenfolgeabhängigkeiten zwischen Schritten unterschiedlicher Freigabeprozesse durch gestrichelte Kontrollflusskanten visualisiert sind.

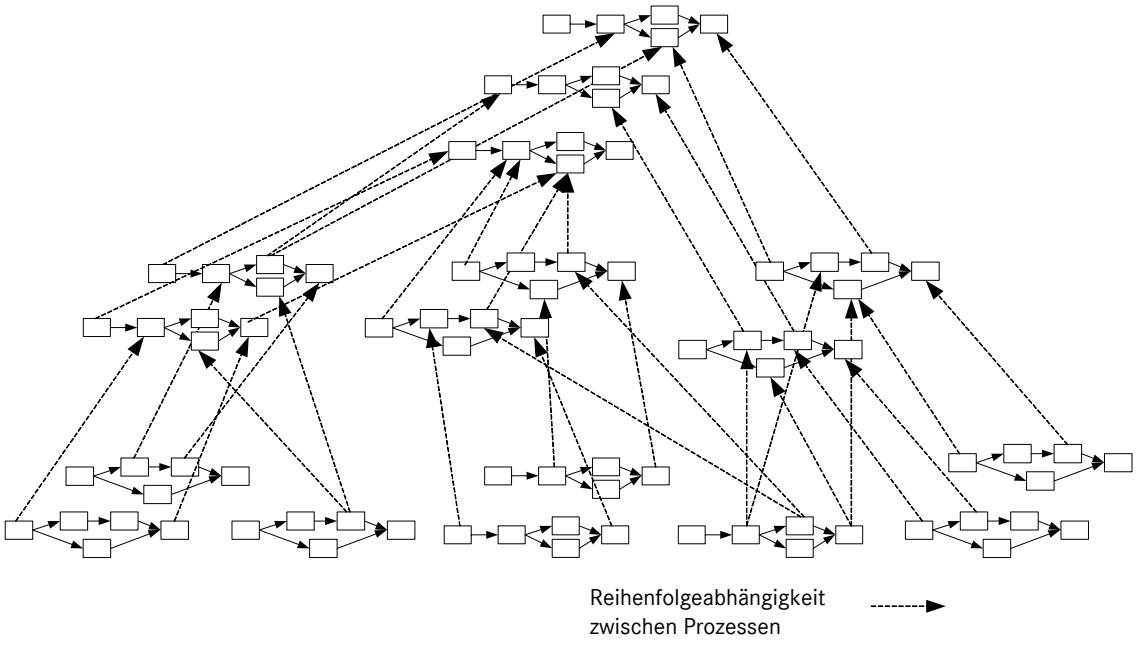

Abbildung 5.1: Komplexes Geflecht der Freigabeprozesse

Anhand dieser Fakten ist offensichtlich, dass eine adäquate IT-Unterstützung zur koordinierten Ausführung dieser Prozesse im Sinne von Workflow-Management benötigt wird. Eine manuelle Koordination und Synchronisation der Prozesse ist dagegen aufwändig und fehleranfällig. Das Hauptziel für die informationstechnische Unterstützung der Freigabeprozesse liegt demnach darin, die *korrekte* Ausführung der Prozesse zu gewährleisten. Dies schließt die Kontrolle und Überwachung der Abhängigkeiten zu anderen Freigabeprozessen ein. Durch Erreichen dieses Hauptziels trägt die Workflow-Unterstützung auch zur Realisierung von ökonomischen Zielen (z.B. Kostenreduzierung und Verkürzung der Prozessdauern) bei.

Durch den Einsatz eines WfMS wird als Nebeneffekt auch zur Lösung eines weiteren aktuellen Problems im Automobilbereich beigetragen. Nach der Änderung des Produkthaftungsgesetzes im Jahre 2002 [ProdHaftG02] sind die Automobilhersteller dazu verpflichtet, alle Qualitätssicherungsmaßnahmen für ein Produkt zu dokumentieren, bevor dieses auf den Markt kommt. Kann der Hersteller bei Produktfehlern nicht nachweisen, dass er alle Vorkehrungsmaßnahmen nach dem jeweiligen Stand der Technik getroffen hat, muss er im Schadensfall mit hohen Ersatzansprüchen rechnen. In den meisten Fällen bieten WfMS eine Logging-Funktion an, mit der alle ausgeführten Aktivitäten aufgezeichnet werden. Die dabei entstehenden Daten bieten eine gute Grundlage für diese Dokumentation.

5.2 Vision der IT-Unterstützung für Freigabeprozesse

In dieser Arbeit wurden Anforderungen für die Vision einer IT-Unterstützung der Freigabeprozesse analysiert. Das Ergebnis dieser Analyse sind fünf funktionale Hauptanforderungen an ein IT-System, das basierend auf Workflow-Management-Technologie Freigabeprozesse unterstützen soll. Wenn in der folgenden Beschreibung der Anforderungen davon die Rede ist, dass etwas getan werden muss, soll oder darf, so ist dies immer als Anforderung an das IT-System zu verstehen, wenn nicht explizit etwas anderes gesagt wird.

5.2.1 Starten von Freigabeprozessen für Konfigurationen (Anforderung 1)

Der für eine Konfiguration Verantwortliche sollte prinzipiell in der Lage sein, einen entsprechenden Freigabeworkflow zu einem „beliebigen“ Zeitpunkt zu starten. Jedoch darf dieser nur gestartet werden, wenn die Freigabeworkflows der Teilkonfigurationen bereits gestartet oder schon erfolgreich beendet worden sind.

Zu diesem Zeitpunkt müssen die Teilkonfigurationen der betroffenen Konfiguration festgelegt sein, da sich die Freigabe auf das Zusammenspiel der Teilkonfigurationen bezieht und keine Aussagekraft mehr hätte, wenn die Teilkonfigurationen erst während des Freigabeprozesses festgelegt werden. In welchen Konfigurationen die freizugebende Konfiguration als Teilkonfiguration verwendet wird, muss hingegen zum Zeitpunkt des Starts und der Durchführung des Freigabeprozesses *nicht* bekannt sein (vgl. Abschnitt 3.3.1).

5.2.2 Unterstützung der Benutzer bei Prüfschritten (Anforderung 2)

Für jeden Schritt in einem Freigabeworkflow sollte der Bearbeiter Zugriff auf die Konfiguration erhalten und darüber informiert werden, welchen Prüfschritt er mit dieser Konfiguration ausführen soll. Nach Beendigung seiner Aufgabe sollte der Benutzer die Möglichkeit haben, dem System mitzuteilen, ob ein Fehler in der Konfiguration entdeckt wurde.

Die Granularität der Prüfschritte ist sehr grob. Prüfschritte können einige Zeit dauern und werden im Regelfall komplett außerhalb des WfMS ausgeführt. Nur das Ergebnis eines Prüf-

schritts soll dem WfMS zurückgemeldet werden. Eine automatisierte Prozessausführung im Sinne des Verknüpfens und des Aufrufs von Anwendungen liegt in diesem Kontext nicht im Fokus und ist allenfalls in Ausnahmefällen denkbar, beispielsweise bei automatisierten Tests von Softwarekomponenten.

5.2.3 Zuweisen des Freigabestatus (Anforderung 3)

Nachdem ein Freigabeworkflow beendet wurde, soll der zugehörigen Konfiguration der passende Freigabestatus zugewiesen werden. Für den Fall, dass in der Konfiguration mindestens ein Fehler gefunden wurde, ist dies „Nicht freigegeben“, ansonsten der Status „Freigegeben“ (vgl. Abschnitt 3.1.2). Dieser Freigabestatus kann von allen Bearbeitern gesehen werden, die im Laufe des Entwicklungsprozesses Zugriff auf die Konfiguration benötigen.

Folgende Randbedingung ist zu beachten: Eine Konfiguration darf den Freigabestatus „Freigegeben“ erst dann erhalten, wenn *alle* Teilkonfigurationen den gleichen Status besitzen.

5.2.4 Beachtung hierarchischer Kontrollflussabhängigkeiten (Anforderung 4)

Bei Ausführung der Freigabeworkflows sollen die Kontrollflussabhängigkeiten zwischen Freigabeworkflows auf verschiedenen Ebenen der Konfigurationshierarchie beachtet werden. Der Freigabeprozess einer Konfiguration auf einer höheren Ebene darf an manchen Stellen nicht fortgesetzt werden, bis alle Freigabeprozesse der Teilkonfigurationen in ihrer Ausführung bestimmte Punkte erreicht haben. Konkrete Beispiele für solche Abhängigkeiten sind in Abschnitt 3.3.3 erläutert.

5.2.5 Flexible Reaktionsmöglichkeiten bei Prüffehlern (Anforderung 5)

In jedem Prüfschritt eines Freigabeprozesses können potenziell Fehler in einer Konfiguration gefunden werden. Wird ein solcher Fehler entdeckt, darf der entsprechende Freigabeworkflow nicht normal weiterlaufen. Vielmehr soll in diesem Fall vom Prozessverantwortlichen für diese Konfiguration entschieden werden, wie mit dem Freigabeworkflow weiter verfahren wird. Betroffen sind von einem Fehler in einer Konfiguration nicht nur der Freigabeworkflow dieser Konfiguration, sondern auch andere Freigabeworkflows in der Konfigurationshierarchie.

Aus Gründen der besseren Darstellbarkeit wird diese Anforderungen in zwei Teilanforderungen unterteilt: Zunächst soll als erste Teilanforderung dargestellt werden, welche Reaktionen auf Fehler bei einem einzelnen Freigabeworkflow möglich sein sollen, bevor dies in einer zweiten Teilanforderung für die Konfigurationshierarchie beschrieben wird.

5.2.5.1 Fehlerreaktion bei einzelnen Konfigurationen (Anforderung 5.1)

Wenn in einem Prüfschritt in einer Konfiguration ein oder mehrere Fehler gefunden werden, soll der Prozessverantwortliche benachrichtigt werden und die Möglichkeit haben, flexibel über das weitere Vorgehen zu entscheiden. Ein Fehler in einer Konfiguration bedeutet, dass diese *nicht mehr* freigegeben werden kann. Alle zu diesem Zeitpunkt noch ausstehenden Schritte können keine Freigabe mehr bewirken. Für einige Schritte kann es jedoch trotzdem sinnvoll sein, sie durchzuführen, weil sie für den weiteren Entwicklungsprozess wertvolle Ergebnisse produzieren. Bei anderen Schritten, wie formalen Genehmigungen, macht dies weniger Sinn.

Das System sollte somit dem Prozessverantwortlichen in Bezug auf den Freigabeworkflow die folgenden beiden Alternativen ermöglichen:

- Abbruch des Workflows
- Weitere Ausführung des Workflows mit der Möglichkeit, bestimmte Schritte von der Ausführung auszunehmen

Bei diesen Entscheidungen sollte das System den Benutzer so unterstützen, dass diese effektiv getroffen werden können.

5.2.5.2 Fehlerreaktion Unter- und Oberkonfigurationen (Anforderung 5.2)

Das Finden eines Fehlers in einer Konfiguration hat nicht nur Konsequenzen für den direkt betroffenen Freigabeworkflow, in dem der Fehler gefunden wurde, sondern auch für die Freigabeworkflows der über- und untergeordneten Konfigurationen.

Wie in Anforderung 5.1 dargelegt, sollen die Prozessverantwortlichen der betroffenen Konfigurationen ebenfalls benachrichtigt werden und dieselben Möglichkeiten zur Fehlerreaktion für ihren Workflow haben. Hinzu kommt, dass über die weitere Gültigkeit von Kontrollflussabhängigkeiten zu anderen Prozessen entschieden werden muss. Da diese Reaktion vom Zustand der Freigabeprozesse anderer Konfigurationen abhängt, benötigt der Nutzer vom System die Möglichkeit, sich schnell einen Überblick über den Zustand und die Abhängigkeiten der betroffenen Freigabeworkflows zu verschaffen.

Aber welche Konfigurationen (und jeweilige Freigabeworkflows) müssen berücksichtigt werden, wenn in einer bestimmten Konfiguration ein Fehler gefunden wurde?

Zuerst einmal müssen im Sinne einer „Bottom-Up-Fehlerbehandlung“ nacheinander alle Oberkonfigurationen bearbeitet werden (siehe Abbildung 5.2). Dies ist unbedingt erforderlich, da eine Konfiguration den Freigabezustand „Freigegeben“ nicht erreichen kann, wenn eine ihrer Teilkonfigurationen nicht erfolgreich freigegeben wurde. Deshalb muss, wenn für eine bestimmte Konfiguration ein Fehler gefunden wurde, der Prozessverantwortliche der jeweiligen Oberkonfiguration entscheiden, ob es Sinn macht, den betroffenen Freigabeprozess fortzusetzen (oder zu starten), selbst wenn die Oberkonfiguration niemals freigegeben werden kann.

In dem in Abbildung 5.2 dargestellten Beispiel bedeutet dies Folgendes: Wir nehmen an, dass bei Ausführung eines Prüfschritts für die Konfiguration des linken Türsteuergeräts (TSL) ein Fehler entdeckt wird. Zuerst wird der Verantwortliche für diese Konfiguration benachrichtigt. Er kann dann seine Entscheidungen bezüglich des weiteren Prozessverlaufs treffen. In einem zweiten Schritt werden die Verantwortlichen der Systeme *Spiegelverstellung* und *Außenlicht* benachrichtigt, so dass sie ebenfalls auf die Fehlersituation reagieren können. Zum Schluss wird der Verantwortliche für die Gesamtkonfiguration informiert. Er kann, wie die anderen Prozessverantwortlichen zuvor, auf die weitere Ausführung des Freigabeprozesses Einfluss nehmen.

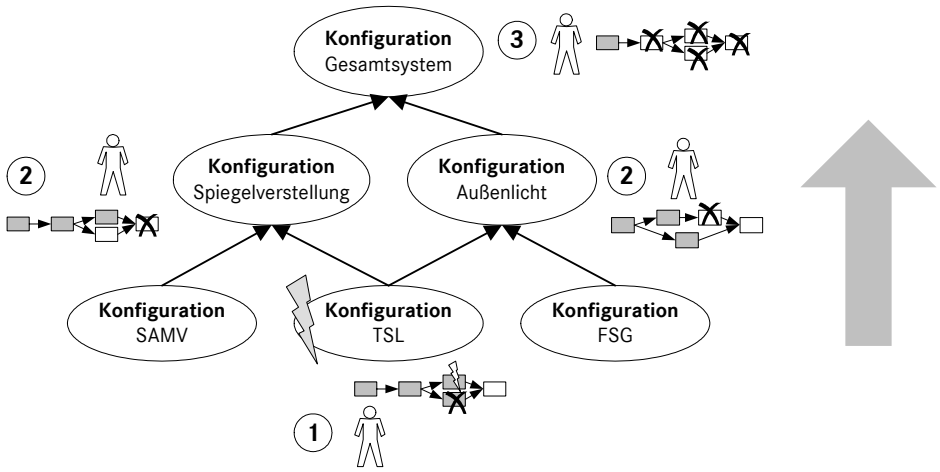

Abbildung 5.2: Bottom-Up-Fehlerbehandlung

Zusätzlich ist es möglich, dass beim Entdecken eines Fehlers die ursächliche Teilkonfiguration identifiziert werden kann. Wenn sich der Freigabeworkflow noch in Ausführung befindet, soll der Verantwortliche benachrichtigt werden und dieselben Möglichkeiten zur Einflussnahme auf den weiteren Prozessverlauf zur Verfügung haben wie in allen anderen Fällen zuvor. Diese Konfiguration soll ebenfalls nicht mehr in den Freigabezustand „Freigegeben“ gelangen können. Diese „Top-Down-Fehlerbehandlung“ kann zusätzlich „Bottom-Up-Fehlerbehandlung“ für betroffene Oberkonfigurationen erfordern, da alle Oberkonfigurationen dieser Konfiguration nun auch nicht mehr freigegeben werden können.

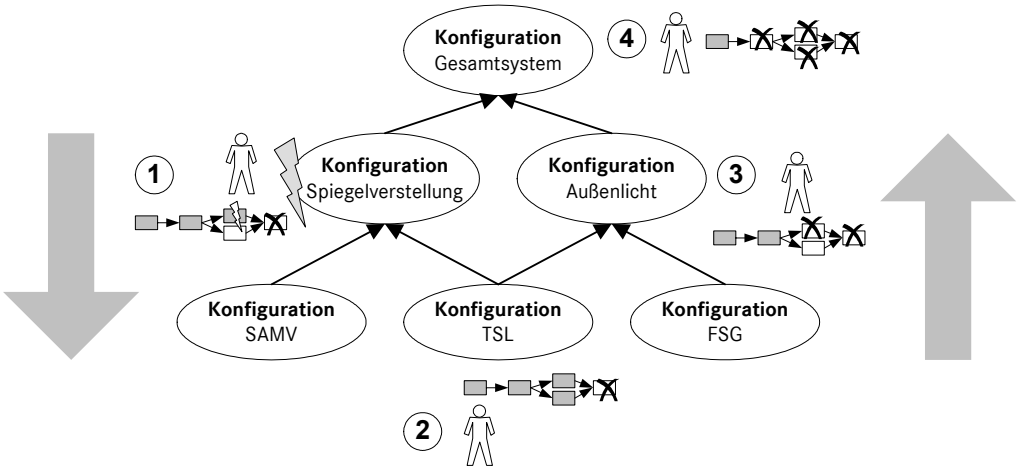

Abbildung 5.3: Top-Down- und Bottom-Up-Fehlerbehandlung

Abbildung 5.3 zeigt hierfür ein Beispiel. In der Konfiguration des *Spiegelverstellungssystems* wird ein Fehler entdeckt. Dieser Fehler kann auf einen Fehler in der Konfiguration des Türsteuergeräts zurückgeführt werden. Somit wird nach der Reaktion auf den Fehler in der Spiegelverstellungskonfiguration der Verantwortliche für die Türsteuergerätekonfiguration benachrichtigt, damit er den Freigabeprozess entsprechend beeinflussen kann. Um die Konsistenz der Freigabezustände zu erhalten, beginnt wiederum eine „Bottom-Up-Fehlerbehandlung“ für die Konfiguration *Außenlicht* und anschließend für die Gesamtsystemkonfiguration. (Anmerkung: Da die Gesamtssystemkonfiguration eine Oberkonfiguration des Spiegelverstellungssystems ist, wäre die Fehlerbehandlung für die Gesamtkonfiguration

trotzdem angestoßen worden, auch wenn es nicht zur Top-Down-Fehlerbehandlung gekommen wäre.)

5.2.6 Einordnung der Freigabeworkflows

Bezogen auf die in Abschnitt 4.6 vorgestellte Klassifikation lassen sich Freigabeworkflows folgendermaßen einordnen:

- *Geschäftswertigkeit:* Die Bedeutung der Freigabeprozesse ist für den Automobilhersteller nicht ganz so hoch wie die der Kernprozesse „Produktion“ und „Bestellung“. Die mit den Freigabeprozessen unmittelbar verbundene Produktqualität ist für Hersteller jedoch ein wesentlicher Erfolgsfaktor. Damit ist auch die Wertigkeit der Freigabeworkflows relativ hoch anzusetzen.
- *Wiederholung:* Freigabeworkflows werden im Laufe der Entwicklung sehr oft angestoßen. Damit ist der Grad der Wiederholung als hoch einzustufen.
- *Automatisierbarkeitsgrad:* Workflow-Unterstützung dient bei Freigabeprozessen nicht der Automatisierung von Prüfschritten, sondern der Koordinierung des Freigabeprozesses und dessen Synchronisation mit anderen Prozessen. Der Grad der Automatisierung ist also niedrig und die menschliche Interaktion steht im Vordergrund.

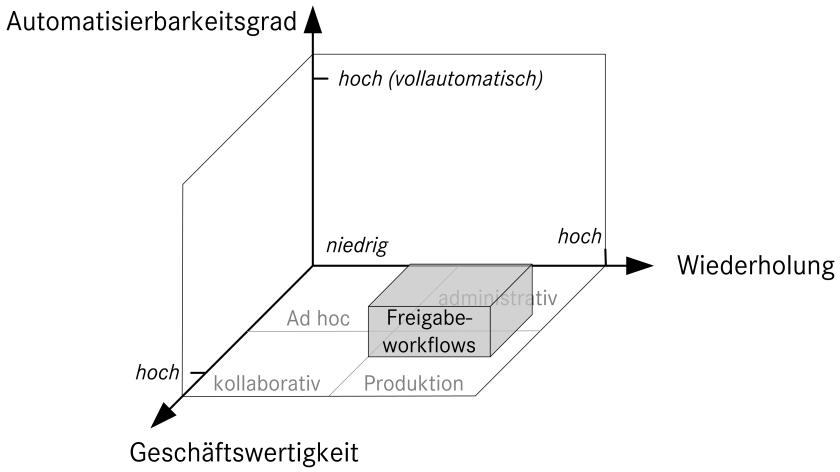

Abbildung 5.4: Einordnung der Freigabeworkflows in Workflowkategorien nach [LeRo00]

5.3 Umsetzbarkeit mit derzeitiger Workflow-Technologie

Die IT-Unterstützung von Freigabeprozessen stellt besondere Anforderungen an WfMS. Im Folgenden soll skizziert werden, wie weit die im vorangegangenen Abschnitt beschriebenen funktionalen Anforderungen mit den Konzepten eines gängigen WfMS umgesetzt werden können. Aufgrund der Nähe zu Produktionsworkflows fiel die Wahl dabei mit *IBM Websphere MQWorkflow* auf einen typischen Vertreter prozessorientierter WfMS, dessen konzeptuelle Möglichkeiten in [Re03] und [IBM03] beschrieben sind. Ziel ist es hierbei, grundsätzliche Probleme beim Einsatz marktüblicher WfMS für Freigabeprozesse zu erkennen und *nicht*, detaillierte Produktbewertung vorzunehmen. Deshalb wird weitgehend auf die Darstellung produktspezifischer Details verzichtet. Workflows werden in MQWorkflow mit *Aktivitätennetzen* modelliert (vgl. Abschnitt 4.5). Abschnitt 5.3.1 beschreibt eine mögliche Umsetzung der Anforderungen mit diesem Formalismus, Abschnitt 5.3.2 fasst die sich daraus ergebenden Erkenntnisse zusammen.

5.3.1 Umsetzung der Anforderungen

In diesem Abschnitt soll skizziert werden, wie mit Aktivitätennetzen die funktionalen Anforderungen aus Abschnitt 5.2 umgesetzt werden können. Die Darstellung der möglichen Umsetzung geschieht schrittweise: Nach der Modellierung eines erfolgreichen Freigabeprozesses wird dieses Modell für die Reaktion auf Prüffehler erweitert und abschließend die mögliche Umsetzung der hierarchischen Abhängigkeiten aufgezeigt.

5.3.1.1 Modellierung eines erfolgreichen Freigabeprozesses

Sind alle Prüfschritte eines Freigabeprozesses erfolgreich durchgeführt worden, ohne dass in der entsprechenden Konfiguration ein Fehler gefunden wurde, so kann die entsprechende Konfiguration freigegeben werden.

Ob in einem Prüfschritt einer Konfiguration ein Fehler gefunden wurde oder nicht, kann durch ein Datenfeld im Ausgabedatencontainer der Aktivität modelliert werden. Für unseren Fall nehmen wir an, dass jede Aktivität das Feld „Ergebnis“ im Ausgabedatencontainer mit den mögliche Werten „OK“ oder „Fehler“ besitzt, das angibt, ob durch diese Aktivität ein Fehler in der Konfiguration gefunden wurde. Das Durchführen einer Aktivität ohne Finden von Fehlern ist Voraussetzung für die Durchführung der Folgeaktivitäten. Das kann durch einfache Transitionsbedingungen ausgedrückt werden. In einem ersten Schritt ergibt sich dann das in Abbildung 5.5 dargestellte Workflow-Modell für den Beispielfreigabeprozess eines Steuergeräts (siehe Abschnitt 3.3.3). Datenfluss und Datencontainer sind aus Gründen der besseren Übersichtlichkeit nicht dargestellt.

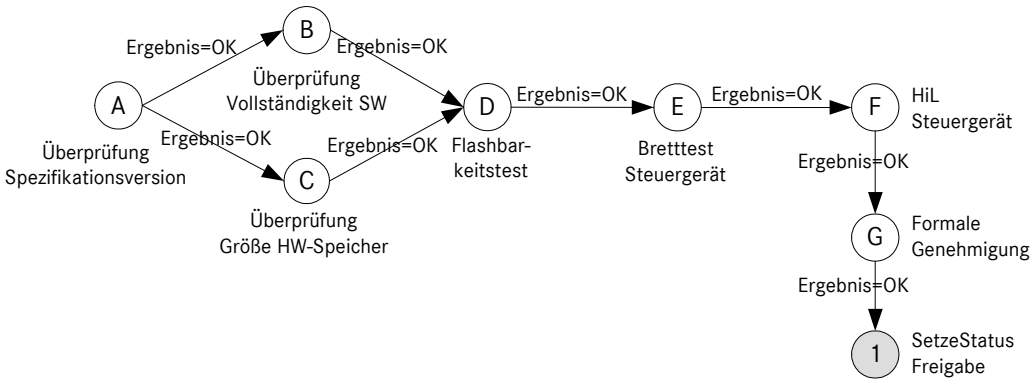

Abbildung 5.5: Workflow-Modell für einen erfolgreichen Freigabeprozess

Als zusätzliche Aktivität wurde die Aktivität *SetzeStatusFreigabe* eingefügt. Diese wird nach dem letzten erfolgreichen Prüfschritt automatisch ausgeführt. Dabei wird ein Programm gerufen, das im System, in dem die Konfigurationen verwaltet werden, den Status der Konfiguration entsprechend setzt.

5.3.1.2 Reaktion auf Prüffehler im Einzelworkflow

Bis jetzt ist im Workflow-Modell nur der Fall komplett abgebildet, bei dem alle Prüfschritte *erfolgreich* ausgeführt werden.

Anforderung 5.1 (siehe Abschnitt 5.2.5.1) beschreibt, welches Verhalten des Gesamtsystems bei einem Prüffehler im einzelnen Workflow gewünscht ist. Wenn in einem Prüfschritt in einer Konfiguration Fehler gefunden worden sind, soll der Verantwortliche informiert werden. Dieser soll dann entscheiden können, ob der Prozess weiterläuft und welche Schritte ggf.

wegzulassen sind. In jedem Fall sollte der Status der Konfiguration anschließend auf „Keine Freigabe“ gesetzt werden.

Diese Anforderung lässt sich durch die gegebenen Mittel nur durch Vormodellieren *aller* möglichen Fälle realisieren. Für jede Aktivität muss für den Fall eines Fehlers zu einer Aktivität verzweigt werden, die den Verantwortlichen benachrichtigt. Bei dieser Aktivität muss sich dieser für einen von mehreren vormodellierten Pfaden entscheiden. Nach dieser Aktivität müssten nun also alle möglichen Kombinationen modelliert werden, die beim Auslassen von Schritten vom aktuellen Zustand aus möglich sind. Abbildung 5.6 zeigt dies beispielhaft für die Aktivität D. Das Setzen des Status „Keine Freigabe“ erfolgt wiederum durch eine weitere automatische Aktivität „SetzeStatusKeineFreigabe“. Für ein vollständiges Modell müsste diese Vormodellierung natürlich für *alle* Aktivitäten erfolgen.

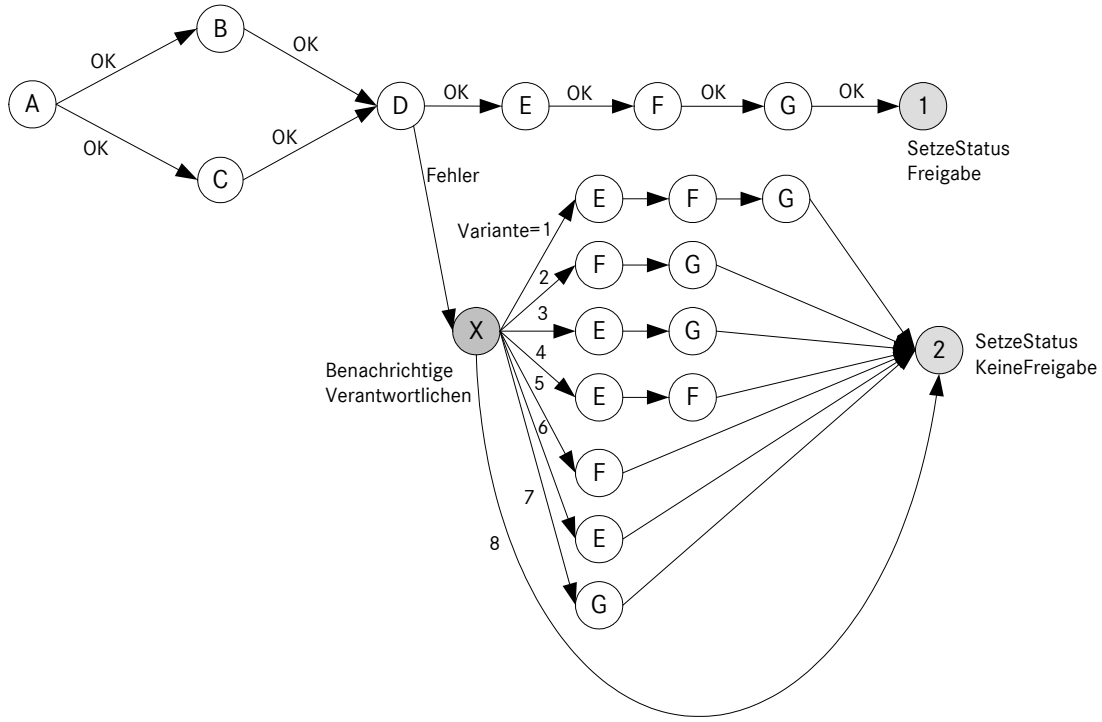

Abbildung 5.6: Vormodellieren der Prüffehler im Einzelworkflow

5.3.1.3 Umsetzung der hierarchischen Abhängigkeiten

Anforderung 4 (siehe Abschnitt 5.2.4) erfordert die Berücksichtigung der hierarchischen Abhängigkeiten zwischen parallelen Workflow-Instanzen. Diese Abhängigkeiten können mit den Mitteln eines Standard-WfMS nicht direkt beschrieben werden, sondern müssten als „Workaround“ über den Aufruf von Hilfsanwendungen in speziellen Prozessschritten des Workflow-Modells realisiert werden. Wenn man stark vereinfachend davon ausgeht, dass es jeweils ein separates Workflow-Schema für Freigabeprozesse auf Komponentenebene und eines auf Systemebene gibt, könnte ein solcher Workaround folgendermaßen aussehen: An jeder Stelle, an der der Freigabeprozess auf bestimmte Zustände der Freigabeprozesse der Teilkonfigurationen warten muss, wird ein Anwendungsprogramm gerufen, das feststellt, ob die Workflow-Instanzen aller Teilkonfigurationen schon so weit fortgeschritten sind, wie dies zum weiteren Ausführen des Workflows der Oberkonfiguration notwendig ist. Sollte dies der Fall sein, wird mit der Ausführung fortgefahren, falls nicht, wird das Anwendungsprogramm nach einer gewissen Zeit erneut gestartet.

Dieses Anwendungsprogramm müsste über Zugriff auf das Konfigurationsmanagement-System die Teilkonfigurationen für die entsprechende Konfiguration bestimmen und anschließend über die API des WfMS oder ein separat angelegtes Verzeichnis die Workflow-Instanzen zu diesen Teilkonfigurationen finden und dann überprüfen, ob der gewünschte Fortschritt schon erreicht ist.

5.3.2 Zusammenfassende Bewertung

Der eben dargestellte Versuch, auch nur Teile der Anforderungen mit den konzeptuellen Möglichkeiten eines gängigen WfMS umzusetzen, hat gezeigt, dass dies nur mühsam unter Zuhilfenahme von Workarounds möglich ist. Es existieren keine Konzepte, um unsere Anforderungen direkt umsetzen zu können. Diese fehlenden Konzepte müssen mit von Hand programmierten, „hart-verdrahteten“ Anwendungsprogrammen umgesetzt werden.

Die dem Workflow-Ansatz zu Grunde liegende Idee der *expliziten* Modellierung der Prozesslogik (siehe Kapitel 4) wird durch diese implizite, harte Kodierung ad absurdum geführt und zieht eine Reihe von Nachteilen nach sich: Hart kodierte und nicht explizit gemachte Abhängigkeiten lassen sich schlecht warten und führen damit in der Folge zu Fehlern bzw. hohen Wartungskosten sowie unflexibler Anpassbarkeit. Das Fehlen geeigneter Konzepte verursacht extrem große und damit sehr unübersichtliche Workflow-Modelle, bei denen der ursprünglich zu Grunde liegende Prozess nicht mehr erkennbar ist und das Modell damit unverständlich und schlecht wartbar wird.

5.4 Weitergehende Anforderungen an WfMS

Abschnitt 5.3 hat gezeigt, dass die funktionalen Anforderungen an ein IT-System zur Unterstützung der Freigabeprozesse (siehe Abschnitt 5.2) mit den konzeptuellen Mitteln gängiger WfMS nicht vollständig und adäquat umgesetzt werden können. Daraus können die folgenden *weitergehenden* Anforderungen an WfMS selbst abgeleitet werden, deren konzeptuelle Umsetzung in WfMS Voraussetzung dafür ist, darauf aufbauend ein IT-System zur Unterstützung der Freigabeprozesse realisieren zu können.

