Überführung von EPK-Modellen in ausführbare Grid- und Cloud-Prozesse

Andreas Hoheisel[1], Thorsten Dollmann[2], Michael Fellmann[3]

[1] Fraunhofer-Institut für Rechnerarchitektur und Softwaretechnik (FIRST)
Kekuléstraße 7, 12489 Berlin
andreas.hoheisel@first.fraunhofer.de

[2] Institut für Wirtschaftsinformatik (IWi) im DFKI
Stuhlsatzenhausweg 3, 66123 Saarbrücken
thorsten.dollmann@iwi.dfki.de

[3] Universität Osnabrück
Institut für Informationsmanagement und Unternehmensführung (IMU)
Lehrstuhl für Informationsmanagement und Wirtschaftsinformatik
Katharinenstraße 3, 49069 Osnabrück
michael.fellmann@uos.de

Zusammenfassung: Die Überführung von fachlichen Geschäftsprozessen in technische, ausführbare IT-Prozesse bleibt aufgrund unterschiedlicher Modellierungsansätze, Zielstellungen und Abstraktionsniveaus eine Herausforderung. Dieser Artikel beschreibt ein Vorgehen, mit dem fachliche Prozesse, die als Ereignisgesteuerte Prozessketten modelliert wurden, automatisch in Petrinetz-basierte, ausführbare Prozesse übersetzt werden können. Der Ansatz bietet darüber hinaus eine automatische Abbildung der Prozesse auf verteilte IT-Ressourcen im Rahmen von Service-Orientierten Architekturen, Cloud-Plattformen und Computing-Grids unter Berücksichtigung von Aspekten wie Lastausgleich und Skalierbarkeit.

1 Einleitung

Die Überführung von Geschäftsprozessen in technische Prozesse, welche IT-unterstützt in verteilten Systemen ausgeführt werden sollen, ist derzeit meist umständlich und nur zum geringen Teil automatisiert. Der Grund hierfür liegt insbesondere in der Kluft zwischen der oftmals informalen und abstrakten Modellierung von Geschäftsprozessen einerseits, und der formalen, mit konkreten technischen Details versehenen Modellierung von IT-Prozessen andererseits. Neben dem unterschiedlichen Abstraktionsniveau sind beide Arten der Prozessbeschreibung auch in der Regel organisatorisch unterschiedlich im Unternehmen verankert und unterscheiden sich in Bezug auf den Zeitpunkt ihrer Durchführung. So wird in den entsprechenden Softwareentwicklungsprojekten zum einen sehr früh und zum anderen sehr spät modelliert. Die frühe Modellierung umfasst die Spezifikation eines Systems aus fachlicher Sicht (z. B. Lastenheft). Die späte Modellierung definiert die Komponenten und deren Zusammenspiel aus einer technischen

Sicht und ist oft eng verbunden mit den zur Implementierung verwendeten Entwicklungsumgebungen. Durch die bestehende Lücke zwischen fachlichem und technischem Modell ist eine konsistente Überführung fachlicher Anforderungen in unterstützende IT-Systeme nicht gewährleistet. Die mangelnde Durchgängigkeit der Modellierung von der fachlichen zur technischen Ebene birgt – neben dem Risiko einer Mehrfachimplementierung von identischen Funktionalitäten, die aus fachlicher Sicht in verschiedenen Modellen unterschiedlich beschrieben werden – auch eine Synchronisationsproblematik zwischen fachlichem und technischen Prozessmodell. So spiegeln sich beispielsweise aufgrund betriebswirtschaftlicher Notwendigkeiten vorgenommene Ad-hoc-Änderungen an den Geschäftsprozessen zwar in den Implementierungsmodellen wieder, nicht aber in den fachlichen Modellen.

Weitere Unterschiede bestehen hinsichtlich des Gegenstandsbereichs. Während fachliche Prozessbeschreibungen beispielsweise Ziele und organisatorische Zuständigkeiten aufzeigen, sind für die technischen Prozesse die Datenflüsse und die Aufteilung von Ressourcen wie etwa Rechenleistung von zentraler Bedeutung. In der Summe führen die genannten Unterschiede und Probleme dazu, dass für die Modellierung von Geschäftsprozessen und technischen (ausführbaren) Prozessen in der Regel unterschiedliche und oftmals inkompatible Ansätze verwendet werden, die nur schwer zu überbrücken sind.

Unser Beitrag besteht darin, ein möglichst einfaches Vorgehen aufzuzeigen, um Geschäftsprozessmodelle in IT-Prozesse zu überführen, die in unterschiedlichen verteilten Systemen, wie zum Beispiel Service-Orientierten Architekturen (SOA), Cloud-Plattformen oder Computing-Grids zur Ausführung kommen können. Der vorgestellte Ansatz zeichnet sich *erstens* durch ein generatives Verfahren aus, bei dem ausgehend von einem fachlichen Prozessmodell vollautomatisch anhand der zuvor in der Workflow-Engine hinterlegten Konfigurationsparameter ein ausführbares Modell erzeugt wird. Zur fachlichen Modellierung wird hierbei die betriebswirtschaftlich orientierte, semiformale Modellierungssprache EPK (Ereignisgesteuerten Prozesskette) [KS92] verwendet, welche in Wissenschaft und Praxis weite Verbreitung erlangt hat. *Zweitens* zeichnet sich der Ansatz dadurch aus, dass die im Grid- und Cloud-Computing relevanten Aspekte wie Lastausgleich und Skalierung der Ressourcen berücksichtigt werden.

Der Artikel ist wie folgt aufgebaut. Zunächst werden im Abschnitt 2 Alternativen zu dem in diesem Beitrag vorgestellten Technologieansatz erläutert. Anschließend erfolgt eine Einführung in das Speicher- und Austauschformat EPML, das zur Repräsentation von auszuführenden EPK-Modellen genutzt wird. Auf die grundsätzlichen Besonderheiten hinsichtlich der Ausführung von Prozessen in verteilten Systemen geht anschließend Abschnitt 4 ein. Der beschrittene Weg der Transformation von EPK-Modellen in ausführbare Prozesse wird in Abschnitt 5 detailliert beschrieben und in Abschnitt 6 anhand einer Fallstudie beispielhaft veranschaulicht.

2 Verwandte Ansätze

Um dem im vorhergehenden Abschnitt beschriebenen Problem einer mangelnden Durchgängigkeit der Modelle von der fachlichen zur technischen Ebene zu begegnen, ist ein in Wissenschaft und Praxis vielfach reflektierter Weg die Verwendung von BPMN (Business Process Modeling Notation) [Ob09] zur Modellierung der Geschäftsprozesse und die anschließende Überführung in WS-BPEL (Web Services Business Process Execution Language) [Oa07] anhand der im Anhang A des BPMN-Standards enthaltenen Transformationsregeln. Eine Implementierung für eine Untermenge der durch die BPMN-Spezifikation definierten Sprachkonstrukte steht zum Beispiel durch das Projekt Babel zur Verfügung (http://www.bpm.fit.qut.edu.au/research/projects/babel/). Gleichwohl hiermit prinzipiell eine Durchgängigkeit im Sinne eines über alle Phasen des Entwicklungszyklus existierenden Prozessmodells erreicht wird, zeigt dieses Vorgehen jedoch für einige Anwendungsfälle gewichtige Nachteile:

- Sowohl WS-BPEL als auch BPMN haben eine relativ komplexe Semantik und sehr umfangreiche Syntax, was zu einer komplexen und fehlerträchtigen Transformation führt, die auch eine Überprüfung der Modelle hinsichtlich formaler Kriterien (Model Checking) erschwert. So besitzt WS-BPEL zum Beispiel drei Möglichkeiten um eine Schleife zu definieren (while, repeatUntil, forEach), obwohl aus technischer Sicht ein Schleifenkonstrukt ausreichen würde. Dies führt dazu, dass eine Überführung von BPMN-Modellen in ausführbare Modelle typischerweise manuelle Arbeitsschritte erfordert, die eine enge Kopplung zwischen Modellierung und Ausführung verhindern und folglich den Aufwand zur Unterstützung dynamischer Prozesse erhöhen.

