T U M

I N S T I T U T F Ü R I N F O R M A T I K

Modulare Spezifikation von Projektabläufen

Klaus Bergner, Jan Friedrich

TUM-I0912
April 09

T E C H N I S C H E U N I V E R S I T Ä T M Ü N C H E N

TUM-INFO-04-I0912-0/1.-FI

Druck: Institut für Informatik der
Technischen Universität München

Modulare Spezifikation von Projektabläufen

Eine formale Fundierung von Syntax und Semantik der Projektdurchführungsstrategien des V-Modell XT 1.3

Klaus Bergner, Jan Friedrich
4Soft GmbH
Mittererstraße 3
D-80336 München, Deutschland
{bergner|friedrich}@4soft.de

Zusammenfassung

Das V-Modell® XT ist der deutsche Standard für die Durchführung von IT-Projekten des Bundes.[1] Es legt die zeitlichen Abläufe in einem Projekt mit Hilfe sogenannter Projektdurchführungsstrategien fest. Der vorliegende Bericht stellt diese Beschreibungstechnik vor und zeigt, wie sie zur modularen Spezifikation von Projektabläufen verwendet werden kann. Weiterhin enthält er eine formale Spezifikation der abstrakten Syntax und der Semantik von Projektdurchführungsstrategien sowie der damit verbundenen Konzepte für Version 1.3 des V-Modell XT. Die Formalisierung geschieht über eine Abbildung der Projektdurchführungsstrategien auf Petri-Netze und die Ableitung von konsistenten Petri-Spuren oder Prozessnetzen. Eine konsistente Abbildung von Projektmeilensteinplänen auf diese Petri-Spuren definiert dann schließlich, wann ein Projektplan konsistent zum Vorgehensmodell ist. Damit legt der Bericht insbesondere eine klare Grundlage für die Implementierung entsprechender Werkzeuge.

Keywords: Software Development Process, V-Modell XT, Werkzeuge, Prozessspezifikation

CR-Classification: D.2

[1] V-Modell® ist eine eingetragene Marke der Bundesrepublik Deutschland.

Inhaltsverzeichnis

1 Einführung

1.1 Motivation und Ziele

Als eines der ersten Vorgehensmodelle baut das V-Modell XT [Bun09a] vollständig auf einem Metamodell auf. Dadurch können Werkzeuge das Vorgehensmodell von sich aus „verstehen“ - die Inhalte müssen nicht hart im Quellcode programmiert werden.

Die im V-Modell enthaltenen Projektdurchführungsstrategien spezifizieren die Reihenfolge von Meilensteinen und definieren dadurch gültige Projektabläufe bzw. Projektpläne. Mit dem V-Modell 1.3 wurden Beschreibungstechnik und Metamodell der Projektdurchführungsstrategien komplett umgestaltet und erweitert, weil die zuvor verwendete Beschreibungstechnik viele sinnvolle Projektabläufe nicht abbilden konnte.

Die neue Beschreibungstechnik bietet im Vergleich zu vergleichbaren Ansätzen einige grundlegende Neuerungen und erlaubt damit eine modulare Spezifikation realer Projektabläufe. Allerdings war bisher die Bedeutung der Projektdurchführungsstrategien für die im V-Modell definierten Abläufe zwar mehr oder weniger intuitiv verständlich, aber nicht formal definiert. Einer eindeutig definierte Semantik ist besonders für zwei Personengruppen relevant:

- *Projektleiter* planen ihre Projekte konform zum Vorgehensmodell und müssen deswegen genau verstehen, welche Meilensteinabfolgen gültig sind.
- *Werkzeughersteller* unterstützen mit ihren Projektplanungswerkzeugen die Arbeit der Projektleiter. Sie müssen genau verstehen, was das Vorgehensmodell bedeutet, um die Vorgaben des Vorgehensmodells nicht zu verletzen und die Projektleiter dabei zu unterstützen, es optimal anzuwenden und auszureizen.

Dieser Bericht zeigt in der Einführung zunächst kurz die neuen Möglichkeiten des V-Modell XT 1.3. Anschließend stellt er Syntax und Semantik der neuen Beschreibungstechnik auf eine mathematisch-formale Basis. Damit ist eindeutig definiert, welche Projektpläne im Sinne des V-Modells gültig sind und welche nicht. Wir beschränken uns dabei ausschließlich auf die Reihenfolge von Meilensteinen; Aktivitäten bzw. Vorgänge betrachten wir nicht.

Wissenschaftlern im Bereich Prozess- und Projektmodellierung bietet der Bericht ein durchgängiges Beispiel für die vollständige mathematische Formalisierung einer praktisch relevanten, komplexen Beschreibungstechnik. Daneben sind insbesondere Werkzeughersteller in der Zielgruppe, da sie auf eine eindeutige und unmissverständliche Spezifikation angewiesen sind. Von Projektleitern kann dagegen im Allgemeinen nicht erwartet werden, dass sie die Inhalte dieses Berichts vollständig durcharbeiten – der Kurzüberblick in Abschnitt 1.3 bietet Ihnen jedoch einen ersten Einstieg in die Thematik.

1.2 Abgrenzung von anderen Ansätzen

Zur Spezifikation von Projektabläufen werden heute meist Techniken verwendet, die aus dem Gebiet der Geschäftsprozessmodellierung stammen. Geschäftsprozesse werden dabei als wiederkehrende Unternehmensabläufe definiert, die möglichst effizient durchgeführt werden sollen. Letztendliches Ziel der Geschäftsprozessmodellierung ist es deshalb oft, die Prozesse auf Workflows abzubilden und diese dann mit Hilfe einer Workflow Engine und Sprachen wie BPEL (Business Process Execution Language) [OAS07] zu automatisieren [Wit05].

Projekte sind jedoch im Gegensatz zu typischen Geschäftsprozessen einmalige Vorhaben: Zu jedem Projekt gehört ein individueller Projektplan. Viele wissenschaftliche Ansätze zur Operationalisierung von Vorgehensmodellen verkennen diesen fundamentalen Unterschied und zielen darauf ab, Vorgehensmodelle genauso wie Geschäftsprozessmodelle zu automatisieren. So schlägt [BCCG07] beispielsweise im Kontext des SPEM2.0-Standards eine Abbildung von Abläufen auf die BPEL vor, [YL07] versucht es mit der Workflow-Sprache XPDL. [Gna07] ist sich des fundamentalen Unterschieds bewusst und beschreibt die Verbindung zwischen Vorgehensmodell und Projektplan; allerdings ist der Ansatz nur ansatzweise formalisiert, eine Werkzeugunterstützung wird nur teilweise skizziert.

Das V-Modell XT setzt gegenüber den „vollautomatischen Ansätzen“ auf flexibel vom Projektleiter kombinierbare *Ablaufbausteine*, die jeweils für das betreffende Projekt ausgewählt und angewendet werden, um einen passenden Projektplan zu erstellen. Zudem bietet es besondere Unterstützung für Projektiterationen und den Wechsel von Durchführungsstrategien im Projekt. Das folgende Beispiel soll das veranschaulichen:

Bei der iterativen Entwicklung eines Systems mit mehreren Komponenten kann es vorkommen, dass das Entwicklungsteam in einer ersten Iteration mit der Entwicklung einer Komponente beginnt, diese aber erst in einer späteren Iteration ins System integriert. Abbildung 1.1 zeigt dieses Vorgehen schematisch: Ein Pfeil nach unten bedeutet, dass der jeweilige Ablauf gestartet wird, ein Pfeil nach oben zeigt, dass der Ablauf endet. Die Entwicklung von Komponente 2 bildet damit praktisch einen „Tunnel“ zwischen Iteration 1 und Iteration 2 und führt dazu, dass der Ablaufgraph nicht baumartig ist.

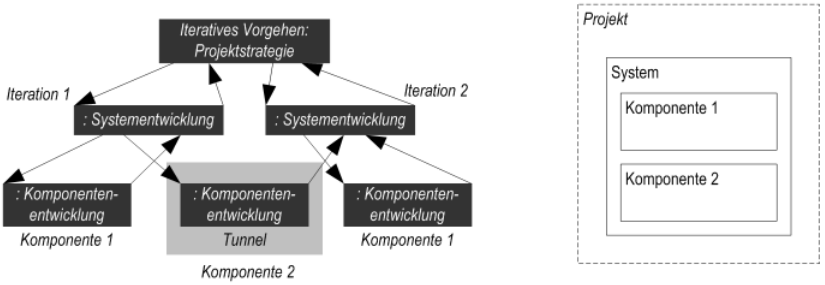

Abbildung 1.1: Ablauftunnel zwischen mehreren Iterationen

Diese Situation ist besonders dadurch kompliziert, dass in Iteration 2 durchaus eine andere Strategie zur *Systementwicklung* verwendet werden kann als in Iteration 1 – beispielsweise wäre denkbar, dass in Iteration 1 zunächst ein Basissystem mit einer Prototyping-Strategie entwickelt wird, das dann in Iteration 2 gemäß einer „traditionellen“ inkrementellen Entwicklungsstrategie fortentwickelt wird. Die Entwicklung von Komponente 2 endet also in einem anderen Ablaufbaustein, als sie begonnen hatte.

Diese Möglichkeit wurde durch die Semantik des alten V-Modell-Metamodells nicht unterstützt: Hatte ein Ablauf einen Entwicklungsstrang angestoßen, so musste er ihn auch wieder beenden – wie in allen uns bekannten anderen Ablaufbeschreibungstechniken waren Tunnel nicht erlaubt. Diese Unzulänglichkeit führte zur Entwicklung der neuen Strategien, die seit Version 1.2 in grafischer Form vorhanden sind und seit Version 1.3 nun endlich auch vollständig vom Metamodell unterstützt werden.

1.3 Veranschaulichung der Semantik

Abbildung 1.2 zeigt die Modellierungsmöglichkeiten des V-Modell XT anhand des Ablaufbausteins *Inkrementelle Systementwicklung*.

Der *Ablaufbaustein* „Inkrementelle Systementwicklung“ genügt der *Ablaufbausteinspezifikation* „Entwicklungsstrategie“. Der darin definierte Teilprozess beschreibt die Entwicklung eines Systems gemäß einem Top-down-Vorgehen: Zunächst wird das System spezifiziert und entworfen, dann realisiert, integriert und schließlich geliefert. Die grau dargestellten Parallelogramme entsprechen dabei den betreffenden *Entscheidungspunkten* im Projekt.

Über einen sogenannten *Ablaufbausteinpunkt* bindet der Ablaufbaustein „Inkrementelle Systementwicklung“ einen (nicht in der Abbildung gezeigten) Unterablauf ein, der der *Ablaufbausteinspezifikation* „Unterauftrag“ gehorchen muss. Welche Unterabläufe hier möglich sind, bestimmt sich aus der Menge der Ablaufbausteine, die der Projektleiter nach dem Tailoring seines Projekts zur Verfügung hat. Einer dieser Ablaufbausteine ist als *Wurzelbaustein* gekennzeichnet. Im Beispiel könnte der (ebenfalls nicht in der Abbildung dargestellte) Wurzelbaustein als (Gesamt-)Projektdurchführungsstrategie beispielsweise Vor- und Nachlauf des Projekts inklusive Angebotserstellung und Projektabschluss spezifizieren und dabei die „Entwicklungsstrategie“ einbinden.