5.4.1 Kontrollflussabhängigkeiten zwischen parallelen Workflows

Das WfMS sollte es ermöglichen, Kontrollflussabhängigkeiten zwischen parallelen Workflows schon auf Schema-Ebene zu modellieren und zur Laufzeit zu berücksichtigen. Hierbei sollte eine Modellierung möglich sein, die auf die Daten eines Workflows Bezug nimmt.

Diese Kontrollflussabhängigkeiten können nicht nur zwischen zwei Workflows, sondern zwischen beliebig vielen Workflows existieren: Eine Konfiguration kann beliebig viele Teilkonfigurationen besitzen. Somit gibt es Kontrollflussabhängigkeiten zu allen Freigabeworkflows der Teilkonfigurationen. Ebenso existieren Kontrollflussabhängigkeiten zu mehr als einem Freigabeworkflow der Oberkonfigurationen, da eine Konfiguration Teilkonfiguration mehrerer Konfigurationen sein kann (Abschnitt 3.3.1).

Die Datenstrukturen, mit denen die Workflows assoziiert sind und auf deren Grundlage die Kontrollflussabhängigkeiten definiert sind, können sich zur Laufzeit teilweise noch ändern. So ist es durchaus möglich, dass die zugehörigen Oberkonfigurationen einer Teilkonfiguration erst festgelegt wird, wenn der Freigabeworkflow der Teilkonfiguration schon läuft oder bereits beendet ist.

5.4.2 Anpassen von Kontrollflussabhängigkeiten zwischen parallelen Workflows

Es soll möglich sein, im Rahmen der Reaktion auf Fehler in einer Konfiguration Kontrollflussabhängigkeiten zwischen Workflows dynamisch zur Laufzeit anzupassen (siehe Abschnitt 5.2.5).

5.4.3 Löschen von Aktivitäten

Das WfMS soll ebenfalls ermöglichen, dynamisch Aktivitäten aus einem laufenden Workflow entfernen und dabei die Auswirkungen auf parallel laufende Workflows berücksichtigen zu können.

5.4.4 Unterstützung für spezielle Fehlerbehandlung

Wie in Abschnitt 5.3 deutlich wurde, ist eine explizite Vormodellierung von alternativen Pfaden für Prüffehler kein gangbarer Weg, da es zu einer exponentiellen Explosion von Aktivitäten im Workflow-Modell kommt. Das WfMS sollte also mit adäquaten Konzepten Möglichkeiten zur Behandlung dieser besonderen Art von Fehlern (vgl. Abschnitt 5.2.5) bieten.

In diesem Zusammenhang ist darauf hinzuweisen, dass die „Prüffehler“ in diesem Zusammenhang *nicht* mit den Fehlern verwechselt werden dürfen, wie sie normalerweise im Workflow-Management betrachtet werden. Details zu dieser Abgrenzung werden in Abschnitt 6.2.5 diskutiert.

5.4.5 Weitere spezielle Anforderungen

Die obigen weitergehenden Anforderungen stellen die Kernanforderungen dar, um überhaupt eine adäquate Workflow-Unterstützung für Freigabeprozesse realisieren zu können. Daneben gibt es eine Reihe von weiteren für die Praxis interessanten Fragestellungen:

- Prozessvisualisierung: Wie kann das komplexe Geflecht der Workflows so visualisiert werden, dass ein schneller Überblick möglich ist, z.B. im Fall von Prüffehlern und der damit verbundenen Entscheidung über das weitere Vorgehen?
- Prozessmonitoring: Wie kann Entscheidungsträgern und Managern eine aggregierte Sicht auf das Workflowgeflecht ermöglicht werden, die ihnen erlaubt, Prozesse effizient zu überwachen?
- Wie kann die Anbindung an ein Datenmanagementsystem zur Verwaltung von Konfigurationen bewerkstelligt werden?

5.5 Zusammenfassung

In diesem Kapitel sind die wichtigsten funktionalen Anforderungen zur Workflow-Unterstützung der Freigabeprozesse diskutiert und deren mögliche Umsetzung mit einem gängigen WfMS skizziert worden. Das Ergebnis dieser Betrachtungen besteht darin, dass die konzeptuellen Möglichkeiten von konventionellen WfMS nicht ausreichen, um Freigabeprozesse adäquat unterstützen zu können und WfMS diesbezüglich weitergehenden Anforderungen genügen müssen.

6 Stand der Wissenschaft und verwandte Arbeiten

Im vorangegangenen Kapitel wurden weitergehende Anforderungen an WfMS diskutiert, um Freigabeprozesse adäquat unterstützen zu können. In diesem Kapitel wird der Stand der Wissenschaft in Bezug auf die Kernanforderungen herausgearbeitet und bewertet.

6.1 Hierarchische Workflows

Aufgrund der hierarchischen Struktur der Konfigurationen könnte der Eindruck entstehen, es handle sich hierbei um *hierarchische Workflows*. Dies ist jedoch nicht der Fall. Deshalb wird im Folgenden dargestellt, was üblicherweise unter hierarchischen Workflows verstanden wird und warum diese für die vorliegende Problemstellung nicht ausreichend sind.

6.1.1 Aktivitätennetze

Grundlegendes zu Aktivitätennetzen wurde bereits in Abschnitt 5.3 gesagt. An dieser Stelle werden die zusätzlichen Konzepte zur hierarchischen Strukturierung von Aktivitätennetzen beschrieben [LeRo00]. Bei Aktivitätennetzen wird zwischen *Informationsaktivitäten*, *Programmaktivitäten* und *Prozessaktivitäten* unterschieden. Informations- und Programmaktivitäten sind elementar. Prozessaktivitäten hingegen sind Aktivitäten, die durch einen anderen Workflow (den sog. *Subworkflow*) implementiert werden. Bei der Ausführung von Prozessaktivitäten wird eine neue Workflow-Instanz erzeugt und abgearbeitet. Nach Beendigung dieser Workflow-Instanz geht die Kontrolle wieder zurück an den aufrufenden Workflow. Die Ein- und Ausgabedaten einer Prozessaktivität werden auf die Ein- und Ausgabedatencontainer des Subworkflow abgebildet.

Abbildung 6.1 zeigt ein Beispiel. Zur Ausführung der Prozessaktivität A4 wird eine Instanz des Workflow-Schemas B erzeugt. Wenn dieser Workflow mit Beendigung der Aktivität B5 terminiert, geht die Kontrolle zurück an Workflow A.

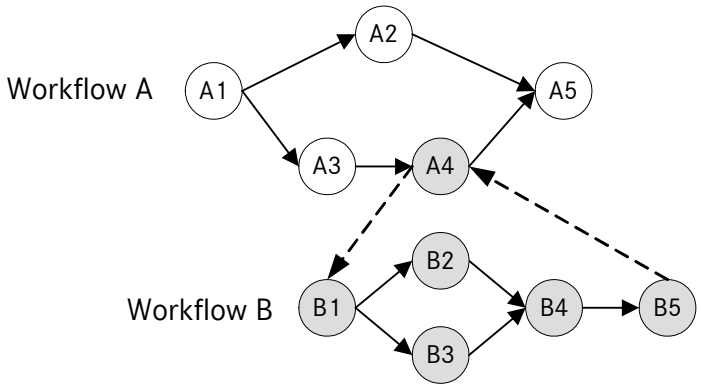

Abbildung 6.1: Hierarchische Workflows bei Aktivitätennetzen nach [LeRo00]

6.1.2 FunSoft-Netze

Konzepte zur hierarchischen Strukturierung findet man auch bei FunSoft-Netzen [DeGr94]. Bei FunSoft-Netzen handelt es sich um einen Petri-Netz-basierten Formalismus zur Workflow-Modellierung. Hauptbeschreibungselemente von FunSoft-Netzen sind *Kanäle* (Stellen bei Petri-Netzen) und *Instanzen* (Transitionen bei Petri-Netzen).

Kanäle können getypte Objektmarken aufnehmen. Instanzen entsprechen Aktivitäten eines Prozesses. Diese können entweder mit einem Anwendungsprogramm verknüpft oder durch ein weiteres Sub-Netz verfeinert werden. Kommt eine durch ein Sub-Netz verfeinerte Instanz zur Ausführung, so wird das Sub-Netz ausgeführt und nach dessen Beendigung mit der Ausführung des aufrufenden Workflows fortgesetzt.

Abbildung 6.2 zeigt ein Beispiel für ein hierarchisch strukturiertes FunSoft-Netz. Sollte in diesem die komplexe Aktivität A2 zur Ausführung kommen, wird hierfür Workflow B ausgeführt. Nach Beendigung dieses Workflows werden die entsprechenden Objektmarken in die Kanäle von Workflow A gelegt, um mit der Ausführung von A fortschreiten zu können.

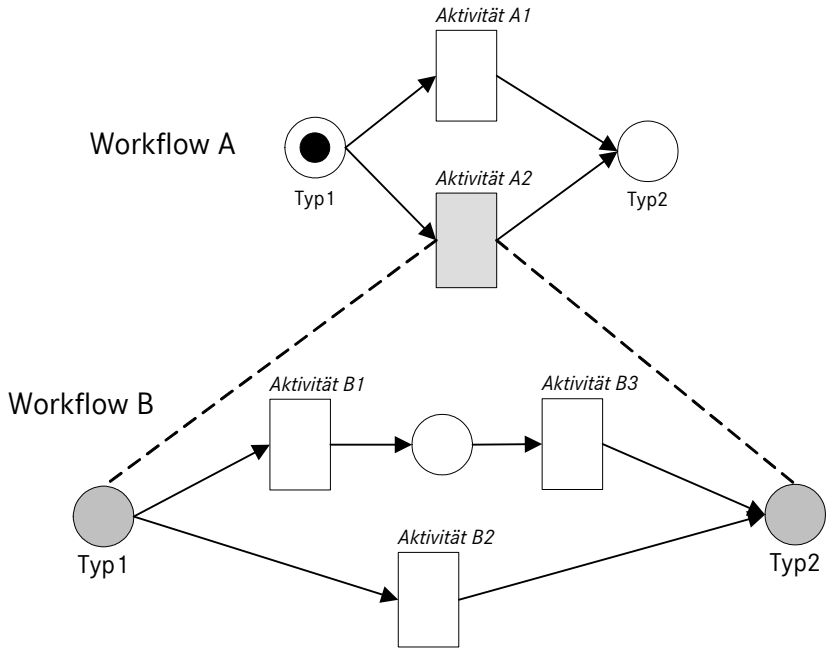

Abbildung 6.2: Hierarchische Workflows bei FunSoft-Netzen

6.1.3 WEP und dynamische Parallelität

Als letztes Beispiel für hierarchische Workflow-Konzepte soll *WEP (Workflow-Management for Engineering Processes)* [BDS98, Be02] näher betrachtet werden. Bei WEP handelt es sich um ein Workflow-Management-Konzept für Entwicklungsprozesse im Maschinenbau. Diese Prozesse sind teilweise unstrukturiert und kreativ – eine klassische Workflow-Unterstützung ist hier zu rigide. WEP bietet deshalb die Möglichkeit, unstrukturierte Teilprozesse innerhalb eines strukturierten Prozesses zu unterstützen. Im Kontext dieser Arbeit spielen jedoch unstrukturierte Teilprozesse keine wesentliche Rolle. Vielmehr soll hier neben hierarchischen Workflows eine Besonderheit betrachtet werden: In WEP können, abhängig von den Daten eines Workflows, dynamisch parallele Ausführungspfade modelliert werden. Da es sich bei Konfigurationen ebenfalls um Daten handelt, die in Bezug zu parallelen Workflows stehen, lohnt sich eine nähere Betrachtung von WEP.

6.1.3.1 Metadatenmodell

Zentral für Entwicklungsprozesse sind die bearbeiteten komplexen Daten. WEP erlaubt die Datenmodellierung mittels eines objektorientierten Metamodells. Als Modellierungskonstrukte stehen im Wesentlichen Klassen mit Einfachvererbung, Relationen und Attribute zur Verfügung. Eine Besonderheit des Metadatenmodells sind sog. *Qualitätsstufen*, die für diese Arbeit jedoch keine Rolle spielen.

6.1.3.2 Zielorientierte Aktivitäten

Ein wesentliches Merkmal von WEP ist die Beschreibung unstrukturierter Teilprozesse innerhalb eines strukturierten Prozesses. Basis hierfür sind *zielorientierte Aktivitäten*, aus denen sich WEP-Workflow-Modelle zusammensetzen.

Wie Aktivitäten bei klassischen Workflows besitzen zielorientierte Aktivitäten Ein- und Ausgabeparameter. Zu einer Aktivität gehören mehrere *Schrittprogramme*. Ein Schrittprogramm kann dabei entweder ein bestimmtes Anwendungsprogramm, eine andere zielorientierte Aktivität oder ein WEP-Workflow sein. Durch die letztgenannte Möglichkeit wird eine hierarchische Workflow-Modellierung möglich. Dem Anwender selbst ist es zur Laufzeit überlassen, wie er mit Hilfe der Schrittprogramme die Ausgabeparameter der Aktivität erzeugt. Die Ausgabeparameter stellen somit das Ziel einer Aktivität dar. Für den Fall, dass eine zielorientierte Aktivität nur ein Schrittprogramm besitzt, entspricht diese einer einfachen Aktivität bei klassischen Workflows.

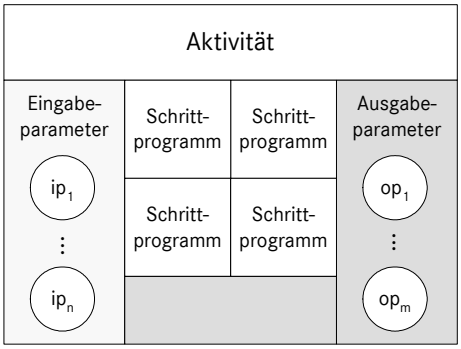

Abbildung 6.3: Zielorientierte Aktivität in WEP

6.1.3.3 Kontrollfluss

Zur Spezifikation der Ausführungsreihenfolge zielorientierter Aktivitäten bietet WEP Konstrukte zur Beschreibung sequenzieller, alternativer und paralleler Ausführungspfade und Schleifen an. Wie auch in anderen Ansätzen (vgl. [Re00]) wird bei der Kontrollflussmodellierung das Prinzip der *regelmäßigen Blockstrukturierung* verwendet. Dies bedeutet, dass ein Workflow aus Blöcken aufgebaut ist, die entweder völlig ineinander geschachtelt oder disjunkt sind. Diese Art der Modellierung bietet einige Vorteile, auf die bei der Beschreibung des ADEPT-Ansatzes in Abschnitt 6.4.2 detaillierter eingegangen wird.

Als Beispiel für Kontrollflusskonstrukte in WEP sollen Sequenz und Parallelität dienen, möglich sind auch Schleifen und bedingte Verzweigungen. Eine Sequenz beschreibt das Nacheinanderausführen von Aktivitäten. Jede Aktivität besitzt genau einen Vorgänger und einen Nachfolger, die durch gerichtete Kontrollflusskanten miteinander verbunden sind (siehe Abbildung 6.4).

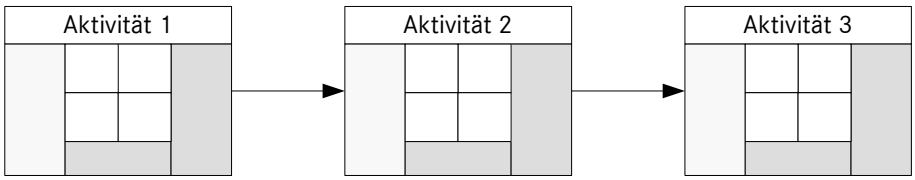

Abbildung 6.4: Modellierung einer Sequenz in WEP

Parallelität wird durch einen speziellen Kontrollflussblock modelliert. Dabei gibt es eine Startaktivität und eine Endaktivität für diesen Kontrollflussblock. Wurde die Startaktivität beendet, werden alle nachfolgenden Kontrollflussstränge parallel „freigeschaltet“. Bevor die Endaktivität ausgeführt werden darf, müssen alle parallelen Kontrollflussstränge beendet sein.

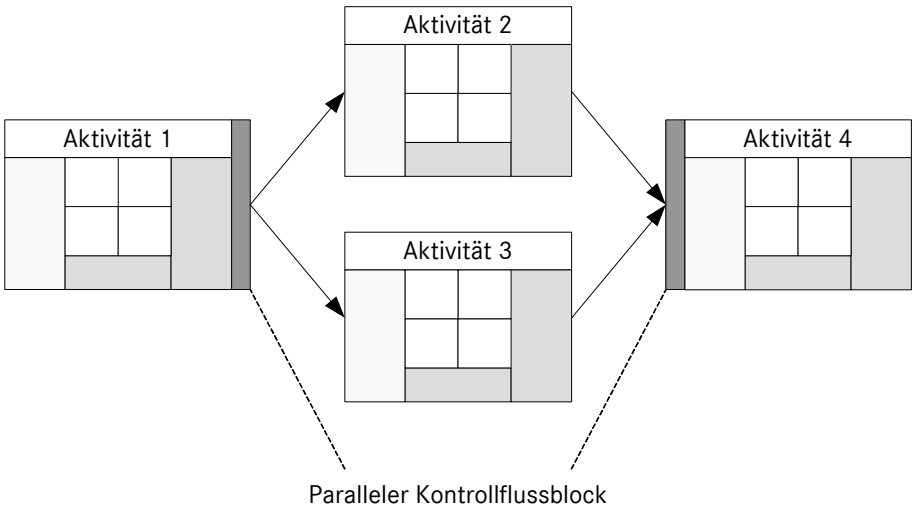

Abbildung 6.5: Modellierung von Parallelität in WEP

6.1.3.4 Datenfluss

Die Objekte, die ein WEP-Workflow bearbeitet, werden in Produktdatenmanagementsystemen verwaltet. Im WfMS werden die für den Workflow wichtigen Produktdaten durch ein *globales Objekt* repräsentiert, das Zugriff auf die Datenobjekte ermöglicht und alle Versionen eines Objektes verwaltet. Ein globales Objekt ist einer Klasse des Datenmodells zugeordnet.

Der Datenfluss wird in WEP dadurch modelliert, dass über spezielle, gerichtete Datenflusskanten die Ein- und Ausgabeparameter der Aktivitäten mit einem globalen Objekt verknüpft werden. Eine Datenflusskante kann entweder einen Ausgabeparameter einer Aktivität mit dem globalen Objekt oder das globale Objekt mit einem Eingabeparameter einer Aktivität verbinden (siehe Abbildung 6.6).

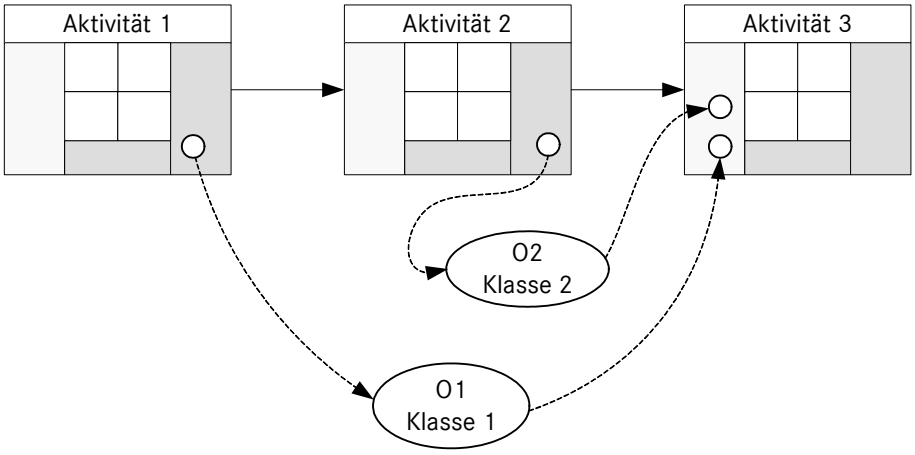

Abbildung 6.6: Datenfluss über globale Objekte in WEP

6.1.3.5 Dynamische Parallelität

Eine Besonderheit von WEP ist Modellierung *dynamischer Parallelität*. Dies bedeutet, dass abhängig von den Daten, die mit einem Workflow verknüpft sind, zur Laufzeit *dynamisch* parallele Pfade erzeugt werden. Zu diesem Zweck existiert das Modellierungskonstrukt der *Traversierung*. Durch die Traversierung wird u.a. beschrieben, wie ein komplex strukturiertes Eingabeobjekt in Subobjekte zerlegt wird. Für jedes dieser Subobjekte wird zur Laufzeit ein paralleler Pfad der von der Traversierung umschlossenen Workflowblöcke erzeugt, ausgeführt und nach der Traversierung wieder zusammengeführt.

Abbildung 6.7 zeigt die Modellierungssicht. Durch die Traversierung sind die Aktivitäten 2 und 3 eingeschlossen. Abbildung 6.8 zeigt eine Ausführung dieses Workflows. Es wurden durch die Traversierung drei Subobjekte des Objekts 1 gefunden und für diese Subobjekte drei parallele Ausführungspfade mit den Aktivitäten 2 und 3 erzeugt. Nach Ausführung der Aktivitäten für alle Subobjekte werden diese wieder zu einem komplexen Objekt zusammengesetzt.

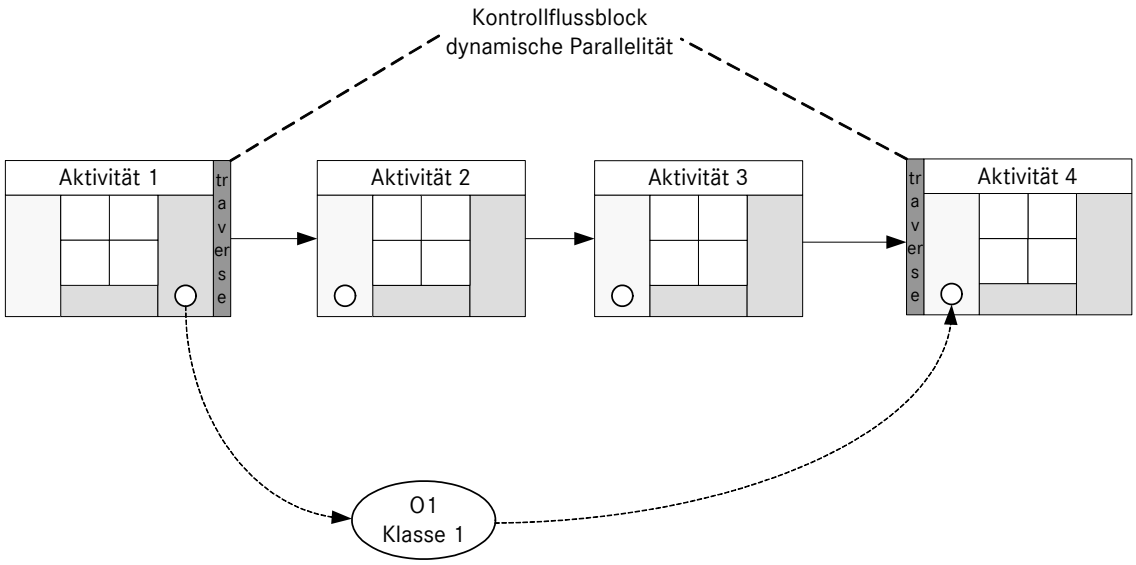

Abbildung 6.7: Modellierungssicht dynamischer Parallelität (vereinfacht)

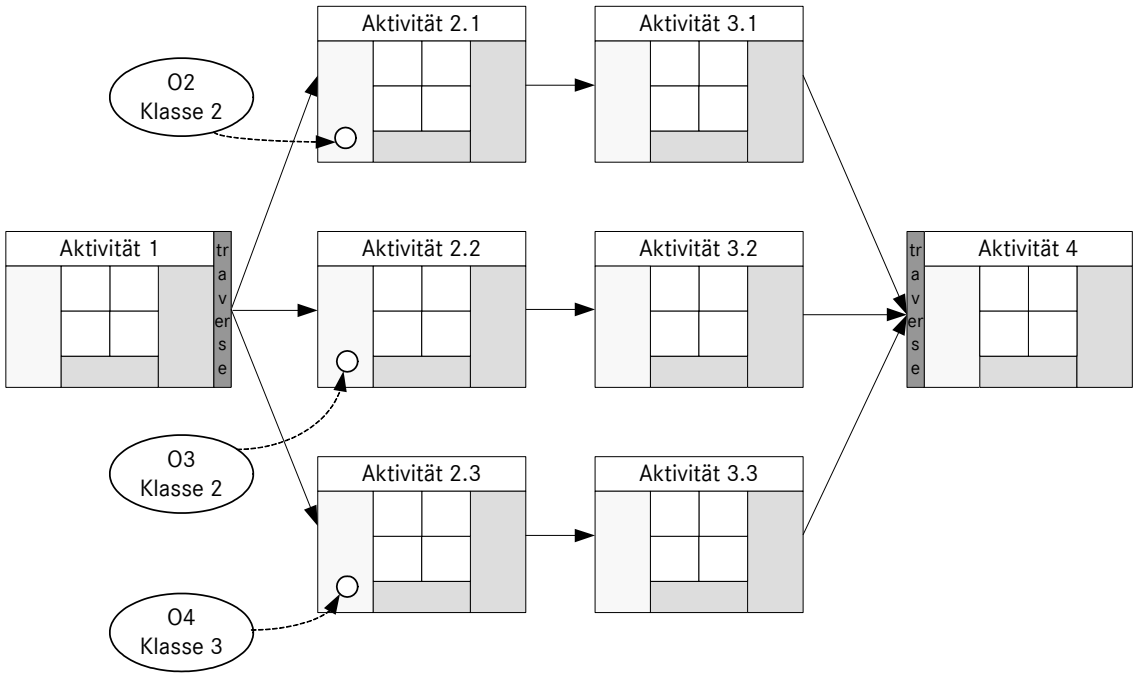

Abbildung 6.8: Ausführung dynamischer Parallelität (vereinfacht)

Die Aufsplittung in Pfade wird durch sog. *Traversierungsmerkmale* definiert. Ein Traversierungsmerkmal gibt durch Regeln an, wie ein komplexes Objekt in Subobjekte zerlegt werden soll. Die Regeln bestehen dabei im Wesentlichen aus Pfaden, die ausgehend vom Ursprungsobjekt den im Datenmodell festgelegten Relationen folgen und dann das Objekt am Ende dieses Pfades auswählen. Zusätzlich ist auch eine Verfeinerung der Objektauswahl durch Attributeinschränkungen möglich. Für jeden der *Traversierungstreffer* wird zur Laufzeit dynamisch ein neuer Ausführungspfad erzeugt.

6.1.4 Anwendbarkeit hierarchischer Workflows für Freigabeprozesse

Wie in Abschnitt 5.2 beschrieben, sollen für alle Konfigurationen echte, eigenständige Freigabeworkflows (mit gewissen Restriktionen) ausgeführt werden. Kann dies durch klassische hierarchische Workflows erreicht werden?

Den in den letzten Abschnitten vorgestellten Ansätzen für hierarchische Workflows ist gemein, dass es *einen* ursprünglichen Workflow gibt, bei dessen Ausführung andere Workflows aufgerufen werden können. Dies ist bei Freigabeworkflows jedoch unpassend. Versucht man aus Sicht einer Gesamtkonfiguration einen Workflow zu definieren, führt dies dazu, dass man Teilkonfigurationen erst dann in den Freigabeworkflow bringen kann, wenn die Gesamtkonfiguration komplett ist. Dies ist jedoch nicht gewünscht: Man möchte Teilkonfigurationen auch unabhängig von deren Verwendung in einer Oberkonfiguration freigeben können. Meist entstehen die Gesamtkonfigurationen erst im Laufe der Zeit. Ein anderes Problem würde dadurch entstehen, dass eine Teilkonfiguration auch in mehreren Gesamtkonfigurationen enthalten sein kann: Würde man aus Sicht der Gesamtkonfiguration Freigabeworkflows modellieren, so würde diese Teilkonfiguration fälschlicherweise zweimal einen Freigabeprozess durchlaufen.

Dadurch ist klar, dass alleine die Einschränkung auf *einen* Ursprungsworkflow in unserem Fall zu rigide ist. Dieses grundsätzliche Problem kann auch durch die *dynamische Parallelität* von WEP nicht beseitigt werden. Klassische hierarchische Workflows sind demnach *nicht*

geeignet, um Freigabeprozesse adäquat unterstützen zu können. Zur Thematik, ob man echt parallele Workflows wirklich benötigt, findet sich in [He00] eine ausführliche Diskussion.

6.2 Synchronisation paralleler Workflows

Wie in Abschnitt 6.1 deutlich gemacht und im Anforderungskapitel beschrieben, können Freigabeprozesse nur durch *parallele* Workflows für jede Konfiguration unterstützt werden. Zwischen diesen existieren jedoch Kontrollflussabhängigkeiten (siehe Abschnitt 5.4.1). In diesem Abschnitt werden typische Ansätze aus der Literatur diskutiert, die sich mit der Synchronisation paralleler Workflows beschäftigen, und deren Eignung für die Problemstellung dieser Arbeit bewertet. Der Schwerpunkt liegt auf der Tauglichkeit der Modellierungskonzepte.

6.2.1 Workflowsynchronisation bei organisationsübergreifenden Workflows

Kontroll- und Datenflussabhängigkeiten zwischen parallelen Workflows treten auch im Kontext organisationsübergreifender Workflows auf (sog. Workflow-Choreographie). Im Folgenden werden zwei Ansätze aus diesem Forschungsgebiet im Detail betrachtet: *COW* [JKK+02] und *CrossFlow* [GAHL00].

Bei organisationsübergreifenden Workflows stehen neben Abhängigkeiten zwischen Workflows auch andere Fragestellungen im Fokus, wie beispielsweise die Überwindung der Heterogenität verschiedener WfMS, der Erhalt der Autonomie der beteiligten Organisationen oder die Geheimhaltung wettbewerbskritischer Workflowdetails. Da diese Aspekte im Kontext dieser Arbeit keine Rolle spielen, werden hier nur die Aspekte von COW und CrossFlow diskutiert, die sich mit Kontrollflussabhängigkeiten zwischen parallelen Workflows beschäftigen.

6.2.1.1 COW (Cross-Organizational Workflows)

COW [JKK+02, Sc03, KRS01] zielt auf die Unterstützung unternehmensübergreifender Workflows. COW sieht eine indirekte Spezifikation der Abhängigkeiten zwischen Workflows vor. Ein Partner in einer Workflowkooperation nimmt im Sinne von COW eine bestimmte *Kopplungsrolle* ein. In einem *Kopplungsschema (KS)* werden globale Daten- und Kontrollflussabhängigkeiten zwischen unterschiedlichen Kopplungsrollen beschrieben, jedoch ohne hier schon eine Anknüpfung an Workflow-Schemata vorzunehmen. Vielmehr werden im Kopplungsschema globale Konnektoren und deren Verhalten zueinander (sequenziell, parallel, Schleife ...) festgelegt. Jeder Partner spezifiziert mit einem *externen Workflow-Schema (EWS)* für eine Kopplungsrolle eine Sicht auf sein internes Workflow-Schema und verknüpft dann bestimmte Endpunkte des Kopplungsschemas mit Aktivitäten im externen Workflow-Schema.

Ein Beispiel zeigt Abbildung 6.9. Das Kopplungsschema (Mitte) legt fest, dass es zwei Kontrollflussabhängigkeiten c1 und c2 zwischen den Kopplungsrollen A und B gibt. Für das externe Workflow-Schema jeder Kopplungsrolle werden die Endpunkte der Kontrollflusskonnektoren mit konkreten Aktivitäten verknüpft (dargestellt durch die gestrichelten Linien). So ist im Beispiel festgelegt, dass die Aktivität A3 erst nach B1 ausgeführt werden darf und die Aktivität A4 vor Start von B3 beendet sein muss. In COW lassen sich nicht nur sequenzielle Kontrollflussabhängigkeiten, sondern auch Parallelität und Schleifen beschreiben. Auch Datenflüsse zwischen parallelen Workflows sind modellierbar.

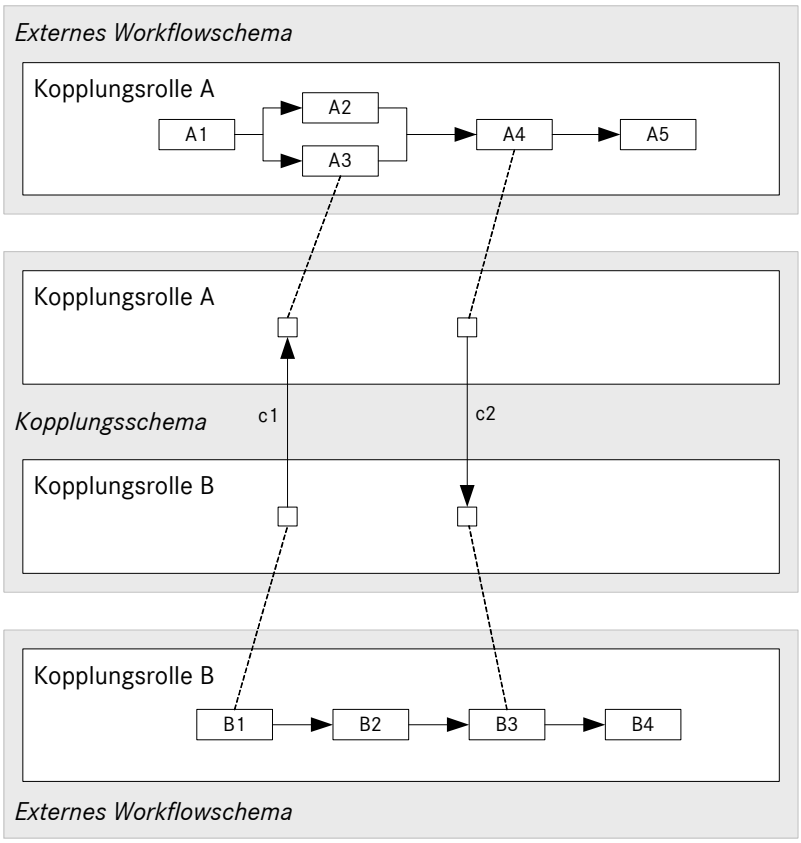

Abbildung 6.9: Kopplungsschema in COW

COW sieht ein dezentrales Ausführungsmodell vor, das bei jeder an der Kooperation teilnehmenden Organisation eine *Workflow Integration Component (WIC)* vorsieht, die mit den WICs der anderen Organisationen dafür sorgt, dass die im Kopplungsschema spezifizierten Abhängigkeiten umgesetzt werden.