- WS-BPEL ist ohne spezifische Erweiterungen wie BPEL-J auf die Orchestrierung von Web Services im Sinne von SOAP-Diensten spezialisiert und lässt sich nicht einfach auf andere Komponentenmodelle anwenden. Die Kapselung aller Komponenten als Web Services ist zwar für viele Anwendungsfälle ein gangbarer Weg, birgt jedoch zusätzliche Komplexität und Geschwindigkeitseinbußen. IT-Prozesse, die zum Beispiel überwiegend aus Datenbankaufrufen bestehen, lassen sich effektiver direkt über ODBC-Schnittstellen realisieren. Ein weiteres Beispiel sind rechen- oder datenintensive Prozesse, die auf der Ausführung von Kommandozeilenprogrammen in Multicore-, Cluster- oder Grid-Comuputing-Umgebungen basieren und schneller z.B. über POSIX-Schnittstellen oder dazwischen geschaltete Batch-Systeme (LSF, PBS etc.) ausgeführt werden können. Darüber hinaus muss eine nebenläufige Ausführung von Prozessen explizit in den BPEL-Prozessen vorgegeben werden, sie kann nicht automatisiert erfolgen.

- WS-BPEL bietet nur eingeschränkte Möglichkeiten, Prozesse unabhängig von der ausführenden Infrastruktur zu definieren. Diese Abstraktionsmöglichkeiten sind insbesondere in dynamischen verteilten Umgebungen notwendig (Cloud, Grid), da dort Teile der Infrastruktur während der Prozessausführung wegfallen, neu hinzukommen, oder fehlschlagen können. Zudem lässt sich durch abstrakte Prozesse eine bessere Wiederverwendbarkeit für unterschiedliche Infrastrukturen erzielen. So sollte

z. B. der gleiche Prozess für Tests zunächst lokal auf einem Laptop und später in Produktion auf einem großen Cluster ausgeführt werden können.

Im Unterschied zu einer Transformation von BPMN zu BPEL besitzt der von uns vorgeschlagene Ansatz eine vergleichsweise einfache Syntax und Semantik, die auf Ereignisgesteuerten Prozessketten (EPK) sowie Petrinetzen aufbaut, die beide (im Gegensatz etwa zu BPMN und WS-BPEL) theoretisch bereits umfangreich erforscht worden sind. Die EPK wird hierbei zur fachlichen Prozessbeschreibung aufgrund ihrer weiten Verbreitung in Wissenschaft und Praxis und ihrer strukturellen Einfachheit und Ähnlichkeit zu Petrinetzen ausgewählt. Für die Beschreibung ausführbarer Prozesse wird die auf Petrinetzen basierende Sprache GWorkflowDL gewählt, die durch eine Abstraktionsschicht für IT-Prozesse in verteilten Rechnerarchitekturen eine wichtige Voraussetzung zur Unterstützung von Grid- und Cloud-Prozessen erfüllt. Somit ist der vorgeschlagene Ansatz ist nicht auf Web Services als Ausführungs-Technologie beschränkt und gestattet es, Prozesse unabhängig von der ausführenden Infrastruktur zu definieren.

Weitere Ansätze, die ebenfalls eine Verbesserung der Überführung von fachlichen Prozessbeschreibungen in ausführbare Prozesse intendieren, finden sich im Umfeld der *Semantic Web Services* (für eine Übersicht vgl. [CS05] sowie [Ca04]). Durch eine maschinenverarbeitbare, semantische Beschreibung von Web Services soll hierbei vor allem eine leichteres Auffinden (Discovery) und Auswählen (Selection) ermöglicht werden sowie eine Überbrückung semantischer Differenzen im Rahmen von Service-Kompositionen (Orchestration) erreicht werden. Hierzu werden allerdings umfangreiche ontologiebasierte Beschreibungen benötigt, wozu in der Vergangenheit eigene Ontologiesprachen und -Vokabulare wie WSMO (Web Service Modeling Ontology) [Ar05] und OWL-S (Web Ontology Language for Web Services) [Ma04] entwickelt worden sind.

Semantic Web Services können als eine komplementäre Technologie zu dem in diesem Beitrag beschriebenen Ansatz aufgefasst werden, da sie einerseits eine Überbrückung semantischer Differenzen zwischen verschiedenen Diensten fokussieren, die nicht das primäre Ziel des von uns vorgestellten Ansatzes ist. Andererseits werden von den genannten Technologien Aspekte wie Lastausgleich und Skalierung der Ressourcen nicht per se berücksichtigt, die jedoch ein grundlegender Bestandteil der hier vorgestellten Technologie sind.

3 EPML als Austauschformat für EPK-Modelle

Zum Austausch von Prozessmodellen, die mit Hilfe einer Ereignisgesteuerten Prozesskette beschrieben werden, hat sich die *Event-driven Process Markup Language (EPML)* durchgesetzt [MN04]. Wesentliche Merkmale der EPML ist die Graph-orientierte Repräsentation, die Trennung zwischen Definition und Ausprägung, die Erweiterbarkeit sowie die XML-basierte Notation, welche spezifisch als Austauschformat für EPK-Modelle entwickelt wurde. Eine graph-orientierte Repräsentation liegt vor, da die mit EPML beschriebenen EPK-Modelle als gerichteter Graph aufgefasst werden, dessen Knoten durch die Modellelemente eines EPK-Modells konstituiert werden und dessen Kanten

dem Kontrollfluss eines EPK-Modells entsprechen. Eine Trennung zwischen Definition und Ausprägung erlaubt in Analogie zu datenbankgestützten Modellierungswerkzeugen eine redundanzfreie Speicherung der Beschreibung von Modellinhalten wie etwa auszuführender Funktionen unabhängig von deren konkretem Auftreten (Ausprägung) in einem prozessualen Kontext. Eine Erweiterbarkeit von EPML ist über Attribute gegeben, die beliebige Werte aufnehmen können und im Kopfbereich eines EPML-Dokumentes deklariert werden. Die XML-basierte Notation gestattet eine Plattform- und sprachunabhängige Repräsentation von EPK-Modellen. Als Werkzeug zur Erzeugung von EPML wurde im Rahmen dieser Arbeit das am Institut für Wirtschaftsinformatik in Saarbrücken entwickelte Werkzeug CoMoMod eingesetzt.