Die in diesem Bericht beschriebene Semantik orientiert sich an Petri-Netzen: Auf dem Ablaufbaustein befinden sich Marken, die entlang der Pfeile (genannt *Übergänge*) „gezogen“ werden. Bildlich gesprochen ist Projektplanung damit nichts anderes als ein großes „Markenverschiebespiel“ nach der folgenden Regel: Zieht eine Marke über einen Entscheidungspunkt, so entspricht dies einem Meilenstein im Projektplan. Die zeitliche Reihenfolge der Meilensteine ergibt sich durch die Reihenfolge, in der die Marken die

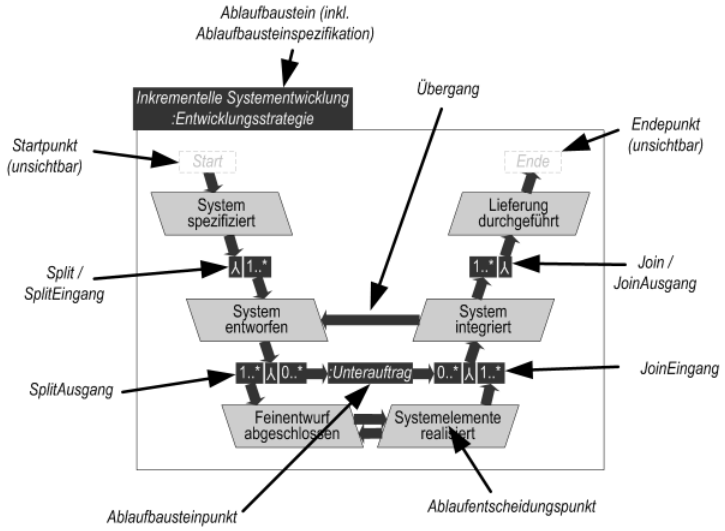

Abbildung 1.2: Graphische Syntax der Ablaufbausteine des V-Modell XT

Entscheidungspunkte passieren. Abbildung 1.3 zeigt solch eine einfache Zugfolge und den dazugehörigen Projektplan.

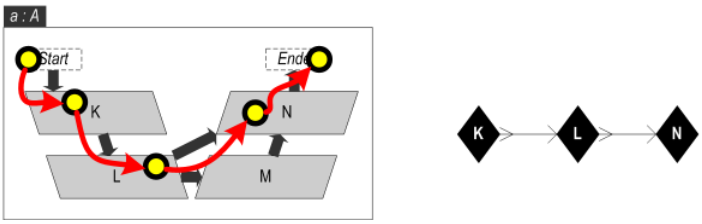

Abbildung 1.3: Beispiel für eine einfache Zugfolge und den zugehörigen Projektplan

An einem *Split* können aus einer Marke mehrere Marken werden: Splits entsprechen damit den Punkten im Projekt, an denen sich der Projektverlauf in Teilstränge verzweigen kann. Analog dazu werden an einem *Join* mehrere Marken zu einer Marke zusammengefasst. Damit entsprechen Joins den Punkten im Projekt, an denen Teilstränge wieder zu einem einzigen Strang zusammengefasst werden. Die notierten Kardinalitäten an den *Splitausgängen* und *Joineingängen* geben dabei jeweils an, wie viele Marken erzeugt bzw. zusammengefasst werden können. Abbildung 1.4 zeigt eine Zugfolge, bei der Marken vervielfacht und wieder zusammengefasst werden, sowie den dazugehörigen Projektplan.

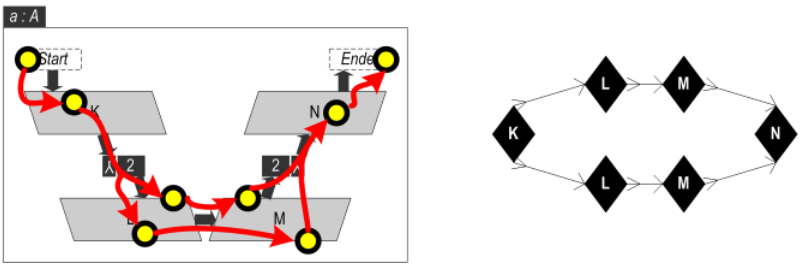

Abbildung 1.4: Beispiel für eine aufgeteilte Zugfolge und den zugehörigen Projektplan

Jeder Ablaufbaustein hat einen *Startpunkt* und einen *Endepunkt*. Wird ein Ablaufbaustein für die Projektplanung verwendet, so beginnt die Planung mit genau einer Marke auf dem Startpunkt. Eine Marke, die den Endepunkt erreicht hat, darf den Ablaufbaustein wieder verlassen. Die Verknüpfung und das Zusammenspiel der einzelnen Ablaufbausteine wird durch *Ablaufbausteinpunkte* erreicht: Zieht man eine Marke

entlang eines Pfeils auf einen Ablaufbausteinpunkt, so verlässt sie den aktuellen Ablaufbaustein und findet sich auf dem Startpunkt eines beliebigen Ablaufbausteins wieder - vorausgesetzt dieser entspricht der vorgegebenen Spezifikation.

Im Beispiel aus Abbildung 1.2 benötigt man also, wie bereits beschrieben, einen Baustein, der der Spezifikation „Unterauftrag“ genügt. Eine Marke, die den Ablaufbaustein über einen Endepunkt verlässt, wird auf einen Ablaufbausteinpunkt in einem beliebigen Ablaufbaustein gesetzt. Bedingung hierbei ist, dass die Spezifikation des Ablaufbausteinpunktes der des verlassenen Bausteins entspricht. Abbildung 1.5 zeigt eine gültige Zugfolge als Beispiel. Dabei sieht man, dass die Marke vom Baustein *c* nicht in den Baustein *a* zurückkehren muss, sondern auch im Baustein *b* „auftauchen“ kann. Damit lassen sich „Tunnel“ wie in Abbildung 1.1 gezeigt realisieren.

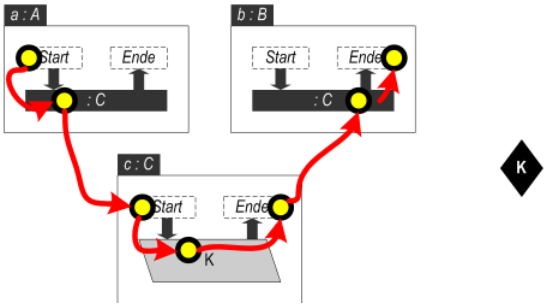

Abbildung 1.5: Beispiel für eine Zugfolge über mehrere Ablaufbausteine

Abbildung 1.6 zeigt abschließend noch einige ungewöhnlich anmutende, aber zugelassene Abläufe: Ablaufbaustein *a* besitzt prinzipiell vier unterschiedliche Durchläufe, die jeweils über *M* führen: *Start* → *Ende*, *D* → *Ende*, *Start* → *E* und *D* → *E*. Der Ablaufbaustein *b* kann zwar über einen Startpunkt betreten werden, dies führt aber zum sofortigen Ende. Dafür ist er in der Lage, Marken von Ablaufbausteinen der Spezifikation : *G* zu solchen der Spezifikation *H* zu schleusen. Außerdem realisiert der Ablaufbaustein eine Schleife auf Ablaufbausteinspezifikation *F*. So kann man beispielsweise eine iterative Entwicklung modellieren. Der Ablaufbaustein *c* ist insofern besonders, dass es bei ihm nicht möglich ist, Marken *innerhalb* des Bausteins vom Start- auf den Endepunkt zu ziehen.

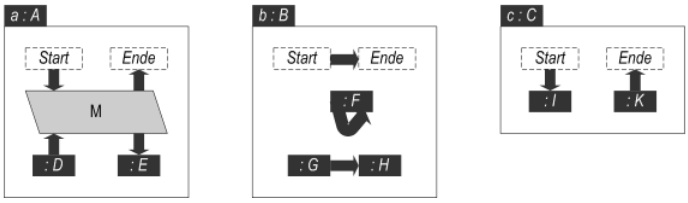

Abbildung 1.6: Ungewöhnliche, aber zugelassene Abläufe

1.4 Gliederung und Aufbau

Dieser Bericht stellt die im vorangehenden Abschnitt dargestellten Konzepte auf eine formale Basis. Ausgangspunkt für unsere Betrachtung ist das durchgeführte Tailoring mit einer (vielleicht nicht konsistenten) Menge an Ablaufbausteinen, von denen einer als Wurzelbaustein gekennzeichnet ist. Wir bauen davon ausgehend eine Brücke zwischen diesen Ablaufbausteinen im V-Modell XT (Kapitel 2) und Projektplänen in der Welt des Projektleiters (Kapitel 5). Diese Brücke besteht aus Petri-Netzen (Kapitel 3) und Petri-Spuren (Kapitel 4): Wir konstruieren aus Ablaufbeschreibungen des V-Modells konsistente Petri-Netze und definieren, wann eine Petri-Spur konsistent zu einem Petri-Netz ist; weiterhin definieren wir die Konsistenz zwischen Petri-Spuren und Projektplänen und schließen somit die Verbindung. Abbildung 1.7 zeigt unser Vorgehen. In Kapitel 6 fassen wir die Ergebnisse und unsere neuen Erkenntnisse zusammen und geben schließlich einen Ausblick auf noch offene Forschungsfragen.

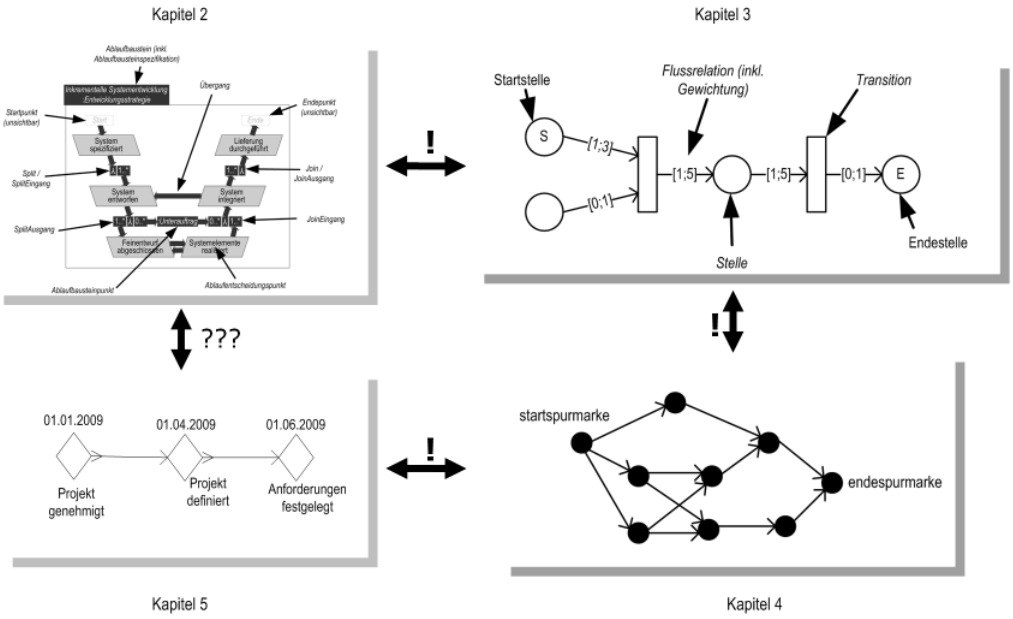

Abbildung 1.7: Übersicht über Aufbau der Beschreibung

2 Ablaufbausteine

Das Metamodell des V-Modell XT findet sich im Überblick in [KT09]. Dort beschreiben Kuhrmann und Ternité sowohl die Modellierung als UML-Modell als auch die technische Umsetzung als XML-Schema. Abbildung 2.1 zeigt die dort verwendete und an UML-Klassendiagramme angelehnte Überblicksgrafik.