Bei parallelen Workflows verschiedener Organisationen stellt sich die Frage, *welche* Workflow-Instanzen auf beiden Seiten miteinander kooperieren und wann diese Instanzen erzeugt werden. Dies wird in COW durch sog. *Bindeabhängigkeiten* im Kopplungsschema spezifiziert. Aus der vorliegenden Literatur zu COW ergibt sich leider nicht, wie diese Bindeabhängigkeiten spezifiziert werden können. Im Gespräch mit einem Projektbeteiligten wurde klar, dass bei COW an einen *Broker* gedacht wird, der andere Workflow-Instanzen kennt und der Suche von Workflow-Instanzen bei Partnern dient. Details zur Abhängigkeitsspezifikation selbst liegen für das Projekt leider (noch) nicht vor. Damit bleibt die Frage offen, ob Abhängigkeiten basierend auf Daten von Workflows in COW beschreibbar sind. Dies wäre Voraussetzung, um die Synchronisationskonzepte für Freigabeworkflows anwenden zu können. Somit lässt sich noch keine Aussage treffen, ob die Konzepte von COW für die Anforderungen bezüglich Workflow-Synchronisation in unserem Fall ausreichend sind.

6.2.1.2 CrossFlow

Der grundlegende Ansatz bei CrossFlow [GAHL00] wird als *dynamisches Service-Outsourcing* bezeichnet. Dies bedeutet, dass ein Unternehmen Teile seiner Prozesse durch andere Unternehmen ausführen lassen kann. Dynamisch meint, dass erst zur Ausführungszeit des Workflows entschieden wird, ob und von welchem Partner bestimmte Dienste dazu in Anspruch genommen werden. Dies geschieht über einen Service-Marktplatz, auf dem *Service-*

Provider ihre Dienste (respektive Workflows) anbieten und *Service-Consumer* nach Diensten suchen können. Im Beispiel in Abbildung 6.10 werden die Workflow-Schritte D und E dynamisch an einen Provider ausgelagert.

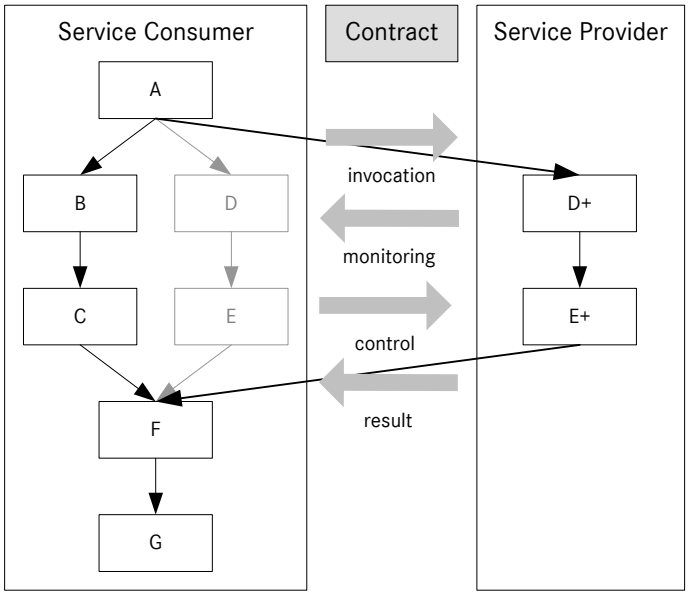

Abbildung 6.10: Outsourcing von Workflow-Schritten in CrossFlow

Das Besondere an CrossFlow ist, dass die Einbindung eines Dienstes nicht als Black-Box geschieht, sondern der Konsument den Workflow auf Providerseite steuern und überwachen kann. Die Ausführung des Services auf Providerseite ist zu großen Teilen für den Konsumenten sichtbar. Die Interaktion zwischen Provider und Consumer basiert auf sog. *Contracts*. Dort ist u.a. in einem Prozessmodell beschrieben, welche Schritte von einem Service-Provider aus Sicht des Service-Consumers durchgeführt werden müssen. Dieses Prozessmodell kann der Provider auf sein internes Prozessmodell abbilden und so dem Consumer entsprechende Nachrichten über den Fortgang des Prozesses schicken.

Als Teile der sog. *Cooperation Support Services* bekommt der Service-Consumer eine Reihe von Operationen an die Hand, um den Workflow auf Providerseite zu steuern, beispielsweise kann er mit einer *Stop*-Operation den Workflow anhalten oder mittels *Abort* abbrechen.

Der CrossFlow zu Grunde liegende Ansatz des Service-Outsourcing entspricht im Wesentlichen dem Aufruf eines *Subworkflows* und damit von der Grundidee klassischen hierarchischen Workflows (siehe Abschnitt 6.1). Deshalb ist dieser Ansatz für die in dieser Arbeit betrachtete Problemstellung nicht ausreichend.

6.2.2 Event-basierte Ansätze

Bei Event-basierten Ansätzen zur Synchronisation werden zwischen parallelen Workflows Ereignisse verschickt bzw. auf Ereignisse gewartet, um die Ausführung zu synchronisieren. Als repräsentative Beispiele für diese Ansätze werden die entsprechenden Konzepte von OPERA [HaAl99] bzw. WIDE [Ca99, Ca98] diskutiert. Weitere Ansätze derselben Kategorie sind auch in [LeRo00, S. 185ff.] sowie [BaHo99] zu finden.

6.2.2.1 Workflow-Synchronisation in OPERA

In OPERA [HaAl99] können *Eventtypen* deklariert werden. Ein Eventtyp besteht aus einem Namen und einer Parameterliste. Während der Ausführung werden Eventinstanzen der Eventtypen von Aktivitäten oder Workflows erzeugt. In den Parametern eines Events kann relevante Kontextinformation übertragen werden. Eventtypen können sowohl für Aktivitäten als auch für ganze Workflows deklariert werden. Wenn ein Eventtyp für eine Aktivität oder einen Workflow deklariert wurde, bedeutet dies, dass dieses Event während der Ausführung der Aktivität beliebig oft (oder auch gar nicht) auftreten kann (siehe Beispiel Abbildung 6.11).

Events von Aktivitäten werden von deren Implementierung selbst erzeugt, d.h. das Erzeugen dieser Events ist für das WfMS transparent. Das Erzeugen von Events eines Workflow wird in OPERA durch *Pseudoaktivitäten* realisiert. Events können sowohl zur Synchronisation innerhalb eines Workflows als auch zur Synchronisation zwischen Workflows verwendet werden.

Die eventbasierte Synchronisation zwischen Aktivitäten innerhalb eines Workflows erfolgt über sog. *Event-based Control Connectors (ECCs).* Im Wesentlichen bedeutet ein ECC zwischen zwei Aktivitäten A und B, dass die Aktivität B gestartet werden kann, sobald Aktivität A ein Event eines bestimmten Typs signalisiert. ECCs können in einem Workflow-Modell zusammen mit normalen Kontrollflusskonnektoren verwendet werden.

Das Beispiel in Abbildung 6.11 zeigt einen Prozess, bei dem die Aktivität A Events der Typen EVENT_1 und EVENT_2 erzeugt. Die Aktivitäten B und C werden gestartet, sobald die an den ECCs deklarierten Events durch die Aktivität A erzeugt werden. Die Aktivitäten D und E werden im normalen Kontrollfluss ausgeführt, d.h. mit D wird erst nach Beendigung von A begonnen.

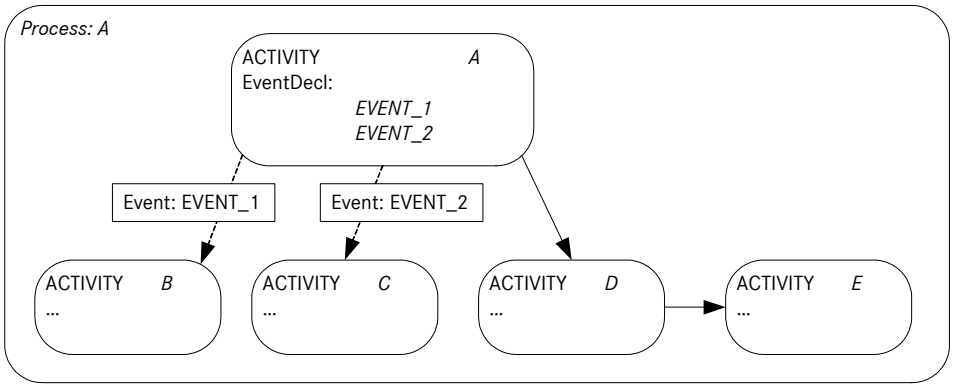

Abbildung 6.11: Intraworkflow-Kommunikation über Events

Die Sichtbarkeit von Events ist, ähnlich wie lokale Variablen einer Programmiersprache, auf einen Workflow beschränkt. Die Synchronisation *zwischen* Workflows wird über einen *Subscription-Mechanismus* realisiert. Ein Workflow kann einen bestimmten Eventtyp eines anderen Workflows bzw. einer Aktivität dieses Workflows importieren. Damit ist dieser Eventtyp im Workflow sichtbar, er kann zur Vermeidung von Namenskonflikten umbenannt werden. Jeder Workflow besitzt eine sog. *Imported Event Queue (IEQ),* die alle importierten Ereignisse den Aktivitäten des Workflows zur Verfügung stellt. Die IEQ kann durch ECCs mit einzelnen Aktivitäten verbunden werden. Die Bedeutung ist dieselbe wie innerhalb eines Workflows: Gelangt ein Ereignis in die IEQ, so können auf dieses Ereignis wartende Aktivitäten gestartet

werden. In der Exportdefinition gibt ein Workflow an, welche Events anderen Workflows signalisiert werden können.

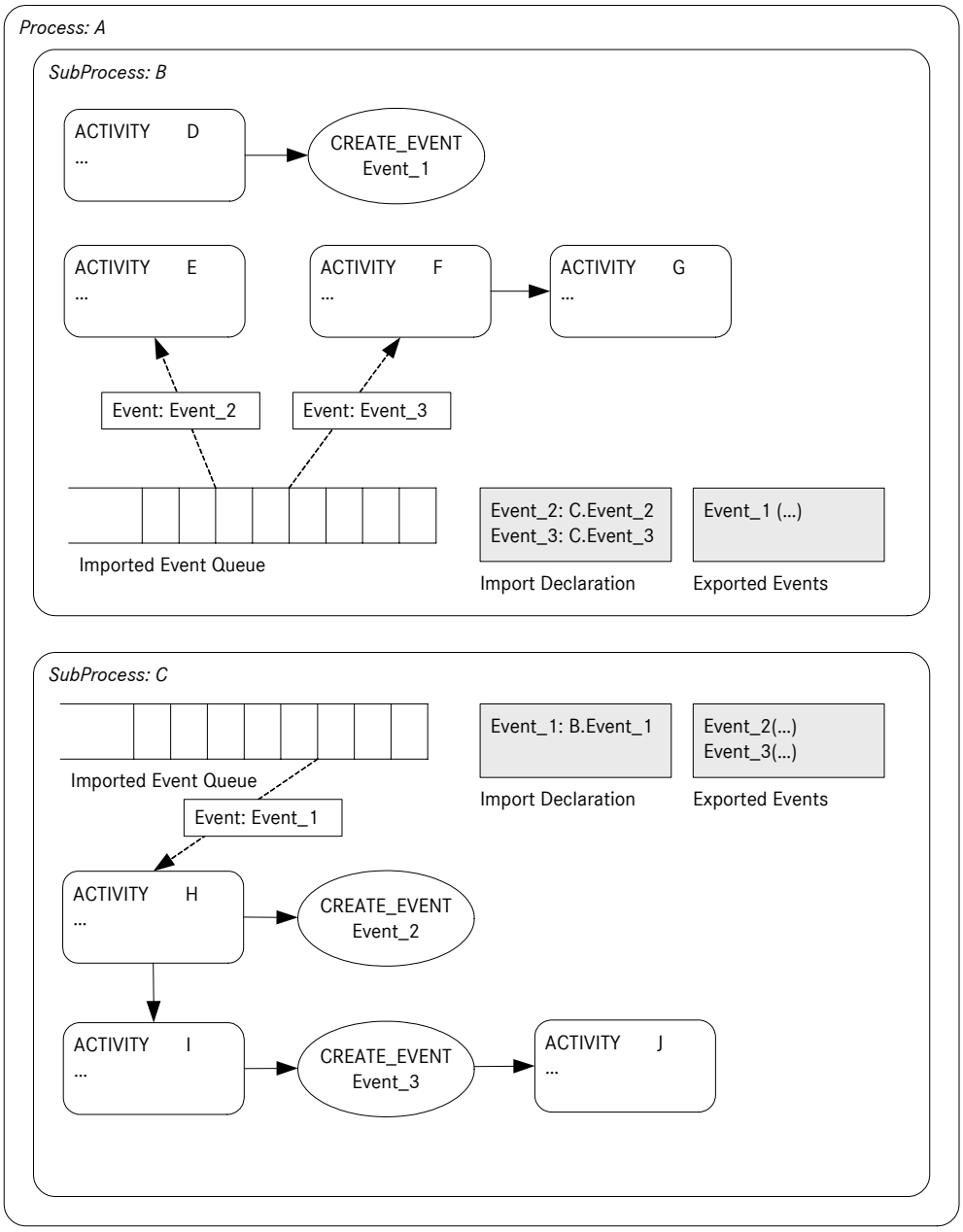

Abbildung 6.12: Interworkflow-Kommunikation über Events

Abbildung 6.12 zeigt ein Beispiel für Interworkflow-Kommunikation in OPERA. Zur Synchronisation werden von Workflows generierte Events verwendet. So erzeugt Subprozess B Events des Typs EVENT_1 und Subprozess C Events der Typen EVENT_2 und EVENT_3. Die Verwendung der Events 2 und 3 in Subprozess B bzw. von Event 1 in Subprozess C wird durch *ImportDeclarations* spezifiziert. *ExportedEvents* geben für einen Prozess die Ereignisse an, die in anderen Prozessen verwendet werden können. Die Subprozesse B und C interagieren folgendermaßen miteinander: Nach der Ausführung von D wird Event 1 erzeugt, damit kann die Aktivität H in Subprozess C gestartet werden. Nach Beendigung von H wird Event 2 generiert. Dadurch kann in Subprozess B die Aktivität E ausgeführt werden. Nach der Aus-

führung von I in Subprozess C wird Event 3 erzeugt. Dadurch kann in Subprozess B die Aktivität F ausgeführt werden.

Soweit der Ansatz in [HaAl99] beschrieben ist, können Events immer nur zwischen Subworkflows eines übergreifenden Workflows ausgetauscht werden, nicht jedoch zwischen beliebigen Workflow-Instanzen, die auf Schemaebene nicht als Subworkflows in ein gemeinsames Workflow-Schema eingebettet sind. Dies ist für Freigabeworkflows jedoch nicht ausreichend: die Beschränkung auf einen einzigen, übergreifenden Workflow lässt beispielsweise das unabhängige Starten von parallelen Workflows nicht zu und führt zu ähnlichen Restriktionen wie die Umsetzung mit hierarchischen Workflows (siehe Abschnitt 6.1.4). Außerdem müsste die Zusammensetzung der Konfigurationen auf Instanzebene schon zur Modellierungszeit der Freigabeworkflows bekannt sein, da für jede Teilkonfiguration ein eigener Subworkflow im Workflow-Modell vormodelliert werden müsste. In der in [HaAl99] präsentierten Form sind die Konzepte von OPERA daher für unsere Anforderungen nicht ausreichend.

6.2.2.2 Workflow-Synchronisation in WIDE

In WIDE [Ca98] ist ebenfalls ein Konzept zur eventbasierten Workflow-Synchronisation enthalten [Ca99]. Events besitzen in WIDE ebenfalls unterschiedliche Eventtypen. Spezielle Eventknoten, um die das Workflow-Schema ergänzt wird, markieren Punkte im Kontrollfluss, an denen Events bestimmter Typen empfangen bzw. verschickt werden. Dabei wird zwischen Sende- und Empfangsknoten unterschieden. Abbildung 6.13 zeigt die graphische Repräsentation dieser Knoten.

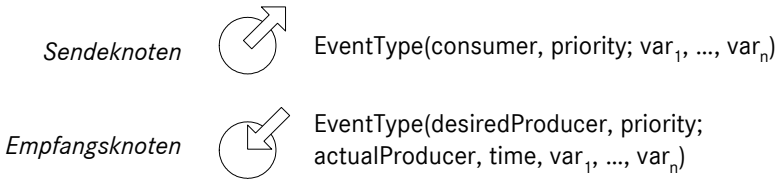

Abbildung 6.13: Sende- und Empfangsknoten bei WIDE

Ein *Sendeknoten* verschickt Events an andere Workflows. Neben dem Namen des Eventtyps können hier beliebig viele Parameter angegeben werden. Die ersten beiden Parameter *consumer* und *priority* steuern das Zustellen der Events. Über den Parameter *consumer* kann festgelegt werden, an welche Workflow-Instanz das Event geschickt werden soll. Die Angabe von *any* an dieser Stelle bedeutet, dass das Event an eine beliebige Workflow-Instanz ausgeliefert wird, die auf Events dieses Typs wartet. *Priority* gibt eine Priorität (als Integer) an, wobei höher priorisierte Events bevorzugt ausgeliefert werden, wenn bei einer Eventanfrage eines Workflows mehrere passende Events zur Auswahl stehen. In den restlichen Parametern var_1 – var_n können die Inhalte beliebiger Workflowvariablen an andere Workflow-Instanzen übermittelt werden.

Empfangsknoten markieren Stellen im Workflow, an denen auf Events gewartet wird. Ein Empfangsknoten blockiert so lange, bis ein Event des spezifizierten Typs durch einen Sendeknoten eines anderen Workflows erzeugt wird. Auch Empfangsknoten können mit Parametern versehen werden. Über den ersten Parameter *desiredProducer kann* eine bestimmte Workflow-Instanz angegeben werden, von der das Event empfangen werden soll. Die Konstante *any* bedeutet an dieser Stelle, dass es keine Rolle spielt, von welcher Instanz dieses

Event erzeugt worden ist. Der Parameter *priority* gibt die Dringlichkeit der Anforderung (kodiert als Integer) an. Eventnachfragen mit höherer Dringlichkeit werden zuerst behandelt. Die Parameter nach dem Semikolon sind Workflow-Variablen, in die beim Empfang eines Events die entsprechenden Eventdaten kopiert werden. Die erste Variable *actualProducer* enthält beim Empfang des Events die interne ID der Workflow-Instanz, die dieses Event verschickt hat. In der Variable *time* wird der Zeitpunkt der Eventerzeugung festgehalten. In den folgenden Variablen var_1 – var_n werden die Workflowdaten abgelegt, die bei Erzeugung des Events als Parameter übergeben wurden.

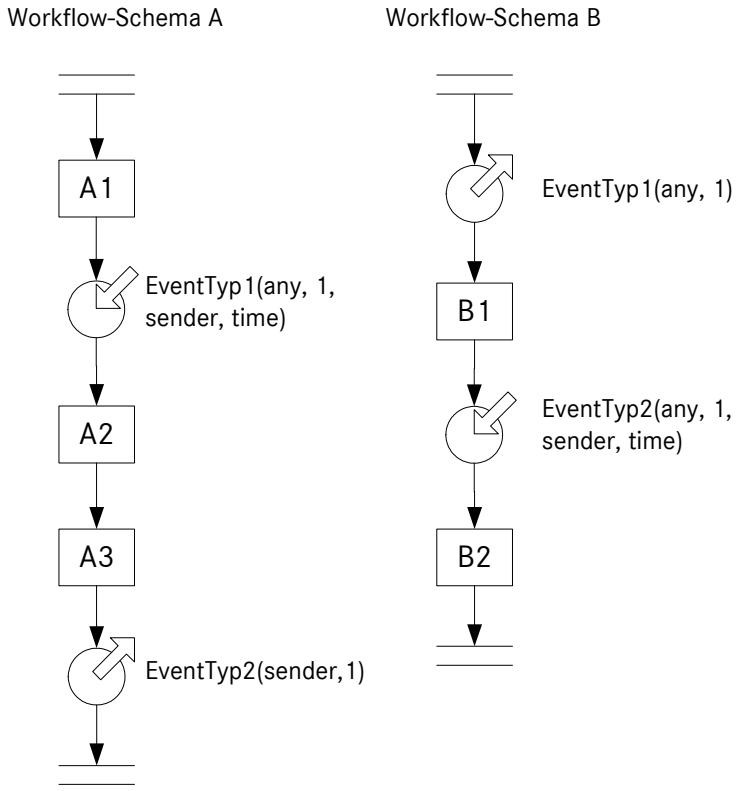

Abbildung 6.14: Beispiel für Workflowsynchronisation in WIDE

Zur Illustration dieses Konzepts sollen die in Abbildung 6.14 dargestellten Workflow-Schemata dienen, deren Instanzen über Events miteinander interagieren. Nach Start einer Workflow-Instanz des Schemas A wird Aktivität A1 ausgeführt und anschließend auf das Eintreten eines Events des Typs *EventTyp1* gewartet. *Any* bedeutet, dass die Eventinstanz von jeder beliebigen Workflow-Instanz kommen kann. Nach Empfang des Events wird in der Variable *sender* die ID der absendenden Workflow-Instanz abgelegt. Nach Ausführung von A2 und A3 wird an *genau diese* Instanz - festgelegt durch den Parameter *sender* an erster Stelle – ein Event des Typs *EventTyp2* abgeschickt. Nach Start einer Workflow-Instanz des Schemas B wird ein Event des *EventTyp1* an beliebige Workflow-Instanzen verschickt. So könnte dies bei einer Instanz von Schema A beim ersten Empfängerknoten eintreffen. Nach Ausführung von B1 wird auf ein Event des Typs *EventTyp2* gewartet. An dieser Stelle kann man *nicht* festlegen, dass dieses Event von derjenigen Workflow-Instanz kommen soll, die das vorige, selbst erzeugte Event erhalten hat. Die ID einer Workflow-Instanz wird erst beim erstmaligen Erhalt eines Events von dieser Instanz übermittelt und kann erst dann verwendet werden, um Events gezielt von bestimmten Instanzen zu empfangen bzw. zu verschicken. Ein Event des

Typs *EventTyp2* wird beispielsweise im letzten Schritt von Workflow A erzeugt und gezielt an den Sender des ersten Events (z.B. eine Instanz des Schemas B) übermittelt.

Der Ansatz zur Prozesssynchronisation in WIDE ist auf semantisch niedrigem Level angesiedelt. Dadurch ist zum einen ein hoher Modellierungsaufwand nötig, zum anderen ist die Wahrscheinlichkeit von Fehlern sehr viel höher. So können beispielsweise Deadlocks entstehen, wenn vergeblich auf ein bestimmtes Event gewartet wird und dieses aufgrund der falschen Prioritätsvergabe immer von anderen Workflow-Instanzen konsumiert wird. Um Events an bestimmte Workflow-Instanzen zu senden, muss mindestens ein Event dieser Instanz vorher erhalten worden sein (siehe auch Diskussion zum obigen Beispiel). Dadurch muss mindestens eine der beteiligten Instanzen mit der Konstanten *any* Events an beliebige Instanzen schicken. Dies ist jedoch in unserem Fall überhaupt nicht das gewünschte Verhalten. Es sollen nicht beliebige Workflow-Instanzen miteinander synchronisiert werden, sondern Freigabeworkflows von Teilkonfiguration mit den *entsprechenden* Freigabeworkflows der Oberkonfigurationen.

Ein weiteres Problem, das die Anwendbarkeit dieses Konzepts für unseren Fall unmöglich macht, entsteht durch die Semantik der Event-Instanzen: Jeder Sendeknoten erzeugt genau eine Eventinstanz. Nachdem diese an genau einen Konsumentenworkflow ausgeliefert wurde, existiert sie nicht mehr. Eine Teilkonfigurationen kann zu mehreren Oberkonfigurationen gehören, deren Anzahl sich im Laufe der Zeit dynamisch vergrößern kann. Es existieren also Kontrollflussabhängigkeiten zwischen dem Freigabeworkflow einer Teilkonfiguration und beliebig vielen anderen Workflow-Instanzen der Oberkonfigurationen. Bei der Umsetzung mit dem Eventkonzept von WIDE hieße dies, dass man zur Synchronisierung in der Workflow-Instanz für die Teilkonfiguration mit einer Schleife so viele Eventinstanzen wie Oberkonfigurationen erzeugen müsste. Da deren Anzahl zur Ausführungszeit des Freigabeworkflows der Teilkonfiguration jedoch nicht bekannt ist, ist dies nicht umsetzbar.

6.2.3 Transaktional geprägte Ansätze

6.2.3.1 Atomare Folgen von Aktivitäten

In [AAE96] wird ein Ansatz vorgestellt, der es erlaubt, mögliche Überlappungen paralleler Workflows zu beschreiben. Zudem wird ein Protokoll beschrieben, dass die Einhaltung dieser Randbedingungen gewährleistet.

Workflows werden „objektorientiert“ betrachtet: Ein Objekt ist ein logisches Konstrukt, das eine Ressource oder einen Dienst *außerhalb* des WfMS repräsentiert. Eine Workflow-Instanz stellt eine (total) geordnete Folge von Methodenaufrufen an diesen Objekten dar. Da die Semantik der Methodenaufrufe dem WfMS nicht bekannt ist, wird davon ausgegangen, dass jeder Methodenaufruf ein Update an diesem Objekt vornimmt. Somit wird angenommen, dass jedes Objekt *eine* Operation *op* besitzt, die den Objektzustand ändert. Workflow-Schemata werden hier als *Prozessklassen* bezeichnet. Abbildung 6.15 zeigt vier Objekte W, X, Y und Z und deren logische Operation *op*. Für jeweils zwei Instanzen der Prozessklassen A und B wird deren Betrachtung als total geordnete Folge von Objektmethodenaufrufen an diesen Objekten dargestellt.

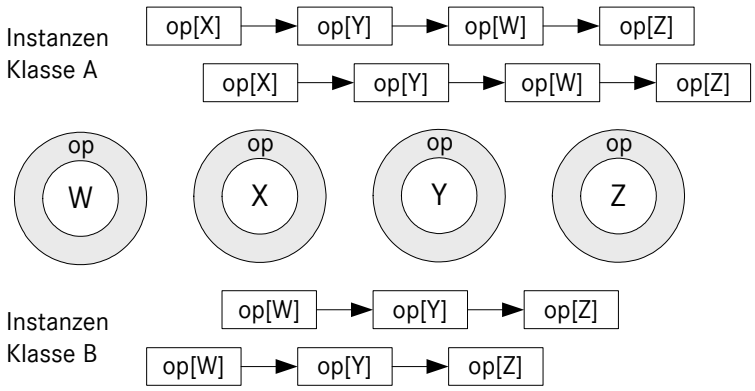

Abbildung 6.15: Workflow-Instanzen als Folge von Methodenaufrufen an Objekten

Die Autoren gehen davon aus, dass sich Workflows nicht beliebig überlappen dürfen, sondern dass es bestimmte Folgen von Aktivitäten innerhalb einer Workflow-Instanz gibt, die für andere Workflow-Instanzen nur als atomare Einheit „sichtbar“ sein sollen.

Abbildung 6.16 zeigt das Prinzip der atomaren Aktivitätseinheiten an einem Beispiel. Aus Sicht der Instanzen der Klasse B sollen die ersten beiden Aktivitäten der Instanzen von A als atomare Einheit erscheinen – umgekehrt sollen die letzten beiden Aktivitäten von Instanzen der Klasse B für Instanzen der Klasse A als atomare Einheit wirken. Im ersten abgebildeten Zustand wurde für zwei Instanzen der beiden Klassen jeweils die erste Aktivität ausgeführt. Die Instanz der Klasse B muss nun mit Ausführung der nächsten Aktivität warten, bis die zweite Aktivität der Instanz von A ausgeführt wurde. Die Gründe hierfür sind das gemeinsam zugegriffene Objekt Y und die atomare Wirkung der ersten beiden Schritte aus Sicht von B. Ein späterer Zustand ist darunter abgebildet. Hier muss die Instanz von A vor Ausführung der letzten Aktivität warten, da das gemeinsame Objekt Z (aus Sicht von A) schon in einer atomaren Aktivitätseinheit einer Instanz der Klasse B zugegriffen wird.

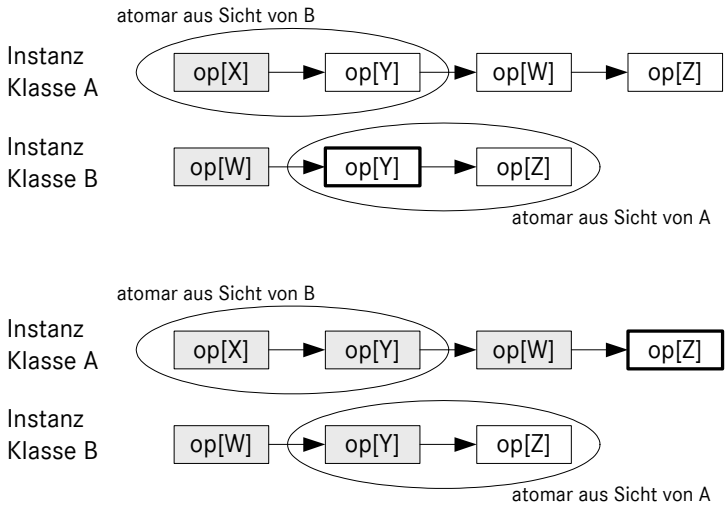

Abbildung 6.16: Beispiel für atomare Aktivitätseinheiten

Spezifiziert werden für jede Prozessklasse sog. *Protected Operations* im Hinblick auf eine andere Prozessklasse. So kann für die Prozessklasse A festgelegt werden, dass für alle Prozessinstanzen von B die Operationen (op[X], op[Y]) als atomare Einheit wirken sollen, d.h. eine Instanz von B darf - nachdem op[X] von einer Instanz der Prozessklasse A ausgeführt wurde – erst wieder auf die Objekte X und Y zugreifen, wenn op[Y] von der Instanz von A ausge-

führt wurde. Umgekehrt kann natürlich auch aus Sicht von B eine derartige Einschränkung für Instanzen der Prozessklasse A getroffen werden. Auch Einschränkungen zwischen Instanzen derselben Prozessklasse können spezifiziert werden. Für die Aufrechterhaltung dieser Randbedingungen werden ein Protokoll und Algorithmen vorgeschlagen, die Semaphoren in Betriebssystemen ähneln. Dabei werden im Wesentlichen prozessklassenbezogene Sperren an diesen logischen Objekten angebracht.

Mit dem hier vorgestellten Ansatz kann lediglich ein Art gegenseitiger Ausschluss von Workflows modelliert (und ausgeführt) werden, nicht jedoch sequenzielle Reihenfolgeabhängigkeiten wie wir sie für unser Anwendungsgebiet benötigen. Zusätzliche Limitierungen ergeben sich dadurch, dass dieser gegenseitige Ausschluss nur für Prozessklassen definiert werden kann und immer für *alle* Instanzen einer Prozessklasse gilt.

6.2.3.2 Parallele Workflows in LAWS

Die Workflowsprache LAWS (Language for Workflow Specification), die im Projekt CREW (Correct & Reliable Execution of Workflows) für die Spezifikation von Workflows entwickelt wurde, enthält Elemente, mit denen Abhängigkeiten zwischen nebenläufigen Workflows ausgedrückt werden können [KaRa98, Ka98]. Dem dort verfolgten Ansatz zur Synchronisation paralleler Workflows liegt die Denkweise zu Grunde, dass Workflow-Instanzen um dieselben Ressourcen konkurrieren und ein gleichzeitiger Zugriff auf eine Ressource von verschiedenen Workflow-Instanzen zu Konflikten führt. Dies ist z.B. dann der Fall, wenn gleichzeitig auf die*selben* Daten in einer Datenbank zugegriffen werden soll. Ressourcen im Sinne dieses Konzepts können jedoch auch Objekte der realen Welt sein, wenn beispielsweise in zwei parallelen Instanzen eines Bestellprozesses gleichzeitig *dasselbe* Produkt verschickt werden soll. Als Beispiel sollen die in Abbildung 6.17 abgebildeten Workflowsschemata WF1 und WF2 dienen.