4 IT-Prozesse in verteilten Systemen

Bei der IT-gestützten Automatisierung von Prozessen kommen zunehmend verteilte Systeme zum Einsatz, bei denen die unterschiedlichen Prozessaktivitäten nicht auf einem zentralen Server, sondern auf räumlich zum Teil weltweit verteilten Rechnern ausgeführt werden. Die Kommunikation zwischen diesen Rechnern erfolgt in der Regel über das Internet oder über lokale Netzwerke. Eine typische Realisierungsform eines solchen verteilten Systems ist eine *Service-orientierte Architektur (SOA)*, bei der die Dienste „top-down“ an den einzelnen Funktionalitäten der Geschäftsprozesse ausgerichtet werden und oftmals in Form von Web-Services auf räumlich verteilten Servern zur Verfügung gestellt werden.

Eine weitere Methode zur Realisierung von besonders skalierbaren Rechennetzen zur Ausführung von IT-Prozessen sind sogenannte *Clouds*. Der Begriff *Cloud* wird im Rahmen dieses Artikels entsprechend der Definition von Forrester Research verwendet, wonach eine Cloud ein Pool aus abstrahierter, hochskalierbarer und verwalteter IT-Infrastruktur ist, die Kundenanwendungen vorhält und nach Verbrauch abgerechnet wird. Der Schwerpunkt von Cloud-Computing liegt im Bereich der ungekoppelten oder lose gekoppelten Massendienste. Spezielle Dienste, wie zum Beispiel parallele Simulationsrechnungen, welche eng gekoppelte Parallelrechner oder Computer-Cluster mit schnellen internen Netzwerkverbindungen benötigen, lassen sich derzeit hingegen besser auf eine Grid-Infrastruktur abbilden, wie sie in Deutschland zum Beispiel durch D-Grid (http://www.d-grid.de/) gegeben ist. Zudem werden beim Grid-Computing in der Regel komplexe virtuelle Organisationen besser unterstützt, welche einen Zusammenschluss von vielen Ressourcenbetreibern und Anwendern für die Lösung einer Aufgabe bezeichnen. Im Unterschied zu Clouds zielt Grid-Computing auf die transparente, gemeinsame Nutzung von Ressourcen in einem Netzwerk innerhalb einer Organisation, in Gruppen oder auch in weltweiten Verbünden ab. Zu den Ressourcen eines Grids zählen hierbei neben Rechen- oder Speicherressourcen z.B. auch Anwendungen, Software, Web-Services, Lizenzen, oder Sensoren zur Datenerfassung.

4.1 Grid Workflow Description Language (GWorkflowDL)

Für die Beschreibung von IT-gestützten Prozessen in SOAs, Clouds oder Grids entwickelt Fraunhofer FIRST die XML-basierte Workflow-Beschreibungssprache *GWorkflowDL*, welche auf High-Level-Petrinetzen basiert, um die Kontroll- und Datenflüsse von verteilten IT-Prozessen zu modellieren [Al06, Al06b, HA06]. Der Begriff *Workflow* wird hierbei als „Automatisierung von IT-Prozessen mittels Graphen" verstanden. Ziel der GWorkflowDL ist neben der Modellierung und Analyse von Workflows insbesondere deren effektive Ausführung und Überwachung. Dieser Artikel bezieht sich auf die aktuelle GWorkflowDL Version 2.0, welche erstmalig die für die Abbildung von Hinterlegungen wichtigen Unternetze direkt unterstützt und zudem besondere Kantentypen für Verbesserung des Datenflusses in verteilten Systemen vorsieht. Da diese Neuerungen noch nicht ausführlich in anderen Veröffentlichung dokumentiert wurden, wird die GWorkflowDL 2.0 im Folgenden zumindest informal eingeführt. Für eine formale Spezifikation der GWorkflowDL sei auf das XML-Schema der GWorkflowDL [Ho09], sowie auf Kapitel 3 von [Vo08] verwiesen.

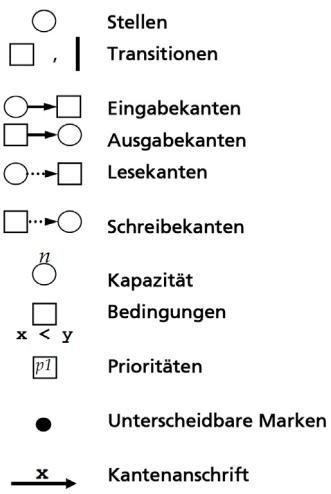

Abbildung 1: Komponenten zur grafischen Notation der GWorkflowDL.

Abbildung 1 zeigt die grafische Notation der einzelnen Bestandteile der GWorkflowDL. Entsprechend des Standards ISO/IEC 15909-1 [Is04] werden *Stellen* durch Kreise, *Transitionen* durch Quadrate und *Eingabe-* sowie *Ausgabekanten* durch durchgezogene Pfeile dargestellt. Im IT-Prozess repräsentieren die *Stellen* Platzhalter für Daten oder Kontrollmarken. Die *Kapazität* gibt die maximale Anzahl von Marken an, die auf einer Stelle liegen können. Falls keine Kapazität angegeben ist, haben die Stellen standardmäßig eine unendliche Kapazität. Die *Transitionen* dienen zur Modellierung abstrakter oder konkreter Aktivitäten, die zum Beispiel auf Web-Service-Methodenaufrufe in einer SOA, Programmausführungen in einem Grid oder auf Unterprozesse – die selber wieder als Petrinetz dargestellt werden können – abgebildet werden. Die *Eingabekanten* sind Pfeile von Stellen nach Transitionen, *Ausgabekanten* sind Pfeile von Transitionen nach Stellen. Die Stellen, die über Eingabekanten mit einer Transition verbunden sind, nennt man *Eingabestellen* dieser Transition. Analog hierzu nennt man die Stellen, die über Ausgabekanten mit einer Transition verbunden sind, *Ausgabestellen*. Jede Stelle kann mehrere unterscheidbare *Marken* enthalten, die als ausgefüllte Kreise dargestellt werden und im XML-Format der GWorkflowDL entweder einen Verweis auf Daten (z.B. als URL) oder die Daten selber enthalten. Eine spezielle Form der Marken sind die Kontrollmarken, welche lediglich die booleschen Werte „wahr" oder „falsch" annehmen können. Transitionen können *Bedingungen* enthalten, die in Form von XPath-Ausdrücken [CD99] als boolesche Funktionen der Eingabemarken formuliert werden.

Eine Transition nennt man *aktiviert*, wenn auf allen Eingabestellen der Transition mindestens eine Marke vorhanden ist und keine der Ausgabestellen ihre Kapazität erreicht hat. Eine aktivierte Transition kann *eintreten* (*ausgeführt werden, schalten*), wenn alle ihre *Bedingungen* erfüllt sind. Beim Eintreten einer Transition wird von jeder Eingabestelle eine Marke entfernt und anschließend auf jede Ausgabestelle eine neue Marke gelegt. Sind mehrere Transitionen zugleich aktiviert, tritt die Transition ein, welche die höchste *Priorität* besitzt. Durch die Einführung von Prioritäten wird die GWorkflowDL Turing-Vollständig und erhält die gleiche Beschreibungskraft wie Inhibitor-Netze. Dadurch werden zwar die Analysemöglichkeiten eingeschränkt (zum Beispiel der Beweis der Beschränktheit), da aber die Workflow-Ausführung und nicht deren Analyse im primären Fokus steht, wird dies in Abwägung der besseren Ausdrucksmöglichkeiten in Kauf genommen. Zudem bietet das Konzept der Prioritäten gute Möglichkeiten, die Ausführung von Workflows zu beschleunigen, z. B. indem bei nebenläufigen Transitionen alle Transitionen des kritischen Pfades mit hohen Prioritäten versehen werden.