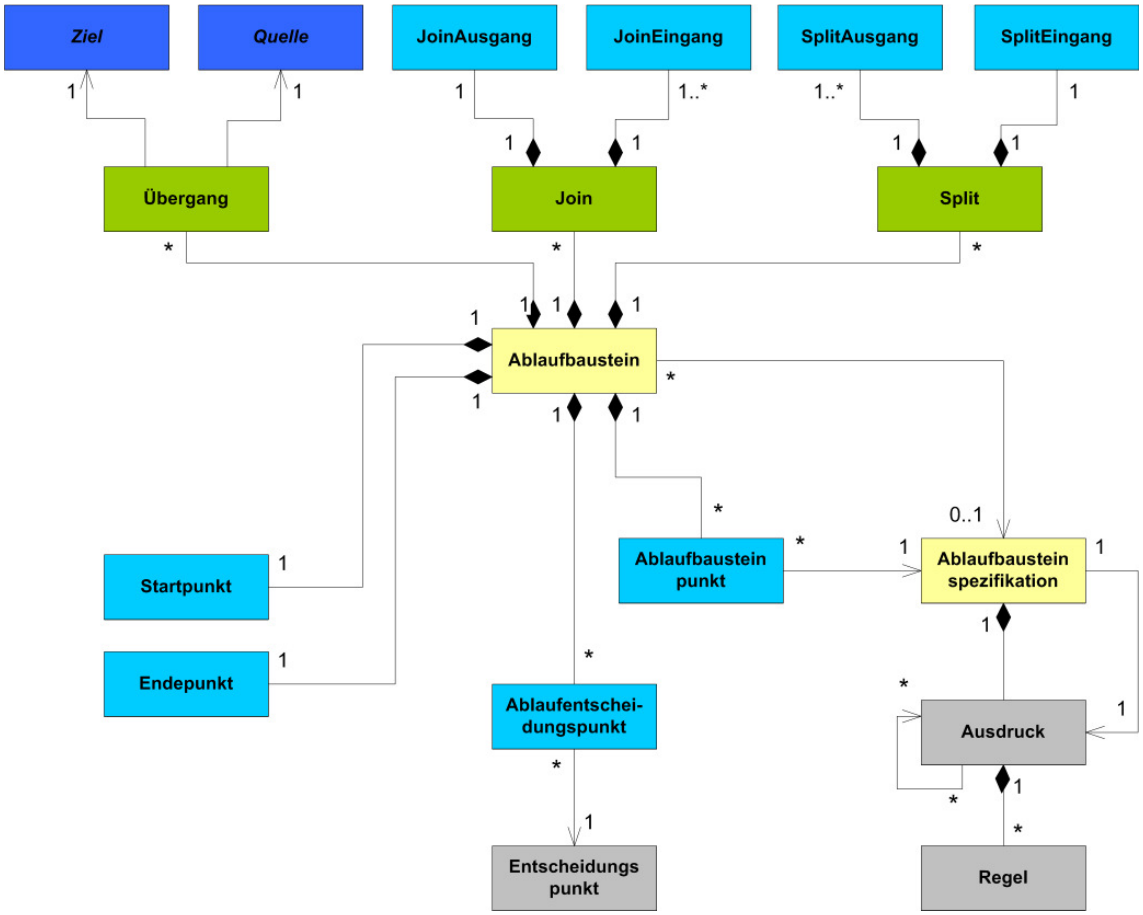

Abbildung 2.1: Das Paket Dynamik im V-Modell-XT-Metamodell (aus [KT09], Farben geändert)

Als mathematische Beschreibungsform der verwendeten Klassen und Assoziationen verwenden wir einfache Mengen, Vereinigungsmengen, Relationen (bzw. Funktionen) und prädikatenlogische Ausdrücke. Diese Beschreibungsformen decken sich mit den Klassen und Assoziationen aus Abbildung 2.1 wie folgt:

1. Gelbe und hellblaue Klassen realisieren wir durch einfache Mengen.
2. Die hellblauen Klassen bilden zusammen als Vereinigungsmenge die Menge der *Ablaufknoten*.
3. Die dunkelblauen Klassen definieren wir als Vereinigungsmenge von jeweils fünf hellblauen Klassen und damit folglich auch als Teilmenge der *Ablaufknoten*.
4. Die Kompositionsbeziehungen zwischen *Ablaufbausteinen* und den hellblauen Klassen (teilweise tranistiv über *Split* und *Join*) realisieren wir als totale Funktionen über den jeweiligen Mengen.
5. Die grünen Klassen und ihre Assoziationen realisieren wir über Relationen, Funktionen und prädikatenlogische Ausdrücke.
6. Die grauen Klassen berücksichtigen wir nicht.

2.1 Definitionen

Basiskonzepte Wir bezeichnen das 19-Tupel $VMA = (Ablauf, Typ, Start, Ende, Jein, Jaus, Sein, Saus, Alep, Sub, wurzel, typ, ablauf, subtyp, join, jmult, split, smult, \longrightarrow)$ als Ablaufbeschreibung im Sinne des VMXT, wenn folgendes gilt:

Sei *Ablauf* die endliche Menge der Ablaufbausteine im Tailoring. (2.1)

Sei *Typ* die endliche Menge der Ablaufbausteintypen im Tailoring. (2.2)

Sei $wurzel \in Ablauf$ der Wurzel-Ablaufbaustein nach dem Tailoring. (2.3)

Jeder Ablaufbaustein hat dabei genau einen Ablaufbausteintyp, was durch die folgende totale Funktion ausgedrückt wird.

$$typ \quad : \quad Ablauf \rightarrow Typ \tag{2.4}$$

Knoten Im folgenden werden die Inhalte eines Ablaufbausteins beschrieben. Dazu zunächst die Definition der grundlegenden, paarweise disjunkten Mengen:

Sei *Start* die endliche Menge der Startpunkte. (2.5)

Sei *Ende* die endliche Menge der Endepunkte. (2.6)

Sei *Jein* die endliche Menge der Join-Eingänge. (2.7)

Sei *Jaus* die endliche Menge der Join-Ausgänge. (2.8)

Sei *Sein* die endliche Menge der Split-Eingänge. (2.9)

Sei *Saus* die endliche Menge der Split-Ausgänge. (2.10)

Sei *Alep* die endliche Menge der Ablaufentscheidungspunkte. (2.11)

Sei *Sub* die endliche Menge der Unterablaufpunkte. (2.12)

Die Menge aller Ablaufknoten *Alk* [Wik09] sei gegeben durch

$$Alk \quad = \quad Start \cup Ende \cup Jein \cup Jaus \cup Sein \cup Saus \cup Alep \cup Sub \tag{2.13}$$

Die totale Funktion

$$ablauf \quad : \quad Alk \rightarrow Ablauf \tag{2.14}$$

bezeichne die Zuordnung von Ablaufknoten zu Ablaufbausteinen. Der Ausdruck $ablauf(k) = a$ bedeutet zum Beispiel, dass Knoten k in dem Ablaufbaustein a enthalten ist. Jeder Ablaufknoten gehört also genau einem Ablaufbaustein an.

Unterablaufpunkte kann man sich als Aufruf anderer Ablaufbausteine vorstellen. Dabei sind Varianten möglich: Der aufgerufene Ablaufbaustein wird nicht direkt aufgerufen, sondern auf dem Umweg über seinen Typ. Die totale Funktion

$$subtyp \quad : \quad Sub \rightarrow Typ \tag{2.15}$$

ordnet jedem Unterablaufpunkt genau einen Ablaufbausteintyp zu. Die totale Funktion

$$varianten \quad : \quad Sub \rightarrow \wp(Ablauf) \tag{2.16}$$

gibt nun die möglichen Ablaufbaustein-Varianten für einen Unterablaufpunkt an. Sie ist wie folgt definiert:

$$varianten(s) \quad = \quad \{al \in Ablauf \mid typ(al) = subtyp(s)\} \tag{2.17}$$

Joins und Splits Ein Join-Ausgang besitzt eine endliche, nichtleere Menge von zugehörigen Join-Eingängen. Dabei ist jeder Join-Eingang mit genau einem Join-Ausgang verbunden. Die totale Funktion

$$join \quad : \quad Jein \rightarrow Jaus \tag{2.18}$$

ordnet jedem Join-Eingang seinen Join-Ausgang zu.
Die totale Funktion

$$jmult \quad : \quad Jein \rightarrow [\![\mathbb{N}_0, \mathbb{N}_1^\infty]\!] \tag{2.19}$$

gibt die Vielfachheiten von Join-Eingängen an. Eine Vielfachheit entspricht mathematisch einem nichtleeren Intervall auf der Menge der natürlichen Zahlen, wobei die untere Grenze 0 und die obere Grenze ∞ eingeschlossen sein können. Für den Typ dieser Intervalle benutzen wir die Schreibweise $[\![\mathbb{N}_0, \mathbb{N}_1^\infty]\!]$. Es gilt also beispielsweise $[2;4] \in [\![\mathbb{N}_0, \mathbb{N}_1^\infty]\!]$.
Für Split-Eingänge und Split-Ausgänge wird analog zu Joins die totale Funktion

$$split \quad : \quad Saus \rightarrow Sein \tag{2.20}$$

definiert. Sie ordnet jeden Split-Ausgang eindeutig genau einem Split-Eingang zu.

$$smult \quad : \quad Saus \rightarrow [\![\mathbb{N}_0, \mathbb{N}_1^\infty]\!] \tag{2.21}$$

Übergänge Ablaufbausteine definieren nicht nur Ablaufknoten, sondern auch Übergänge zwischen diesen. Ein Übergang besitzt eine Quelle und ein Ziel. Dabei kann aber nicht jeder Ablaufknoten sowohl Quelle als auch Ziel sein:

$$Quelle \quad = \quad Start \cup Jaus \cup Saus \cup Alep \cup Sub \tag{2.22}$$
$$Ziel \quad = \quad Ende \cup Jein \cup Sein \cup Alep \cup Sub \tag{2.23}$$

Die Relation

$$\longrightarrow \quad \subseteq \quad Quelle \times Ziel \tag{2.24}$$

bezeichne die *Übergangsbeziehungen* auf den Knoten eines Ablaufs. Wir schreiben also beispielsweise $a \longrightarrow b$, wenn Ablaufentscheidungspunkt b Nachfolger von Ablaufentscheidungspunkt a ist. Graphisch wird dies durch einen Pfeil vom Quelle-Knoten zum Ziel-Knoten dargestellt.

Kurzschreibweisen Um die folgenden Kapitel möglichst kurz und prägnant zu halten, wollen wir hier noch Abkürzungen für zwei Sachverhalte einführen.
Die Relationen

$$\xrightarrow{ein} \quad \subseteq \quad Sub \times Start \tag{2.25}$$
$$\xrightarrow{aus} \quad \subseteq \quad Ende \times Sub \tag{2.26}$$

beschreiben den Wechsel (oder das Springen) in einen Unterablaufbaustein hinein und wieder heraus. Sie sind wie folgt definiert:

$$e \xrightarrow{ein} f \;\Leftrightarrow\; f \in \{alstart(x) \mid x \in varianten(e)\} \tag{2.27}$$
$$e \xrightarrow{aus} f \;\Leftrightarrow\; e \in \{alende(x) \mid x \in varianten(f)\} \tag{2.28}$$

Die Relationen

$$\xrightarrow{split} \quad \subseteq \quad Sein \times \wp(Saus) \tag{2.29}$$
$$\xrightarrow{join} \quad \subseteq \quad \wp(Jein) \times Jaus \tag{2.30}$$

beschreiben jeweils zusammengehörige Ein- und Ausgänge von Splits und Joins:

$$se \xrightarrow{split} Sa \;\Leftrightarrow\; Sa = \{sa \in Saus \mid split(sa) = se\} \tag{2.31}$$
$$Je \xrightarrow{join} ja \;\Leftrightarrow\; Je = \{je \in Jein \mid join(je) = ja\} \tag{2.32}$$

Schließlich definieren wir die Menge der Sprünge in Ablaufbausteinen wie folgt:

$$Sprung \quad = \quad \longrightarrow \cup \xrightarrow{join} \cup \xrightarrow{split} \cup \xrightarrow{ein} \cup \xrightarrow{aus} \tag{2.33}$$

Ein Sprung dient als Oberbegriff für Übergänge, Joins, Splits und das Wechseln zwischen Ablaufbausteinen und Unterablaufbausteinen.