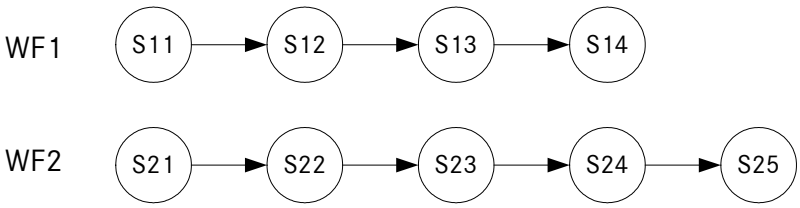

Abbildung 6.17: Beispielworkflows für LAWS

Die Spezifikation von Abhängigkeiten geschieht in zwei Schritten: Zuerst wird festgelegt, *wann* ein Konflikt vorliegt. Dies wird in LAWS als *Step Conflict Specification* bezeichnet. Anschließend wird basierend auf diesen Konfliktdefinitionen in der *Coordinated Execution Specification* festgelegt, *wie* auf Konflikte reagiert werden soll.

Die *Step Conflict Specification* ist Teil der Definition eines Workflow-Schemas. Die Definition eines Konflikts geschieht über Boole'sche Ausdrücke, in denen auf Ein- und Ausgabedaten von Aktivitäten desselben Workflow-Schemas und auf Aktivitäten anderer Workflow-Schemata Bezug genommen werden kann. Das folgende Beispiel stellt eine *Step Conflict Specification* im Workflow-Schema WF1 dar:

```
IF (S12.I1 == WF2.S23.I2) THEN S12 CONFLICTS_WITH WF2.S23
```

Dies bedeutet Folgendes: Wenn das Eingabedatum I1 des Schritts S12 im Workflow-Schema WF1 mit dem Eingabedatum I2 des Schritts S23 im Workflow-Schema WF2 übereinstimmt, liegt ein Konflikt zwischen S12 und S23 der jeweiligen Workflow-Instanzen vor.

Die *Coordinated Execution Specification* gehört ebenfalls zu einem bestimmten Workflow-Schema und definiert die Art und Weise, wie das WfMS auf Konflikte reagieren soll. Dabei gibt es zwei grundsätzliche Möglichkeiten zur Reaktion: *gegenseitiger Ausschluss* und *relative Reihenfolge*.

Mit *gegenseitigem Ausschluss* ist gemeint, dass das WfMS bei einem Konflikt sicherstellen soll, dass die betroffenen Schritte im Sinne einer Serialisierung nicht gleichzeitig ausgeführt werden. Gegenseitiger Ausschluss kann nicht nur zwischen einzelnen Schritten, sondern auch zwischen Gruppen von Schritten definiert werden.

Ein für sich selbst sprechendes Beispiel für die Definition gegenseitigen Ausschlusses ist das folgende:

```
IF (S12 CONFLICTS_WITH WF2.S23)
THEN S12 MUTUALLY_EXCLUDE WF2.S23
```

Als alternative Reaktionsmöglichkeit kann festgelegt werden, dass zwischen Workflows bei Konflikten *relative Reihenfolgen* erhalten bleiben sollen. Nachfolgend ein Beispiel für eine solche relative Reihenfolgespezifikation in der Workflowdefinition WF1:

```
IF (S12 CONFLICTS_WITH WF2.S23) AND (S14 CONFLICTS WITH WF2.S25)
AND S12 RELATIVE_ORDER WF2.S23 THEN S14 RELATIVE ORDER WF2.S25
```

Damit wird (nicht unbedingt intuitiv verständlich) Folgendes ausgedrückt: Wenn Schritt S12 mit S23 eines Workflows des Schemas WF2 kollidiert und ein Konflikt zwischen S14 mit S25 besteht, sollen die Schritte S14 und S25 in derselben relativen Reihenfolge ausgeführt werden, wie die Schritte S12 und S23. D.h., wurde S12 vor S23 ausgeführt, so muss anschließend S14 vor S25 ausgeführt werden. Umgekehrt gilt natürlich: Sollte S23 vor S12 ausgeführt worden sein, so muss S25 vor S14 ausgeführt werden.

Zusätzlich kann in LAWS auch das Verhalten im Fehlerfall und bei Rücksetzen des Workflows spezifiziert werden. Dies ist jedoch im vorliegenden Kontext nicht relevant.

Mit dem Konstrukt des gegenseitigen Ausschlusses von Aktivitäten können sequenzielle Abhängigkeiten zwischen parallelen Freigabeworkflows nicht beschrieben werden. Die relativen Reihenfolgen in LAWS sehen zwar auf den ersten Blick nach den zu modellierenden sequenziellen Kontrollflussabhängigkeiten zwischen Workflows aus, bei näherer Betrachtung (vgl. Diskussion des obigen Beispiels) wird jedoch klar, dass damit *nicht* festgelegt werden kann, dass eine Aktivität A des einen Workflows vor Beginn der Aktivität B eines anderen Workflows beendet sein *muss*. Es geht hier um die Erhaltung von zur Laufzeit aufgetretenen Reihenfolgen bei der weiteren Ausführung. Somit sind auch die Konzepte von LAWS nicht für unsere Anforderungen bezüglich Synchronisation paralleler Workflows geeignet.

6.2.4 Interaktionsausdrücke und –graphen

In [He00] wird ein sehr umfassender und ausdrucksstarker Mechanismus vorgestellt, parallele Workflows zu synchronisieren. Die Idee hierbei ist, die Synchronisation von Workflows *völlig getrennt* von den Schemata der Einzelworkflows mittels sog. *Interaktionsgraphen* zu beschreiben. Die Interaktionsgraphen stellen allgemeine Integritätsbedingungen dar, die zur Laufzeit zu Einschränkungen führen können, welche Aktivitäten der Workflow-Instanzen zu einem bestimmten Zeitpunkt ausgeführt können. Interaktionsgraphen werden ähnlich wie Syntaxdiagramme nach gewissen Regeln von links nach rechts durchlaufen. Die bei einem Weg durch diesen Graphen passierten Aktivitäten werden als *eine* gültige Ausführungsreihenfolge interpretiert. Es gibt jedoch nicht nur einen, sondern beliebig viele (in vielen Fällen sogar unendlich viele) Wege durch einen Interaktionsgraphen.

Ein Interaktionsgraph beschreibt somit eine *Menge* zulässiger Ausführungsfolgen von Aktivitäten für *alle* ausgeführten Workflow-Instanzen. Zur Laufzeit muss dann für eine bestimmte Aktivität entschieden werden, ob diese ausgeführt werden darf, ohne dass dadurch eine unzulässige Ausführungsreihenfolge entsteht. Diese Fragestellung wird auch als *Aktionsproblem* bezeichnet. Zur Lösung des Aktionsproblems werden die Interaktionsgraphen durchlaufen. Dies ist jedoch nicht trivial, da zu einem Zeitpunkt zwei alternative Wege möglich sein können. Die Entscheidung, in einem Interaktionsgraphen einen bestimmten Weg einzuschlagen, kann also oft nicht sofort nach Ausführung einer bestimmten Aktivität getroffen, sondern muss aufgeschoben werden. Das Durchlaufen von Interaktionsgraphen zur Lösung des Aktionsproblems soll jedoch nicht näher betrachtet werden. Im Vordergrund soll die Ausdrucksmächtigkeit von Interaktionsgraphen in Bezug auf die Anforderungen stehen.

Anhand von Beispielen sollen nun die Möglichkeiten von Interaktionsgraphen diskutiert werden. Basisbaustein von Interaktionsgraphen sind sog. *Aktionen*. Diese Aktionen werden durch Operatoren miteinander verknüpft.

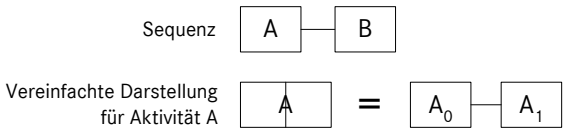

Abbildung 6.18: Graphische Repräsentation von Sequenz und Aktivitäten

Einfachster Operator ist die *Sequenz*. Sind zwei Aktionen A und B mit einer Sequenz verknüpft, so muss im Fall einer gültigen Ausführungsreihenfolge B direkt nach A ausgeführt werden. Aktionen haben konzeptuell keine Zeitdauer. Aktivitäten von Workflows sind jedoch mit einer Zeitdauer behaftet. Um diese beschreiben zu können, wird eine Aktivität in zwei punktuelle Aktionen A_0 (Beginn der Aktivität) und A_1 (Ende der Aktivität) zerlegt, die durch eine Sequenz verknüpft sind. Aus Gründen der Lesbarkeit wird hierfür eine vereinfachte graphische Darstellung gewählt (siehe Abbildung 6.18).

Mit einer *Entweder-Oder-Verzweigung* kann ausgedrückt werden, dass zum Durchlaufen des Graphen entweder der obere oder der untere Pfad eingeschlagen werden kann. Das Beispiel in Abbildung 6.19 zeigt eine Verschachtelung von Entweder-Oder-Verzweigungen. Durch diesen Graphen werden folgende möglichen Ausführungsreihenfolgen beschrieben: ABF, CDF *oder* CEF.

Bei Verschachtelungen in Interaktionsgraphen muss das Prinzip der Blockstrukturiertheit gewährleistet sein, d.h. die von Operatoren eingeschlossene Blöcke dürfen zwar verschachtelt sein, sich jedoch nicht überlappen.

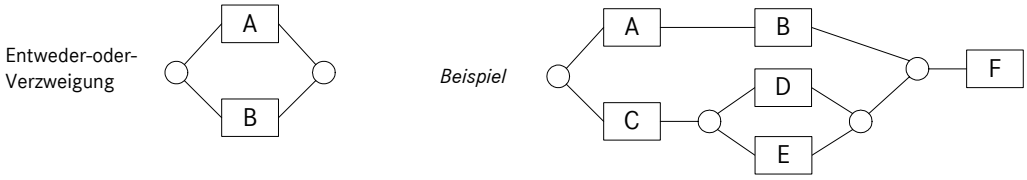

Abbildung 6.19: Entweder-oder-Verzweigung

Das Pendant zur Entweder-Oder-Verzweigung ist die *Sowohl-als-auch-Verzweigung*. Sie drückt aus, dass man beide Zweige *parallel* und *unabhängig voneinander* durchläuft. Durch eine Sowohl-als-auch-Verzweigung werden alle möglichen Verschränkungen der Aktionen ihrer Teilzweige beschrieben. Durch das Beispiel in Abbildung 6.20 sind demnach folgende Ausführungsreihenfolgen definiert: ABCD, ACBD, ACDB, CDAB, CADB, CABD. Nicht möglich ist beispielsweise die Folge BACD, da die Aktion A immer relativ vor B ausgeführt werden muss.

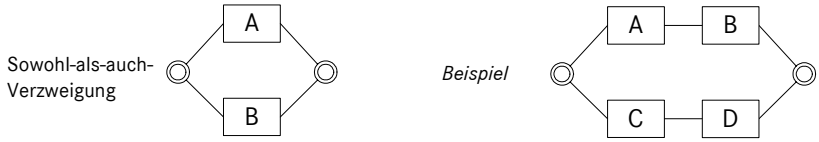

Abbildung 6.20: Sowohl-als-auch-Verzweigung

Eine Verallgemeinerung der Sowohl-als-auch-Verzweigung stellt die *Beliebig-oft-Verzweigung* dar (Abbildung 6.21). Eine Beliebig-oft-Verzweigung entspricht der Menge von Interaktionsgraphen, die man erhält, wenn man die Beliebig-oft-Verzweigung durch eine Sowohl-als-auch-Verzweigung ersetzt und dabei bei die Anzahl der Verzweigungen schrittweise von 0 bis ∞ erhöht. Das Beispiel in Abbildung 6.21 illustriert dies: Durch die Beliebig-oft-Verzweigung der Aktionssequenz A-B werden all diejenigen Ausführungsfolgen beschrieben, die durch eine Sowohl-als-auch-Verzweigung mit einem *leeren* Teilpfad beschrieben sind (die leere Ausführungsfolge), diejenigen, die durch eine Sowohl-als-auch-Verzweigung mit *einem* Teilpfad beschrieben sind, diejenigen, die durch eine Sowohl-als-auch-Verzweigung mit *zwei* Teilpfaden beschrieben sind, etc..

Gültige Ausführungsfolgen des Beispiels wären also: die leere Ausführungsfolge, AB, AABB, ABAB, AAABBB, AABABB, etc.. Anders ausgedrückt werden durch den Interaktionsgraphen im Beispiel alle Folgen von A und B beschrieben, die dieselbe Anzahl der beiden Aktionen enthalten und bei denen an *jeder* Stelle garantiert ist, dass zeitlich vorher (d.h. bildlich gesprochen links davon) die Aktion A *mindestens* so oft ausgeführt wurde wie Aktion B.

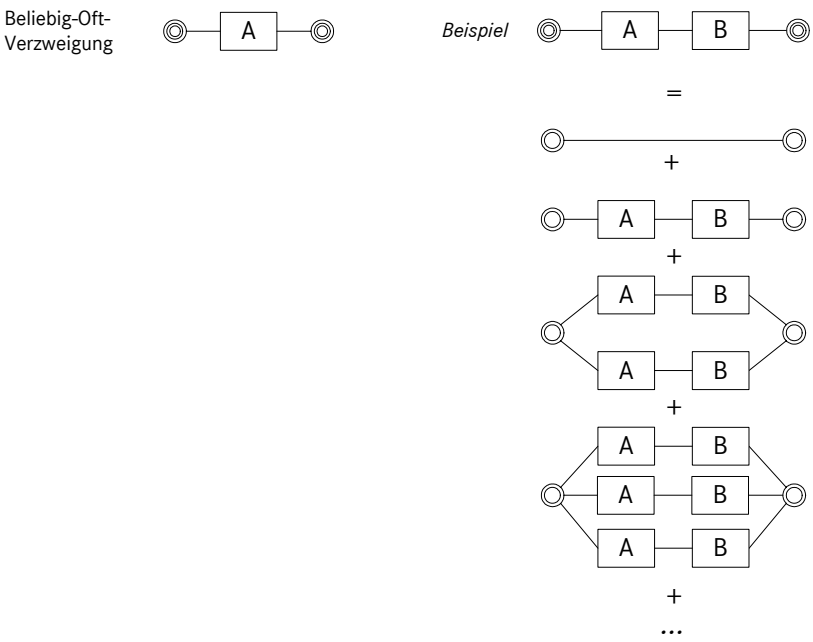

Abbildung 6.21: Beliebig-oft-Verzweigung

Mit den bisher vorgestellten Konzepten ist es möglich, unterschiedlichste Überlappungen von Aktivitäten zu beschreiben, jedoch kann dabei nicht auf Aktivitäten ganz bestimmter Workflow-Instanzen Bezug genommen werden. Das in Abbildung 6.22 dargestellte Beispiel soll dies verdeutlichen. Neben zwei Workflow-Schemata WFS 1 und WFS 2 ist mittels eines Interaktionsgraphen eine Integritätsbedingung zwischen Aktivitäten der beiden Schemata angegeben. Damit sind beispielsweise folgende Ausführungsreihenfolgen zulässig: EB, EEBB oder EBEB. (Aktivitäten, die nicht im Interaktionsgraphen vorkommen, werden bei der Auswertung der Integritätsbedingung einfach ignoriert.) Was dies für die Workflow-Instanzen bedeutet, zeigen die darunter abgebildeten Workflow-Instanzen WFI 1-4. In der Ausgangssituation links wurde in WFI 3 die Aktivität E ausgeführt. Nun kann *eine* Aktivität B einer *beliebigen* Instanz (ob WFI 1 oder WFI 2 oder eine ganz andere Workflow-Instanz ist offen) ausgeführt werden. Im Beispiel wird nun Aktivität B in WFI 2 ausgeführt. Bevor Aktivität B in WFI 1 ausgeführt werden kann, muss in einer anderen Workflow-Instanz (beispielsweise WFI 4) wieder Aktivität E ausgeführt werden. Nicht möglich wäre es, mit den bisher vorgestellten Konzepten zu beschreiben, dass beispielsweise genau WFI 2 und WFI 3 miteinander zu koppeln sind (d.h., dass die Aktivität E in WFI 3 der Aktivität B in WFI 2 vorhergehen soll.)

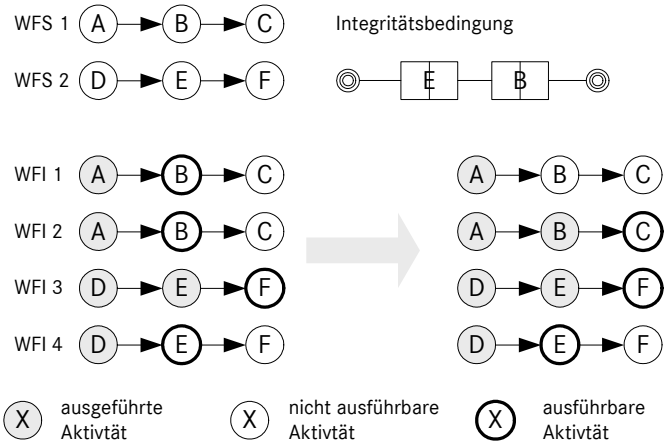

Abbildung 6.22: Beispiel Interaktionsgraph ohne Parameter

Um über bestimmte Aktivitäteninstanzen Integritätsbedingungen formulieren zu können, existiert das Konzept der *parametrisierten Aktionen*. Eine Aktion kann im Interaktionsgraphen mit beliebig vielen Parametern versehen werden. Die Aktionen werden zur Laufzeit mit konkreten Belegungen für diese Parameter ausgeführt. Gleiche Aktionen mit unterschiedlichen Parametern werden als voneinander verschiedene Aktionen betrachtet. Mit Parametern versehene Aktionen müssen sich innerhalb einer *Für-alle-Verzweigung* oder einer *Für-ein-Verzweigung* des jeweiligen Parameters befinden.

Eine *Für-alle-Verzweigung* entspricht einer *Sowohl-als-auch-Verzweigung*, die für jeden Wert des Parameters einen Teilzweig enthält und in dem jedes Vorkommen des Parameters in einer Aktion durch den jeweiligen Parameterwert ersetzt ist. Abbildung 6.23 macht dies an einem Beispiel nochmals deutlich. Der Parameter k repräsentiert einen Kunden und hat den endlichen Wertebereich {kunde1, kunde2, kunde3}. Der Interaktionsgraph bedeutet nun, dass für einen *bestimmten* Kunden k die Schritte E und B nacheinander ausgeführt werden sollen. Für die im unteren Teil der Abbildung dargestellten Workflow-Instanzen hat dies folgende Auswirkungen: Da in WFI 3 der Schritt E für *kunde1* ausgeführt wurde, kann in WFI 1 der Schritt E für *kunde1* ausgeführt werden. Es ist nicht wie in Abbildung 6.22 möglich, dass der Schritt B in WFI 2 ausgeführt werden kann. Hierzu müsste vorher Schritt E in WFI 3 für *kunde2* ausgeführt werden.

Zur besseren Verständlichkeit wurde im Beispiel ein endlicher Wertebereich für den Parameter verwendet. In der Realität soll die durch den Interaktionsgraphen definierte Integritätsbedingung *für alle* Kunden gelten, für die jemals Workflow-Instanzen laufen, d.h. der Wertebereich für k ist unendlich groß. Für die Auswertung des Beispielinteraktionsgraphen zur Laufzeit bedeutet dies stark vereinfacht, dass logisch gesehen ein neuer Zweig in die Sowohl-als-auch-Verzweigung eingefügt wird, sobald eine Aktivität mit einem neuen Parameterwert auftritt.

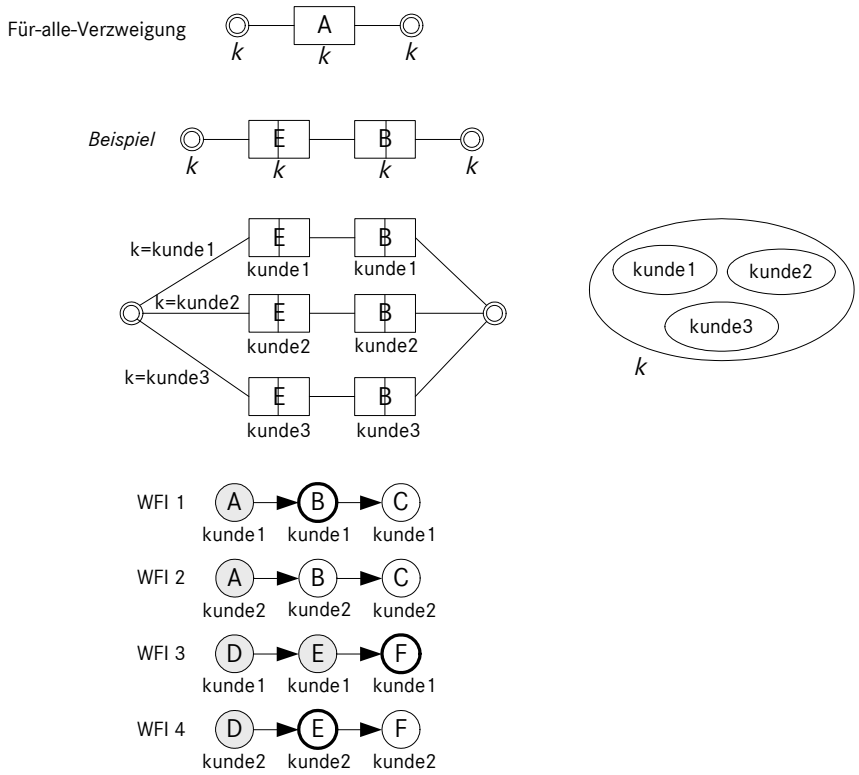

Abbildung 6.23: Für-alle-Verzweigung

Die *Für-ein-Verzweigung* ist das Pendant zur *Für-alle-Verzweigung* und entspricht einer *Entweder-oder-Verzweigung*, die für jeden Wert des Parameters einen Teilzweig enthält. In diesem Teilzweig ist jeder Parameter einer Aktion durch den konkreten Wert ersetzt. Abbildung 6.24 zeigt ein Beispiel, in dem der Parameter p ein Produkt repräsentiert und den Wertebereich {prod1, prod2, prod3} besitzt. Der Interaktionsgraph bedeutet für die den Produkten zugeordneten Workflow-Instanzen Folgendes: Wird der Schritt E für ein bestimmtes Produkt ausgeführt, so kann anschließend der Schritt B für dieses Produkt ausgeführt werden – alle Schritte E und B für andere Produkte können dann *nicht mehr* ausgeführt werden. Im Beispiel wurde in WFI 3 der Schritt E für *prod1* ausgeführt. Damit kann nur noch Schritt B in WFI 1 ausgeführt werden. Die Schritte E(prod2) und B(prod2) und damit auch deren Nachfolgeaktivitäten können nicht mehr ausgeführt werden.

Die *Für-ein-Verzweigung* bedeutet also, dass man sich beim Durchlaufen des Interaktionsgraphen an dieser Stelle für *irgendeine* Belegung des Parameters entscheiden muss. Ein Vergleich zu prädikatenlogischen Ausdrücken liegt hier nahe: Die Für-ein-Verzweigung könnte man als Existenz-Quantor interpretieren, dementsprechend die Für-alle-Verzweigung als All-Quantor.

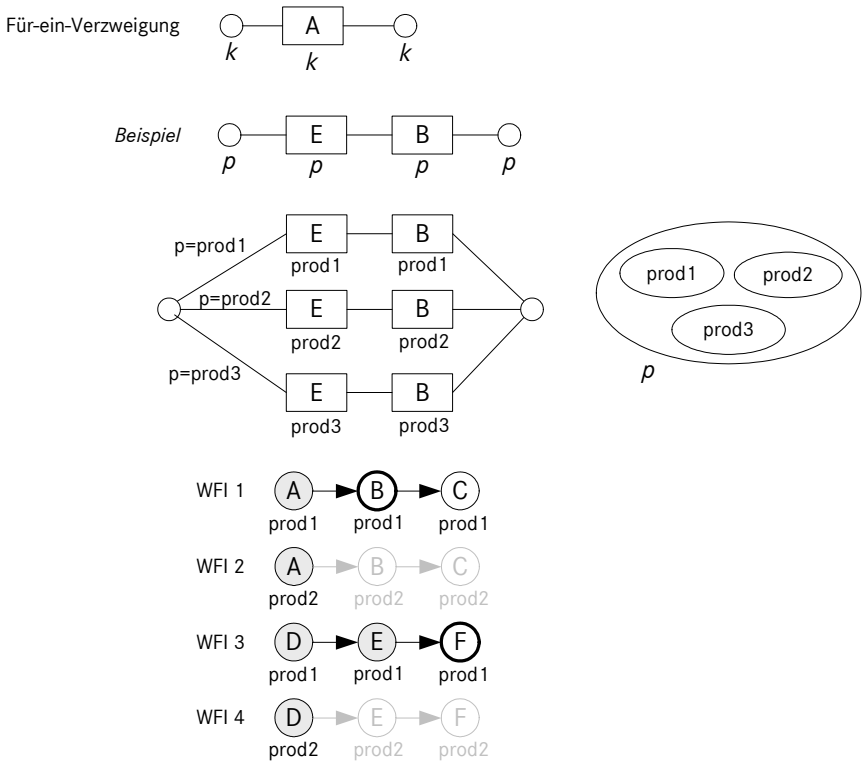

Abbildung 6.24: Für-ein-Verzweigung

Die bisher vorgestellten Konstrukte stellen nur eine Auswahl der wichtigsten Konzepte der Interaktionsgraphen dar. Es existieren noch weitere Konzepte zur Schleifenbildung, Wiederverwendung von Teilgraphen und Definition von Vorlagen für Interaktionsgraphen, die jedoch für die in dieser Arbeit betrachtete Problemstellung eine untergeordnete Rolle spielen.

Trotz der großen Ausdrucksmächtigkeit reichen Interaktionsgraphen nicht aus, um die Abhängigkeiten zwischen Freigabeworkflows beschreiben zu können. Dies liegt hauptsächlich darin begründet, dass die Parameter untypisiert sind und es kein Datenmodell gibt, auf das in Interaktionsgraphen Bezug genommen werden kann.

Dies soll durch das folgende Beispiel erläutert werden: Abbildung 6.25 zeigt zwei Workflow-Schemata für Freigabeprozesse. Dabei soll *WFS 1* den Freigabeworkflow für die untere Ebene der Konfigurationen beschreiben und *WFS 2* denjenigen für die direkt darüber liegende. Für die rechts dargestellte Konfigurationshierarchie bedeutet dies, dass die Freigabeworkflows für die Konfigurationen *konf1*, *konf2* und *konf3* durch WFS 1 und Freigabeworkflows für *konf4* und *konf5* durch *WFS 2* beschrieben sind. Links neben der Konfigurationshierarchie sind Instanzen der Workflow-Schemata für die Konfigurationen *konf1*, *konf2*, und *konf4* abgebildet. Abhängigkeiten existieren zwischen den Aktivitäten D_1 und D_2. Für die abgebildeten Workflow-Instanzen bedeutet dies, dass vor Aktivität D_2(konf4) die Aktivitäten D_1(konf1) und D_1(konf2) ausgeführt werden müssen. Die Frage ist nun, ob sich diese Abhängigkeiten allgemeingültig durch Interaktionsgraphen beschreiben lassen.

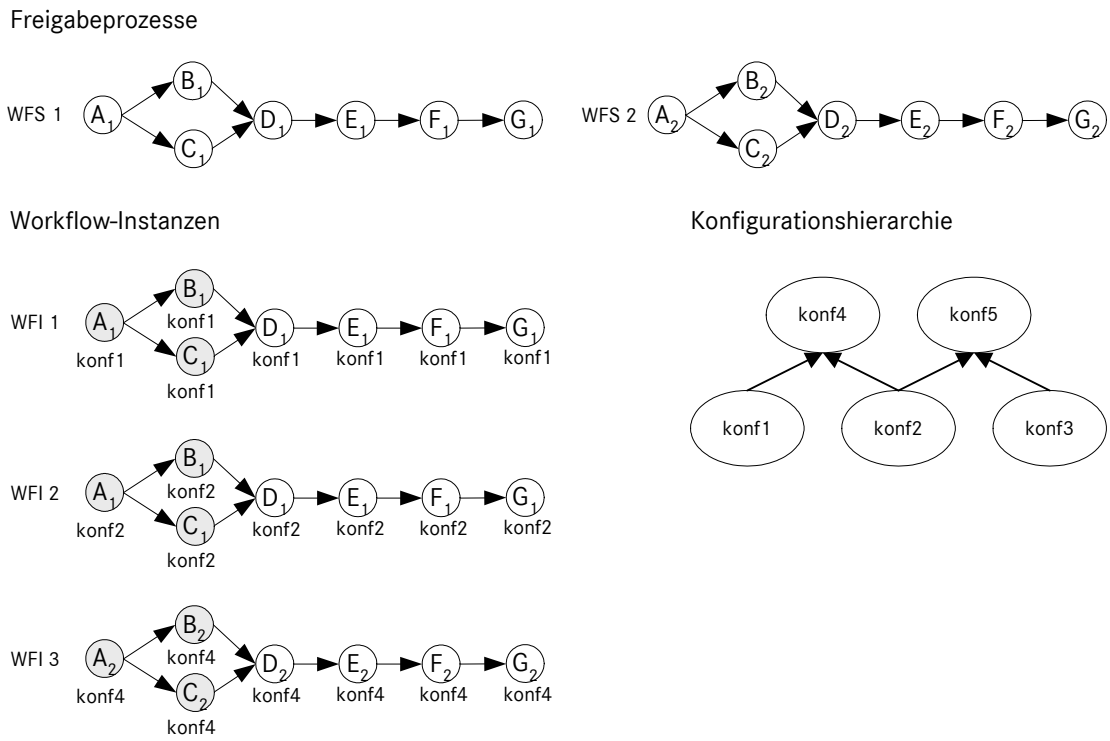

Abbildung 6.25: Beispielszenario

Für Interaktionsgraphen soll nun der Parameter k einer Aktivität die Konfiguration ausdrücken, für die diese Aktivität ausgeführt wird. Im Wertebereich für k sind demnach alle Konfigurationen. Der in Abbildung 6.26 als Versuch 1 abgebildete Interaktionsgraph scheint diese Abhängigkeiten auf den ersten Blick auszudrücken: D_1 soll für die Konfigurationen vor D_2 ausgeführt werden. Durch den Interaktionsgraphen wurde jedoch fälschlicherweise festgelegt, dass für jede Konfiguration die Schritte D_1 und D_2 nacheinander ausgeführt werden sollen. Dies ist jedoch falsch: Es soll D_1 in allen Unterkonfigurationen vor D_2 der Oberkonfiguration ausgeführt werden. Die hier formulierte Integritätsbedingung hat zur Folge, dass die Aktivität *D_2(konf4)* in *WFI 3* niemals ausgeführt wird, da vorher auf die Beendigung einer Aktivität *D_1(konf4)* gewartet werden muss, die es jedoch gar nicht gibt.

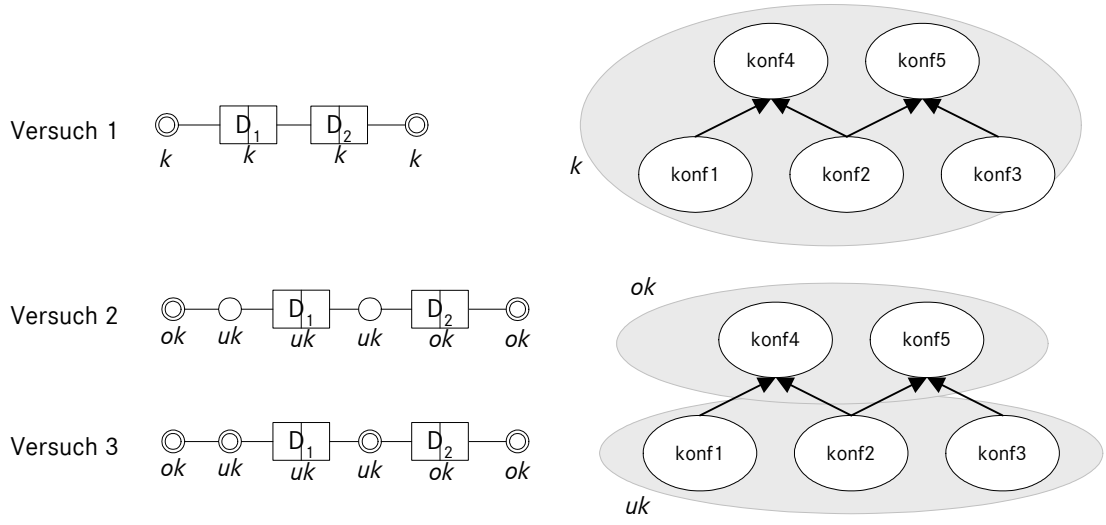

Abbildung 6.26: Spezifikationsversuche mit Interaktionsgraphen

In einem zweiten Versuch soll nun gezeigt werden, dass auch die Unterscheidung zwischen Unter- und Oberkonfiguration in den Parametern keine Verbesserung bringt. Dazu werden die möglichen Konfigurationen in zwei Teilmengen aufgeteilt: In der einen Menge sind alle Oberkonfigurationen und in der anderen Menge alle Unterkonfigurationen enthalten. Alle Aktivitäten in *WFS 1* sollen mit einem Parameter *uk* versehen werden, der als Wertebereich alle Unterkonfigurationen besitzt, alle Aktivitäten in *WFS 2* entsprechend mit einem Parameter *ok* und dem Wertebereich aller möglichen Oberkonfigurationen. Dies ist stark vereinfacht, da es im Allgemeinen mehr als zwei Ebenen innerhalb der Konfigurationen gibt.