In der GWorkflowDL kann über das XML-Element `<operation>` eine Transition mit einer bestimmten Workflow-Aktivität verknüpft werden, sodass jedes Eintreten der Transition eine Ausführung der entsprechenden Aktivität zur Folge hat. Jede Aktivität konsumiert dabei einen Satz Eingabedaten – also die Daten jeweils einer Marke aller Eingabestellen – und erzeugt einen Satz Ausgabedaten, der anschließend in Form von Marken auf alle Ausgabestellen gelegt wird (jeweils eine Marke pro Ausgabestelle). Anders als in der Theorie erfolgt das Eintreten einer Transition nicht instantan, sondern benötigt eine unbestimmte Zeitdauer, da das Schalten der Ausführung einer realen Workflow-Aktivität entspricht. Um die nebenläufige Verarbeitung eines GWorkflowDL-Prozesses zu ermöglichen, müssen daher während der Ausführung die Eingabemarken gesperrt und bei beschränkter Kapazität Platz für die Ausgabemarken reserviert werden.

Zur besseren Modellierung von Workflow-Aktivitäten, die nicht nur Daten konsumieren (= lesen + löschen) und neue Daten erzeugen, sondern auch Daten lesen bzw. vorhandene Daten (ggf. teilweise) überschreiben, wurden zusätzlich *Lesekanten* bzw. *Schreibekanten* eingeführt, die durch gestrichelte Pfeile zwischen Stellen und Transitionen bzw. Transitionen und Stellen dargestellt werden. In der Theorie sind Lese- und Schreibekanten gleichbedeutend mit einer Schleife. Eine *Schleife* bezeichnet hierbei einen Bereich im Petrinetz, in dem eine Transition und eine Stelle sowohl durch eine Eingabekante, als auch durch eine Ausgabekante miteinander verbunden sind, die Stelle somit zugleich Eingabestelle und Ausgabestelle einer Transition ist. Um festzustellen, ob eine Transition aktiviert ist, kann man alle Lese- und Schreibekanten jeweils durch eine Schleife von Eingabe- und Ausgabekante ersetzen und oben definierte Regel zum Begriff „aktiviert“ anwenden.

In der Praxis erlauben *Lesekanten* die effektive Modellierung von nebenläufigem Zugriff von Workflow-Aktivitäten auf gemeinsame Daten. Anstatt also beim Eintreten der Transition eine Marke von einer Stelle zu entfernen und sie anschließend unverändert wieder auf die selbe Stelle zu legen, können die Marke und die damit verbundenen Daten an Ort und Stelle verbleiben. Damit die Transition aktiviert ist, muss die Marke jedoch auf der mit der Lesekante verbunden Stelle bereits vorhanden sein. Bei der Ausführung der Workflow-Aktivität werden die Daten gelesen, aber nicht verändert. Im Gegen-

satz hierzu werden bei *Schreibekanten* Daten überschrieben ohne sie zuvor zu lesen. Beim Eintreten der Transition wird jeweils eine Marke, die bereits auf einer mit einer Schreibekante verbundenen Stelle liegt, durch eine neue Marke ersetzt. Eine Schleife, die sowohl eine Lese- als auch eine Schreibekante darstellen soll, kann alternativ als *Aktualisierungskante* durch eine gestrichelte Linie mit Pfeilspitzen an beiden Enden modelliert werden. Eine Aktualisierungskante beschreibt das Lesen und anschließende (ggf. teilweise) Überschreiben von Daten. Analog hierzu beschreibt eine *Auswechselkante* eine Schleife aus Eingabe- und Ausgabekante, bei der die entsprechende Aktivität Daten nimmt (also liest und löscht) und anschließend auf derselben Stelle neu erzeugt. Eine Auswechselkante kann alternativ als durchgezogener Strich mit Pfeilspitzen an beiden Enden dargestellt werden. Das Ersetzen von Schleifen durch Lese-, Schreibe-, Aktualisierungs- oder Auswechselkanten ändert jedoch nichts an der prinzipiellen Abfolge eines Petrinetzes. Sie dienen lediglich zur Optimierung der Datenflüsse bei der Ausführung der mit den Transitionen verbundenen Aktivitäten.

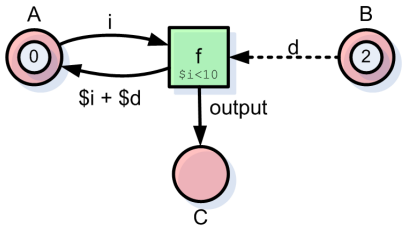

Abbildung 2: GWorkflowDL-Beispiel eines iterativen Aufrufs einer Aktivität f.

Die Bedeutung der *Kantenanschriften* weicht in der GWorkflowDL etwas von der sonst gebräuchlichen Notation in High-Level-Petrinetzen ab. Üblicherweise werden die Kantenanschriften dazu verwendet, um die Typen von Marken zu bezeichnen, die beim Eintreten einer Transition von der Transition konsumiert bzw. erzeugt werden [Is04]. In der GWorkflowDL dienen die Kantenanschriften hingegen dazu, die Marken den einzelnen Parametern einer Workflow-Aktivität zuzuordnen. Bei Eingabe- oder Lesekanten gibt die Kantenanschrift eine Variable vor, über die im Kontext der Transition auf die entsprechenden Marken zugegriffen werden kann. Befindet sich zum Beispiel auf einer Stelle, welche über eine Eingabekante mit der Kantenanschrift `i` mit einer Transition verbunden ist, eine Marke mit dem Inhalt `5`, so wird beim Eintreten der Transition der Wert `5` der Variable `i` zugewiesen: `i=5`. Im Kontext der Transition kann auf den Inhalt der Variablen mit einem vorangesellten „`$`“ zugegriffen werde, z.B. in der Bedingung `$x>4`. Die Kantenanschriften an Ausgabe- bzw. Schreibekanten geben hingegen entweder einen Ausgabeparameter einer Aktivität an, oder eine Verarbeitungsanweisung in Form eines XPath-Ausdruckes, der eine Funktion der Ausgabedaten der Workflow-Aktivität sowie der durch die eingehenden Kantenanschriften definierten Variablen ist, z.B. `$i+6`.

Die Notation eines ausführbaren GWorkflowDL-Prozesses ist exemplarisch in Abbildung 2 dargestellt. Die Transition `f` ist aktiviert da die Stellen `A` und `B` eine Marke enthalten und die Stelle `C` eine unendliche Kapazität besitzt. Die Transition `f` kann eintreten, da die Bedingung `$i<10` mit `i=0` erfüllt ist. Wenn die Transition eintritt wird die Aktivität `f(i,d)` ausgeführt mit `i=0` und `d=2`. Anschließend wird die Marke auf Stelle `A` mit dem Inhalt „`0`“ entfernt und eine neue Marke mit dem Inhalt `$i+$d=2` auf Stelle `A` gelegt. Zudem wird eine neue Marke mit dem Wert des Ausgabeparameters

`output` auf die Stelle `C` gelegt. Insgesamt tritt die Transition `f` fünfmal ein und zwar mit den Eingabedaten `i={0,2,4,6,8}` und `d={2,2,2,2,2}`.