2.2 Konsistenzbedingungen

Eindeutige Start- und Endepunkte Jeder Ablaufbaustein enthält genau einen Start- und einen Endepunkt. Die beiden totalen Funktionen

$$alstart \quad : \quad Ablauf \rightarrow Start \tag{2.34}$$
$$alende \quad : \quad Ablauf \rightarrow Ende \tag{2.35}$$

legen für jeden Ablaufbaustein einen Start- und einen Endepunkt fest. Diese müssen selbst im Ablaufbaustein enthalten sein:

$$ablauf(alstart(a)) = ablauf(alende(a)) = a \tag{2.36}$$

Übergänge nur innerhalb eines Ablaufbausteins Knoten können durch die Übergangsbeziehung nur innerhalb ein- und desselben Ablaufs verbunden sein, nicht ablaufübergreifend:

$$a \longrightarrow b \;\Rightarrow\; ablauf(a) = ablauf(b) \tag{2.37}$$

Aufbau von Splits und Joins Join-Ausgänge bzw. Split-Eingänge können nicht für sich alleine existieren. Zu jedem gehört mindestens ein Join-Eingang bzw. Split-Ausgang:

$$\forall ja \in Jaus : \exists je \in Jein : join(je) = ja \tag{2.38}$$
$$\forall se \in Sein : \exists sa \in Saus : split(sa) = se \tag{2.39}$$

Außerdem liegen zusammengehörige Join-Eingänge und Join-Ausgänge (bzw. Split-Eingänge und Split-Ausgänge) im selben Ablaufbaustein:

$$ablauf(je) = ablauf(join(je)) \tag{2.40}$$
$$ablauf(sa) = ablauf(split(sa)) \tag{2.41}$$

Erreichbarkeit und Zusammenhang Beim Durchlaufen eines Ablaufbausteins sind nicht nur die explizit durch Pfeile dargestellten Übergangsbeziehungen relevant, sondern auch die impliziten Übergänge von Join- bzw. Split-Eingängen zu Join- bzw. Split-Ausgängen. Die Relation

$$\xrightarrow{al} \quad \subseteq \quad Alk \times Alk \tag{2.42}$$

bezieht alle Übergänge zwischen Ablaufknoten innerhalb eines Ablaufbausteins ein:

$$a \xrightarrow{al} b \;\Leftrightarrow\; a \longrightarrow b \vee join(a) = b \vee split(b) = a \tag{2.43}$$

Darauf aufbauend kann die Erreichbarkeit

$$\xrightarrow{al\star} \quad \subseteq \quad Alk \times Alk \tag{2.44}$$

innerhalb eines Ablaufbausteins als reflexive, transitive Hülle aller Übergänge definiert werden:

$$a \xrightarrow{al\star} b \;\Leftrightarrow\; a = b \vee a \xrightarrow{al} b \vee \exists x \in Alk : a \xrightarrow{al} x \wedge x \xrightarrow{al\star} b \tag{2.45}$$

Auf Basis der Erreichbarkeit lassen sich weitere Bedingungen für gültige Abläufe definieren. Jeder Ablaufpunkt (außer Unterablaufpunkte) muss von einem Startpunkt oder einem Unterablaufpunkt aus erreichbar

sein. Weiterhin muss von jedem Ablaufpunkt aus (wieder ohne Unterablaufpunkte) der Endpunkt bzw. ein Unterablaufpunkt erreichbar sein:

$$\forall a \in Alk \setminus Sub : (\exists e \in Start \cup Sub : e \xrightarrow{al\star} a) \wedge (\exists e \in Ende \cup Sub : a \xrightarrow{al\star} e) \tag{2.46}$$

Für Unterablaufpunkte gelten spezielle Bedingungen. Jeder Unterablaufpunkt muss von einem Startpunkt oder einem Unterablaufpunkt erreichbar sein, oder von ihm aus muss der Endpunkt bzw. ein Unterablaufpunkt erreichbar sein:

$$\forall a \in Sub : (\exists e \in Start \cup Sub : e \xrightarrow{al\star} a) \vee (\exists e \in Ende \cup Sub : a \xrightarrow{al\star} e) \tag{2.47}$$

Daraus leiten sich die folgenden, etwas anschaulicheren Bedingungen ab:

- Startknoten müssen mindestens einen ausgehenden Übergang haben.
- Endeknoten müssen mindestens einen eingehenden Übergang haben.
- Jeder Ablaufentscheidungspunkt hat mindestens einen eingehenden und einen ausgehenden Übergang.
- Jeder Split- und Join-Eingang hat mindestens einen eingehenen Übergang.
- Jeder Split- und Join-Ausgang hat mindestens einen ausgehenden Übergang.
- Jeder Unterablaufpunkt hat mindestens einen ein- oder ausgehenden Übergang.

Wir sichern mit diesen Bedingungen die lokale, auf einzelne Ablaufbausteine bezogene Konsistenz. Dadurch ist sichergestellt, dass für jede Marke, die den Ablaufbaustein betritt, zumindest ein Weg zu einem Ausgang existiert. Wir betrachten allerdings auf diesen Wegen *keine* Kardinalitäten für Splits und Joins, da hierfür im Allgemeinen eine ablaufbausteinübergreifende Betrachtung notwendig ist. Abbildung 1.7 zeigt dies am Beispiel: während die Kardinalitäten in Ablaufbaustein *a* erfüllbar sind (drei Marken auf *Start* ergeben zwei Marken auf *Ende*, ist es in Ablaufbaustein *b* niemals möglich, dass *alle* Marken den Baustein verlassen, sobald er von einer Marke betreten wurde.

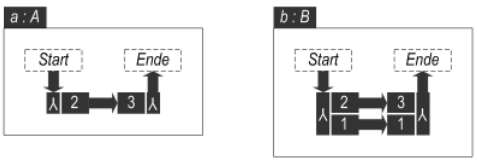

Abbildung 2.2: Fehlende Überprüfung der Kardinalitäten

3 Petri-Netze

Petri-Netze sind eine weit verbreitete und wohlverstandene Beschreibungstechnik für nebenläufige Prozesse, die starke Ähnlichkeiten zu den graphischen Abläufen des V-Modell XT aufweist.

Um die Semantik der Ablaufspezifikationen des V-Modell XT zu bestimmen, definieren wir zunächst eine Variante von Stellen-/Transitionsnetzen [Rei86]. Danach bilden wir die komplexen Modellierungskonstrukte der Ablaufspezifikationen (insbesondere die getypten Unterabläufe) auf Stellen und Transitionen ab. Anstelle hierarchisch strukturierter Ablaufbausteine treten damit „flachgeklopfte“ Stellen-/Transitionsnetze, die sich in der Folge wesentlich einfacher handhaben und auf Projektpläne abbilden lassen.

Abbildung 3.1 zeigt an einem Beispiel, wie die Abbildung von Ablaufbausteinen des V-Modell XT in Stellen-/Transitionsnetze funktioniert. Der Ablaufbaustein *Komponentenbasierte Systementwicklung* sei dabei der Wurzelablauf.

1. Ablauf-Entscheidungspunkte werden zu Stellen (in der Abbildung grau gekennzeichnet).
2. Übergangspfeile werden zu einfachen Transitionen mit jeweils einem Ein- und Ausgang.
3. Start- und Endepunkte von Abläufen werden zu Start- und Endestellen (in der Abbildung sind die Start- und Endestellen des Wurzelablaufs mit den Buchstaben S und E gekennzeichnet).
4. Joins bzw. Splits werden zu Transitionen mit den entsprechenden Vielfachheiten und einer entsprechenden Zahl von Ein- bzw. Ausgangsstellen (in der Abbildung durch schwarz umrandete Stellen gekennzeichnet).
5. Der Ein- bzw. Austritt in einen bzw. aus einem Unterablaufbaustein wird jeweils durch eine Transition mit zugeordneter Stelle abgebildet (in der Abbildung durch Stellen in Form eines liegenden und eines stehenden **O** dargestellt).

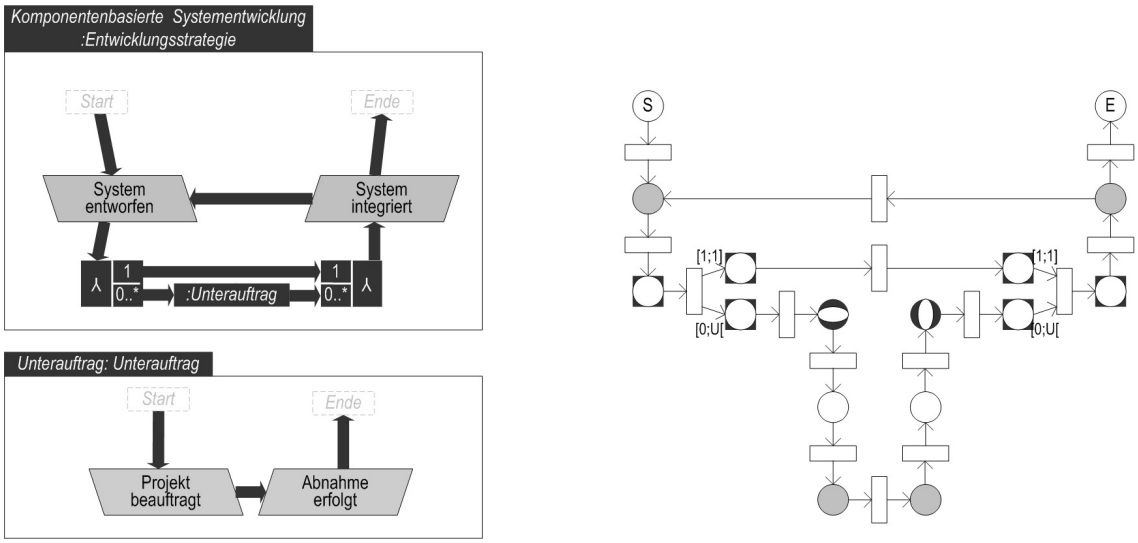

Abbildung 3.1: Abbildung von Ablaufbausteinen auf ein Stellen-/Transitionsnetz

In Beispiel aus Abbildung 3.1 gibt es nur einen passenden (Unter-)ablaufbaustein zu dem Ablaufbausteintyp *Unterauftrag*. Gäbe es hier mehrere vom Typ her passende Unterablaufbausteine, so würde jeder in eine eigene Ein- und Austrittstransition umgesetzt.