Doch auch für diese vereinfachte Variante reichen die Möglichkeiten von Interaktionsgraphen nicht aus. Versuch 2 der Spezifikation drückt Folgendes aus: Voraussetzung der Ausführung von D_2 für eine Oberkonfiguration ist, dass der Schritt D_1 für *eine* Unterkonfiguration ausgeführt worden ist. Dies kann jedoch jede beliebige Konfiguration in der Menge der Unterkonfigurationen sein und muss daher nicht zwangsläufig die Unterkonfiguration der jeweiligen Oberkonfiguration sein. Außerdem muss D_1 für alle Unterkonfigurationen einer Konfiguration beendet sein. Dies wird durch diesen Interaktionsgraphen ebenfalls nicht ausgedrückt.

Versuch 3 kommt den auszudrückenden Abhängigkeiten am nächsten: Für *alle* Oberkonfigurationen soll für *alle* Unterkonfigurationen zuerst D_1 und dann D_2 ausgeführt werden. Da aber wiederum nicht ausgedrückt werden kann, dass für eine Oberkonfiguration nicht alle Unterkonfigurationen in der Menge der Unterkonfigurationen berücksichtigt werden dürfen, sondern nur diejenigen, die in einer entsprechenden Relation zur Oberkonfiguration stehen, wird das gewünschte Verhalten nicht erreicht.

Die dargestellten Versuche haben gezeigt, dass mit Interaktionsgraphen die Abhängigkeiten zwischen Freigabeworkflows nicht adäquat modelliert werden können. Ein weiteres Manko von Interaktionsgraphen stellen die mitunter sehr komplexen Modellierungskonstrukte dar, für die zumindest bezweifelt werden darf, dass sie von allen betroffenen Anwendern verstanden werden. Sehr positiv ist die saubere Trennung zwischen Workflow-Schemata und deren Instanzen einerseits und den Abhängigkeiten zwischen parallelen Workflows andererseits. Dies ermöglicht es, Interaktionsgraphen unabhängig von einer konkreten Workflow-Sprache zu verwenden.

6.2.5 Zusammenfassung

Die vorangegangene Diskussion von Ansätzen zur Synchronisation paralleler Workflows hat gezeigt, dass diese für Freigabeworkflows nicht ausreichend sind. Die geforderten sequenziellen Kontrollflussabhängigkeiten sind im Prinzip sehr einfach. Für welche Workflow-Instanzen diese gelten sollen, hängt jedoch von der Konfigurationshierarchie, d.h. dem Datenmodell, ab. An dieser Stelle fehlt *allen* betrachteten Ansätzen die Möglichkeit, bei der Modellierung der Abhängigkeiten auf ein Datenmodell Bezug nehmen zu können.

6.3 Ausnahme- und Fehlerbehandlung

6.3.1 Klassifikation von Fehlern

In einer grundlegenden Arbeit zum Thema Ausnahme- und Fehlerbehandlung [EdLi95] werden vier Klassen von Fehlern bzw. Ausnahmen identifiziert: *fundamentale* Fehler, *Anwendungs*fehler, *erwartete* Ausnahmen und *unerwartete* Ausnahmen (vgl. Tabelle 6.1).

	Typ
Ausnahmen	unerwartete Ausnahmen
	erwartete Ausnahmen/semantische Fehler
Fehler	Anwendungsfehler
	fundamentale Fehler

Tabelle 6.1: Klassifikation von Fehlern in Workflows

Als fundamentale Fehler werden Fehler auf Systemebene bezeichnet, wie beispielsweise Systemabstürze, Deadlocks oder Kommunikationsprobleme. Anwendungsfehler hingegen sind Fehler, die beim Aufrufen einer mit einer Aktivität verknüpften Anwendung auftreten. Typische Beispiele hierfür sind Programmierfehler in den Anwendungen oder fehlerhafte Datenversorgung.

Während sich die gerade besprochenen Fehler auf der technischen Seite ereignen, sind die Ausnahmen auf der Seite der Realwelt (d.h. der Geschäftsprozesssemantik) anzusiedeln. *Erwartete* Ausnahmen sind *vorhersehbare* Abweichungen vom normalen Prozessverhalten und entsprechen dem semantischen Fehlschlagen einer Aktivität. Deshalb werde diese Ausnahmen oft auch als *semantische Fehler* bezeichnet. Eine solche Abweichung kann beispielsweise in einem Reisebuchungsprozess das Fehlschlagen einer Flugbuchung sein, weil das Flugzeug bereits ausgebucht ist. Ein weiteres Beispiel hierfür wäre, wenn in einem Support-Prozess ein Kunde angerufen werden soll, und dieser den Anruf nicht entgegennimmt.

Unerwartete Ausnahmen sind solche, die darauf zurückgehen, dass der Prozess der Realwelt nicht mehr vollständig dem ausgeführten Workflow entspricht. Ein Beispiel für eine solche Ausnahme ist, wenn in einem Patientenbehandlungsprozess auf Grund des Zustands des Patienten bestimmte Untersuchungsschritte eingefügt, vorgezogen oder weggelassen werden müssen.

6.3.2 Einordnung

Bei Fehlern in Prüfschritten handelt es sich weder um technische, noch um semantische Fehler im engeren Sinn. Ein (semantischer) Fehler in einer Aktivität bedeutet, dass diese logisch fehlgeschlagen ist. In unserem Falle bedeutet „Fehler in einem Prüfschritt“, dass die Aktivität selbst erfolgreich durchgeführt wurde, aber in der zugehörigen Konfiguration Fehler gefunden wurden. Es handelt sich bei einem Prüffehler also um ein mögliches, reguläres Ergebnis einer Aktivität und kann bei einer Ausweitung des Begriffs „semantischer Fehler“ auch als solcher verstanden werden.

Nach einem Prüffehler sind manche Schritte des Prozesses nicht mehr relevant, d.h. der Realprozess entspricht nicht mehr dem laufenden Workflow. Der Prüffehler selbst wird zwar als

reguläres Ergebnis einer Aktivität durchaus erwartet, jedoch ist die Abweichung danach in ihrer Art und Weise unerwartet. Die nicht mehr relevanten Schritte können erst zu dem Zeitpunkt bestimmt werden, an dem der Prüffehler passiert ist. Hier handelt es sich der Definition nach also um eine *unerwartete* Ausnahme. D.h. den Prüffehler selbst wird man nicht als unerwartete Ausnahme bezeichnen, jedoch die daraufhin möglicherweise notwendigen Änderungen an der Prozessstruktur.

Für alle Klassen von Fehlern wurden zahlreiche Lösungsansätze zu deren Behandlung entwickelt. Die Prüffehler selbst lassen sich jedoch in keine dieser Klassen einordnen und demnach lassen sich hierfür keine ausgearbeiteten Lösungsansätze in der Literatur [ScBi96] finden. Für *unerwartete Ausnahmen,* wie sie durch Prüffehler ausgelöst werden, gibt es im Themenbereich „Dynamische Änderung von Workflows“ zahlreiche Arbeiten, die separat in Abschnitt 6.4 diskutiert werden.

6.3.3 Beispiel Kompensationssphären

Als Beispiel für ein konkretes Konzept zur Fehlerbehandlung soll anhand von *Kompensationssphären* [LeRo00] deutlich gemacht werden, was aus klassischer Sicht semantische Fehler sind und wie sie behandelt werden können.

Semantische Fehler sind Fehler im Sinne einer inkorrekten Ausführung einer Aktivität in der *Real*welt: Ein fehlerhafter Brief wird an einen Kunden verschickt, eine Flugreservierung schlägt fehl, weil keine Plätze mehr vorhanden sind, usw.. Solche semantischen Fehler können nicht mit üblicher Transaktionslogik behoben werden, indem beispielsweise ein bestimmtes Datenbankimage wiederhergestellt wird. Es ist vielmehr erforderlich, dass die Auswirkung des Fehlers auf die Realwelt rückgängig gemacht wird. Zu diesem Zweck wurde das Konzept der *Kompensationssphären* entwickelt.

Eine Kompensationssphäre umfasst eine beliebige Anzahl von Aktivitäten (Abbildung 6.27). Für jede Aktivität kann eine Kompensationsaktivität angegeben werden, die die Auswirkung dieser Aktivität auf die Realwelt rückgängig macht. Im Beispiel des fehlerhaften Briefs könnte dies das Verschicken eines korrekten Schreibens mit einer Entschuldigung für den Fehler sein.

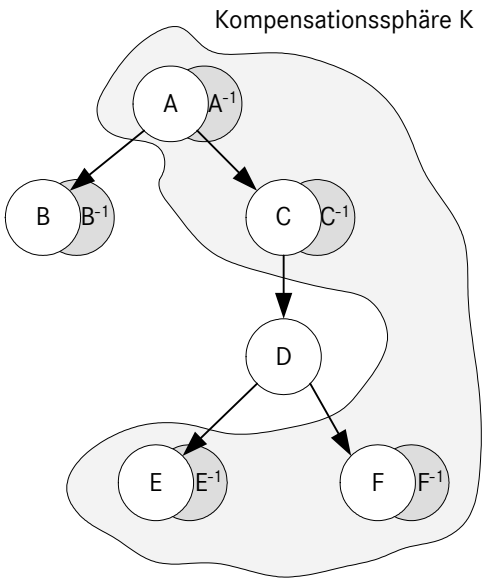

Abbildung 6.27: Kompensationssphären nach [LeRo00]

Wird für eine Aktivität in einer Kompensationssphäre festgestellt, dass sie semantisch inkorrekt ausgeführt worden ist, so werden für alle bereits ausgeführten Aktivitäten der Kompensationssphäre die entsprechenden Kompensationsaktivitäten ausgeführt. Genauer gesagt konstruiert das WfMS *automatisch* ein Workflow-Modell, das aus den Kompensationsaktivitäten besteht und führt dies dann aus. Bei der automatischen Ableitung des Workflow-Modells wird – vereinfacht gesprochen – die ursprüngliche Ausführungsreihenfolge der schon ausgeführten Aktivitäten invertiert.

Abbildung 6.28 zeigt dies an einem Beispiel: Die Aktivitäten A, C, E und F sind Teil der Kompensationssphäre K. Aktivität E wird inkorrekt ausgeführt. Als Folge werden alle bereits ausgeführten Aktivitäten durch Kompensationsaktivitäten in umgekehrter Reihenfolge ihrer Ausführung rückgängig gemacht. Auf der rechten Seite ist das vom WfMS automatisch generierte Workflow-Modell zur Fehlerkompensation abgebildet.

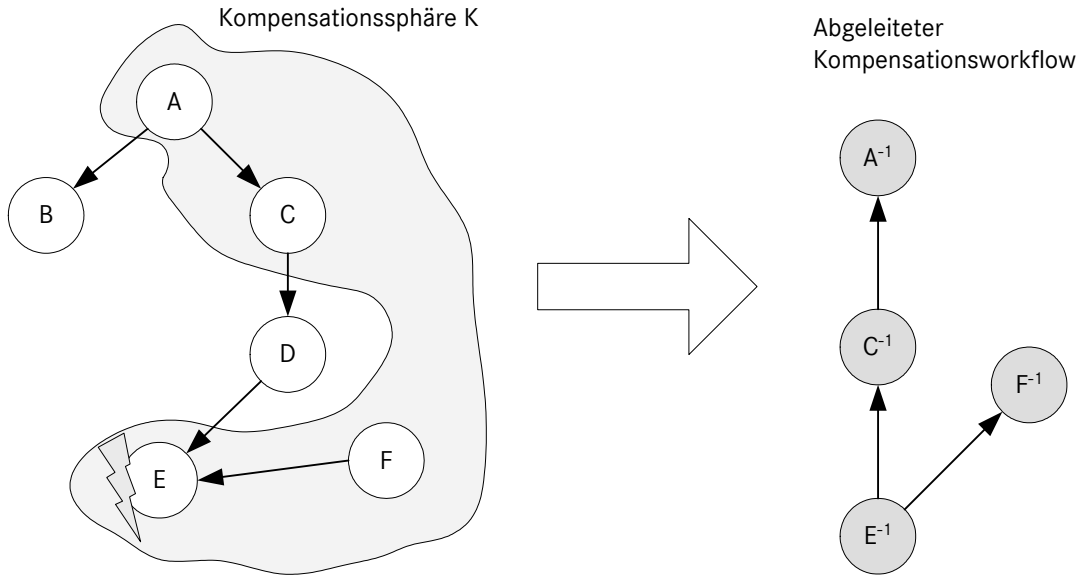

Abbildung 6.28: Ableitung eines kompensierenden Workflow-Modells nach [LeRo00]

Nicht immer genügt für das Kompensieren eines Fehlers die Ausführung der Kompensationsaktivitäten der Einzelaktivitäten. Für solche Fälle ist es möglich, eine Kompensationsaktivität für eine *ganze* Kompensationssphäre zu definieren.

6.3.4 Zusammenfassung

Obwohl die Prüffehler nicht in die klassischen Fehlerkategorien passen, weisen sie in verschiedener Hinsicht Ähnlichkeit mit semantischen Fehlern auf:

- Sie können jederzeit im Workflow auftreten.
- Sie werden erwartet.
- Sie geschehen in der Realwelt.

Trotz dieser Gemeinsamkeiten benötigen Prüffehler eine ganz andere Art von Behandlung (siehe auch 5.2.5), die nicht *automatisch* geschehen kann (wie beispielsweise bei Kompensationssphären), sondern *manuelle* Eingriffe erfordert. Dies liegt darin begründet, dass die Art der Reaktion sehr stark von der jeweiligen Situation abhängt, in der ein Prüffehler auftritt.

Insbesondere die Schwere des gefundenen Fehlers und der Fortschritt anderer Freigabeprozesse in der Konfigurationshierarchie spielen dabei eine Rolle.

6.4 Dynamische Änderungen von Workflows

Eine weitere Kernanforderung aus Abschnitt 5.4 ist die Möglichkeit, dynamisch Aktivitäten aus einem Workflow löschen zu können. Diese Problematik stellt einen Spezialfall dynamischer Workflow-Änderungen dar, wie sie in zahlreichen Forschungsarbeiten thematisiert worden sind. Wie bereits angedeutet, geht es an dieser Stelle um den Umgang mit *unerwarteten Ausnahmen* im Sinne von [EdLi95]. Unerwartet bedeutet, dass der Realprozess nicht mehr dem Workflow-Modell entspricht und somit eine Änderung erforderlich ist.

Dynamische Änderungen von Workflows können prinzipiell auf zwei Ebenen stattfinden (siehe auch [RRD04]): auf der Ebene einzelner Workflow-Instanzen und auf der Ebene des Workflow-Schemas. (Ad-hoc-)Änderungen auf Instanzebene betreffen meist nur eine einzige laufende Instanz. Änderung auf Schemaabene hingegen betreffen potenziell alle Instanzen eines Schemas. Im Zusammenhang mit Freigabeworkflows sind in erster Linie Ad-hoc-Änderungen bzw. instanzbezogene Änderungen von Bedeutung.

Die Ad-hoc-Änderung eines Workflows bedeutet, dass der Nutzer zur Laufzeit in die Ausführung einer bestimmten Workflow-Instanz eingreifen und Änderungen vornehmen kann. Ließe man ihm dabei vollkommen freie Hand, könnten durch evtl. resultierende Inkonsistenzen ernste Probleme entstehen, wie Verklemmungen oder Systemfehler bei Aufruf von Anwendungen mit unvollständigen Eingabedaten. Deshalb sollte das WfMS nur Änderungen zulassen, die die Konsistenz und damit die Korrektheit des Workflows erhalten [ReDa98]. Weitere Aspekte (vgl. [ReDa98]) sind Sicherheit („Wer darf Änderungen durchführen?“) sowie Ausführungs- und Laufzeit-Aspekte: Da die Änderung nur eine Workflow-Instanz eines Schemas betrifft, muss diese (zumindest logisch) durch einen separaten Ausführungsgraphen repräsentiert werden. Auch sollten die Änderungen keine Performance-Einbußen nach sich ziehen und andere Beteiligte des Workflows nicht beeinträchtigen.

Neben „echter Adaptivität“, wie sie gerade beschrieben wurde, gibt es auch weniger mächtige Ansätze für Adaptivität von Workflows. Beispiel hierfür ist das späte Modellieren von Subworkflows in MOVE [Re00]. Für einzelne Transitionen eines Petri-Netzes kann festgelegt werden, ob sie zur Laufzeit durch einen Subworkflow ersetzt werden sollen. Kommt die Ausführung an diese Transition, so kann zur Laufzeit dynamisch ein Subworkflow für diese Transition modelliert werden. Ansätze dieser Art sind jedoch für die Problematik dieser Arbeit nicht tauglich, da Aktivitäten dynamisch gelöscht und nicht hinzugefügt werden sollen.

Im Folgenden werden exemplarisch mit WASA und $\text{ADEPT}_{\text{flex}}$ zwei Ansätze vorgestellt, die „echte Adaptivität“ ermöglichen. Es wird insbesondere darauf eingegangen, ob und wie das Löschen von Aktivitäten möglich ist. Ziel dieses Abschnitts ist es *nicht*, einen umfassenden Überblick über den Stand der Wissenschaft im Bereich „adaptive Workflows“ zu geben, sondern anhand von Beispielen zu zeigen, wo die prinzipiellen Problemstellungen liegen, und ob es für in dieser Arbeit betrachteten Anforderungen bereits ausreichende Lösungsansätze gibt.

6.4.1 WASA

In WASA [WHKS98, Wes99a, Wes99b] wird zur Modellierung von Workflows eine Aktivitätennetzen (siehe Abschnitt 4.5) ähnliche Workflow-Sprache verwendet. Aktivitätennetze werden um Möglichkeiten zur dynamischen Änderung erweitert. WASA verfolgt einen ob-

jektorientierten Ansatz zur Modellierung und Ausführung von Workflows. Workflow-Schemata und –Instanzen sind Objekte eines objektorientierten Metamodells, die über eine *instance-of-Beziehung* verknüpft sind. Die Ausführung von Workflows geschieht bei WASA nicht durch eine zentrale Workflow-Engine, sondern mit verteilten Objekten, die Workflow-Instanzen bzw. deren Aktivitäten repräsentierten. Diese Objekte kennen nur Objekte in ihrer unmittelbaren Umgebung und agieren bei der Ausführung weitgehend autonom. Abbildung 6.29 zeigt hierfür ein Beispiel. Jeder Knoten repräsentiert ein Objekt, die Kanten und Pfeile entsprechen Beziehungen zwischen Objekten. Das Objekt S repräsentiert das Schema und setzt sich aus mehreren Objekten zusammen, die Aktivitäten auf Schemaebene repräsentieren (in den Begriffen von WASA sind dies Workflow-Schemata für einfache Workflows). S1 repräsentiert eine Workflow-Instanz von S und ist über die instance-of-Beziehung mit dem Schemaobjekt S verknüpft. S1 kennt alle (direkten) Aktivitäteninstanzen seiner Ausführung (A1 bis E1). Diese sind ebenfalls über eine instance-of-Beziehung mit ihrem Schema verknüpft. Jedes Objekt, das eine Aktivitäteninstanz repräsentiert, kennt seine direkten Nachfolger im Kontrollfluss und benachrichtigt diese, wenn die eigene Ausführung beendet ist. Auf die Darstellung des Datenflusses wurde hier verzichtet.

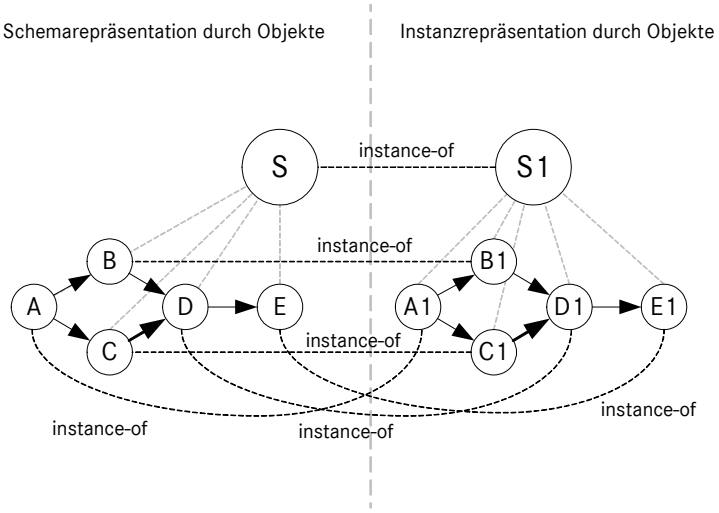

Abbildung 6.29: Repräsentation von Workflow-Instanz und -schema bei WASA

Die dynamische Änderung geschieht durch folgende Schritte [Wes99b]:

- Ein neues Workflow-Schema wird erzeugt oder eine Kopie des Schemas modifiziert.
- WASA bestimmt aufgrund eines Korrektheitskriteriums, welche Instanzen von S auf das neue Schema migriert werden *können*. Das Korrektheitskriterium trifft im Wesentlichen die Aussage, dass eine Workflow-Instanz nur dann zu einem neuen Schema migriert werden kann, wenn sie zum Betrachtungszeitpunkt auch nach dem neuen Schema hätte ausgeführt werden können.
- Aus dieser Menge kann ein Administrator eine Teilmenge auswählen, die tatsächlich migriert werden soll.
- Für jede zu migrierende Workflow-Instanz werden die *instance-of-Beziehungen* vom alten Schema zum neuen Schema angepasst und zusätzliche Änderungen an der Objektstruktur durchgeführt.
- Mit der Ausführung aller Workflow-Instanzen wird fortgefahren.

Abbildung 6.30 zeigt die Migration der Beispiel-Instanz zu einem neuen Schema S', in das zwei Aktivitäten F und G eingefügt und die Aktivität E gelöscht wurde. Die Aktivitätsinstanzen A1, B1, C1 sind in der Workflow-Instanz S1 bereits beendet worden. Zur Migration werden das Objekt E1 und die Beziehungen zwischen B1 und D1 sowie D1 und B2 gelöscht, auch die instance-of-Beziehung zum Schema S wird entfernt. Es werden zwei neue Aktivitätsinstanzobjekte F1 und G1 erzeugt und entsprechend des Schemas S' eingefügt. Neue instance-of-Beziehungen werden zwischen S1 und S' und zwischen F1, D1 und ihren entsprechenden Objekten auf Schemaebene (F und D) angelegt.

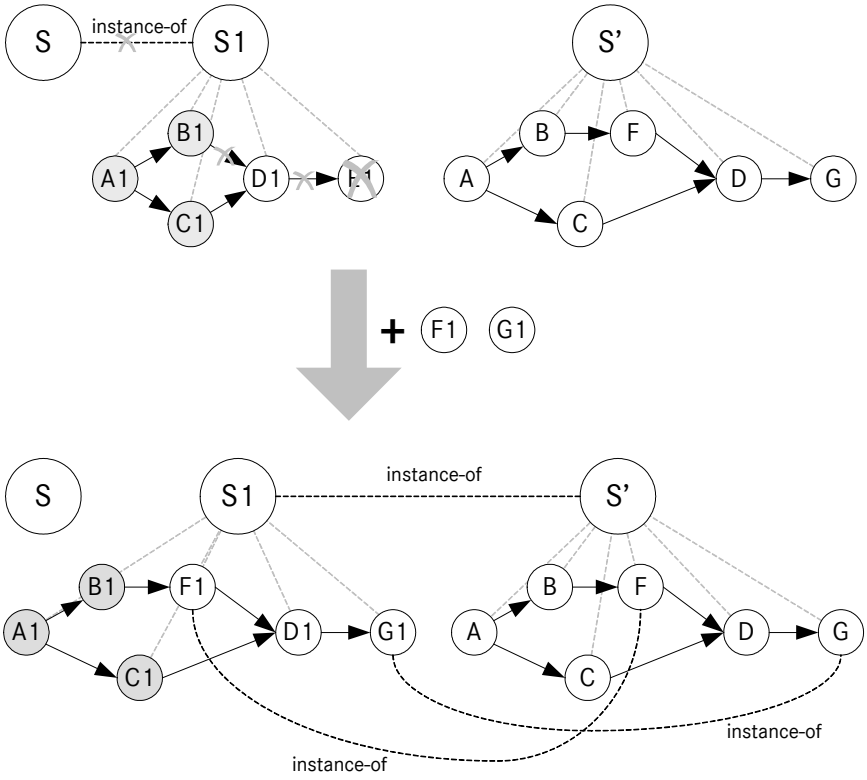

Abbildung 6.30: Dynamische Migration einer Workflow-Instanz bei WASA

Nicht alle Workflow-Instanzen können migriert werden. Die in Abbildung 6.31 dargestellte Workflow-Instanz S2 kann nicht zu S' migriert werden, da die Ausführung zu weit fortgeschritten ist: in S' wird verlangt, dass vor D die Aktivität F ausgeführt werden soll, dies kann für S2 nicht mehr erreicht werden, da D1 schon beendet wurde.

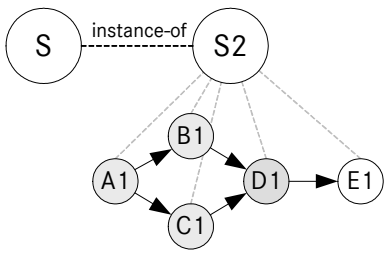

Abbildung 6.31: Nicht migrierbare Workflow-Instanz

Als formales Korrektheitskriterium für die Durchführbarkeit einer Migration wird bei WASA das folgende verwendet: Ist der bereinigte Instanzgraph, der alle ausgeführten Aktivitäten enthält, ein Präfix des Graphen des neuen Schemas, so lässt sich die Workflow-Instanz migrieren [vgl. RRD04].

WASA betrachtet dynamische Änderungen auf einer implementierungsnahen Ebene und befasst sich dabei weniger mit formalen Grundlagen. WASA unterstützt vom Grundprinzip Änderungen am Workflow-Schema und die Propagierung dieser Änderungen auf die Instanzen (Schemaevolution). Änderungen für einzelne Instanzen sind dadurch auch realisierbar, indem für diese Instanz eine neue Version des Schemas erzeugt und anschließend nur diese Instanz migriert wird. Diese Vorgehensweise verlangt jedoch vom ändernden Nutzer, dass dieser die Modellierungssprache für die Workflow-Schemata komplett versteht sowie eine verständliche Benutzeroberfläche für das Editieren der Schemata bereitgestellt wird. Es ist so jedoch nicht sichergestellt, dass der Benutzer nur Änderungen durchführt, nach denen die Instanz immer noch auf das Schema übertragbar bleibt.

6.4.2 ADEPT$_{flex}$

Unter dem Namen ADEPT$_{flex}$ [ReDa98, Re00] wurden auf Basis des formalen Workflow-Modells ADEPT Konzepte für dynamische Änderungen an laufenden Workflow-Instanzen entwickelt. ADEPT$_{flex}$ zielt darauf ab, dass durch Änderungen die Korrektheit und Konsistenz der Workflow-Instanzen gewährleistet bleiben. Dabei kann auf die für Workflow-Modelle in ADEPT formal formulierten Korrektheitskriterien zurückgegriffen werden. Da ADEPT Grundlage für ADEPT$_{flex}$ ist, soll nun zuerst ADEPT und anschließend ADEPT$_{flex}$ vorgestellt werden.

Die Modellierung des Kontrollflusses erfolgt in ADEPT graphbasiert. Dazu wird das Konzept der *regelmäßigen Blockstrukturierung* verwendet. D.h., Kontrollflüsse wie Sequenz oder Parallelität werden als Blöcke mit eindeutiger Start- und Endaktivität beschrieben. Diese Blöcke dürfen ineinander verschachtelt sein, sich jedoch nicht überlappen. Dies führt zu einer symmetrischen Graphstruktur, die einige Vorteile hat: So werden dadurch bestimmte dynamische Eigenschaften schon durch Konstruktion sichergestellt, Korrektheitsüberprüfungen können effizient durchgeführt werden. Die durch diese Blockstruktur beschränkte Ausdrucksmächtigkeit ist durch weitere Konstrukte (z.B. über Blockgrenzen hinweggehende Synchronisationskanten) erhöht worden.

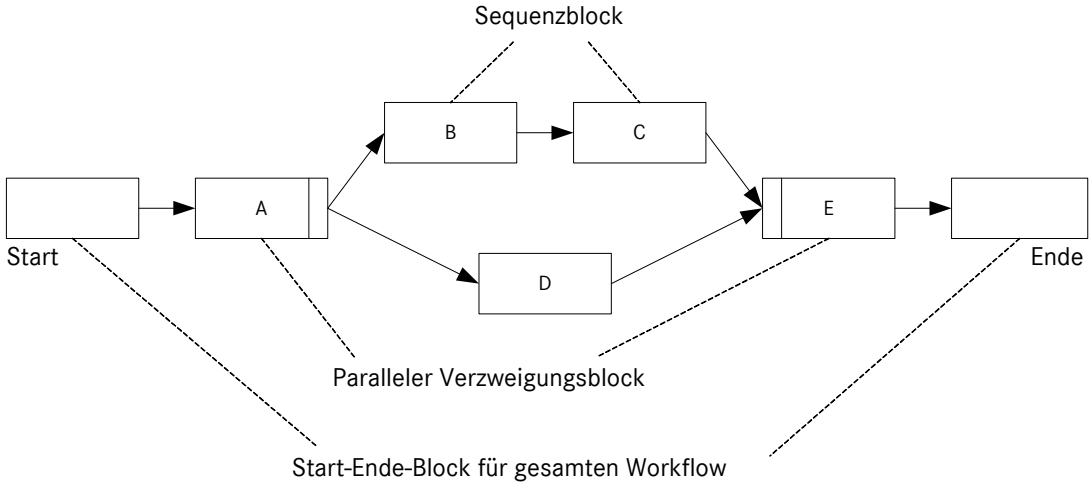

Abbildung 6.32: Kontrollflussgraph in ADEPT nach [Re00]

Datenflüsse werden in ADEPT über *globale Variablen* modelliert. Diese werden über Lese- bzw. Schreibkanten mit Parametern der Aktivitäten verbunden, mit der Bedeutung, dass zur Laufzeit die Aktivität mit dem Wert einer globalen Variablen aufgerufen wird bzw. nach Ausführung der Aktivität diese die globale Variable mit einem Wert belegt. Die Parameter von

Aktivitäten können über Eigenschaften konfiguriert werden, so kann beispielsweise festgelegt werden, ob der Parameter einer Aktivität *obligat* oder *optional* für deren Ausführung ist. Dies hat Auswirkungen darauf, wie auf den Wegfall einer schreibenden Aktivität einer Variablen reagiert werden muss. Für Datenflüsse existieren in ADEPT formale Regeln für Korrektheit und effiziente Verfahren, um Korrektheitsüberprüfung durchzuführen. So ist beispielsweise genau festgelegt, wann ein obligater Parameter einer Aktivität *sicher* mit Daten versorgt wird.

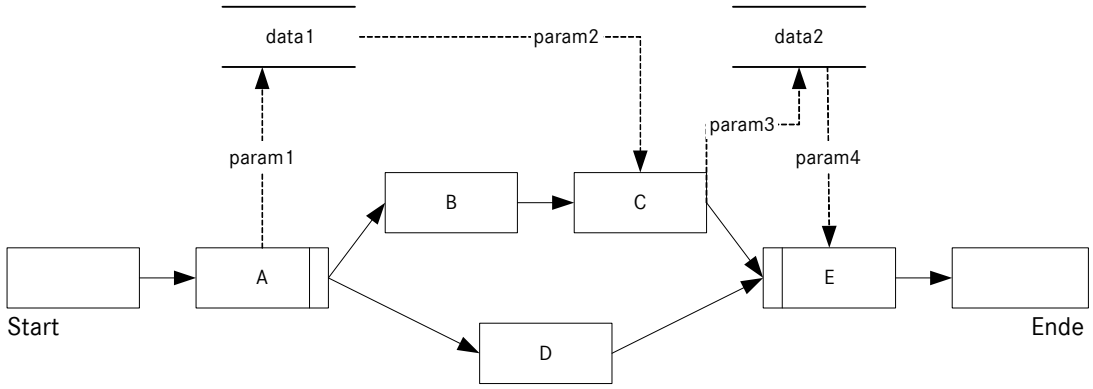

Abbildung 6.33: Datenflussmodellierung in ADEPT nach [Re00]

Zur eben vorgestellten Workflow-Modellierung kommt in ADEPT ein ebenfalls formal fundiertes Konzept zur Ausführung von Workflow-Instanzen. Der Zustand einer Workflow-Instanz wird dabei in einem markierten Graphen festgehalten. Markiert werden sowohl die Aktivitäten selbst als auch die Kanten zwischen Aktivitäten. Die Markierung einer Aktivität gibt den momentanen Ausführungszustand an, wie beispielsweise „aktiviert“, „laufend“ oder „beendet“. Für die Zustandsübergänge in einem markierten Workflowgraphen existieren formale Regeln.