Die XML-Syntax der GWorkflowDL ist eine direkte Umsetzung eines High-Level-Petrinetzes und orientiert sich an der Petrinet Markup Language (PNML) [WK02]. Im Gegensatz zur PNML, welche überwiegend auf die Visualisierung und Analyse von Petrinetzen ausgerichtet ist, bietet die GWorkflowDL jedoch bessere Möglichkeiten, Transitionen mit realen Workflow-Aktivitäten zu verknüpfen. Außerdem werden Kanten nicht als eigenständige Komponenten, sondern als Unterelement von Transitionen definiert, da dies eher dem intuitiven Vorgehen bei der Modellierung von Methoden- oder Funktionsaufrufen in IT-Prozessen entspricht. Die Erweiterbarkeit der GWorkflowDL ist über generische property-Elemente gegeben, mit denen die einzelnen Komponenten des Petrinetzes annotiert werden können. Weitere GWorkflowDL-Beispiele, darunter auch die GWorkflowDL-Repräsentation einiger Workflow-Muster aus [AH02] (*Workflow Patterns*) finden Sie in der Workflow-Registry des myExperiment-Projektes [My09].

Um komplexe Prozesse in verschiedenen Detailabstufungen zu modellieren, können – ähnlich wie die Hinterlegungen in EPKs – einzelne Transitionen mit Unternetzen verknüpft werden. Ein Beispiel ist in Abbildung 3 dargestellt. Die Anschlussbedingungen sind über die Kantenanschriften gegeben. Bei jedem Eintreten der Transition wird eine neue Instanz des Unterprozesses gebildet und jeweils eine Marke von jeder Eingabestellen des übergeordneten Netzes an die Stelle des Unternetzes kopiert, die durch die entsprechende Kantenanschrift gegeben ist. Nachdem das Unternetz vollständig ausgeführt wurde, also keine Transition mehr aktiviert ist, werden jeweils eine Marke von der durch die ausgehenden Kantenanschriften bezeichneten Stellen vom Unternetz wieder in das übergeordnete Netz kopiert.

Durch die Marken des Petrinetzes wird nicht nur ein Prozessmuster, sondern auch der Zustand einer jeden Prozessinstanz beschrieben. Dies ermöglicht die direkte Überwachung der Prozesse sowie eine einfache Realisierung von Fehlertoleranzmechanismen.

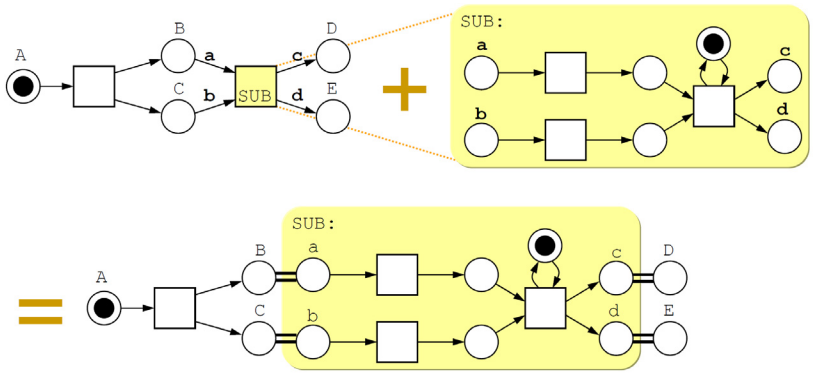

Abbildung 3: Beispiel eines hierarchischen GWorkflowDL-Netzes bei dem die Transition „SUB“ auf ein Unternetz verweist, welches bei jedem Eintreten der Transition ausgeführt wird.

4.2 Grid Workflow Execution Service (GWES)

Der von Fraunhofer FIRST entwickelte *Grid Workflow Execution Service (GWES)* ermöglicht die Automatisierung und das interaktive Management von komplexen und dynamischen Prozessabläufen in Serviceorientierten Architekturen oder Grid-Umgebungen [Ho08, Ho06, GH07, Vo08]. Ein Alleinstellungsmerkmal ist die vollständige Ressourcenvirtualisierung. Die abstrakte Modellierung der Prozessabläufe und Ressourcenbeschreibungen erlaubt die vollständig automatische Abbildung auf jeweils geeignete Hard- und Software-Ressourcen bzw. Dienste. Derzeit eingesetzt wird der GWES unter Anderem in den D-Grid-Projekten MediGRID, MedInfoGrid, Services@MediGRID, BauVOGrid, PneumoGRID und TextGrid.

Der GWES wird in der Regel als Web-Service bereitgestellt, welcher GWorkflowDL-Netze automatisch, persistent und fehlertolerant auf verteilten IT-Ressourcen ausführt. Neben dem GWES besteht das Grid-Workflow-Management-System aus einer Ressourcen- und Workflow-Datenbank, einem *ResourceMatcher* zur Abbildung von abstrakten Job-Anfragen auf verfügbare Hardware- und Software-Ressourcen, einem Monitoring-System, welches über verteilt installierte *ResourceUpdater* alle 10-20s Informationen über die aktuelle Auslastung der Zielsysteme erhält, sowie einem Scheduler, welcher die Auswahl der Zielsysteme optimiert. Das Grid-Workflow-Management-System ermöglicht somit neben einer automatischen Ausführung von Workflows ein Meta-Scheduling von Anwendungen, welches sich im Produktivbetrieb bereits vielfach bewährt hat und bei dem die Anwender keine besonderen Kenntnisse der verteilten Infrastruktur benötigen.

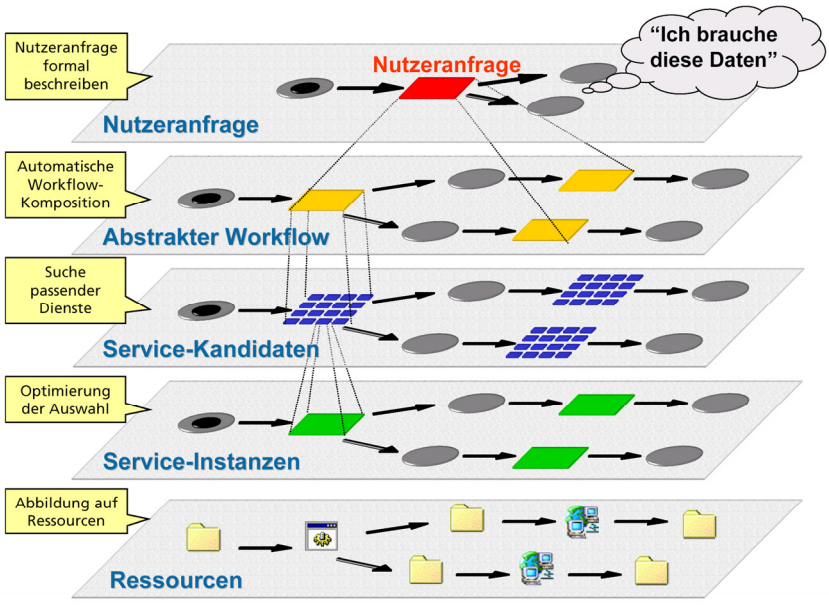

Abbildung 4: Automatische Abbildung von dynamischen Prozessen auf Ressourcen.