3.1 Definitionen

Basiskonzepte Ein 6-Tupel $PN = (Stelle, Transition, \overset{f}{\rightharpoonup}, gewicht, startstelle, endestelle)$ heißt Petri-Netz, wenn gilt:

$$\text{Sei } Stelle \text{ die endliche Menge der Stellen.} \tag{3.1}$$
$$\text{Sei } Transition \text{ die endliche Menge der Transitionen.} \tag{3.2}$$

Die Relation

$$\overset{f}{\rightharpoonup} \quad \subseteq \quad (Stelle \times Transition) \cup (Transition \times Stelle) \tag{3.3}$$

ist die Flussrelation.
Die totale Funktion

$$gewicht \quad : \quad \overset{f}{\rightharpoonup} \rightarrow [\![\mathbb{N}_0, \mathbb{N}_I^\infty]\!] \tag{3.4}$$

weist jedem Element der Flussrelation ein Gewichtsintervall zu.
Die Elemente

$$startstelle \quad \in \quad Stelle \tag{3.5}$$
$$endestelle \quad \in \quad Stelle \tag{3.6}$$

weisen zwei Stellen als Start- und Endestellen aus.
Im Vergleich zur konventionellen Definition bei Reisig [Rei86] haben wir drei Änderungen vorgenommen:

1. Kantengewichte sind nicht als natürliche Zahl, sondern als Intervall definiert. Dies soll ausdrücken, dass der Projektleiter während der Projektplanung eine von ihm zu wählende, passende Anzahl paralleler Teilabläufe vorsehen kann.
2. Wir verzichten auf die Angaben einer Kapazität von Stellen. Die Kapazität von Stellen ist stets unendlich. Die Anzahl der parallelen Teilabläufe im Projekt ist also grundsätzlich nicht beschränkt.
3. Die Anfangsmarkierung wird ersetzt durch das Ausweisen einer Start- und einer Endestelle. Vor der Planung des Projektstarts befindet sich als Startmarkierung genau eine Marke auf der *startstelle*, und nach der Planung des Projekts befindet sich genau eine Marke auf der *endestelle*.

Kurzschreibweisen Um in der Folge die Abbildung von Petri-Netzen auf Projektpläne zu vereinfachen, definieren wir zusätzlich Vor- und Nachbereiche von Transitionen. Jede Transition t hat einen Vorbereich $\bullet t$ und einen Nachbereich $t\bullet$, für die gilt:

$$\bullet t = \left\{ s \in Stelle \mid s \overset{f}{\rightharpoonup} t \right\} \tag{3.7}$$
$$t\bullet = \left\{ s \in Stelle \mid t \overset{f}{\rightharpoonup} s \right\} \tag{3.8}$$

3.2 Konsistenzbedingungen

Für Petri-Netze gelten die folgenden Konsistenzbedingungen:

Unterscheidung von Stellen und Transitionen Stellen und Transitionen sind disjunkte Mengen:

$$Stelle \cap Transition = \emptyset \tag{3.9}$$

Zusammenhang Wir fordern nicht, dass Petri-Netze gemäß unserer Definition zusammenhängend sind. Würden wir das fordern, müssten wir gemäß der folgenden Abbildungsvorschrift auf Ablaufbausteine im folgenden Abschnitt gleichzeitig sicherstellen, dass die Menge der Ablaufbausteine zusammenhängend ist. Die dazu notwendigen Betrachtungen gehen über den Fokus dieses Berichts hinaus und werden den Werkzeugherstellern überlassen.

3.3 Abbildung auf Ablaufbausteine

Im Folgenden geben wir an, wann ein Stellen-/Transitionsnetz wie oben definiert konsistent zu einem Ablaufbaustein des V-Modell XT ist.

Abbildungsfunktionen Die totalen Funktionen

$$alk : Stelle \rightarrow Alk \tag{3.10}$$
$$sprung : Transition \rightarrow Sprung \tag{3.11}$$

weisen jeder Stelle einen Ablaufknoten bzw. jeder Transition einen Sprung zu.
Die Mengen

$$Substart \subset Stelle \tag{3.12}$$
$$Subende \subset Stelle \tag{3.13}$$

enthalten die Start- bzw. Ende-Stellen zu den Ein- bzw. Austritten von Unterablaufbausteinen.

Konsistenz von Start- und Endestelle Start- und Endestelle des Petri-Netzes werden auf Start- und Endepunkt des Wurzelablaufbausteins abgebildet:

$$alk(startstelle) = alstart(wurzel) \tag{3.14}$$
$$alk(endestelle) = alende(wurzel) \tag{3.15}$$

Surjektivität der Abbildung *alk* Allen Ablaufknoten außer den Unterablaufpunkten ist im Petri-Netz genau eine korrespondierende Stelle zugeordnet:

$$\forall a \in Alk \setminus Sub : |\{s \mid alk(s) = a\}| = 1 \tag{3.16}$$

Jeder Unterablaufpunkt besitzt dagegen genau zwei korrespondierende Stellen (eine davon repräsentiert den Eintritt in einen Unterablauf, die andere den Austritt aus dem Unterablauf):

$$\forall a \in Sub : |\{s \mid alk(s) = a\}| = 2 \tag{3.17}$$

Die Funktion *alk* ist somit surjektiv, aber nicht injektiv. Bezüglich der Anzahl von Stellen und Ablaufknoten gilt $|Stelle| = |Alk \setminus Sub| + 2 \times |Sub| = |Alk| + |Sub|$.

Eindeutige Ein- und Austrittsstellen Die beiden Stellen, die auf einen Unterablaufpunkt abgebildet werden, müssen unterscheidbar sein. Die eine davon liegt dazu in der Menge *Substart*, die andere in der Menge *Subende*. Stellen, die mit anderen Ablaufknoten (also nicht den Unterablaufpunkten) korrespondieren, liegen in keiner der beiden Mengen.

$$s \in Substart \cup Subende \Leftrightarrow alk(s) \in Sub \tag{3.18}$$
$$Substart \cap Subende = \emptyset \tag{3.19}$$

Bijektivität der Abbildung *sprung* Die totale Funktion *sprung* bildet jede Transition in die Menge der Sprünge ab. Es muss zusätzlich gelten, dass jeder Sprung Bild einer Transition ist. Die Funktion *sprung* muss also bijektiv sein, was durch die folgenden Bedingungen gesichert wird:

$$sprung(t_1) = sprung(t_2) \Rightarrow t_1 = t_2 \tag{3.20}$$
$$\forall s \in Sprung : \exists t \in Transition : sprung(t) = s \tag{3.21}$$

Konsistenz der Flussrelation Bisher ist sichergestellt, dass die Funktion *alk* Stellen korrekt auf Ablaufknoten und die Funktion *sprung* Transitionen korrekt auf Sprünge abbildet. Es fehlt noch die korrekte Verbindung von Stellen und Transitionen durch die Flussrelation. Diese ist wie folgt definiert:

$$\begin{aligned} s \xrightarrow{f} t \;\Leftrightarrow\; & \exists (a \longrightarrow b) : alk(s) = a \wedge sprung(t) = (a \longrightarrow b) \\ & \vee (\exists (a \xrightarrow{ein} b) : alk(s) = a \wedge sprung(t) = (a \xrightarrow{ein} b) \wedge s \in Substart) \\ & \vee (\exists (a \xrightarrow{aus} b) : alk(s) = a \wedge sprung(t) = (a \xrightarrow{aus} b)) \\ & \vee (\exists (se \xrightarrow{split} Sa) : alk(s) = se \wedge sprung(t) = (se \xrightarrow{split} Sa)) \\ & \vee (\exists (Je \xrightarrow{join} ja) : alk(s) \in Je \wedge sprung(t) = (Je \xrightarrow{join} ja) \end{aligned} \tag{3.22}$$

$$\begin{aligned} t \xrightarrow{f} s \;\Leftrightarrow\; & \exists (a \longrightarrow b) : alk(s) = b \wedge sprung(t) = (a \longrightarrow b) \\ & \vee (\exists (a \xrightarrow{ein} b) : alk(s) = b \wedge sprung(t) = (a \xrightarrow{ein} b)) \\ & \vee (\exists (a \xrightarrow{aus} b) : alk(s) = b \wedge sprung(t) = (a \xrightarrow{aus} b) \wedge s \in Subende) \\ & \vee (\exists (se \xrightarrow{split} Sa) : alk(s) \in Sa \wedge sprung(t) = (se \xrightarrow{split} Sa)) \\ & \vee (\exists (Je \xrightarrow{join} ja) : alk(s) = ja \wedge sprung(t) = (Je \xrightarrow{join} ja) \end{aligned} \tag{3.23}$$

Konsistenz der Gewichtung Schließlich muss noch das Gewichtsintervall für jedes Element der Flussrelation bestimmt werden. Dieses ist außer bei Split-Ausgängen und Join-Eingängen stets $[1;1]$. Für Elemente der Flussrelation, die Stellen enthalten, die auf Join-Eingänge oder Split-Ausgänge abgebildet werden, ergibt sich die Gewichtung aus den entsprechenden Funktionen *jmult* und *smult*:

$$gewicht(p \xrightarrow{f} q) \quad = \quad \begin{cases} jmult(alk(p)), & \text{falls } alk(p) \in Jein \\ smult(alk(q)), & \text{falls } alk(q) \in Saus \\ [1;1] & \text{sonst} \end{cases} \tag{3.24}$$

4 Petri-Spuren

Formal ist eine Petri-Spur ein gerichteter, azyklischer, zusammenhängender Graph, dessen Knoten wir Spurknoten nennen. Anschaulich ergibt sie sich aus einem Stellen-/Transitionsnetz, wenn man es „abspielt“ und dabei den Markenfluss aufzeichnet. Ein neuer Knoten wird immer genau dann hinzugefügt, wenn im Petri-Netz eine Marke auf einer Stelle zu liegen kommt. Dabei werden Kanten von den Vorgänger-Spurknoten hinzugefügt, deren Marken bei der Erstellung der neuen Marke weggenommen wurden. Eine Petri-Spur ähnelt damit der Netzdarstellung eines Prozesses [Rei86], enthält allerdings im Unterschied dazu keine Transitionen.

Die folgende Abbildung zeigt beispielhaft ein Petri-Netz und eine dazu konsistente Petri-Spur. Zu Beginn liegt nur eine Marke auf der Startstelle A. Beim ersten Schaltvorgang wird diese Marke weggenommen und eine neue Marke auf die Stelle B gelegt. In der Petri-Spur rechts ergeben sich daraus die beiden verbundenen Spurknoten A und B oben. Das restliche Petri-Netz wird auf ähnliche Weise durchlaufen, nur dass bei einem Split gleich mehrere Nachfolger-Spurknoten entstehen können und bei einem Join mehrere Vorgänger zusammengeführt werden können. Die Anzahl dieser Nachfolger oder Vorgänger hängt von den Vielfachheiten im Petri-Netz ab und ist typischerweise nicht im Vorhinein festgelegt: Im Beispiel hat der erste Spurknoten C (oben links in der Petri-Spur) zwei Nachfolgerknoten E, während der zweite Spurknoten C (im rechten unteren Bereich der Petri-Spur) nur einen Nachfolgerknoten E hat. Zu einem Petri-Netz wird es also typischerweise mehrere dazu konsistente Petri-Spuren geben.