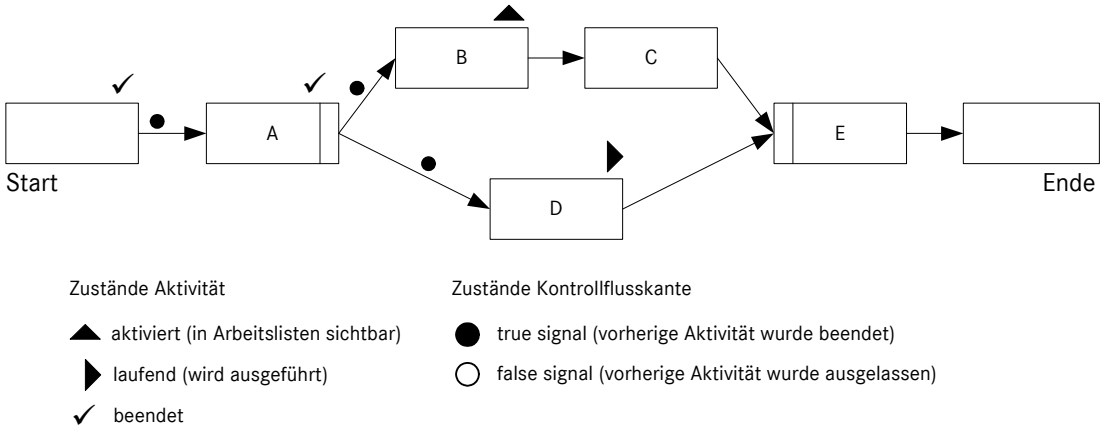

Abbildung 6.34: Markierter Ausführungsgraph in ADEPT nach [Re00]

Durch $ADEPT_{flex}$ wird nun eine Menge von Operationen bereitgestellt, mit denen dynamisch strukturelle Änderungen an Workflows vorgenommen werden können. Besonderes Augenmerk wurde darauf gelegt, dass durch diese Operationen die Korrektheit und Konsistenz erhalten bleibt: Nach Anwendung einer Änderungsoperation muss die Struktur des Workflow-Schema korrekt bleiben und die Instanz darf sich nicht in einem ungültigen Zustand befinden.

Durch Änderungsoperationen können in laufende Workflow-Instanzen Aktivitäten eingefügt und gelöscht, Aktivitäten und ganze Kontrollblöcke verschoben und Änderungen am Datenfluss vorgenommen werden. Logisch gesehen existiert dabei für jede Workflow-Instanz ein eigener Workflowgraph, der durch Graphtransformationen verändert wird. Für jede Operation ist exakt festgelegt, welche formalen Voraussetzungen für ihre Anwendung vorliegen müssen, wie sie durch Graphersetzungen umgesetzt werden und wie auf mögliche Probleme reagiert werden kann.

Das für diese Arbeit relevante dynamische Löschen von Aktivitäten aus Workflow-Instanzen wird in $ADEPT_{flex}$ durch eine Operation *deleteActivity* ermöglicht. Diese Operation kann nur auf „normale“ Aktivitäten angewendet werden, ausgeschlossen davon sind spezielle Knoten, wie beispielsweise der Anfangs- und Endknoten des Kontrollflussgraphen.

Handelt es sich bei der zu löschenden Aktivität X um einen Knoten, von dem *keine* Verzweigungen ausgehen oder einmünden, kann X einfach dadurch gelöscht werden, dass die Aktivität selbst sowie ihre ein- und ausgehenden Kontrollflusskanten aus dem Graphen entfernt und eine neue Kontrollflusskante zwischen Vorgänger- und Nachfolgeaktivität der zu löschenden Aktivität eingefügt werden. Falls X der einzige Knoten in einem Teilzweig einer parallelen Kontrollflussstruktur ist, wird diese neue Kontrollflusskante nicht eingefügt.

Ist X dagegen ein Knoten, bei dem sich der Kontrollfluss aufsplittet bzw. wieder zusammengeführt wird, kann X nicht einfach gelöscht werden. X wird stattdessen durch eine sog. *Nullaktivität* ersetzt. Eine Nullaktivität wird zur Laufzeit so ausgeführt, dass sie unmittelbar nach ihrer Aktivierung sofort wieder beendet wird.

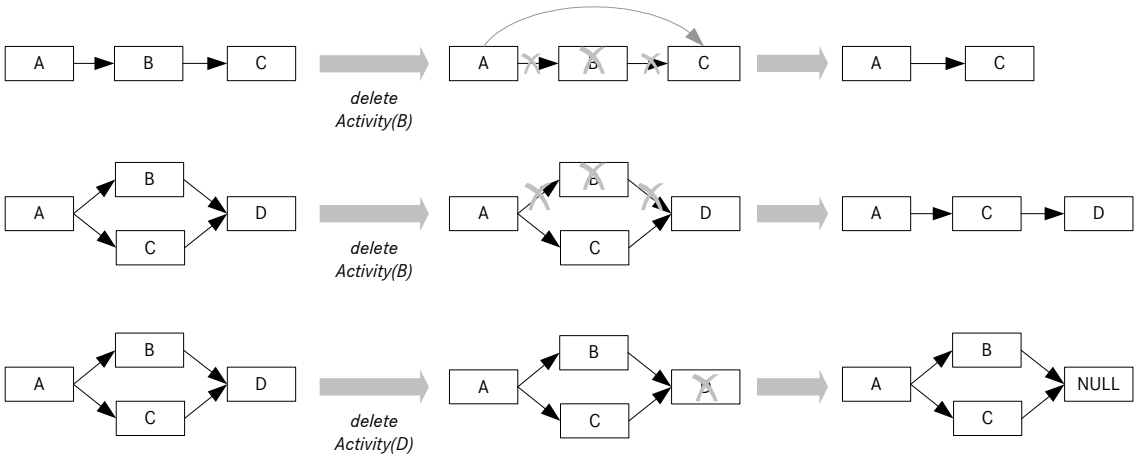

Abbildung 6.35: Löschen von Aktivitäten durch Graphersetzungen nach [Re00]

Damit die eben dargestellten Graphtransformationen nicht zu inkonsistenten Zuständen bei der Ausführung führen, muss vor der Durchführung der Zustand der betroffenen Workflow-Instanz überprüft werden. Für das Löschen einer Aktivität bedeutet dies, dass sie nur dann gelöscht werden kann, wenn sie sich noch nicht in Ausführung befindet. D.h. wird eine Aktivität im Moment gerade ausgeführt oder ist sie bereits beendet, darf sie nicht mehr gelöscht werden, da dies zu inkonsistenten Markierungszuständen führt. Inkonsistent bedeutet, dass die resultierende Markierung bei „normaler“ Ausführung des geänderten Workflows nicht erreicht werden kann.

Durch das Löschen von Aktivitäten werden auch deren Datenzugriffe entfernt. Dies kann für nachfolgende Aktivitäten bedeuten, dass obligate Eingabeparameter fehlen – mit der Folge,

dass das entsprechende Aktivitätenprogramm nicht korrekt ausgeführt wird. Deshalb bietet $ADEPT_{flex}$ den Nutzern Möglichkeiten an, um geeignet auf dieses Problem zu reagieren. Nach Bestimmung der sog. *datenabhängigen Aktivitäten* für die zu löschende Aktivität existieren in $ADEPT_{flex}$ folgende vier Optionen:

- Die datenabhängigen Aktivitäten werden ebenfalls gelöscht.
- Es werden neue Versorgeraktivitäten eingefügt, um die fehlenden Daten zu liefern.
- Nachforderungsdienste werden automatisch aktiviert, die dem Benutzer erlauben, die fehlenden Daten direkt einzugeben.
- Das Datenflussschema wird durch den Nutzer über Änderungsoperationen so lange geändert, bis es wieder korrekt ist.

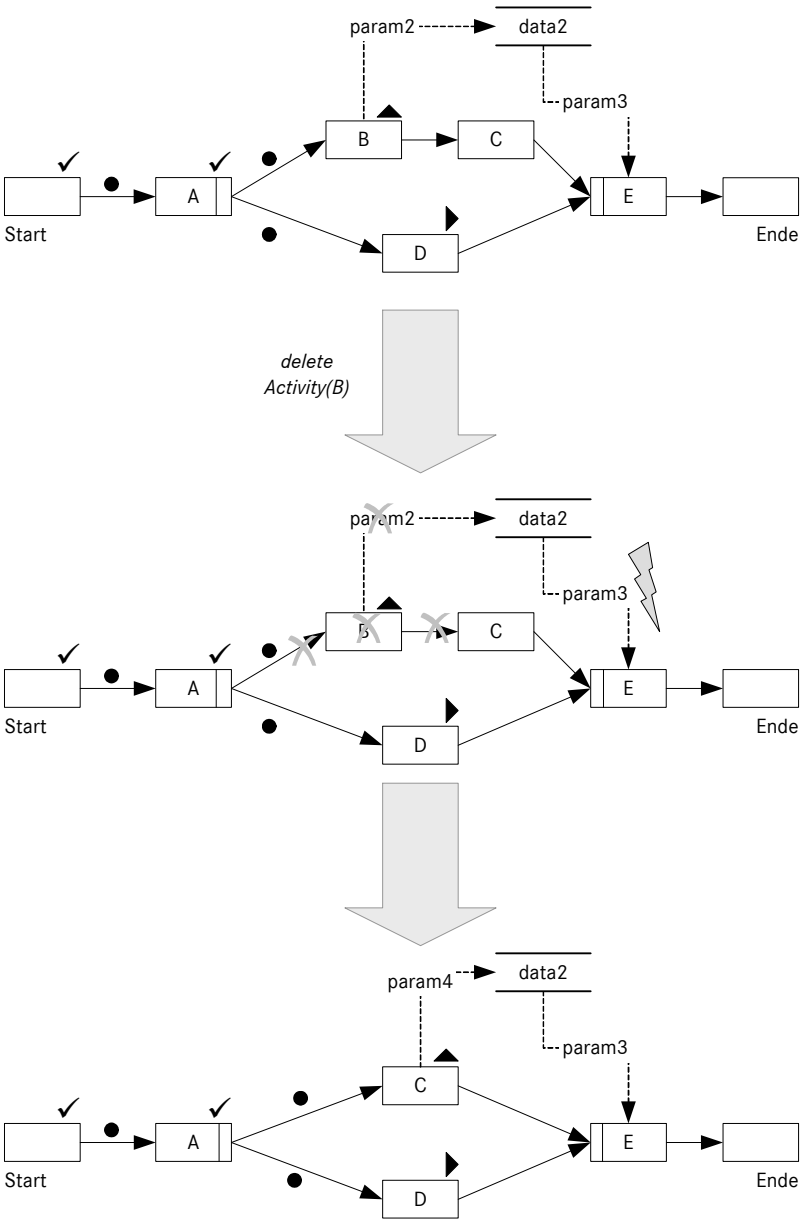

Abbildung 6.36: Beispiel: Löschen einer Aktivität aus einer Instanz mit Datenabhängigkeiten

Abbildung 6.36 zeigt das Löschen einer Aktivität aus einer laufenden Instanz an einem Beispiel. Die aktivierte Aktivität B soll gelöscht werden. Da B noch nicht ausgeführt wird, kann

sie gelöscht werden. Allerdings müssen vorher die entsprechenden Einträge in den Arbeitslisten entfernt werden. Durch Graphersetzungen werden B entfernt und die Zustände der Kontrollkanten angepasst. (Die Ersetzungen geschehen bei Instanzen in ADEPT nur logisch, praktisch wird auch hier an Stelle von B eine Nullaktivität eingefügt.). B schreibt die Datenvariable *data2*, die obligat von Aktivität E gelesen wird. Der Datenfluss muss deshalb angepasst werden. In diesem Fall liefert der Ausgabeparameter der Aktivität C ebenfalls das passende Eingabedatum für E. Somit kann durch eine Änderung des Datenflusses (genauer gesagt durch die Operation *InsertDataEdge*) die Datenflusskorrektheit wiederhergestellt werden.

Hier konnte lediglich ein grober Überblick über die Möglichkeiten von $ADEPT_{flex}$ bezüglich dynamischer Änderung laufender Workflow-Instanzen gegeben werden. Das für unsere Anforderungen benötigte Löschen von Aktivitäten kann durch die Konzepte von $ADEPT_{flex}$ umgesetzt werden. Auch wenn in unserem Fall meist keine Verknüpfung von Aktivitätsprogrammen stattfindet (vgl. Abschnitt 5.2.2), ist die Erhaltung der Datenflusskorrektheit eine wichtige Eigenschaft, die dann wieder für Freigabeworkflows wichtig wird, wenn dort in Zukunft Softwareapplikationen prozessorientiert verknüpft werden sollten.

6.4.3 Zusammenfassung

Das dynamische Ändern einzelner Workflow-Instanzen ist – zumindest was die Anforderungen der Freigabeprozesse angeht – durch konkrete wissenschaftliche Ansätze in einer konsistenzerhaltenden Weise möglich, wie das Beispiel $ADEPT_{flex}$ gezeigt hat.

Allerdings wurden dynamische Änderungen im Zusammenhang mit Workflowgeflechten weder in den diskutierten Ansätzen zur Workflowsynchronisation (vgl. Abschnitt 6.2) noch in den betrachteten Ansätzen im Kontext adaptiver Workflows in diesem Abschnitt berücksichtigt. In [He00] wird diese Problematik zwar erkannt, aber lediglich im Ausblick erwähnt. In diesem Kontext gibt es zwei unterschiedliche Problemstellungen zu betrachten: Zum einen müssen bei dynamischen Änderungen an einzelnen Workflow-Instanzen die Auswirkungen auf das Workflowgeflecht berücksichtigt werden. Zum anderen sind dies dynamische Änderungen an Inter-Workflowabhängigkeiten selbst, wie auch in Abschnitt 5.4.2 gefordert.

6.5 Abschließende Bewertung

Die Diskussionen dieses Kapitels haben gezeigt, dass die besonderen Anforderungen von Freigabeprozessen an WfMS nicht vollständig durch existierende Konzepte umgesetzt werden können. Bei Ansätzen zur *Synchronisation paralleler Workflows* fehlt es an einer Einbeziehung des Datenmodells. Eine *Behandlung von Fehlern*, wie sie in unserem Falle auftreten, wird nicht in Ansätzen zur klassischen Fehlerbehandlung diskutiert. Für dynamische Änderungen an Workflows existieren adäquate Ansätze, jedoch nur für isolierte Änderungen. *Dynamische Änderungen* an Workflowgeflechten spielen in der Wissenschaft bisher keine große Rolle. Somit müssen für die Workflow-Unterstützung von Freigabeprozessen neue Lösungskonzepte erarbeitet werden.

7 Lösungskonzepte

Im Rahmen einer Diplomarbeit können nicht alle angesprochenen offenen Fragen bearbeitet werden. Der Fokus dieser Arbeit liegt auf Fragestellungen zur Synchronisation paralleler Workflows. Für diese Problemstellung wurde im vorangegangenen Kapitel der Stand der Technik ausführlich dargestellt. Die Diskussionen haben gezeigt, dass die Ansätze für unsere Problemstellung nicht ausreichend sind. Konzepte für die Modellierung der Kontrollflussabhängigkeiten sollen in dieser Arbeit getrennt von Ausführungskonzepten betrachtet werden. In Abschnitt 7.1 werden ein Konzept zur Modellierung der Kontrollflussabhängigkeiten beschrieben und Alternativen diskutiert. Der anschließende Abschnitt 7.2 beschreibt mit dynamischen Interworkflow-Kontrollflusskanten ein Konzept für die Ausführungskontrolle dieser Kontrollflussabhängigkeiten zur Laufzeit. In Abschnitt 7.3 werden abschließend erste Konzeptideen zur Fehlerbehandlung vorgestellt.

7.1 Modellierung der Kontrollflussabhängigkeiten zwischen Workflows

Ein Konzept, das eine adäquate Modellierung der Kontrollflussabhängigkeiten zwischen parallelen Freigabeworkflows ermöglicht, wird im Folgenden schrittweise erarbeitet. Das gewählte Lösungskonzept wird zunächst in Abschnitt 7.1.1 vorgestellt. In Abschnitt 7.1.2 werden im Laufe der Arbeit entstandene Alternativen beschrieben und diskutiert.

Da sich die Abhängigkeiten immer auf Workflows bestimmter Konfigurationen beziehen, muss zuerst geklärt werden, was Konfigurationen aus Workflow-Management-Perspektive darstellen: Konfigurationen sollen als Daten eines Workflows betrachtet werden. Im Folgenden wird davon ausgegangen, dass das WfMS zu jedem Workflow die damit verknüpften Daten und deren Beziehungen zu anderen Daten kennt und entlang dieser Beziehungen traversieren kann.

7.1.1 Konzeptvorschlag für Modellierung der Abhängigkeiten

Eine der Kernanforderungen aus Abschnitt 5.4.1 war, die Kontrollflussabhängigkeiten zwischen Workflows in Abhängigkeit von den mit Workflows assoziierten Daten beschreiben zu können. Die Diskussionen in Abschnitt 6.2 haben gezeigt, dass es derzeit keinen Ansatz gibt, der die besonderen Abhängigkeiten in unserem Fall adäquat unterstützen kann.

Eine Grundidee des in dieser Arbeit entwickelten Lösungsansatzes besteht darin, die Workflows und ihre wechselseitigen Abhängigkeiten in drei Teilen zu beschreiben:

- Im *Typmodell* wird die hierarchische Struktur der Konfigurationen auf Typebene modelliert.
- Das *Workflow-Modell* enthält beliebig viele Workflow-Schemata, die bestimmte Typen von Freigabeworkflows beschreiben.
- Im *Abhängigkeitsmodell* werden Typmodell und Workflow-Modell miteinander verknüpft und die Kontrollflussabhängigkeiten zwischen den Workflows auf Schemaebene spezifiziert.

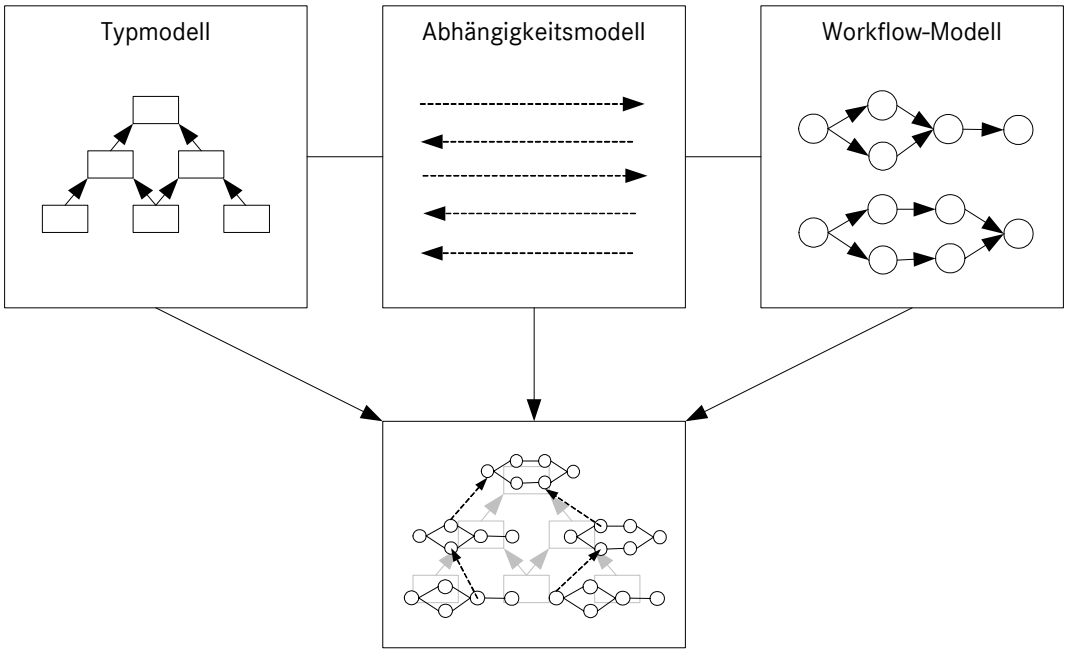

Abbildung 7.1: Zusammenspiel der Teilmodelle

7.1.1.1 Typmodell

Im Typmodell wird die Struktur der Konfigurationen auf Typebene beschrieben. Jede Konfiguration gehört zu *genau* einem *Konfigurationstyp* und ist zugleich eine Instanz dieses Typs. Ein Konfigurationstyp legt zudem fest, welche Konfigurationstypen die Teilkonfigurationen einer Konfiguration besitzen müssen.

Abbildung 7.2 zeigt ein mögliches Typmodell für das Beispiel aus Abschnitt 3.3. Für die Konfigurationen der Steuergeräte sowie der E/E-Systeme wurde jeweils ein Konfigurationstyp definiert. So ist beispielsweise *TSLTyp* ein Teilkonfigurationstyp der Typen *Außenlichttyp* und *SpiegelverstellungsTyp*.

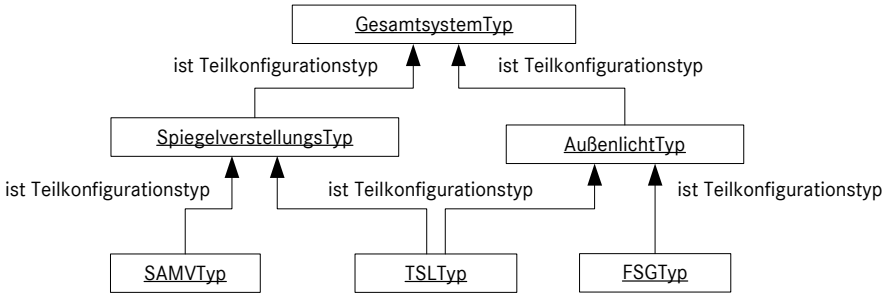

Abbildung 7.2: Beispiel für ein Typmodell

Eine Konfiguration ist bezüglich ihres Konfigurationstyps *gültig*, wenn sie für *jeden* Konfigurationstyp, der mit ihrem Konfigurationstyp über die Beziehung „ist Teilkonfigurationstyp“ assoziiert ist, *genau eine* Teilkonfiguration dieses Typs in der Beziehung „ist Teilkonfiguration“ besitzt. Nur für gültige Konfigurationen kann ein Freigabeworkflow durchgeführt werden. Das in Abbildung 7.3 dargestellte Klassendiagramm veranschaulicht das zu Grunde liegende Metamodell graphisch.

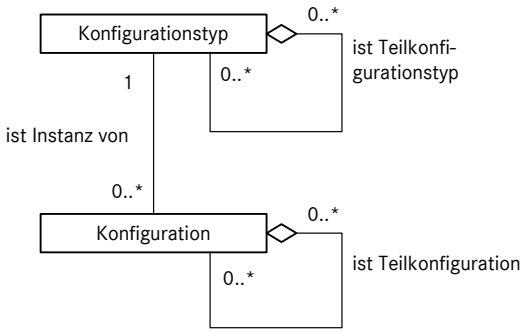

Abbildung 7.3: Klassendiagramm Konfigurationstypen und Konfigurationen

Abbildung 7.4 zeigt ein Geflecht von Konfigurationen, in dem jede Konfiguration gültig ist. Hierbei werden unterschiedliche Konfigurationsinstanzen durch fortlaufende Nummern voneinander unterschieden. Für den *AußenlichtTyp* wurden im Typmodell die Typen *TSLTyp* und *FSGTyp* als Teilkonfigurationstypen festgelegt. Im Beispiel in Abbildung 7.4 existieren zwei Instanzen dieses Typs. Diese sind gültig, da sie sich jeweils aus einer Instanz der Konfigurationstypen *TSLTyp* und *FSGTyp* zusammensetzen. Eine Besonderheit stellt die Konfiguration 3 dar, da diese in beiden Konfigurationsinstanzen des Außenlichttyps (Konfiguration 6 und 7) verwendet wird. Diese Besonderheit bei der Zusammensetzung von Konfigurationen wurde in Abschnitt 3.3.1 diskutiert.

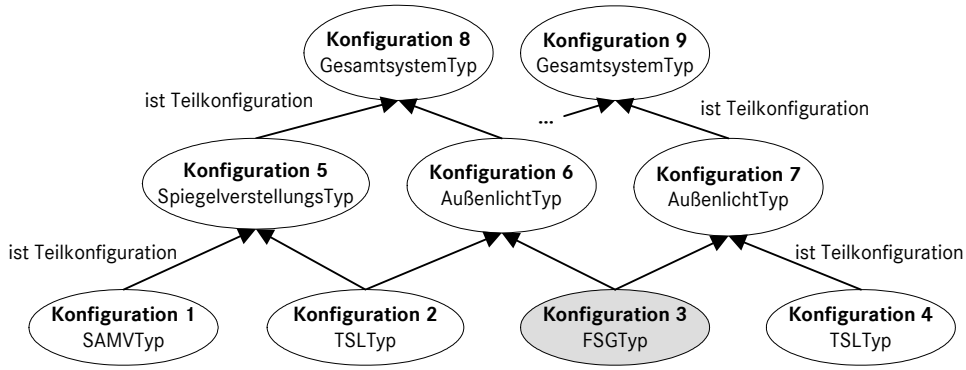

Abbildung 7.4: Beispiele für gültige Konfigurationen des Typmodells

7.1.1.2 Workflow-Modell

Das Workflow-Modell enthält alle Workflow-Schemata, die bestimmte Typen von Freigabeworkflows – unabhängig von Abhängigkeiten zwischen den Workflows – beschreiben. Die gewählte Workflow-Sprache spielt für das grundlegende Konzept der Abhängigkeitsbeschreibung keine Rolle. Es wird lediglich vorausgesetzt, dass Aktivitäten explizit repräsentiert werden.

In Abbildung 7.5 und Abbildung 7.6 sind die Workflow-Schemata der schon bekannten Beispielprozesse abgebildet, die den Kontrollfluss zwischen Prüfaktivitäten beschreiben.

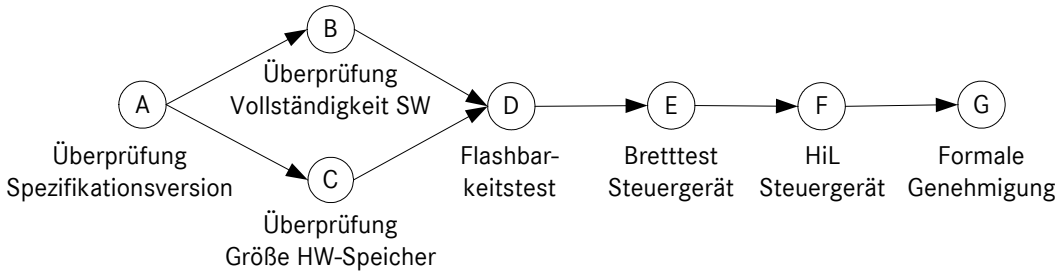

Abbildung 7.5: Workflow-Schema für ein Steuergerät (WFS 1)

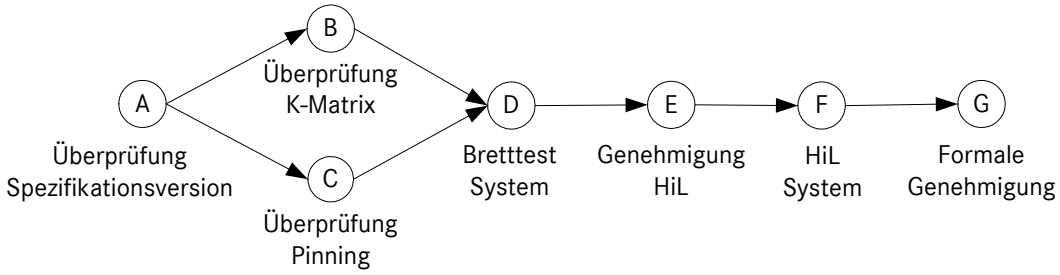

Abbildung 7.6: Workflow-Schema für ein E/E-System (WFS 2)

7.1.1.3 Abhängigkeitsmodell

Im Abhängigkeitsmodell werden das Typmodell und die Workflow-Schemata aus dem Workflow-Modell miteinander verknüpft. Dies geschieht auf zwei Ebenen:

- Im *Typbindungsmodell* wird beschrieben, welches Workflow-Schema für einen Konfigurationstyp des Typmodells verwendet werden darf.
- Im *Kontrollflussmodell* können – abhängig von der Bindung eines Konfigurationstyps an ein Workflow-Schema – Kontrollflussabhängigkeiten zwischen Aktivitäten der Workflows beschrieben werden.

Bevor die beiden Teilmodelle formal eingeführt werden, sollen diese mit einem Beispiel illustriert werden.

a) Beispiel

Im betrachteten Beispielfall soll für alle Konfigurationen auf Ebene der Steuergeräte das Workflow-Schema 1 und für alle Konfigurationen auf Systemebene das Workflow-Schema 2 den Freigabeprozess beschreiben. Das zugehörige Typmodell und die Workflow-Schemata wurden bereits vorgestellt.

Im Typbindungsmodell wird beschrieben, welche Workflow-Schemata für welche Konfigurationstypen verwendet werden sollen. In Tupelschreibweise kann die Typbindung als 3-Tupel *(Typbindungsname, Konfigurationstyp, Workflow-Schema)* ausgedrückt werden. Für das Beispiel ergeben sich folgende Typbindungen:

```
(BindungSAMV, SAMVTyp, WFSchema1)

(BindungTSL, TSLTyp, WFSchema1)

(BindungFSG, FSGTyp, WFSchema1)
```

```
(BindungSpiegelverstellung, SpiegelverstellungssystemTyp,
 WFSchema2)

(BindungAußenlicht, AußenlichtsystemTyp, WFSchema2)
```

Nun werden die Kontrollflussabhängigkeiten zwischen den Workflows beschrieben. Dies kann als 4-Tupel *(TypbindungsnameVon, AktivitätVon, TypbindungsnameNach, AktivitätNach)* ausgedrückt werden.

Der Schritt „Bretttest System“ (Aktivität D des Workflow-Schemas WFS 2) darf erst durchgeführt werden, wenn auf Komponentenebene in allen Teilkonfigurationen der Schritt „Flashbarkeitstest“ (Aktivität D des Workflow-Schemas WFS 1) erfolgreich durchgeführt wurde. Diese Abhängigkeit kann in Tupelschreibweise folgendermaßen ausgedrückt werden:

```
(BindungSAMV, D, BindungSpiegelverstellung, D)

(BindungTSL, D, BindungSpiegelverstellung, D)

(BindungTSL, D, BindungAußenlicht, D)

(BindungFSG, D, BindungAußenlicht, D)
```

Diese Abhängigkeiten können in diesem Beispiel auch einfacher dadurch ausgedrückt werden, dass man eine Festlegung trifft der Art „Schritt D in Workflow-Schema 1 soll immer vor Schritt D in Workflow-Schema 2“ ausgeführt werden. Mit dieser Vorgehensweise wäre es jedoch nicht möglich, Kontrollflussabhängigkeiten zu beschreiben, die vom speziellen Konfigurationstyp abhängen. So könnte es beispielsweise sein, dass die Flashbarkeitsprüfung des Türsteuergeräts *nicht* Voraussetzung für den Bretttest des Außenlichtsystems ist. Mit dem vorgestellten Ansatz kann dies einfach dadurch erreicht werden, dass die Abhängigkeit *(BindungTSL, D, BindungAußenlicht, D)* weggelassen wird.

Wenn man die eben dargestellte Ausnahme mitbetrachtet und die Abhängigkeiten zwischen den Schritten „HiL Steuergerät“ (Aktivität F in WFS2) und „HiL System“ (Aktivität F in WFS2) hinzunimmt, ergibt sich bildlich betrachtet das in Abbildung 7.7 dargestellte Abhängigkeitsmodell.

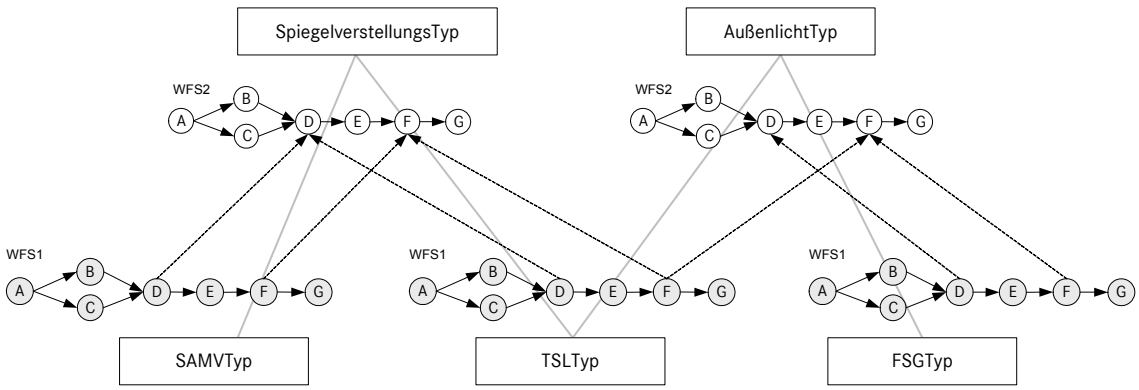

Abbildung 7.7: Beispiel für Abhängigkeitsmodell

b) Typbindungsmodell

Das Typbindungsmodell beschreibt für jeden Konfigurationstyp im Typmodell das passende Workflow-Schema im Workflow-Modell. Diese Verknüpfung wird als *Typbindung* bezeichnet.

Einem Konfigurationstyp ist genau ein Workflow-Schema zugeordnet, so dass beim Start des Freigabeworkflows für eine Konfiguration klar ist, welches Schema dazu instanziiert werden muss. Ein Workflow-Schema kann beliebig vielen Konfigurationstypen zugeordnet werden. Abbildung 7.8 illustriert diesen Sachverhalt als Klassendiagramm.

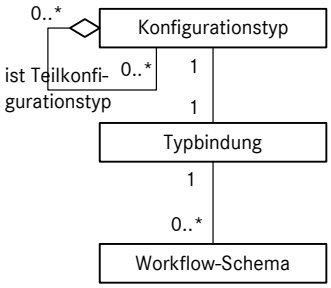

Abbildung 7.8: Klassendiagramm des Typbindungsmodell

c) Kontrollflussmodell

Im Kontrollflussmodell werden die Kontrollflussabhängigkeiten zwischen Freigabeworkflows definiert. Diese Abhängigkeiten werden jedoch *nicht* direkt zwischen den Aktivitäten der Workflow-Schemata definiert, sondern zwischen konkreten Typbindungen eines Workflow-Schemas. Für jeweils zwei Typbindungen kann eine *sequenzielle* Kontrollflussabhängigkeit zwischen Aktivitäten der verknüpften Workflow-Schemata angegeben werden.