Abbildung 4 zeigt den gewählten Lösungsansatz für die automatische Abbildung von dynamischen Workflows auf die jeweils geeigneten und verfügbaren Ressourcen. Nutzeranfragen werden zunächst auf abstrakte Workflows (gelb) abgebildet, in diesem Fall durch die automatische Abbildung von EPKs auf GWorkflowDL-Dokumente (siehe nächstes Kapitel). Jeder Aktivität werden dann zunächst mit Hilfe des ResourceMatcher Service-Kandidaten (blau) zugeordnet, welche die entsprechende Funktionalität bereitstellen. Ein Scheduler wählt einen der Kandidaten aus (grün) und führt die Aktivität auf den entsprechenden Ressourcen aus.

Unter *Scheduling* wird die Abbildung von Workflow-Aktivitäten auf Ressourcen verstanden. Hierbei kann es sich sowohl um IT-Ressourcen (Dienste/Software/Hardware) als auch um Personalressourcen handeln. Abbildung 5 verdeutlicht diesen Prozess. Hierzu werden zunächst alle im aktuellen Prozessschritt nebenläufig ausführbaren Aktivitäten nach ihrer Priorität sortiert. Anschließend werden zu jeder Aktivität prinzipiell geeignete Ressourcen gesucht. Für jede Ressource wird eine Qualitätskennzahl berechnet, die z. B. von der Auslastung der Ressource abhängt. Passende Ressourcen, deren Qualität einen bestimmten Schwellwert (im Beispiel 0,8) überschreiten, werden den Aktivitäten zugeordnet.

5 Überführung von EPKs in ausführbare IT-Prozesse

Um EPKs in ausführbare IT-Prozesse zu überführen, haben wir einen Konverter entwickelt, welcher EPML-Prozessbeschreibungen unter Verwendung einer XSLT-Transformation [Cl99] in GWorkflowDL-Workflowbeschreibungen übersetzt. Zugrunde

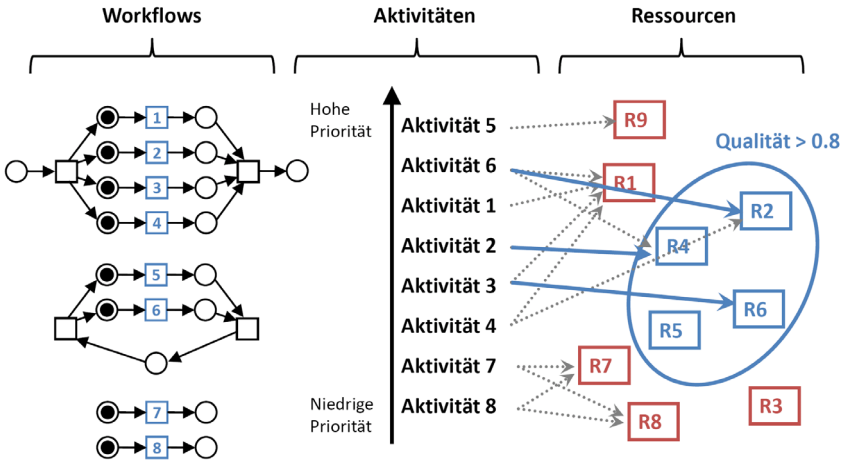

Abbildung 5: Abbildung der Workflow-Aktivitäten von aktivierten Transitionen auf verfügbare Ressourcen.

liegt hierbei eine Abbildung von EPKs auf Petrinetzen ähnlich zu der in [Aa99] beschriebenen. Während van der Aalst jedoch jeweils einzelne EPK-Komponenten auf eine Kombination von Stellen, Transitionen und Kanten abbildet, haben wir eine Abbildungsvorschrift verwendet, welche Ereignisse, Funktionen und Konnektoren einer EPK in genau eine Transition oder Stelle eines Petrinetzes mit den ggf. notwendigen Eingabe- und Ausgabekanten übersetzt. Dadurch wird der Transformationsprozess einfacher und benötigt zudem keine Erweiterungen der EPK, falls zwei Konnektoren (z. B. XOR und AND) direkt miteinander verbunden sind. Analog zu [Aa99] ignorieren auch wir den OR-Operator, zumal er in der Praxis wenig eingesetzt wird und zudem, je nach Definition des OR-Operators, durch eine Kombination aus AND und XOR ersetzt werden kann.

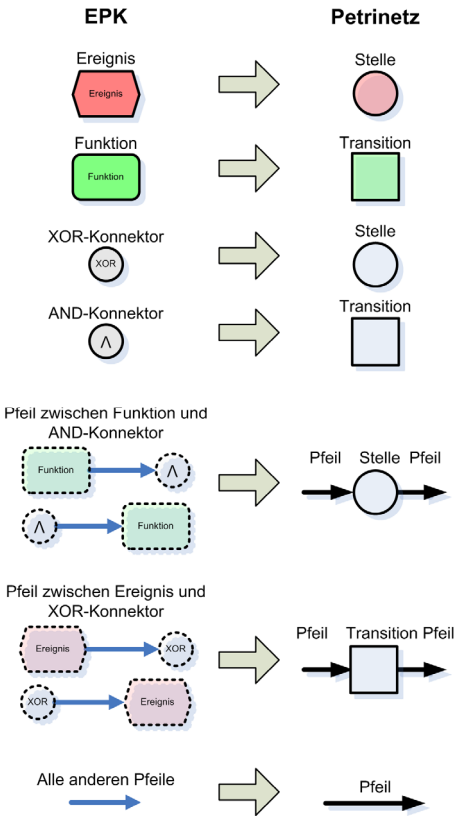

Abbildung 6: Abbildungsvorschriften zur Transformation von EPKs in Petrinetze.

Die verwendeten Abbildungsvorschriften zur Transformation der einzelnen Komponenten von EPKs in Petrinetze sind in Abbildung 6 dargestellt. Während Ereignisse und XOR-Konnektoren von EPKs auf jeweils eine Stelle des Petrinetzes abgebildet werden, werden Funktionen und AND-Konnektoren auf jeweils eine Transition abgebildet. Bei den Kanten gibt es zwei Sonderfälle: Kanten in einer EPK, die eine Funktion mit einem AND-Konnektor verbinden, werden im Petrinetz durch eine zusätzliche Stelle ergänzt, da sowohl Funktionen als auch AND-Konnektoren durch Transitionen abgebildet werden und eine direkte Kante zwischen zwei Transitionen in einem Petrinetz nicht erlaubt ist. Analog hierzu werden Kanten zwischen Ereignissen und XOR-Konnektoren in EPKs durch eine zusätzliche Transition im Petrinetz ergänzt.

Direkte Kanten zwischen zwei Funktionen sind in EPKs zwar oftmals nicht erwünscht, aber in der Praxis durchaus anzutreffen. Ein EPK-Modell kann somit für den Nutzer kompakter dargestellt werden, indem sogenannte Trivialereignisse (z. B. „Brief ist an Post übergeben“ als Ergebnis nach einer Funktion „Brief an Post übergeben“) weggelassen werden. Zudem gestatten auch die meisten Modellierungswerkzeuge derartig vereinfachte Modelle. Um auch diese „unsauberen“ EPKs in korrekte Petrinetze zu überführen, wird eine Kante zwischen zwei Ereignissen im Petrinetz durch die Kombination einer Eingabekante, einer Transition und einer Ausgabekante dargestellt. Analog wird eine

Kante zwischen zwei Funktionen im Petrinetz durch eine Ausgabekante, eine Stelle und eine Eingabekante dargestellt (nicht in Abbildung 6 gezeigt).