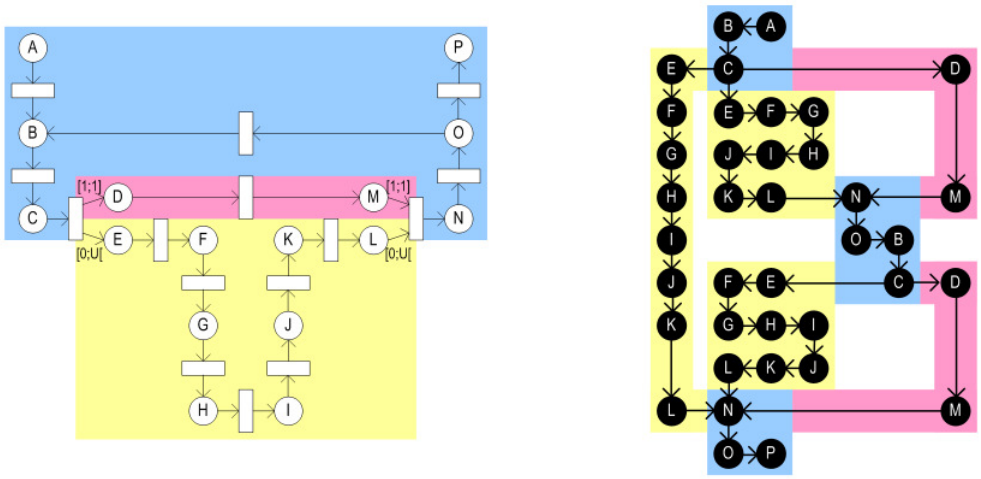

Abbildung 4.1: Abbildung eines Petri-Netzes auf eine dazu konsistente Petri-Spur

Um die Abbildung auf V-Modell-konforme Projektpläne mit einem einzigen Projektstart und einem einzigen Projektende vorzubereiten, schränken wir die Menge der gültigen Petri-Spuren ein und fordern, dass Petri-Spuren zwei ausgezeichnete Spurknoten mit den folgenden Eigenschaften besitzen: Vom sogenannten Startspurknoten aus können alle anderen Spurknoten erreicht werden, und von allen Spurknoten aus kann der sogenannte Endespurknoten erreicht werden.

4.1 Definitionen

Basiskonzepte Ein Quadrupel $PS = (Spuk, \hookrightarrow, startspuk, endespuk)$ heißt Petri-Spur, wenn gilt:

Sei $Spuk$ die endliche Menge der Spurknoten. (4.1)

Die Relation

$$\hookrightarrow \quad \subseteq \quad Spuk \times Spuk \tag{4.2}$$

definiert die Nachfolgerbeziehung auf der Menge der Spurknoten.
Die Elemente

$$startspuk \quad \in \quad Spuk \tag{4.3}$$
$$endespuk \quad \in \quad Spuk \tag{4.4}$$

zeichnen zwei Spurknoten als Start- und Endespurknoten aus.

Kurzschreibweisen Die Menge der Ablaufentscheidungspunkt-Spurknoten *Alepspuk* wird als Teilmenge der Spurknoten definiert. Sie enthält diejenigen Spurknoten, deren zugehörige Stelle auf einen Ablaufentscheidungspunkt verweist:

$$Alepspuk \quad = \quad \{k \in Spuk \mid alk(stelle(k)) \in Alep\} \tag{4.5}$$

4.2 Konsistenzbedingungen

Für Petri-Spuren müssen die folgenden Konsistenzbedingungen erfüllt sein:

Zyklenfreiheit Für die Definition der Zyklenfreiheit definieren wir zunächst die Erreichbarkeit

$$\overset{+}{\hookrightarrow} \quad \subseteq \quad Spuk \times Spuk \tag{4.6}$$

innerhalb einer Petri-Spur als transitive Hülle der Relation $\hookrightarrow$:

$$k \overset{+}{\hookrightarrow} l \;\Leftrightarrow\; k \hookrightarrow l \vee \exists x \in Spuk : k \hookrightarrow x \wedge x \overset{+}{\hookrightarrow} l \tag{4.7}$$

Somit ergibt sich Zyklenfreiheit wie folgt:

$$\neg(k \overset{+}{\hookrightarrow} k) \tag{4.8}$$

Zusammenhang Wie oben bereits erwähnt fordern wir, dass alle Spurknoten einer Petri-Spur vom Startspurknoten aus erreichbar sind und dass der Endespurknoten von allen anderen Spurknoten aus erreichbar ist:

$$m \neq startspuk \;\Rightarrow\; startspuk \overset{+}{\hookrightarrow} m \tag{4.9}$$
$$m \neq endespuk \;\Rightarrow\; m \overset{+}{\hookrightarrow} endespuk \tag{4.10}$$

Zusammen mit den Bedingungen für die Konsistenz von Petri-Netzen und Petri-Spuren schließt diese Bedingung die Erzeugung von Marken „aus dem Nichts“ aus. Dieser Fall könnte beispielsweise bei einem Join mit den beiden Eingangsvielfachheiten $[0;1]$ und $[0;\infty]$ relevant sein. Durch den Zusammenhang der Petri-Spur ist sichergestellt, dass zumindest eine Marke auf einem der beiden Eingänge eingehen muss, damit der Join schaltet. Analoges gilt für den Ausschluss der Vernichtung von Marken „ins Nichts“ bei Splits.

Schaltbarkeit Der Vollständigkeit halber definieren wir eine weitere Konsistenzbedingung, die alle korrekt aus Petri-Netzen abgeleiteten Petri-Spuren erfüllen: Die Vorgänger- und Nachfolgerspurknoten, die zu einem Schaltvorgang im Petri-Netz gehören, müssen immer vollständig verbunden sein. Jede Marke im Vorbereich der entsprechenden Transition wirkt bei der Erzeugung aller Marken im Nachbereich mit.

$$a \hookrightarrow c \wedge a \hookrightarrow d \wedge b \hookrightarrow d \;\Rightarrow\; b \hookrightarrow c \tag{4.11}$$

4.3 Abbildung auf Petri-Netze

Im Folgenden geben wir an, wann eine so definierte Petri-Spur konsistent zu einem Petri-Netz ist.

Abbildungsfunktionen Die totalen Funktionen

$$stelle \quad : \quad Spuk \to Stelle \tag{4.12}$$
$$transition \quad : \quad \hookrightarrow \to Transition \tag{4.13}$$

weisen jedem Spurknoten bzw. jedem Übergang einer Petri-Spur genau eine Stelle bzw. eine Transition im zugehörigen Petri-Netz zu.

Konsistenz von Start- und Endespurknoten Start- und Endespurknoten der Petri-Spur werden auf Start- und Endestelle des Petri-Netzes abgebildet:

$$stelle(startspuk) = startstelle \tag{4.14}$$
$$stelle(endespuk) = endestelle \tag{4.15}$$

Eindeutigkeit von Transitionen Eine Marke kann nur durch genau eine Transition erzeugt werden. Alle eingehenden Kanten eines Spurknotens werden deshalb jeweils auf die selbe Transition abgebildet. Gleiches gilt für die ausgehenden Kanten:

$$transition(k \hookrightarrow m) = transition(l \hookrightarrow m) = transition(k \hookrightarrow n) \tag{4.16}$$

Konsistenz der Spur-Relation Die Elemente der Spur-Relation müssen zu dem Petri-Netz passen. Das wird durch folgende Bedingung sichergestellt:

$$transition(k \hookrightarrow l) = t \;\Leftrightarrow\; stelle(k) \in \bullet t \wedge stelle(l) \in t\bullet \tag{4.17}$$

Konsistenz der Gewichtung Die Anzahl der ein- und ausgehenden Verbindungen in der Petri-Spur muss zu den entsprechenden Vielfachheiten der zugehörigen Transitionen passen. Dies wird durch die folgenden Bedingungen sichergestellt:

$$\begin{aligned} &transition(k \hookrightarrow l) = t \;\Rightarrow \\ &\quad \forall s_v \in \bullet t : |\{m \hookrightarrow l \mid stelle(m) = s_v\}| \in gewicht(s_v \xrightarrow{f} t) \wedge \\ &\quad \forall s_n \in t\bullet : |\{k \hookrightarrow n \mid stelle(n) = s_n\}| \in gewicht(t \xrightarrow{f} s_n) \end{aligned} \tag{4.18}$$

5 Projektpläne

Gemäß dem V-Modell XT erstellen Projektleiter ihre Projektpläne auf der Grundlage der im Projekt verfügbaren Ablaufbausteine. Projektpläne enthalten in der Regel Vorgänge und Meilensteine, die zeitlich geplant sind und über Nachfolgerbeziehungen miteinander verbunden sind. Zur Definition der Semantik der Ablaufbausteine des V-Modell XT genügen allerdings Meilensteine und deren Beziehungen – im Folgenden werden wir deshalb Vorgänge nicht betrachten.

Die folgende Abbildung zeigt eine Petri-Spur mit einer konsistenten Abbildung auf einen Projektplan. Teil a) zeigt die Petri-Spur aus Abbildung 4.1, wobei die Spurknoten zu Ablaufentscheidungspunkten grau eingefärbt sind. Teil b) reduziert die Petri-Spur auf diese grauen Knoten und blendet alle anderen Spurknoten aus. Außerdem ist für jeden grauen Knoten der zugehörige Meilenstein im dargestellten Projektplan notiert. Teil c) zeigt schließlich einen passenden Projektplan, in dem die Meilensteine als Diamanten dargestellt sind. Man erkennt deutlich, dass der Projektplan strenger ist als das Vorgehensmodell (oder die Petri-Spur) vorschreibt: Während das Vorgehensmodell die Meilensteinreihenfolgen $f \rightarrowtail g$ und $e \rightarrowtail g$ fordert, legt der Projektleiter die Reihenfolge $e \rightarrowtail f \rightarrowtail g$ fest und ordnet damit e und f zeitlich an.

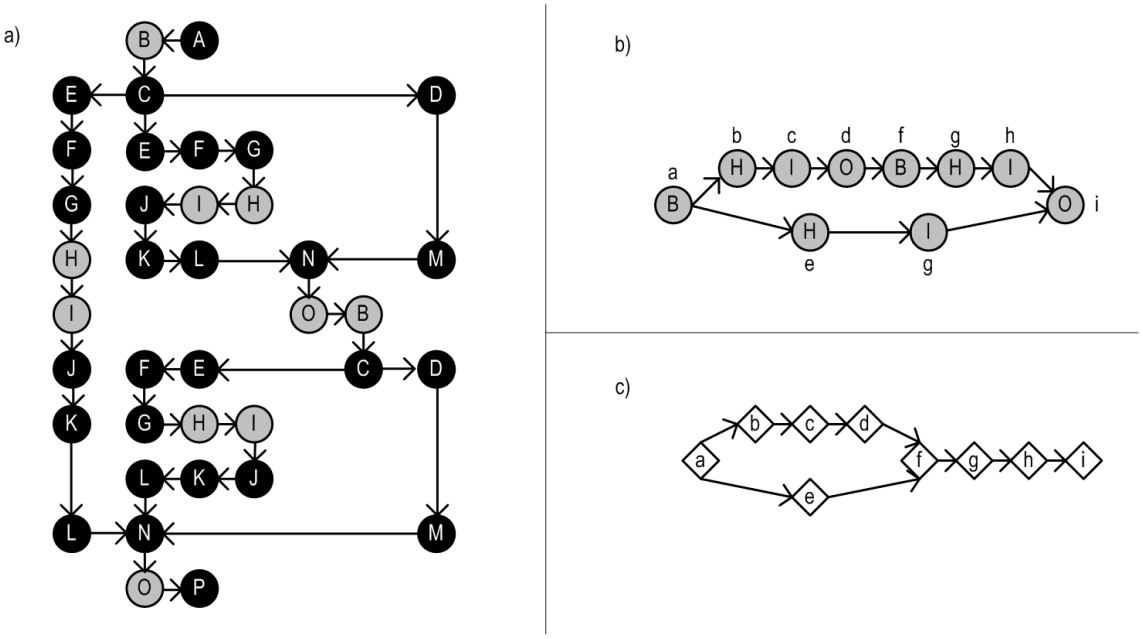

Abbildung 5.1: Petri-Spur und konsistente Abbildung auf Projektplan

Natürlich gibt es viele Möglichkeiten, gültige Projektpläne aus der Petri-Spur zu erzeugen. Beispielsweise wäre auch ein Projektplan zulässig, der wie in Teil b) der Abbildung gestaltet wäre.