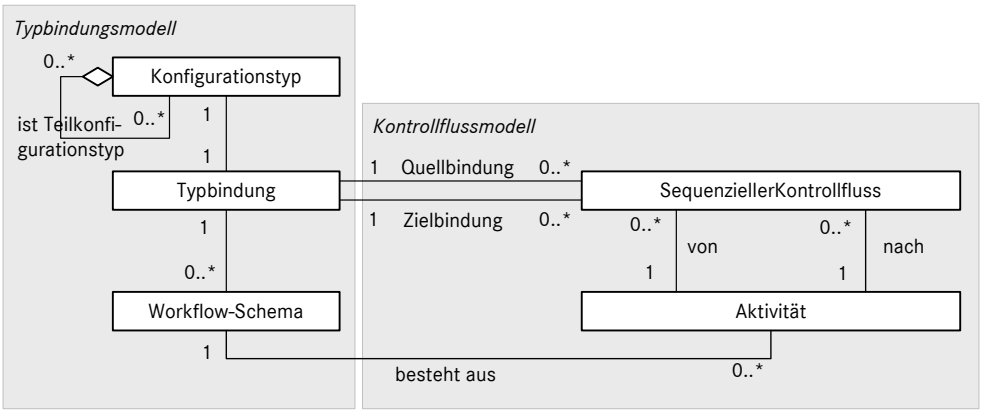

Abbildung 7.9: Klassendiagramm des gesamten Abhängigkeitsmodells

Abbildung 7.9 zeigt das Metamodell in einem Klassendiagramm, die Konsistenzbedingungen dieses Modells werden in Abschnitt d) dieses Unterkapitels definiert.

Definition des sequenziellen Kontrollflusses

Präzise beschrieben bedeutet der sequenzielle Kontrollfluss das Folgende (siehe auch Abbildung 7.10): Seien *KonfTyp1* und *KonfTyp2* Konfigurationstypen und *KonfTyp1* ein Teilkonfigurationstyp von *KonfTyp2*. Seien ferner die Workflow-Schemata *WFS1* mit *KonfTyp1* und *WFS2* mit *KonfTyp2* über die Typbindungen *Bindung1* bzw. *Bindung2* verknüpft. Sei *A* eine Aktivität in *WFS1* und *B* eine Aktivität in WFS2. Es sei dann außerdem eine sequenzielle

Kontrollflussabhängigkeit als 4-Tupel *(Bindung1, A, Bindung2, B)* definiert. Dies bedeutet, dass im Freigabeworkflow des Workflow-Schemas *WFS2* einer Konfigurationen *Konf2* von *KonfTyp2* die Ausführung der Aktivität B erst begonnen werden darf, wenn im Freigabeworkflow des Workflow-Schemas *WFS1* für die Teilkonfiguration des *KonfTyps1* von *Konf2* die Aktivität A bereits beendet wurde.

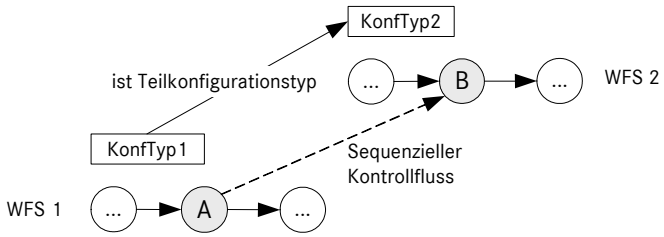

Abbildung 7.10: Sequenzieller Kontrollfluss

Bildlich kann man sich das gesamte Abhängigkeitsmodell so vorstellen, dass für jeden Konfigurationstyp eine logische Kopie eines Workflow-Schemas erzeugt wird und dann zwischen den einzelnen Ebenen spezielle Kontrollflusskanten zwischen Aktivitäten der Workflow-Schemata die Abhängigkeiten ausdrücken. Abbildung 7.7 (S. 81) gibt ein Beispiel hierfür.

d) Konsistenzbedingungen des Abhängigkeitsmodells

Für das Abhängigkeitsmodell müssen zur Modellierungszeit die folgenden Konsistenzbedingungen eingehalten werden. (Der Modellierer sollte dabei natürlich so weit wie möglich durch ein Modellierungswerkzeug unterstützt werden.)

Referenzierung von Aktivitäten

In der Spezifikation des sequenziellen Kontrollflusses zwischen Workflows dürfen (trivialerweise) nur Aktivitäten referenziert werden, die in den Workflow-Schemata der Typbindung tatsächlich enthalten sind. Ferner sollte vermieden werden, dass sich die Aktivität, von der ein sequenzieller Kontrollfluss ausgeht, in einem Pfad des Workflows befindet, der unter Umständen gar nicht ausgeführt wird (XOR-Verzweigung, siehe Abbildung 7.11). Denn sollte dieser Pfad nicht zur Ausführung kommen, würde dies den abhängigen Workflow blockieren, da dieser auf die Beendigung einer Aktivität wartet, die nie mehr ausgeführt wird.

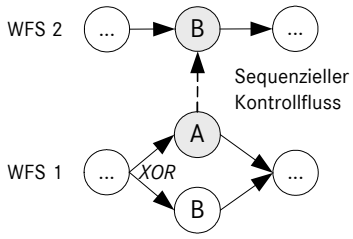

Abbildung 7.11: Referenzierung einer Aktivität eines Verzweigungspfades

Keine Interworkflow-Abhängigkeiten

Sequenzieller Kontrollfluss darf nur zwischen *Workflow-Schemata* unterschiedlicher Bindungen spezifiziert werden. Im anderen Fall würde es sich um (zusätzliche) Kontrollflussabhängigkeiten *in* einem Workflow handeln. Kontrollflussabhängigkeiten zwischen Aktivitäten eines Workflows können jedoch schon in den Workflow-Schemata selbst beschrieben werden

und sollten zur klaren Trennung von Verantwortlichkeiten nicht durch einen weiteren Mechanismus an anderer Stelle spezifiziert werden.

„Spannweite“ des Kontrollflusses

Der sequenzielle Kontrollfluss darf nur zwischen Typbindungen definiert werden, deren Konfigurationstypen über die Beziehung „ist Teilkonfigurationstyp“ *direkt* verknüpft sind. Prinzipiell wäre es auch denkbar, dies *transitiv* zuzulassen. Aus Gründen der Lesbarkeit und Verständlichkeit der Modelle soll dies jedoch nicht möglich sein.

Richtung des Kontrollflusses

Bisher wurde über die Richtung des sequenziellen Kontrollflusses zwischen Workflows nicht *explizit* gesprochen. In der obigen Definition wurde jedoch implizit festgelegt, dass dieser nur „von unten nach oben“ in der Konfigurationstyphierarchie definiert werden kann. Als Beispiele wurden bisher ebenfalls nur Abhängigkeiten diskutiert, bei denen Workflows auf einer unteren Hierarchieebene eine bestimmte Aktivität beendet haben müssen, bevor auf oberer Ebene mit dem Workflow fortgeschritten werden kann. Nun stellt sich die Frage, ob man – um die Ausdrucksmächtigkeit zu erhöhen – auch Kontrollflussabhängigkeiten von „oben nach unten“ zulassen kann und soll (Abbildung 7.12). Dies führt jedoch schon in der Semantik dieser Abhängigkeit zu Problemen: Wie mehrfach angesprochen, kann eine Konfiguration im Laufe der Zeit Teilkonfiguration mehrerer Konfigurationen sein. Dadurch ist nicht klar, auf welche Freigabeworkflows sich die spezifizierten Abhängigkeiten zur Laufzeit beziehen bzw. es ist nicht sichergestellt, dass es zum Ausführungszeitpunkt des Freigabeworkflows überhaupt schon eine „Oberkonfiguration“ gibt.

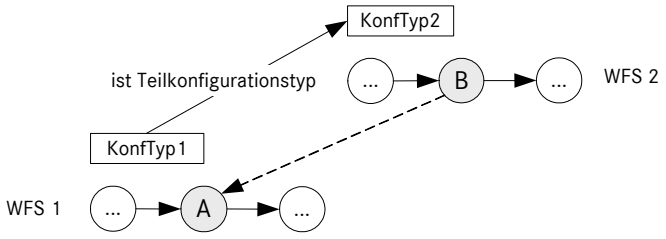

Abbildung 7.12: Möglicher Kontrollfluss in anderer Richtung

Selbst wenn man z.B. festlegt, auf welche der möglichen Oberkonfigurationen sich die sequenzielle Abhängigkeit bezieht, können sich dadurch Probleme bezüglich der Verklemmungsfreiheit ergeben. Grund ist, dass sich durch Kontrollflussabhängigkeiten in beide Richtungen Zyklen zwischen Workflows ausbilden können. Die Spezifikation in Abbildung 7.13 hätte zur Folge, dass eine Instanz des Schemas WFS1 vor Ausführung der Aktivität A blockiert wäre: Voraussetzung für die Ausführung von A ist die Beendigung von B. B kann jedoch erst ausgeführt werden, wenn C ausgeführt wurde. C liegt im Kontrollfluss des WFS1 jedoch hinter A und kann so erst nach Beendigung von A ausgeführt werden. Eine Verklemmungssituation ist erreicht und auch die entsprechende Instanz von WFS2 wäre blockiert. Solche Verklemmungen müssten mit technischem Aufwand verhindert werden.

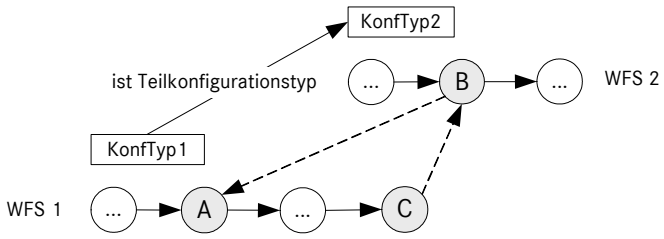

Abbildung 7.13: Mögliche Zyklen bei Kontrollflussabhängigkeiten in beide Richtungen

Aufgrund der dargestellten Probleme sollen ausschließlich Kontrollflussbeziehungen in Richtung der „ist Teilkonfigurationstyp“-Beziehung im Abhängigkeitsmodell erlaubt sein. Da keine praktisch relevanten Beispiele bekannt sind, in denen Kontrollflussabhängigkeiten in beide Richtungen benötigt werden, stellt dies keine signifikante Einschränkung dar.

Durch diese Konsistenzbedingung ist es nicht möglich, mit sequenziellen Kontrollflussabhängigkeiten Verklemmungssituationen zwischen Workflows zu verursachen. Anstelle eines formalen Beweises soll diese Aussage durch folgende Betrachtungen begründet werden:

Um eine Verklemmung zu erreichen, muss sich im Graph ein Zyklus ausbilden. Bei einem Zyklus handelt es sich um eine geschlossene Pfeilfolge, d.h. eine zirkuläre Aneinanderreihung von Pfeilen. Durch die Einschränkung, dass Pfeile zwischen Workflow-Schemata nur in eine Richtung verlaufen dürfen, kann niemals ein Zyklus geschlossen werden.

7.1.2 Diskussion von Alternativen und Entwurfsentscheidungen

Im Laufe der Arbeit sind Alternativen zu dem im vorangegangenen Abschnitt vorgestellten Konzept entstanden, die jedoch wieder verworfen wurden. Diese sollen nun kurz diskutiert werden.

7.1.2.1 Direkte Spezifikation von Abhängigkeiten

Eine Alternative zum vorgeschlagenen Abhängigkeitsmodell besteht darin, Abhängigkeiten nicht über den Umweg der Bindung eines Workflow-Schemas an einen bestimmten Konfigurationstyp zu beschreiben, sondern diese direkt zwischen den Workflow-Schemata festzulegen. Beispielsweise in der Art: „die Aktivität B einer Instanz des Workflow-Schemas 1 soll vor Aktivität A einer Instanz des Workflow-Schemas 2 ausgeführt werden, wenn die Instanzen entsprechenden, in hierarchischen Abhängigkeiten stehenden Konfigurationen zugeordnet sind.“

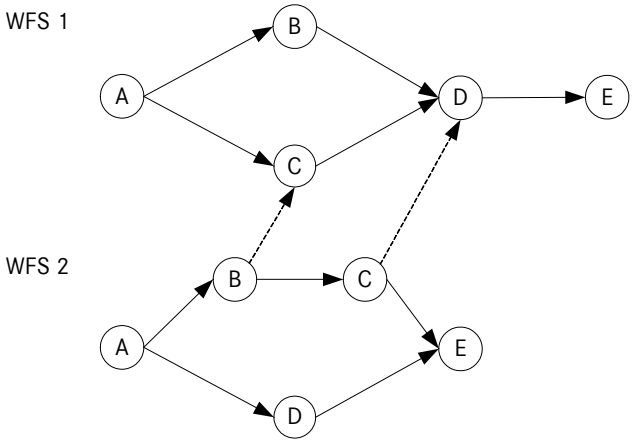

Abbildung 7.14: Direkte Spezifikation von Kontrollflussabhängigkeiten

Vorteil dieser Alternative ist eine kompaktere Spezifikation, da nicht für jeden Konfigurationstyp die Abhängigkeiten einzeln angegeben werden müssen. Allerdings hätte dieses Vorgehen zur Folge, dass Kontrollflussabhängigkeiten, die vom Typ der Konfiguration abhängen, nicht ausgedrückt werden können und für solche Fälle (unnötigerweise) dasselbe Workflow-Schemata repliziert werden müsste. Auch die Verwendung eines Workflow-Schemas wäre eingeschränkt, da durch die Richtung der Abhängigkeiten implizit festgelegt wäre, für welche Hierarchieebenen (relativ gesehen) die Workflow-Schemata verwendet werden dürfen. Insgesamt würde dies dazu führen, dass die Abhängigkeiten im Workflow-Geflecht für den Modellierer weniger durchschaubar und damit schlechter wartbar sind: Zur Laufzeit existieren die Kontrollflussabhängigkeiten zwischen Freigabeworkflows hierarchisch abhängiger Konfigurationen. Durch die Modellierung des Interworkflow-Kontrollflusses in Abhängigkeit von der hierarchischen Struktur wird dies sehr viel anschaulicher, als wenn diese Abhängigkeiten ohne Bezug zur Hierarchie der Konfigurationen geschehen würde. Aus diesen Gründen wurde die Entscheidung zu Gunsten des Abhängigkeitsmodell in Abschnitt 7.1.1.3 getroffen.

7.1.2.2 Modellierung durch Reifegrade

Eine Alternative zum gesamten Lösungskonzept ist die Modellierung der Abhängigkeiten über sog. „Reifegrade“. Diesem alternativen Ansatz liegt die Auffassung zu Grunde, dass der Reifegrad einer Konfiguration höher ist, je weiter der zugehörige Freigabeworkflow fortgeschritten ist. Dies gründet darauf, dass in den vorangegangenen Prüfschritten keine Fehler gefunden wurden. Der Ansatz ist der, diese Reifegrade explizit zu repräsentieren und parallele Freigabeworkflows über die Reifegrade zu synchronisieren. Die Abhängigkeiten zwischen den Freigabeworkflows auf verschiedenen Ebenen werden hier nicht mehr direkt zwischen einzelnen Aktivitäten festgelegt.

Für die Modellierung existieren bei diesem Ansatz zwei Modelle: Im *Reifegradmodell* wird eine globale Ordnung von Reifegraden festgelegt. Im *Workflow-Modell* werden in Workflow-Schemata Typen von Freigabeworkflows definiert. Die Reifegrade und Abhängigkeiten von Reifegraden anderer Workflows werden direkt in den Workflow-Schemata durch neue Modellierungskonstrukte beschrieben. Mit dem Konstrukt des *selbst erreichten Reifegrades* wird im Kontrollfluss des Workflow-Schemas festgelegt, wann ein gewisser Reifegrad erreicht ist. Das Konstrukt des *mindestens zu erreichenden Reifegrades* definiert, welchen Reifegrad *alle Teilkonfigurationen* mindestens erreicht haben müssen, damit mit der Ausführung an dieser Stelle fortgesetzt werden kann. Die Synchronisation der Workflows geschieht somit implizit über globale Reifegrade.

Abbildung 7.15 gibt ein einfaches Beispiel für diesen Ansatz. Im Reifegradmodell sind Reifegrade 1–3 definiert, wobei 3 die höchste Reife ausdrückt. Die Aktivität A darf erst ausgeführt werden, wenn alle Teilkonfigurationen mindestens den Reifegrad 1 besitzen. Nach Ausführung von Aktivität A besitzt die zugehörige Konfiguration ebenfalls den Reifegrad 1. Entsprechendes gilt für Aktivität C und den Reifegrad 2. Anzumerken ist, dass die Freigabeworkflows der Teilkonfigurationen Instanzen beliebiger (auch desselben) Workflow-Schemas sein können.

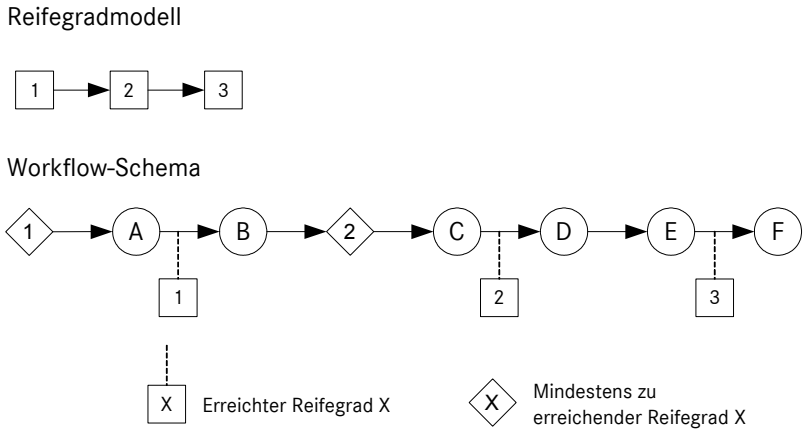

Abbildung 7.15: Workflow-Schema mit Reifegraden

Der Vorteil dieses Ansatzes liegt darin, dass das Modell (zumindest optisch) übersichtlicher ist als im vorgeschlagenen Lösungskonzept. Allerdings stehen diesem Vorteil eine Reihe von Nachteilen gegenüber. Die implizite Beschreibung der Abhängigkeiten führt zu schwerer verständlichen Modellen. Damit ist die Gefahr einer fehlerhaften Modellierung gegeben. Die Beschreibung von Kontrollflussabhängigkeiten, die vom Typ der assoziierten Konfiguration abhängen, ist ebenfalls nicht möglich. Ein weiteres Problem betrifft die Semantik der Reifegrade: Es ist für alle Beteiligten vermutlich sehr schwer, zu einem gemeinsamen Verständnis der Reifegrade zu kommen. Dies ist jedoch Grundvoraussetzung für diesen Ansatz. Ein weiterer Nachteil aus technischer Sicht ist die nötige Erweiterung der Workflow-Sprache, die substanzielle Änderungen an der Workflow-Engine notwendig machen.

7.1.3 Zusammenfassung

Der hier vorgestellte Ansatz zur Beschreibung von Kontrollflussabhängigkeiten zwischen Freigabeworkflows ist ein erster grundlegender Schritt, um Freigabeprozesse durch Workflowtechnologie unterstützen zu können. Es handelt sich hierbei um einen *expliziten* Ansatz, mit dem Kontrollflussabhängigkeiten zwischen Workflows separat und unabhängig von einer konkreten Workflow-Sprache beschrieben werden können.

Das Konzept ist sehr spezifisch auf den Problembereich zugeschnitten. Insbesondere das Metamodell des Typmodells ist speziell für Konfigurationen ausgelegt. Ein nächster Schritt in Richtung eines generischeren Konzepts wäre, die Struktur, die dort gebildet wird, auf ein beliebiges Datenmodell zu verallgemeinern. Damit wäre der Anwendungsbereich nicht mehr auf Konfigurationen eingeschränkt, sondern könnte auf eine Reihe von Anwendungen ausgedehnt werden, in denen es Kontrollflussabhängigkeiten zwischen Workflows gibt, die von einer hierarchischen Datenstruktur abhängig sind.

7.2 Ausführung der spezifizierten Kontrollflussabhängigkeiten

Die Kontrollflussabhängigkeiten zwischen Workflows, die im Wesentlichen durch das Abhängigkeitsmodell beschrieben sind, müssen zur Laufzeit berücksichtigt werden. Zu diesem Zweck wird in dieser Arbeit das Konzept gerichteter *Interworkflow-Kontrollflusskanten (IWK)* vorgeschlagen, die dynamisch zwischen Aktivitäten unterschiedlicher Workflow-Instanzen eingefügt werden. Dieser Ansatz ist prinzipiell unabhängig von einer konkreten Workflow-Sprache und deren Ausführungssemantik. Gefordert wird, dass jede Workflow-Instanz logisch einzeln repräsentiert wird und dass Aktivitäten als Knoten von Graphen betrachtet werden können. Außerdem wird vorausgesetzt, dass Aktivitäten und Workflow-Instanzen bestimmte Zustände in ihrer Ausführung annehmen. Zusätzlich zur „normalen“ Ausführung einer Workflow-Instanz muss vor der Aktivierung von Aktivitäten die Semantik der IWKs berücksichtigt werden, die in Abschnitt 7.2.1 festgelegt wird. Das Vorgehen des Einfügens dieser Kanten beim Start eines Freigabeworkflows ist in Abschnitt 7.2.2 dargestellt.

7.2.1 Semantik der Interworkflow-Kontrollflusskanten

Obwohl das Konzept der Interworkflow-Kontrollflusskanten zuerst einmal von einer Workflow-Sprache und deren konkretem Ausführungsmodell unabhängig ist, ist es für die Semantik der Kanten notwendig, ein logisches Zustandsmodell für Aktivitäten und Workflow-Instanzen vorauszusetzen. In diesem Fall setzen wir das einfache Zustandsmodell aus Abschnitt 4.4 voraus.

Eine IWK verbindet jeweils zwei Aktivitäten unterschiedlicher Workflow-Instanzen. Sei Aktivität A die Ausgangsaktivität der IWK und Aktivität B deren Zielaktivität. Eine IWK besitzt abhängig vom Zustand der Aktivität A einen dieser drei Signalisierungszustände:

- *kein Signal:* A befindet sich noch nicht in einem Endzustand.
- *falsch:* A wurde nicht beendet, d.h. A befindet sich in einem der Endzustände „nicht ausgeführt“ oder „abgebrochen“.
- *wahr:* A wurde beendet, d.h. A befindet sich im Endzustand „beendet“.

Aktivität B kann genau dann aktiviert werden, wenn sie in der normalen Ausführung einer Workflow-Instanz aktiviert werden kann *und* wenn alle eingehenden IWKs den Signalisierungszustand „wahr“ besitzen (Abbildung 7.16).

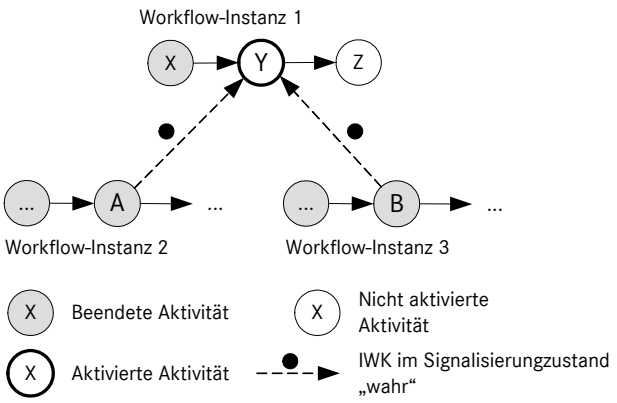

Abbildung 7.16: Beispiel für Aktivierung einer Aktivität durch IWK

Sollte eine der eingehenden IWKs den Signalisierungszustand „kein Signal“ haben, kann die Aktivität noch nicht aktiviert werden. Wenn eine eingehende IWK den Signalisierungszustand „falsch“ besitzt, bedeutet dies zunächst einmal, dass die Aktivität, von der die IWK ausgeht, nicht regulär beendet wurde. Damit darf die Zielaktivität eigentlich nicht mehr ausgeführt werden und blockiert damit u.U. viele transitiv abhängige Workflows. Diese Verklemmungssituation kann nur durch Aufgeben der mit IWKs verbundenen Restriktionen, d.h. dem Löschen der IWK, aufgehoben werden.

7.2.2 Dynamisches Einfügen von Interworkflow-Kontrollflusskanten

Das Konzept der IWKs ist zunächst einmal unabhängig von dem Modell, mit dem die Abhängigkeiten beschrieben werden. Hier soll nun für die Abhängigkeiten zwischen Freigabeworkflows dargestellt werden, wie durch dynamisches Einfügen von IWKs beim Start eines Workflows die im Abhängigkeitsmodell von Freigabeworkflows beschriebenen Kontrollflussabhängigkeiten umgesetzt werden können (vgl. Abbildung 7.18). Dabei soll davon ausgegangen werden, dass die Konfigurationen der Freigabeworkflows bezüglich ihres Konfigurationstyps gültig sind.

1. Starten von Workflows für Teilkonfigurationen

Ein Freigabeworkflow wird immer für eine bestimmte Konfiguration gestartet. Durch deren Konfigurationstyp ist dem WfMS bekannt, ob und welche Teilkonfigurationen diese Konfiguration besitzen muss. Sind für diese Konfiguration keine Teilkonfigurationen im Typmodell festgelegt, wird dasjenige Workflow-Schema instanziiert und gestartet, welches im Typbindungsmodell für diesen Konfigurationstyp angegeben wurde. Damit ist für diesen Fall der Start des Workflows beendet, ohne dass Kontrollflusskanten zwischen den Workflows eingefügt werden müssen. Sind weitere Teilkonfigurationen definiert, so wird zwar das Workflow-Schema instanziiert, jedoch noch nicht ausgeführt.

Für *jede* Teilkonfiguration der Konfiguration werden nun die folgenden Schritte durchgeführt: Wurde der Freigabeworkflow der Teilkonfiguration noch nicht gestartet, so ist für diese Teilkonfiguration rekursiv der hier beschriebene Startvorgang *vollständig* durchzuführen, bevor mit der Bearbeitung fortgefahren werden kann. Der Start der Freigabeworkflows für Teilkonfigurationen sollte allerdings nicht automatisch, sondern erst nach Bestätigung durch den Prozessverantwortlichen für diese Teilkonfiguration erfolgen.

2. Einfügen von Interworkflow-Kontrollflusskanten

Nach dem ersten Schritt ist sichergestellt, dass für jede Teilkonfiguration eine Instanzrepräsentation des zugehörigen Freigabeworkflows existiert. Nun kann mit dem Einfügen der IWKs begonnen werden. Jede Teilkonfiguration und die zugehörige Workflow-Instanz wird dabei einzeln abgearbeitet.

Im Abhängigkeitsmodell ist unter Bezugnahme auf die Konfigurationstypen beschrieben, welche sequenziellen Kontrollflussabhängigkeiten zwischen Freigabeworkflows existieren. Diese müssen nun auf die Instanzebene übertragen und durch entsprechende IWKs umgesetzt werden. Sowohl für die Konfiguration als auch für die jeweils betrachtete Teilkonfiguration ist dem WfMS der Konfigurationstyp bekannt. Für jeden sequenziellen Kontrollfluss im Abhängigkeitsmodell werden nun zwischen den Workflow-Instanzen entsprechende IWKs eingefügt. Der Signalisierungszustand der IWKs wird nach dem Einfügen auf „kein Signal“ gesetzt. Ist die Aktivität, von der die IWK ausgehen soll, schon in einem Endzustand, soll

diese automatisch den Signalisierungszustand entsprechend Abschnitt 7.2.1 bekommen. Abbildung 7.17 gibt ein Beispiel hierfür. Dieses zeigt darüber hinaus, dass zusätzlich auch schon Kontrollflussabhängigkeiten zu anderen Workflow-Instanzen bestehen können.

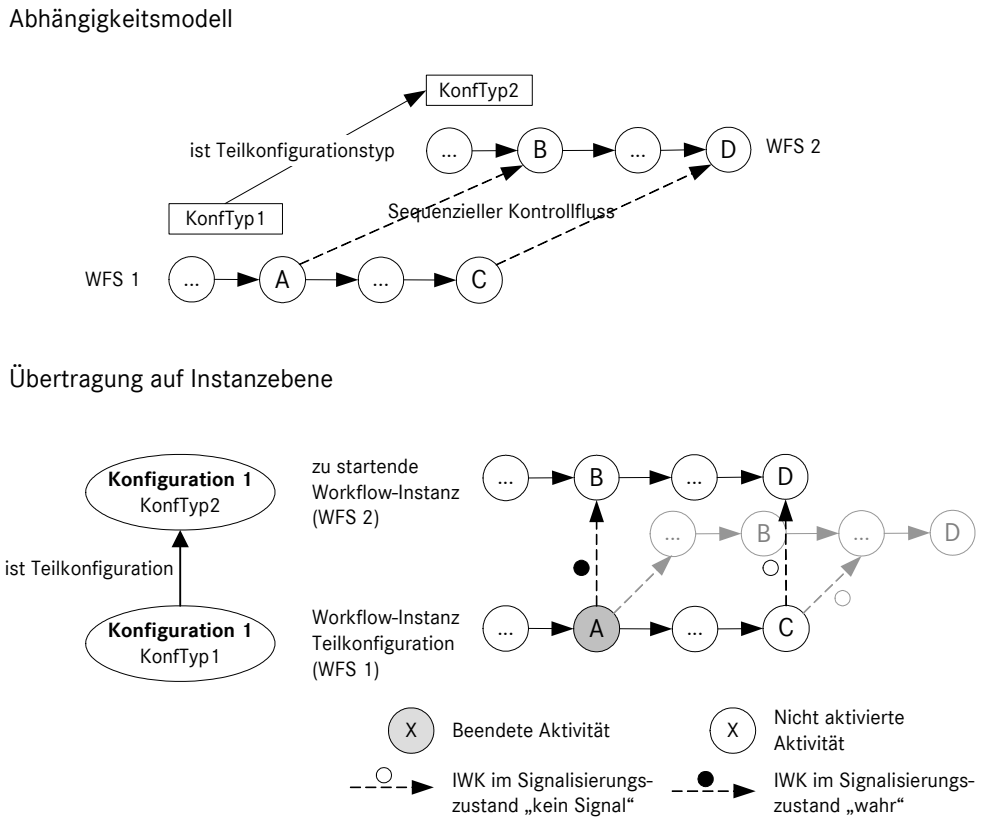

Abbildung 7.17: Umsetzung der Kontrollflussabhängigkeiten durch IWKs

3. Überführung in normale Ausführung

Nach dem Einfügen von IWKs für alle Teilkonfigurationen kann die Workflow-Instanz in den Ausführungszustand „laufend“ überführt und mit der eigentlichen Ausführung begonnen werden.

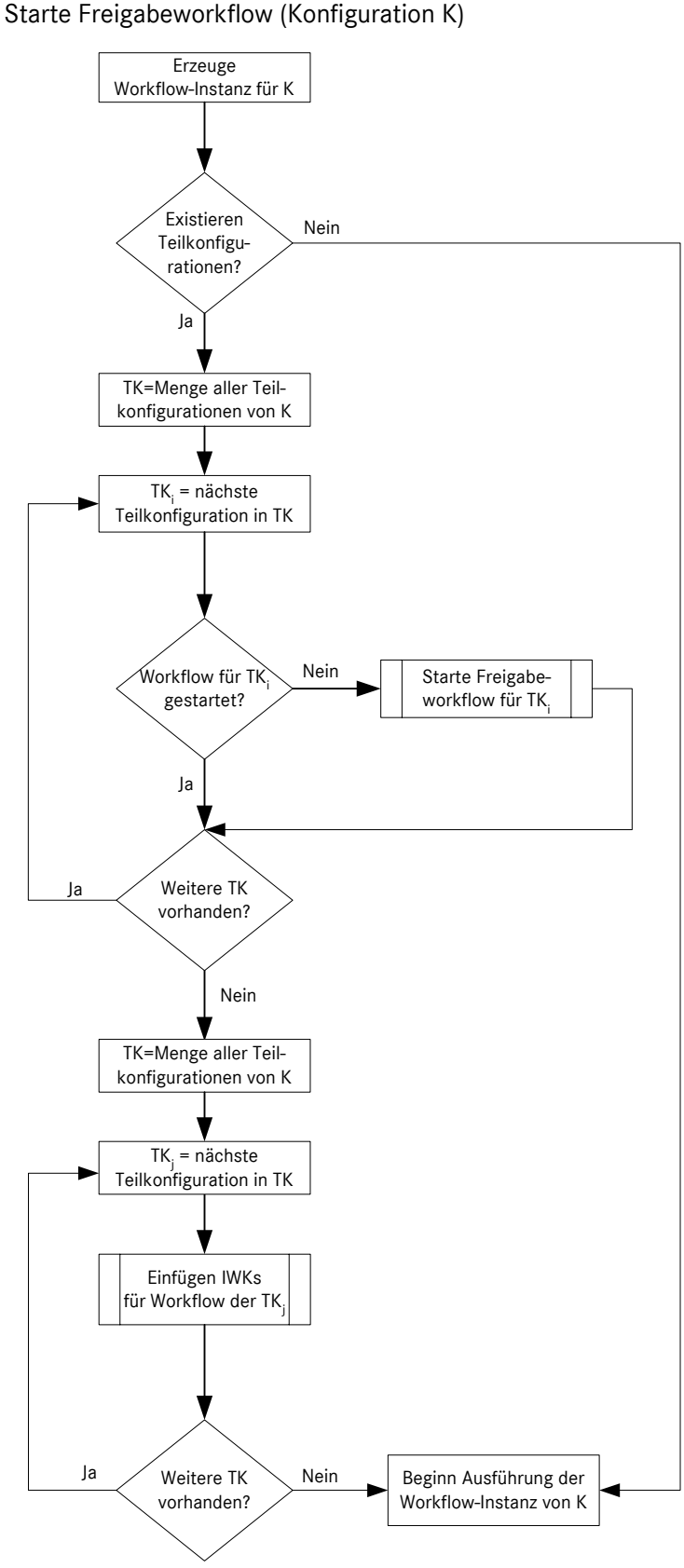

Abbildung 7.18: Ablauf beim Einfügen von IWKs beim Workflow-Start

7.2.3 Bewertung und offene Fragen

Ein Vorteil der vorgeschlagenen, *expliziten* Repräsentation der sequenziellen Kontrollflussabhängigkeiten ist die direkte Sichtbarkeit und damit bessere Verständlichkeit der Abhängigkeiten für Anwender. Durch die direkte Repräsentation als Kanten sind auch dynamische Änderungen an einem Workflowgeflecht einfacher umzusetzen. Aus Zeitgründen konnte in dieser Arbeit das dynamische Löschen von IWKs nicht mehr betrachtet werden.