Ein einfaches Beispiel einer Transformation einer EPK in ein Petrinetz zeigt Abbildung 7. Die Ereignisse A, B und D werden zu den Stellen A, B und C; die Funktionen D und E zu den Transitionen D und E. Die beiden XOR-Konnektoren werden jeweils durch eine Stelle abgebildet, der AND-Konnektor durch eine Transition. Zwischen die Stellen A und XOR, sowie zwischen B und XOR wird jeweils eine Transition eingefügt, alle anderen Kanten werden 1:1 abgebildet. Mit den Transformationsvorschriften aus Abbildung 6 lässt sich somit die Übersetzung einer EPK in ein Petrinetz in einem Schritt durchführen, ohne wie bei [Aa99] zunächst die EPK um zusätzliche Komponenten erweitern zu müssen. Dieses Vorgehen produziert zwar in einigen Fällen einige redundante Stellen/Transitionen im Netz, welche sich aber leicht bei Bedarf wieder reduzieren lassen.

Ein gewichtiger Unterschied zwischen EPKs und GWorkflowDL-Netzen ist der, dass EPKs in der Regel den Kontrollfluss von fachlichen Prozessmustern beschreiben, während GWorkflowDL-Netze konkrete Instanzen von ausführbaren (technischen) Prozessen inklusive der darin zu verarbeitenden Daten sind. Um eine EPK auf einen ausführbaren Prozess abzubilden, benötigt man neben den Funktionen, Ereignissen und deren Verknüpfungen daher folgende zusätzliche Informationen:

Anfangsmarkierung (Kontrollmarken): Die Anfangsmarkierung gibt die Verteilung von Marken auf den Stellen des Petrinetzes zu Beginn der Ausführung an. Dabei wird in

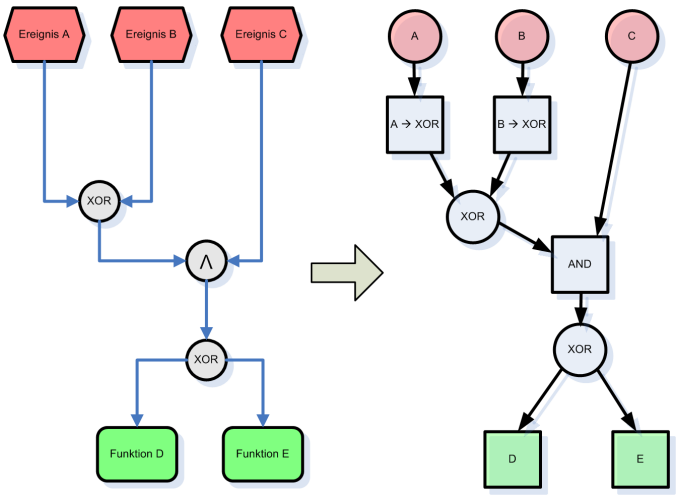

Abbildung 7: Transformation einer EPK in ein Petrinetz analog zum Beispiel aus [Aa99]. Die Anschriften „XOR“ und „AND“ im Petrinetz dienen lediglich der Information, aus welchen Teilen der EPK die Komponenten herrühren und sind für die Funktion des Petrinetzes nicht notwendig.

der GWorkflowDL zwischen Kontrollmarken und Datenmarken unterschieden. Die Kontrollmarken der Anfangsmarkierung können dadurch generiert werden, dass die Startereignisse der EPK durch ein Attribut am Ereignis annotiert werden (z.B.: <`attribute typeRef="workflowStart"/>`).

Anfangsmarkierung (Datenmarken): Komplexer stellt sich die Generierung der notwendigen Datenmarken auf den Stellen dar. Eine Möglichkeit stellt die Verwendung von Transitionen dar, die mit einer Workflow-Aktivität verknüpft sind, die aktiv im Rahmen des Prozesses aus den Kontrollmarken der Anfangsmarkierung Datenmarken erzeugen („hole-Daten-Aktivität"). Bei diesem Vorgehen besteht die Anfangsmarkierung also nur aus Kontrollmarken. Eine zweite Möglichkeit ist die explizite Modellierung von Daten unter Verwendung von *Informationsobjekten* der EPKs. Diese werden üblicherweise mit Funktionen verbunden, um deren Relation zu Dokumenten oder anderen Daten in einer EPK darzustellen. Diese Informationsobjekte können in Datenmarken übersetzt werden, die auf ggf. zusätzlich zu erzeugende Eingabe- bzw. Ausgabestellen gelegt werden. Eine dritte Möglichkeit ist die Verwendung von speziellen Attributen an Ereignissen, analog zu den Kontrollmarken. So kann zum Beispiel in der EPML über ein Attribut `<attribute typeRef="data" value="http://server/data.dat"/>` ein Ereignis mit dem Vorhandensein von bestimmten Daten in Verbindung gebracht werden.

Ausführbare Aktivitäten (Operation): Die GWorkflowDL zielt auf die konkrete Ausführung von Workflows in einem verteilten System ab. Als Mindestinformation benötigt es hierfür das `<oc:operationClass>`-Element, welches die Klasse von Aktivitäten festlegt, die beim Eintreten einer Transition ausgeführt werden soll. Die technisch einfachste Art, diese Information in einer EPK vorrätig zu halten, ist über eine Namenskonvention, bei der der Name einer Funktion in einer EPK identisch ist mit dem Namen des entsprechenden `<oc:operationClass>`-Elements. Der *ResourceMatcher*, der für die Suche passender Dienste zuständig ist, muss dies dann in seiner Konfiguration berücksichtigen. Des Weiteren können auch Ontologiedienste verwendet werden, um Funktionsnamen oder zusätzliche Kennungen auf passende Aktivitätsklassen abzubilden. Die verschiedenen Möglichkeiten der Annotierung sind in Abbildung 8 exemplarisch dargestellt.

Kantenanschriften: Die Kantenanschriften der GWorkflowDL stellen eine Beziehung zwischen den konkreten Daten (Marken auf Stellen) und bestimmten Funktionsparametern einer Workflow-Aktivität her. Eine EPK beschreibt jedoch einen abstrakten Prozessablauf, der in der Regel unabhängig von konkreten Daten ist und daher diese Beziehung nicht enthält. Um trotzdem die Kantenanschriften für den Datenfluss in einem GWorkflowDL-Netz generieren zu können, benötigt man somit eine externe Wissensba-

```
<function id="4">
 <name>f</name>
 <attribute typeRef="ontologyClassId" value="fService"/>
 <attribute typeRef="operationClass" value="urn:dgrdl:service:f"/>
 <attribute typeRef="wsOperation" value="http://server:8080/f?wsdl f"/>
</function>
```

Abbildung 8: EPML-Beispiel einer Annotierung einer EPK-Funktion durch Attribute, die für die Zuordnung einer Funktion zu ausführbaren Workflow-Aktivität verwendet werden können.

sis, die zum Beispiel Profile der Workflow-Aktivitäten mit deren Eingabe- und Ausgabeparametern vorhält. Wenn bei der Benennung der Daten gewisse Namenskonventionen eingehalten werden, können die dazu gehörenden Kantenanschriften z. B. auch direkt aus der WSDL-Schnittstellenbeschreibung des entsprechenden Web-Services extrahiert werden.