5.1 Definitionen

Basiskonzepte Ein 6-Tupel $PP = (Meilenstein, \rightarrowtail, startstein, endestein, termin, aleps)$ heißt Projektplan, wenn gilt:

Sei *Meilenstein* die endliche Menge der Meilensteine. (5.1)

Die Elemente

$$startstein \in Meilenstein \tag{5.2}$$
$$endestein \in Meilenstein \tag{5.3}$$

zeichnen zwei Meilensteine als Start- und Endemeilenstein aus.
Die Relation

$$\rightarrowtail : Meilenstein \rightarrow Meilenstein \tag{5.4}$$

definiert eine Nachfolgerbeziehung auf der Menge der Meilensteine. $q \rightarrowtail r$ bedeutet, dass der Meilenstein q erreicht werden muss, bevor der Meilenstein r erreicht werden darf.
Die totale Funktion

$$termin : Meilenstein \rightarrow Datum \tag{5.5}$$

definiert die vorgenommene Zeitplanung. $termin(p) = 20.09.2009$ bedeutet, dass der Meilenstein p auf den 20.09.2009 terminiert ist. Die Elemente aus der Menge *Datum* sind chronologisch durch die Relation $\leq_t$ total geordnet.
Die totale Funktion

$$aleps : Meilenstein \rightarrow \wp(Alep) \tag{5.6}$$

stellt den Bezug zum Vorgehensmodell her und weist jedem Meilenstein eine Menge von Ablaufentscheidungspunkten zu, die mit dem geplanten Meilenstein erfüllt werden sollen. $aleps(p) = \{System\ entworfen\}$ bedeutet, dass der Meilenstein p die Vorgaben des Vorgehensmodells zum Ablaufentscheidungspunkt *System entworfen* erfüllt. Sogenannte „freie Meilensteine“ ohne Referenz im Vorgehensmodell sind ebenfalls erlaubt. Für einen freien Meilenstein f gilt $aleps(f) = \emptyset$.

5.2 Konsistenzbedingungen

Verbot von Zeitschleifen Projektpläne müssen zyklenfrei sein, da sie einen konkreten zeitlichen Ablauf modellieren und ein Zyklus eine „Zeitschleife“ beschreiben würde. Dazu definieren wir zunächst die projektweite Nachfolgerbeziehung

$$\overset{+}{\rightarrowtail} \subseteq Meilenstein \times Meilenstein \tag{5.7}$$

innerhalb eines Projektplans als transitive Hülle der Relation $\rightarrowtail$:

$$k \overset{+}{\rightarrowtail} l \Leftrightarrow k \rightarrowtail l \vee \exists x \in Meilenstein : k \rightarrowtail x \wedge x \overset{+}{\rightarrowtail} l \tag{5.8}$$

Somit ergibt sich die Zyklenfreiheit aus der folgenden Bedingung:

$$\neg(k \overset{+}{\rightarrowtail} k) \tag{5.9}$$

Eindeutiger Anfang - eindeutiges Ende Der *startstein* muss vor, der *endestein* nach allen anderen Meilensteinen liegen. Der Projektleiter kann dies im Übrigen immer durch entsprechende freie Meilensteine am Projektanfang und -ende erreichen.

$$ms \neq startstein \Rightarrow startstein \overset{+}{\rightarrowtail} ms \tag{5.10}$$
$$ms \neq endestein \Rightarrow ms \overset{+}{\rightarrowtail} endestein \tag{5.11}$$

Korrekte Chronologie Die Relation $\rightarrowtail$ muss mit der chronologischen Reihenfolge ihrer Meilensteine zusammenpassen:

$$p \rightarrowtail q \;\Rightarrow\; termin(p) \leq_t termin(q) \tag{5.12}$$

Damit ist zugelassen, dass zwei Meilensteine zwar in einer Reihenfolgebeziehung stehen, aber zum selben Datum eingeplant werden.

5.3 Abbildung auf Petri-Spuren

Im Folgenden geben wir an, wann eine Petri-Spur konsistent zu einem Projektplan ist.

Abbildungsfunktionen Die totale Funktion

$$meilenstein \quad : \quad Alepspuk \rightarrow Meilenstein \tag{5.13}$$

weist jedem Ablaufentscheidungspunkt-Spurknoten einen korrespondierenden Meilenstein im Projektplan zu. Durch die Existenz dieser Funktion ist sichergestellt, dass jeder Ablaufentscheidungspunkt auch tatsächlich im Projektplan berücksichtigt wurde.

Konsistenz bei Ablaufentscheidungspunkten Zwischen Petri-Spuren und Ablaufbausteinen existieren (wie Abbildung 1.7 zeigt) zwei „Verbindungen“: Über die Petri-Spuren mittels der totalen Funktionen *stelle* und *alk* sowie über die Projektpläne mit den totalen Funktionen *meilenstein* und *aleps*. Als eine Konsistenzbedingung muss dabei gelten, dass beide Wege zum gleichen Ziel führen, also bei den gleichen Ablaufentscheidungspunkten enden:

$$\forall k \in Alepspuk : alk(stelle(k)) \in aleps(meilenstein(k)) \tag{5.14}$$

Keine Überladung Nach der vorangegangenen Bedingung könnte der Projektleiter einfach jedem Meilenstein alle Ablaufentscheidungspunkte zuweisen und damit sicher einen korrekten Projektplan erhalten. Um dies zu verbieten, fordern wir, dass für jeden Verweis von einem Meilenstein auf einen Ablaufentscheidungspunkt auch ein entsprechender Ablaufentscheidungspunkt-Spurknoten existiert:

$$a \in aleps(m) \;\Rightarrow\; (\exists k \in Alepspuk : meilenstein(k) = m \wedge alk(stelle(k)) = a) \tag{5.15}$$

Korrekte Meilensteinreihenfolge Damit ein Projektplan und eine Petri-Spur konsistent zueinander sind muss abschließend noch die Reihenfolgen der Spurknoten und der Meilensteine konsistent zueinander sein. Konsistent sind sie dann, wenn der Projektplan mindestens genauso restriktiv hinsichtlich der Reihenfolge ist wie die Petri-Spur. Das heißt, der Projektleiter muss im Projektplan alle Restriktionen des Vorgehensmodells berücksichtigen, darf aber selbst weitere hinzufügen.

Dazu sollen zunächst die durch die Abbildung der Petri-Spur auf einen Projektplan induzierten Restriktionen zwischen den Meilensteinen als Relation

$$\overset{\times}{\rightarrowtail} \quad \subseteq \quad Meilenstein \times Meilenstein \tag{5.16}$$

definiert werden. Dabei gilt:

$$m \overset{\times}{\rightarrowtail} n \;\Leftrightarrow\; \exists k,l \in Alepspuk : k \overset{+}{\hookrightarrow} l \wedge meilenstein(k) = m \wedge meilenstein(l) = n \tag{5.17}$$

Damit ein Projektplan mindestens so restriktiv ist wie durch die Abbildung der Petri-Spur auf diesen Projektplan gefordert, muss also gelten:

$$\overset{\times}{\rightarrowtail} \subseteq \overset{+}{\rightarrowtail} \tag{5.18}$$

6 Anwendung und Ausblick

Im vorliegenden Bericht haben wir die Projektdurchführungsstrategien des V-Modell XT formalisiert und präzise definiert, wann ein Projektplan dem Vorgehensmodell entspricht. Abschließend wollen wir darlegen, wie die Ergebnisse in der Praxis umgesetzt werden können und welche Forschungsfragen offen bleiben.

6.1 Anwendung und Werkzeugunterstützung

Der V-Modell XT Projektassistent [Bun09b] unterstützt den Projektleiter bei der Ableitung eines initialen Projektdurchführungsplans aus dem Vorgehensmodell heraus und setzt die hier gegeben Spezifikation teilweise um. Er kann Konstellationen wie in den Abbildungen 1.3 und 1.4 (fast) korrekt planen. Dagegen unterstützt er Modelle wie in den Abbildungen 1.5 und 1.6 nicht oder nur teilweise. Insbesondere die für die Erweiterung des Metamodells verantwortlichen Tunnel werden nicht in vollem Umfang unterstützt. Ziel muss es sein, den Projektassistenten und die verfügbaren kommerziellen Werkzeuge analog zu dieser Spezifikation zu implementieren, damit sich ein Prozessingenieur darauf verlassen kann, dass seine Vorgaben des Vorgehensmodells auch vom Werkzeug und dem Projektleiter umgesetzt werden.

Grundsätzlich stehen einem Werkzeug auf Basis dieser Spezifikation mehrere Arten der Planung zur Verfügung, die auch miteinander kombiniert werden können. Sie werden im Folgenden beschrieben:

Vorwärts- und Rückwärtsplanung Die gegebene Spezifikation ist hinsichtlich der Zeit und dem Verschieben von Marken symmetrisch, wie an folgenden Beispielen gezeigt wird:

- Ablaufbausteine besitzen genau einen Startpunkt und genau einen Endepunkt.
- Joins und Splits sind strukturell identisch aufgebaut.
- Petri-Netze enthalten genau eine Start- und eine Endestelle.
- Unterablaufpunkte werden auf eine Startstelle und eine Endestelle abgebildet.
- Petri-Spuren besitzen genau einen Startspurknoten und einen Endespurknoten.
- Projektpläne besitzen genau einen Start und ein Ende.

Durch die Symmetrie und der Eigenschaft, dass das Markenverschiebespiel mit genau einer Marke auf der Startstelle beginnt und mit genau einer Marke auf der Endestelle endet, ist Vorwärts- und Rückwärtsplanung aus formaler Sicht das selbe. Es ist also egal, ob der Projektleiter eine Marke auf die Startstelle setzt und diese so zieht, bis er schließlich genau eine Marke auf der Endestelle liegen hat, oder ob er den umgekehrten Weg wählt und mit dem Projektende beginnt. Ein Verfahren für die eine Richtung unterstützt automatisch auch die andere.

Vorwärts- und Rückwärtsplanung können auch kombiniert werden. Der Projektleiter beginnt dann mit einer „Vorwärtsmarke“ auf der Startstelle und einer „Rückwärtsmarke“ auf der Endestelle und „spielt“ wie in diesem Bericht angegeben. Wenn sich eine Vorwärts- und eine Rückwärtsmarke auf der selben Stelle befinden, darf der Projektleiter beide Marken aus dem Spiel nehmen. Sein Ziel ist es in diesem Fall, alle Marken aus dem Spiel zu bekommen.

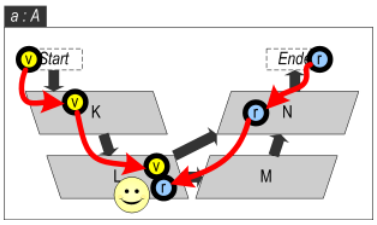

Abbildung 6.1: Beispiel für eine Zugfolge bei gleichzeitiger Vorwärts- und Rückwärtsplanung

Abbildung 6.1 zeigt dieses Vorgehen an einem einfachen Beispiel: Zu Beginn befindet sich auf dem Startpunkt eine (gelbe) Vorwärtsmarke und auf dem Endepunkt eine (blaue) Rückwärtsmarke. Der Projektleiter bewegt die Marken dann mit bzw. gegen die Richtung der gegeben Übergänge. Auf dem Entscheidungspunkt L treffen sich beide Marken und der Projektleiter kann sie entfernen und damit die Planung erfolgreich beenden.