Weitere Fragen stellen sich auch noch in Bezug auf die Verklemmungen, die sich mit IWKs ergeben können. Dies sind zum einen Verklemmungen, die aufgrund zyklischer IWKs entstehen. Ob es solche Verklemmungen geben kann, hängt vom Modell ab, mit dem die Abhängigkeiten beschrieben werden. Im Falle von Freigabeworkflows wurden ganz bewusst Restriktionen im Abhängigkeitsmodell eingeführt, um solche Verklemmungen zu verhindern (siehe Abschnitt 7.1.1.3d)). Wenn man IWKs als allgemeines Konzept in WfMS integrieren möchte, muss man sich dieser Problematik bewusst sein, und Mechanismen vorsehen, die solche Verklemmungen verhindern oder zumindest erkennen können. Zum anderen kann es zu einer weitreichenden Verklemmung kommen, wenn eine Quellaktivität einer IWK nicht erfolgreich beendet wurde und somit eigentlich die Voraussetzung für die Ausführung der Zielaktivität fehlt. Momentan wird hier von einer sehr strengen Semantik der IWKs ausgegangen: Da die durch eine IWK ausgedrückte Vorbedingung fehlt, kann diese Verklemmung nur dadurch aufgelöst werden, dass die Bedingung von einem Prozessverantwortlichen außer Kraft gesetzt wird. Eine interessante Fragestellung an dieser Stelle wäre, wie eine „weichere“ Semantik einer IWK aussehen könnte, bei der mit solchen Verklemmungen flexibler umgegangen werden kann. Dazu sollte auf jeden Fall in der Praxis evaluiert werden, ob es solche Fälle gibt, und welches Verhalten für diese Fälle wünschenswert wäre.

7.3 Behandlung der Prüffehler

Nach der Vorstellung von Konzepten zur Beschreibung und Ausführung der Kontrollflussabhängigkeiten, sollen in diesem Abschnitt erste Ansätze für Fehlerbehandlungen beschrieben werden (vgl. Abschnitt 5.2.5). Ein Grundgedanke dazu ist, in besonderen Ausführungszuständen der Aktivitäten und der Workflow-Instanz festzuhalten, ob Fehler in Konfigurationen gefunden wurden. Durch die Erweiterung der Ausführungszustände von Aktivitäten einer Workflow-Instanz kann eine korrekte und den Anforderungen entsprechende Behandlung von Prüffehlern sichergestellt werden.

7.3.1 Erweiterung der Ausführungszustände von Aktivitäten

Ein verallgemeinertes Modell der Ausführungszustände und –übergänge konventioneller WfMS wurde bereits in Abschnitt 4.4 vorgestellt. Dieses Zustandsmodell wird um zwei Zustände (und entsprechende Übergänge) erweitert, die eine Unterscheidung ermöglichen, ob beim Ausführen einer Aktivität in der Konfiguration ein Fehler gefunden wurde.

Im Einzelnen sind dies die folgenden Zustände:

- *beendet ohne Prüffehler:* die Aktivität wurde beendet und in der entsprechenden Konfiguration wurde kein Fehler gefunden.
- *beendet mit Prüffehler:* die Aktivität wurde beendet und in der entsprechend Konfiguration wurde ein Fehler gefunden.

Diese beiden Zustände ersetzen den Zustand „beendet“. Das gesamte erweiterte Modell für die Zustände einer Aktivität zeigt der in Abbildung 7.19 abgebildete Zustandsautomat.

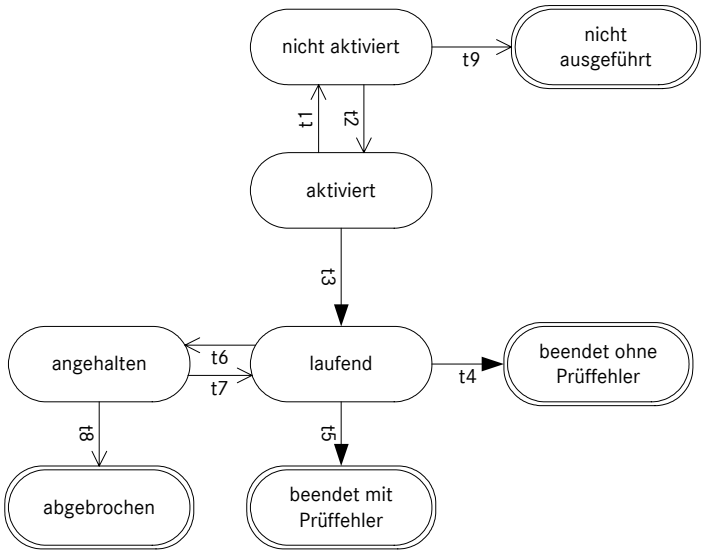

Abbildung 7.19: Ausführungszustände von Aktivitäten

7.3.2 Erweiterung der Ausführungszustände von Workflow-Instanzen

Das Zustandsmodell für Workflow-Instanzen muss für die Ausführung der Freigabeworkflows ebenfalls erweitert werden. Dabei ist es insbesondere notwendig, in Zuständen festzuhalten, ob schon einmal ein Prüffehler gefunden wurde, damit nach Beendigung des Workflows der Freigabestatus korrekt gesetzt werden kann.

Hinzugekommen sind die folgenden Zustände (der Zustand „beendet“ wurde durch den Zustand „beendet mit Freigabe“ ersetzt):

- *beendet mit Freigabe:* Alle Aktivitäten der Workflow-Instanz wurden beendet, und in keiner der Aktivitäten wurde ein Prüffehler gefunden, so dass die zugehörige Konfiguration freigegeben ist.
- *angehalten Prüffehler:* In einer Aktivität wurde ein Fehler gefunden. Die Ausführung der Workflow-Instanz wurde angehalten, damit die Prozessverantwortlichen die nötigen Maßnahmen treffen können.
- *laufend Prüffehler:* Obwohl ein Fehler in der Konfiguration gefunden wurde, wird die Workflow-Instanz weiter ausgeführt.
- *beendet ohne Freigabe:* In der Konfiguration wurde mindestens ein Fehler gefunden, die Ausführung der Instanz ist beendet und die Konfiguration *nicht* freigeben.

Abbildung 7.20 zeigt das vollständige Zustandsmodell für eine Workflow-Instanz.

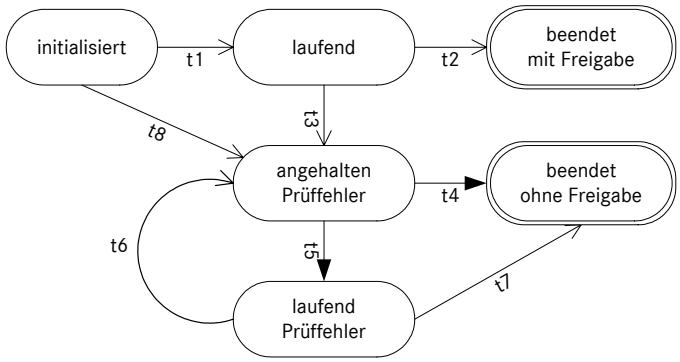

Abbildung 7.20: Zustandsautomat für Workflow-Instanz

7.3.3 Ausführung einer Workflow-Instanz

Nachdem in den beiden vorangegangenen Abschnitten das erweiterte Zustandsmodell vorgestellt wurde, sollen in diesem Abschnitt die Interaktion der Zustandsmodelle und die Zustandsübergänge bei der Ausführung einer Workflow-Instanz beschrieben werden. Nach Beschreibung des Ausführungsverhaltens im Normalfall wird das Verhalten im Falle von Prüffehlern beschrieben. Hierbei werden auch eventuell vorhandene IWKs mit berücksichtigt.

7.3.3.1 Ausführungsverhalten im Normalfall

Nachdem eine Workflow-Instanz erzeugt worden ist, befindet sie sich im Zustand „initialisiert“. Nach dem Starten (vgl. Abschnitt 7.2.2) wird sie vom System in den Zustand „laufend“ gebracht. Die Ausführung des Workflows beginnt. Ausführung des Workflows bedeutet, dass alle zu aktivierenden Aktivitäten bestimmt werden und diese vom Zustand „nicht aktiviert“ in den Zustand „aktiviert“ gelangen. Eine Aktivität kann genau dann aktiviert werden, wenn sie innerhalb des normalen Kontrollflusses aktiviert werden kann *und* alle eingehenden IWKs den Signalisierungszustand „wahr“ besitzen.

Aktivierte Aktivitäten erscheinen in den Arbeitslisten der möglichen Bearbeiter. Selektiert ein Bearbeiter diese Aktivität, so gelangt diese in den Zustand „laufend“ und verschwindet von den Arbeitslisten der anderen Bearbeiter. Wurde die Prüfaktivität erfolgreich ohne Prüffehler beendet, so teilt der Benutzer dies dem System mit. Die Aktivität gelangt über die Transition t_4 in den Endzustand „beendet ohne Prüffehler“. Alle IWKs, die von der Aktivität ausgehen, erhalten den Signalisierungszustand „wahr“ und die Ausführung des Workflows wird durch die Bestimmung von neu zu aktivierenden Aktivitäten fortgesetzt.

Falls in einer Workflow-Instanz alle Aktivitäten beendet wurden und sich im Zustand „beendet ohne Prüffehler“ befinden, führt das System für die Workflow-Instanz automatisch den Übergang t_2 in den Zustand „beendet mit Freigabe“ durch – die zugehörige Konfiguration ist jetzt freigegeben.

7.3.3.2 Ausführungsverhalten bei Prüffehlern

Ein Prüffehler bedeutet aus Systemsicht, dass der Bearbeiter einer laufenden Aktivität dem System mitteilt, dass während des Prüfschritts Fehler in der Konfiguration gefunden wurden. Dies bewirkt im Zustandsautomaten für die betreffende Aktivität, dass diese durch die Transition t_5 in den Zustand „beendet mit Prüffehler“ gelangt. Gleichzeitig werden vom System die Workflow-Instanz in den Zustand „angehalten Prüffehler“ und alle laufenden Aktivitäten dieser Instanz in den Zustand „angehalten“ gebracht. Aktivierte Aktivitäten gelangen in den

Zustand „nicht aktiviert“, die normale Ausführungslogik des Workflows wird angehalten und der Prozessverantwortliche benachrichtigt. Dies geschieht nicht nur mit der direkt betroffenen Workflow-Instanz, sondern auch mit allen Workflow-Instanzen der Oberkonfigurationen (siehe Abschnitt 5.2.5).

Der Prozessverantwortliche kann in diesem Zustand den Workflow modifizieren, indem er Aktivitäten löscht und Abhängigkeiten zwischen Aktivitäten anpasst. Anschließend kann er den Freigabeworkflow weiterlaufen lassen. Dies bedeutet, dass alle Aktivitäten, die vorher im Zustand „angehalten“ waren, jetzt abermals „laufend“ sind und die Abarbeitung des Workflows wieder beginnt (Bestimmung der zu aktivierenden Aktivitäten etc.). Die Workflow-Instanz selbst befindet sich danach jedoch im Zustand „laufend Prüffehler“. Die Prüffehlerreaktion wird in dieser Weise von den Prozessverantwortlichen aller betroffenen Konfigurationen durchgeführt.

Sollte nochmals ein Prüffehler auftreten, gelangt die Instanz wiederum über Transition t_6 in den Zustand „angehalten Prüffehler“ und das eben beschriebene Verfahren beginnt von Neuem. Die andere Reaktionsmöglichkeit des Prozessverantwortlichen ist, den Workflow abzubrechen (Transition t_4). Dadurch gelangen alle angehaltenen Aktivitäten in den Zustand „abgebrochen“ (t_4) und alle nicht aktivierten Aktivitäten in den Zustand „nicht ausgeführt“ (t_9). Die Workflow-Instanz gelangt in den Zustand „beendet ohne Freigabe“ – die zugehörige Konfiguration ist nicht freigegeben.

Tabelle 7.1 zeigt beispielhaft die Zustandsänderungen einer Workflow-Instanz und zweier Aktivitäten dieser Instanz bei Prüffehlern. Dazu sind die Zustände für aufeinander folgende Zeitpunkte notiert.

	Ausgangssituation	*Nach Fehler in Aktivität 1*	*Direkt nach Fehlerreaktion*	*Nach Auswählen der Aktivität 2 durch Benutzer*	*Nach weiterem Prüffehler in Aktivität 2*
Workflow-Instanz	laufend	angehalten Prüffehler	laufend Prüffehler	laufend Prüffehler	angehalten Prüffehler
Aktivität 1	laufend	beendet mit Prüffehler	beendet mit Prüffehler	beendet mit Prüffehler	beendet mit Prüffehler
Aktivität 2	aktiviert	nicht aktiviert	aktiviert	laufend	angehalten Prüffehler

Tabelle 7.1: Zustandsübergänge bei Behandlung von Prüffehlern

7.3.4 Zusammenfassung

Die Berücksichtigung von Prüffehlern in den Zuständen von Aktivitäten und Workflow-Instanzen ist ein erster Schritt zur Ermöglichung einer Prüffehlerbehandlung, wie sie erforderlich ist. Zu einer vollständigen Unterstützung bei der Behandlung dieser Fehler sind allerdings noch einige Fragen offen.

Eine dieser offenen Fragen betrifft die Fehlerreaktion bei Ober- und Unterkonfigurationen. Dort ist zu klären, welches Protokoll für die Koordinierung der Reaktionen auf den verschiedenen Ebenen geeignet ist. Denkbar wäre im einfachsten Fall, dass zuerst die Fehlerreaktion auf einer unteren Ebene abgeschlossen sein muss, bevor auf der darüberliegenden Ebene reagiert werden kann. In diesem Fall würde es allerdings sehr lange dauern, bis eine Fehlerreaktion beendet wäre. Passender wäre an dieser Stelle eine parallelisierte Fehlerreaktion, die jedoch nicht zu inkonsistenten Zuständen führen darf. Auch Fragen in Bezug auf das

dynamische Ändern von Workflow-Instanzen und der Abhängigkeiten zwischen Workflows wurden hier noch ausgeklammert.

7.4 Abschließende Bemerkungen

Die Lösungsansätze dieses Kapitels liefern eine erste Grundlage für eine adäquate Workflow-Unterstützung von Freigabeprozessen. Im Rahmen dieser Arbeit konnten jedoch nicht alle offenen Problemstellungen angegangen werden. Beispielsweise wurden dynamische Änderungen (Löschen von Aktivitäten und Anpassen der Kontrollflussabhängigkeiten) noch nicht berücksichtigt. Auch in den Lösungskonzepten selbst haben sich weiterführende Fragestellungen ergeben, die es noch zu lösen gilt. Nach Lösung der konzeptuellen Probleme, stellt sich dann die Frage, wie man diese Konzepte technisch umsetzen möchte: Kann man ein bestehendes WfMS um diese Konzepte erweitern oder ist die Entwicklung eines neuartigen WfMS zu bevorzugen?

Die Anzahl der sich neu ergebenden und noch offenen Fragestellungen zeigt, dass die Lösungskonzepte dieser Arbeit erst der Beginn der Entwicklung eines ganzheitlichen, in sich abgeschlossenen Lösungsansatzes sind.

8 Zusammenfassung und Ausblick

Die steigende Komplexität der eingebetteten Systeme in Fahrzeugen verlangt Änderungen der bestehenden Entwicklungsprozesse, um hohe Qualität zu erreichen bzw. zu halten. Die Bereiche Test und Freigabe eingebetteter Systeme wurden in dieser Arbeit näher betrachtet. Dabei wurde deutlich, dass eine IT-Unterstützung für Freigabeprozesse auf Basis von Workflow-Management-Technologie unbedingt erforderlich ist. Hierzu wurden die funktionalen Anforderungen ermittelt und deren Umsetzbarkeit mit Konzepten marktüblicher Workflow-Management-Technologie überprüft. Aus den Mängeln, die dort zu Tage kamen, konnten neue, weitergehende Anforderungen an WfMS abgeleitet werden.

Für diese Anforderungen wurden verwandte Arbeiten ausführlich analysiert. Doch nicht für alle ermittelten Anforderungen existieren schon adäquate Konzepte, so dass es neuer Lösungskonzepte bedarf. Hauptsächlich für den Aspekt der Synchronisation paralleler Workflow-Instanzen wurden erste Lösungskonzepte erarbeitet, die in fortführenden Arbeiten noch zu vervollständigen und vor allem in der Praxis zu erproben sind.

Obwohl hier ausschließlich Konfigurationen eingebetteter Systeme betrachtet wurden, lässt sich die grundsätzliche Problematik auch auf andere Domänen übertragen, in denen Einzelteile schrittweise zusammengebaut, getestet und freigegeben werden müssen. Dies ist beispielsweise in der reinen Softwareentwicklung genauso der Fall [Pa04] wie in der klassischen, mechanischen Fahrzeugentwicklung. Hier ergibt sich die Frage, wie weit sich die Anforderungen überdecken und in wie weit die Lösungskonzepte in diesen Domänen anwendbar sind. Daraus können generische Konzepte entwickelt werden, die für alle Anwendungsgebiete tauglich sind. Mit der breiteren Anwendbarkeit steigt gleichzeitig auch die Chance, dass derartige Konzepte in kommerzielle Produkte einfließen und somit von Unternehmen genutzt werden können, die bei Workflow-Management-Systemen nicht individuelle Spezialentwicklungen, sondern Standardsoftware verwenden möchten.

Weitere interessante Fragestellungen ergeben sich, wenn man die von Freigabeprozessen abhängige Prozesse ebenfalls unterstützen möchte, wie beispielsweise Bestellprozesse von zu testenden Produktkomponenten. So müssen beispielsweise bei Abbruch eines Freigabeprozesses eventuell Bestellungen für Teile einer Oberkonfiguration rückgängig gemacht werden. Andererseits sind Freigabeprozesse auch zeitlich abhängig von der termingerechten Lieferung der zu testenden Komponenten, so dass bei Verzögerung ebenfalls entsprechende Maßnahmen getroffen werden müssten. Daraus könnte sich weiterer Forschungsbedarf ergeben.

Bis die in dieser Arbeit skizzierte Vision der IT-Unterstützung für Freigabeprozesse in die Realität umgesetzt wird, ist es sicherlich noch ein weiter Weg. Aufgrund der zu erwartenden Fortschritte in der Produktqualität ist dies jedoch langfristig auf jeden Fall lohnend.

Literatur

[AAE96] Alonso, G.; Agrawal, D.; El Abbadi, A.: *Process Synchronization in Workflow Management Systems.* Proc. 8th IEEE Symposium on Parallel and Distributed Processing, New Orleans, Oktober 1996.

[BaHo99] Barros, A.P.; ter Hofstede, A.H.M.: *Modelling Concurrent Process Coordination in Workflow Specifications.* Proc. Information System Concepts: An Integrated Discipline Emerging, IFIP TC8/WG8.1, Leiden, September 1999, S. 141-162.

[BDS98] Beuter, T.; Dadam, P.; Schneider, P.: *The WEP Model: Adquate Workflow-Management for Engineering Processes.* Proc. European Concurrent Engineering Conference 1998, Erlangen-Nürnberg, April 1998.

[Be02] Beuter, T.: *Workflow-Management für Produktentwicklungsprozesse.* Dissertation, Universität Ulm, 2002.

[BHR05] Bestfleisch, U.; Herbst, J.; Reichert, M.: *Requirements for the Workflow Based Support of Release Management Processes in the Automotive Sector.* Proc. ECEC 2005, Toulouse, April 2005, S. 130-134.

[Ca98] Casati, F.: *Models, Semantics, and Formal Methods for the design of Workflows and their Exceptions.* Dissertation, Universität Mailand, 1998.

[Ca99] Casati, F.: *Semantic Interoperability in interorganizational workflows.* Proc. Workshop on Cross-Organisational Workflow Management and Coordination, San Francisco, Februar 1999.

[DeGr94] Deiters, W.; Gruhn, V.: *The FUNSOFT Net Approach to Software Process Management.* Int'l Journal of Software Engineering and Knowledge Engineering, Vol.4, No.2, 1994, S. 229-256.

[DoKö04] Dold, A.; König, C.: *SW-Archivierung in Smaragd.* Internes DaimlerChrysler-Dokument.

[DRK00] Dadam, P.; Reichert, M.; Kuhn, K.: *Clinical workflows - The Killer Application for Process-oriented Informations Systems?* Proc. 4th Int'l Conference on Business Information Systems, Poznan, April 2000, pp. 36-59.

[EdLi95] Eder, J.; Liebhart, W.: *The Workflow Activity Model WAMO.* Proc. of the 3rd Int. Conference in Cooperative Information Systems, Wien, Mai 1995.

[EP04] Abteilung EP/QEM: *Das Mercedes-Benz Development System.* Internes DaimlerChrysler-Dokument.

[GAHL00] Grefen, P.; Aberer, K.; Hoffner, Y.; Ludwig, H.: *CrossFlow: cross-organizational workflow management in dynamic virtual enterprises.* Int'l Journal of Computer Systems Science & Engineering, Vol. 15, No. 5, September 2000, S. 277-290.

[Gr05] Grinberg, M.: *Entwicklung eines toolgestützten Konfigurationsmanagements für den E/E-Testprozess von Fahrzeugen.* Diplomarbeit, Universität Stuttgart, 2005.

[HaAl99] Hagen, C.; Alonso G.: *Beyond the Black Box: Event-based Inter-Process Communication in Process Support Systems.* Proc. 19th Int'l Conference on Distributed Computing Systems, Austin, Juni 1999, S. 450-457.

[He00] Heinlein, C.: *Workflow- und Prozeßsynchronisation mit Interaktionsausdrücken und -graphen.* Dissertation, Universität Ulm, 2000.

[HRKH04] Herbst, J.; Rodefeld, C.; König, C.; Handke, N.: *Freigabe- und Änderungsmanagement.* Internes DaimlerChrysler-Dokument.

[IBM03] IBM Corp.: *Getting Started with Buildtime. Version 3.5.* Handbuch zu MQSeries Workflow, 2003.

[ISO 10007] DIN ISO 10007:2003: *Leitfaden für Konfigurationsmanagement.*

[JBS97] Jablonski, S.; Böhm, M.; Schulze W. (Hrsg.): *Workflow-Management – Entwicklung von Anwendungen und Systemen.* dpunkt.verlag, 1997.

[JKK+02] Jordan, J.; Kleinhans, U.; Kulendik, O.; Porscha, J.; Pross, A.; Siebert, R.; Storch, M.: *Transparent and Flexible Cross-Organizational Workflows for Engineering Cooperations in Vehicle Development.* Proc. PDT Europe 2002, Turin, Mai 2002, S. 101-108.

[Ka98] Kamath, M.: *Improving correctness and failure handling in workflow management systems.* Dissertation, Universität Massachusetts, 1998.

[KaRa98] Kamath, M.; Ramamritham, K.: *Failure Handling and Coordinated Execution of Concurrent Workflows.* Proc. 14th Int'l Conference on Data Engineering, Orlando, Februar 1998, S. 334-341.

[KnSc04] Knippel, E.; Schulz, A.: *Lessons Learned from Implementing Configuration Management within Electrical/Electronic Development of an Automotive OEM.* Proc. 14th Annual Int'l Symposium of the Int'l Council on Systems Engineering.

[KRS01] Kulendik, O.; Rothermel, K.; Siebert, R.: *Cross-organizational Workflow Management - General approaches and their suitability for engineering processes.* Proc. First IFIP-Conference on E-Commerce, E-Business, E-Government, Zürich, Oktober 2001, S. 143-158.

[LeRo00] Leymann, F.; Roller, D.: *Production Workflow – Concepts and Techniques.* Prentice Hall, 2000.

[Ma04] Maier, A.: *Visualisierung dynamischer, komplexer Projektpläne in der Produktentwicklung.* Diplomarbeit, Universität Ulm, 2004.

[MuAl00] zur Muehlen, M.; Allen, R.: *Workflow Classification: Embedded & Autonomous Workflow Management Systems.* White Paper of the WfMC, 2000.

[Pa04] Partsch, H.: *Softwaretechnik.* Begleitunterlagen zur Vorlesung an der Universität Ulm, SS 2004.

[ProdHaftG02] *Gesetz über die Haftung für fehlerhafte Produkte.* Verkündet im BGBl I 1989, 2198, geändert am 19.7.2002.

[RDMK00] Reichert, M.; Dadam, P.; Mangold, R.; Kreienberg, R.: *Computerbasierte Unterstützung von Arbeitsabläufen im Krankenhaus – Konzepte, Technologien und deren Anwendung.* Zentralblatt für Gynäkologie, Heft 1, 2000, S. 53-67.

[Re00] Reichert, M.: *Dynamische Ablaufänderungen in Workflow-Management-Systemen.* Dissertation, Universität Ulm, 2000.

[Re03] Reichert, M.: *Workflow-Management-Systeme: Grundlagen, Einsatz und Implementierung.* Begleitunterlagen zur Vorlesung an der Universität Ulm, WS 2003/2004.

[ReDa98] Reichert, M.; Dadam, P.: *$ADEPT_{flex}$ – Supporting Dynamic Changes of Workflows Without Loosing Control.* Journal of Intelligent Information Systems, Vol. 10, No.2, März/April 1998, S. 93-129.

[RRD04] Rinderle, S.; Reichert, M.; Dadam, P.: *Correctness criteria for dynamic changes in workflow systems – a survey.* Data & Knowledge Engineering, Vol. 50, No. 1, Juli 2004, S. 9–34.

[Sc02] Schichtel, M.: *Produktdatenmodellierung in der Praxis.* Hanser Verlag, 2002.

[Sc03] Schmid, A.: *Konzeption und Realisierungs komplexer Interaktionen zwischen Workflows.* Diplomarbeit, Universität Stuttgart, Institut für Parallele und Verteilte Systeme, 2003.

[ScBi96] Schreyjak, S.; Bildstein, H.: *Fehlertolerante Abwicklung von Geschäftsprozessen in Workflow-Management-Systemen.* Fakultätsbericht 1996/17, Fakultät für Informatik, Universität Stuttgart, Dezember 1996.

[We00] Wehlitz, P.: *Nutzenorientierte Einführung eines Produktdatenmanagement-Systems.* Dissertation, Technische Universität München, 2000.

[Wes99a] Weske, M.: *State-based modelling of flexible workflow executions in distributed environments.* Journal of Integrated Design and Process Science, Vol. 3, No. 2, Juni 1999, S. 49-62.

[Wes99b] Weske, M.: *Adaptive Workflows based on Flexible Assignment of Workflow Schemas and Workflow Instances.* Proc. Workshop: Enterprise-wide and Cross-enterprise Workflow Management: Concepts, Systems, Applications, 29. Jahrestagung der GI (Informatik ’99), Paderborn, Oktober 1999, S. 42-48.

[WfMC95] Workflow Management Coalition: *The Workflow Reference Model.* WFMC-TC00-1003, Version 1.1, Januar 1995.

[WHKS98] Weske, M.; Hündling, J.; Kuropka, D.; Schuschel, H.: *Objektorientierter Entwurf eines flexiblen Workflow-Management-Systems.* Informatik Forschung und Entwicklung, Vol. 13, No. 4, 1998, S. 179-195.

Glossar

Absicherungsmaßnahme	Maßnahme mit der ein bestimmter Aspekt einer ⇢*Konfiguration* abgesichert wird, d.h. deren Kompatibilität und Korrektheit überprüft wird. Beispiele sind ⇢*Bretttest* oder ⇢*HiL-Test*.
Aktuator	Komponente, die auf ein elektrisches Eingabesignal hin auf seine Umgebung einwirkt, beispielsweise der Motor für den Scheibenheber.
Baureihe	Produkte, die in verschiedensten Ausführungsvarianten gleichartig hergestellt werden. Beispiele aus dem KfZ-Bereich sind die aktuelle A-Klasse von Mercedes-Benz oder die 7er-Reihe von BMW.
Bretttest	Test, bei dem ⇢*Steuergeräte* auf einem Brett verkabelt und die Eingangssignale manuell simuliert werden.
E/E	Elektrik/Elektronik Oberbegriff für alle elektrischen oder elektronischen Komponenten eines Fahrzeugs, einschließlich Software.
E/E-Gesamtsystem	Gesamtheit aller ⇢*E/E-Komponenten* in einem Fahrzeug.
E/E-Komponente	⇢*Produktkomponente* im Bereich Elektrik/Elektronik. Verallgemeinerung von ⇢*Steuergeräten*, ⇢*Sensoren* und ⇢*Aktuatoren*.
E/E-Konfiguration	Bündelung von ⇢*Versionen* von ⇢*E/E-Komponenten*, die Kompatibilität und Korrektheit dieser E/E-Komponenten gemäß ihrer Spezifikation ausdrückt.
E/E-System	Alle ⇢*E/E-Komponenten*, die zur Realisierung einer kundenerlebbaren Funktion beitragen.
ECU	Electronic Control Unit: engl. für ⇢*Steuergerät*.
Feature	Eine vom Kunden erlebbare Funktion eines Fahrzeugs.
Feature-Rollout-Plan	Zeitplan, wann welches ⇢*Feature* umzusetzen ist.
Flashbarkeit, flashbar	Ein ⇢*Steuergerät* ist dann *flashbar*, wenn gemäß bestimmter Richtlinien Software auf das Steuergerät geflasht werden kann. (⇢*Flashen)*
Flashen	Laden von Software in den wiederbeschreibbaren Speicher eines ⇢*Steuergeräts*.

Freigabe	Eine ⇢*Konfiguration* besitzt eine *Freigabe*, wenn sie den ⇢*Freigabezustand* „Freigegeben“ hat. Eine *Freigabe* dient einem bestimmten Zweck.
Freigabeprozess	Gesamtheit aller ⇢*Absicherungsmaßnahmen*, die durchgeführt werden, um eine Konfiguration abzusichern.
Freigabezustand	Drückt aus, ob und wie eine ⇢*Konfiguration* einen ⇢*Freigabeprozess* durchlaufen hat.
FSG	Fondsteuergerät ⇢*Steuergerät*, das im Fond eines Fahrzeugs eingebaut ist.
HiL-Test	Hardware-in-the-Loop-Test Test, bei dem ⇢*Steuergeräte* in einer von einem Echtzeit-Rechner simulierten Umgebung getestet werden.
K-Matrix	Kommunikations-Matrix Matrix, in der die zwischen ⇢*Steuergeräten* ausgetauschten Signale dokumentiert sind.
Konfiguration	In dieser Arbeit: synonym zu ⇢*E/E-Konfiguration*
PDM-System	Produktdatenmanagement-System System, in dem alle Daten eines Produktes verwaltet werden, die im ⇢*Produktentstehungsprozess* erzeugt werden.
Produktentstehungsprozess	Gesamter Prozess der Entstehung eines Produktes einschließlich Produktion.
Produktkomponente	Einzelteil, aus dem sich ein bestimmtes Produkt zusammensetzt. Beispiele sind ⇢*E/E-Komponenten*.
SAMV	Signalerfassungs- und Auswertemodul vorn ⇢*Steuergerät* im vorderen Teil des Fahrzeugs, das die Signale von Sensoren erfasst und auswertet.
Sensor	Elektrische oder elektronische Komponente, die Messungen vornimmt.
Steuergerät	Eigenständiges, in den meisten Fällen mit Software bestücktes elektronisches Gerät zur Steuerung technischer Anlagen wie beispielsweise Fahrzeuge. Moderne Steuergeräte enthalten wiederbeschreibbaren Speicher, auf den die Software beliebig oft geladen werden kann (⇢*Flashen*).
TSL	Türsteuergerät links: ⇢*Steuergerät*, das in der linken Tür eines Fahrzeugs eingebaut ist.
Version	Bestimmter Entwicklungs- und Änderungsstand einer ⇢*Produktkomponente*.

Abbildungsverzeichnis

Erklärung

Diese Arbeit wurde von mir selbständig verfasst. Ich habe dabei nur die angegebenen Quellen und Hilfsmittel verwendet.

Ulm, 25. April 2005

Ulrich Bestfleisch