Modulare Planung Oftmals kann am Projektanfang nur ein unvollständiger Meilensteinplan erstellt werden, der nur die wichtigsten Meilensteine (z.B. Projektanfang, Projektende oder Releases) enthält und die genauen Meilensteine während der Entwicklung ausspart. Wenn die Ablaufbausteinstruktur entsprechend geschnitten ist (z.B. ein Baustein Projektablauf und ein Baustein Entwicklungsablauf) kann der Projektleiter vom genauen Entwicklungsvorgehen zunächst abstrahieren und annehmen, dass der Ablaufbaustein vom Typ *Entw* betreten und wieder verlassen wird. Dieses Vorgehen ist in Abbildung 6.2 dargestellt.

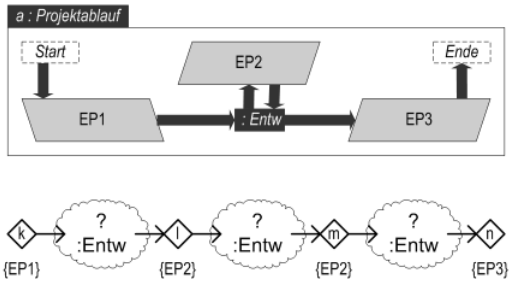

Abbildung 6.2: Vorläufige, modulare Planung auf Basis der Ablaufbausteinstruktur

Zu einem späteren Zeitpunkt kann der Projektleiter die Planung dann verfeinern und die Platzhalter-Wolken durch konkrete Meilensteinpläne ersetzen. Dabei muss er allerdings beachten, dass er im Beispiel davon ausgegangen ist, dass der Ablaufbaustein mit einer Marke betreten und wieder verlassen werden kann. Diese Zusicherung macht der Ablaufbaustein allerdings nicht. Es ist also im Allgemeinen nicht sichergestellt, dass ein vorläufiger Projektplan wie in Abbildung 6.2 zu einem vollständig konsistenten Projektplan ergänzt werden kann – Voraussetzung dafür ist, dass geeignete Vorgehensbausteine verfügbar sind.

Freie Planung mit Konsistenzüberprüfung Die beiden bisher gezeigten Szenarien gehen davon aus, einen neuen Projektplan auf Basis eines Vorgehensmodells zu erzeugen. Eine weitere Frage ist, wie man bestehende Projektpläne gegen ein Vorgehensmodell auf Konsistenz prüfen kann. Die Schwierigkeit besteht dann darin, einen Durchlauf durch das Petri-Netz zu finden, der konsistent auf den bestehenden Projektplan abgebildet werden kann. Dieses Problem wird durch zwei Umstände erschwert:

- Ein Meilenstein kann wie in Formel (5.6) angegeben auf eine Menge von Ablaufentscheidungspunkten verweisen.
- Projektpläne dürfen wie in Formel (5.18) angegeben, strenger sein als die dazugehörige Petri-Spur.

Die Herausforderung ist hier, einen performanten Algorithmus zu finden, der die Konsistenz überprüft. Idealerweise könnte der Algorithmus außerdem Planänderungen hin zu einem konsistenten Plan vorschlagen, falls Plan und Vorgehensmodell inkonsistent sind.

6.2 Weitere Forschungsaufgaben

Neben den angesprochenen Algorithmen ergeben sich aus der in diesem Bericht vorgestellten Spezifikation weitere offene Forschungsfragen.

Erweiterung um Produkt- und Organisationsmodell Trotz präziser Definition der Semantik der Projektdurchführungsstrategien ist es nicht möglich, alle unsinnigen Pläne auszuschließen. Abbildung 6.3 zeigt einen solchen unsinnigen Plan: Der Projektverlauf spaltet sich auf in einen Strang für den Client und einen für den Server. Jeder Strang spaltet sich dananch wieder auf in einen Strang für die GUI und einen Strang für die Anwendungslogik. Soweit ist der Plan korrekt. Bei der Zusammenführung der einzelnen Stränge existiert dann aber ein Meilenstein, der zeitlich von der Client-GUI und der Server-Logik abhängt und ein

Meilenstein, der zeitlich von der Client-Logik und der Server-GUI abhängt. Das ist unsinnig, weil Client-GUI und Server-Logik nicht zu einem Systembestandteil integriert werden können. Das vorgestellte Ablaufmodell kann diesen Fehler allerdings nicht verhindern, da die Marken des Petri-Netzes nicht „wissen“, auf welchen Teil der Produktstruktur sie sich beziehen.

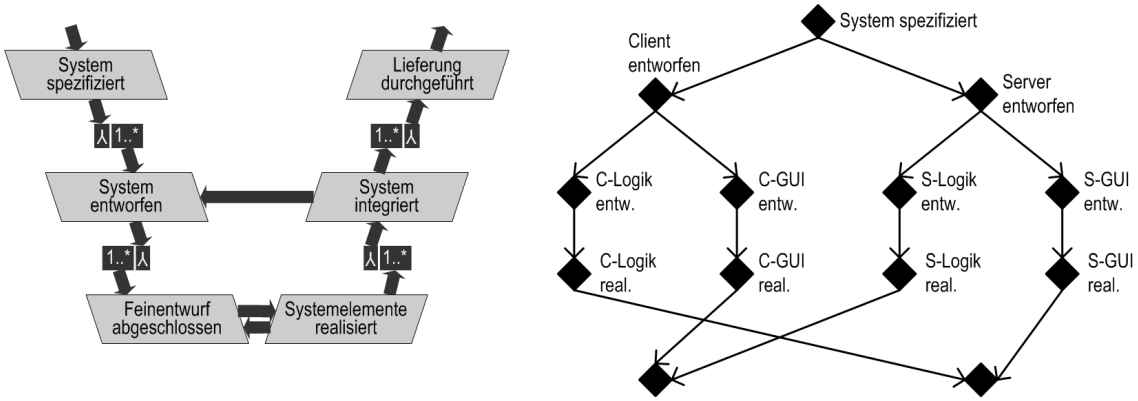

Abbildung 6.3: Sinnlose Abläufe durch fehlende Verknüpfung zum Produktmodell

Die Integration von Produktmodell und Ablaufmodell ist Aufgabe der Zukunft. Ähnliche Szenarien können im Übrigen auch in Abhängigkeit von der Organisationsstruktur beschrieben werden. Eine Integration von Organsisationsmodell und Ablaufmodell ist deshalb ebenfalls notwendig.

Weitergehende Modularisierung In unserem Ansatz ist ausschließlich die Ebene der Ablaufbausteine modular aufgebaut. Im Sinne einer Reduzierung auf das Notwendige benutzen wir für Petri-Netze, Petri-Spuren und Projektpläne „flachgeklopfte“ Strukturen (vgl. dazu auch Kapitel 3). Modulare Projektpläne würden dem Projektleiter allerdings zusätzliche Vorteile verschaffen, wenn die Modulstruktur des Projektplan der Modulstruktur des Vorgehensmodells entspricht: Der Projektleiter könnte durch mehr Struktur beispielsweise besser im Plan navigieren; außerdem kann die Umplanung bei Vorgehensmodelländerungen im Projekt (dynamisches Tailoring) erleichtert werden, da klar ist, welcher Teil des Plan welchen (geänderten oder entfallenen) Modulen entspricht.

Dynamisches Tailoring und Umplanungen In diesem Bericht gehen wir allerdings zunächst davon aus, dass das Tailoring im Verlauf des Projekts stabil bleibt. Diese Annahme entspricht aber oft nicht der Wirklichkeit: Während des Projekts ergeben sich neue Fakten und es ändern sich Ziele, sodass das Vorgehensmodell dementprechend angepasst werden muss. In unserem Modell bedeutet dies, dass sich Teile des Petri-Netzes verändern, *während* sich Marken darin befinden. Dabei können unter anderem folgende Fälle auftreten:

- Bestehende Marken müssen umgelegt werden. Interessant hierbei ist, ob man die Marken „einfach so“ umlegt, oder sie zunächst in einen unveränderten Teil des Petri-Netzes zurückzieht und von dort aus neu beginnt.
- Bestehende Marken müssen vom Spielfeld genommen werden. Dies kann passieren, wenn das veränderte Tailoring angefangene Arbeiten unnötig macht.
- Neue Marken müssen hinzugefügt werden. Dies kann immer dann passieren, wenn zusätzliche Anforderungen oder Prozesse während des Projekts hinzukommen. Dabei ist besonders interessant, wo neue Marken eingesetzt werden dürfen und wie man davon ausgehend zu einem neuen konsistenten Stand kommt.

Umplanungen durch ein geändertes Tailoring kommen in der Praxis häufig vor und sollten deshalb weiter untersucht werden.

Hierarchisches oder stufenweises Tailoring Wir gehen im vorliegenden Bericht davon aus, dass das Tailoring am Projektbeginn *vollständig* erfolgt. Viele Fragen, beispielsweise ob man Unteraufträge vergeben muss oder nicht, klären sich aber erst im Laufe des Projekts. Erst durch die Beantwortung dieser Fragen ergibt sich ein vollständiger Ablauf. Auch kann es sein, dass das Tailoring für unterschiedliche Teile des Projektablaufs unterschiedlich ist: Während das zu entwickelnde System selbst sicherheitskritisch

sein könnte und deshalb mit einer besonders aufwändigen Strategie entwickelt wird, könnte das zugehörige Unterstützungssystem unkritisch sein und mit einer einfachen Strategie entwickelt werden. Das Tailoring ist dadurch nicht mehr global gültig, sondern kann für die einzelnen Ablaufbausteine individuell festgelegt werden. Dabei ist insbesondere interessant, wie eine integrierte Modellierung der Abläufe und der Tailoringentscheidungen aussehen kann und wie die dabei entstehenden Semantik als Fortführung dieses Berichts formal beschrieben werden kann.

Danksagung

Wir danken Andreas Rausch und Thomas Ternité für interessante und anregende Diskussionen.

Literaturverzeichnis

[BCCG07] BENDRAOU, R., B. COMBEMALE, X. CREGUT und M. P GERVAIS: *Definition of an Executable SPEM 2.0*. In: *Proceedings of the 14th Asia-Pacific Software Engineering Conference*, 2007.

[Bun09a] BUNDES-CIO (DER BEAUFTRAGTE DER BUNDESREGIERUNG FÜR INFORMATIONSTECHNIK): *Das V-Modell XT Version 1.3*. http://www.v-modell-xt.de, 2009.

[Bun09b] BUNDES-CIO (DER BEAUFTRAGTE DER BUNDESREGIERUNG FÜR INFORMATIONSTECHNIK): *V-Modell XT Projektassistent: V. 1.3.1*. Apache Licence, 2009.

[Gna07] GNATZ, MICHAEL: *Vom Vorgehensmodell zum Projektplan*. VDM Verlag Dr. Müller, Saarbrücken, 1. Auflage, 2007.

[KT09] KUHRMANN, MARCO und THOMAS TERNITÉ: *Das V-Modell XT 1.3 Metamodell*. Technischer Bericht TUM-INFO-02-I0905-0/1.-FI, Technische Universität München, München, Februar 2009.

[OAS07] OASIS: *Web Services Business Process Execution Language Version 2.0*. http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.pdf, April 2007.

[Rei86] REISIG, WOLFGANG: *Petrinetze: Eine Einführung*. Springer, Berlin, 2., überarb. u. erw. Auflage, 1986.

[Wik09] WIKIPEDIA: *Riesenalk*. http://de.wikipedia.org/wiki/Riesenalk, 2009.

[Wit05] WITTGES, HOLGER: *Verbindung von Geschäftsprozessmodellierung und Workflow-Implementierung*. Deutscher Universitätsverlag, 2005.

[YL07] YUAN, FENG und MINGSHU LI: *Towards Software Process Enactment based on the SPEM2-XPDL Model Transformation*. Journal of Software, 18(9):2141–2152, 2007.