Hinterlegungen: Hinterlegungen einer EPK können wie in Abbildung 3 gezeigt hierarchisch durch Unter-Workflows abgebildet werden, auf die innerhalb einer Transition verwiesen werden kann.

Abbildung 9 zeigt exemplarisch den Ausschnitt aus dem XSLT-Stylesheet, welcher für die Transformation eines Ereignisses der EPK (*event*) in eine Stelle des Petrinetzes (*place*) zuständig ist. Die EPK-Annotation *control.token*, *workflowStart* oder *startEvent* wird dabei in eine Marke (*token*) des Petrinetzes übersetzt, welches Bestandteil der Anfangsmarkierung ist. Der EPK-Gegenpart der GWorkflowDL-Komponente wird über ein <property>-Element im Petrinetz annotiert, sodass das Petrinetz sehr einfach wieder auf die ursprüngliche EPK abgebildet werden kann – z.B. zur Darstellung des Fortschritts des Prozessablaufes in der EPK und zur Synchronisation bei Ad-hoc-Änderungen.

```
<!-- convert event to wf:place -->
<xsl:template match="event">
  <xsl:element name="place"
       namespace="http://www.gridworkflow.org/gworkflowdl">
  <xsl:attribute name="ID">p<xsl:value-of select="@id"/></xsl:attribute>
  <xsl:apply-templates select="name"/>
  <xsl:apply-templates select="graphics/position"/>
  <xsl:apply-templates select="attribute"/>
  <!-- convert token attribute to token -->
  <xsl:for-each select="(attribute[@typeRef='control.token']
       | attribute[@typeRef='workflowStart']
       | syntaxInfo[@implicitType='startEvent'])[1]">
    <xsl:element name="token"
         namespace="http://www.gridworkflow.org/gworkflowdl">
      <xsl:element name="control" name-
space="http://www.gridworkflow.org/gworkflowdl">true</xsl:element>
      </xsl:element>
    </xsl:for-each>
  </xsl:element>
</xsl:template>
```

Abbildung 9: Ausschnitt aus dem XSLT-Stylesheet zur Konvertierung von EPML nach GWorkflowDL.

6 Fallstudie: Mängelmanagement im Bauwesen

Die effektive Zusammenarbeit und Kooperation von Unternehmen sowie die effiziente Nutzung von geeigneten Informationstechnologien sind Grundvoraussetzungen für den Erfolg jedes Zusammenschlusses von Personen, Unternehmen und realen Organisationen. Die spezifische Situation im Bauwesen wird von folgenden Faktoren geprägt: der Einmaligkeit jedes Bauwerks (Produktunikat) und der Einmaligkeit jedes Projekts, das von einem Zusammenschluss vieler, überwiegend kleiner Unternehmen durchgeführt wird (Projektunikat). Weiterhin werden die Anforderungen an die Abwicklung von Bau-

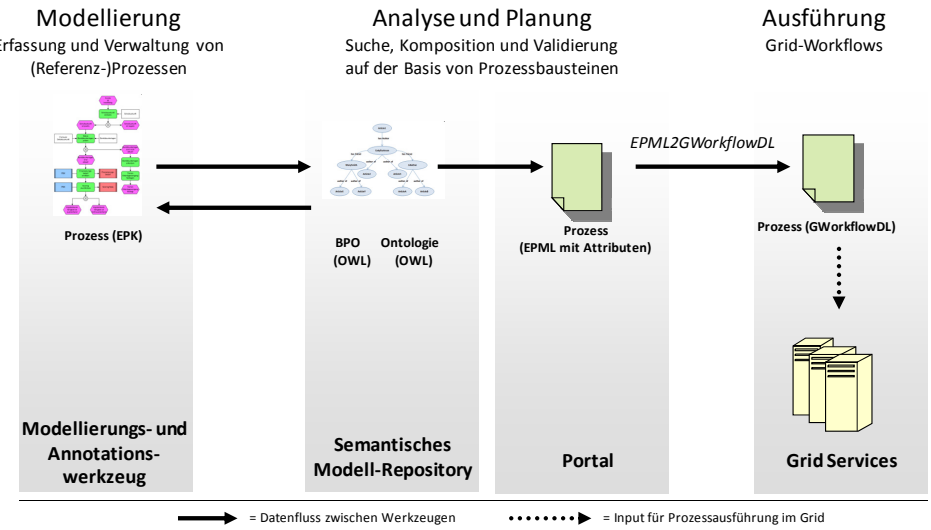

Abbildung 10: BauVOGrid Methodenbaukasten

projekten einerseits und das Bestreben nach Wirtschaftlichkeit und Kostenreduktion andererseits immer höher.

Im Rahmen von BauVOGrid[1] werden verschiedene Werkzeuge zur Prozessmodellierung und Automatisierung eingesetzt, welche die gesamte Kette von der fachlichen Modellierung von Geschäftsprozessen bis hin zur automatischen Ausführung der darunterliegenden IT-Prozesse auf der Grid-Infrastruktur von BauVOGrid abbilden. Da die unterschiedlichen Werkzeuge jeweils eine eigene Sicht auf die Prozesse haben, liegen diesen oftmals auch unterschiedliche Formalismen und Beschreibungssprachen zugrunde, die ineinander überführt werden müssen. Die verschiedenen Werkzeuge und Methoden bilden einen integrierten Werkzeug- und Methodenkasten zur Prozessmodellierung. Bei der Abbildung dynamischer Prozesse werden Zusatzinformationen bei der Modellierung mitgegeben, welche die Integration bzw. Kopplung von Prozessen in Grid-Umgebungen (teil-)automatisiert ermöglichen. Abbildung 10 zeigt die verschiedenen Bestandteile des hierzu entwickelten Methodenbaukastens.

In der Phase der *Modellierung* erfolgt die Erhebung der Referenzprozessmodelle mit einem Standard-Modellierungswerkzeug.

In der Phase der *Analyse und Planung* werden auf der Basis dieser Prozessbausteine konkrete, in einem Bauprojekt ablaufende Prozesse neu erstellt oder geändert. Hierbei wird dem Grundgedanken gefolgt, dass ein Modell nicht gänzlich neu erstellt werden muss, sondern dass Änderungen am Prozess vielmehr durch das Neuarrangieren von

[1] BauVOGrid steht für „Grid-basierte Plattform für virtuelle Organisationen im Bauwesen". Das Projekt wird vom BMBF innerhalb der D-Grid Initiative gefördert(Förderkennzeichen: 01IG07001A).

Prozessbausteinen leicht und im Vergleich zur Neu-Modellierung zeitsparend umgesetzt werden können. Auch die Ausführung von Prozessen zum Zeitpunkt des Prozessablaufes (Runtime) wird so unterstützt und beispielsweise Ad-hoc-Änderungen von real auf einer Baustelle ablaufenden Prozessen ermöglicht.

In der Phase der *Ausführung* werden schließlich die zuvor erstellten Prozessmodelle in auf dem Grid ausführbare Prozessmodelle überführt. Durch diese Modelle wird das Workflow-Management-System GWES konfiguriert, das eine Prozessausführung von Kooperationsverbünden in verteilten Systemen unterstützt. Zur Ausführung von Prozessen werden die aus den fachlichen Prozessmodellen gewonnenen ausführbaren Prozessbeschreibungen herangezogen, welche die Ablauflogik sowie die prinzipiell beteiligten Informationsobjekte und erforderlichen Ressourcen zur Ausführung eines Prozesses angeben.

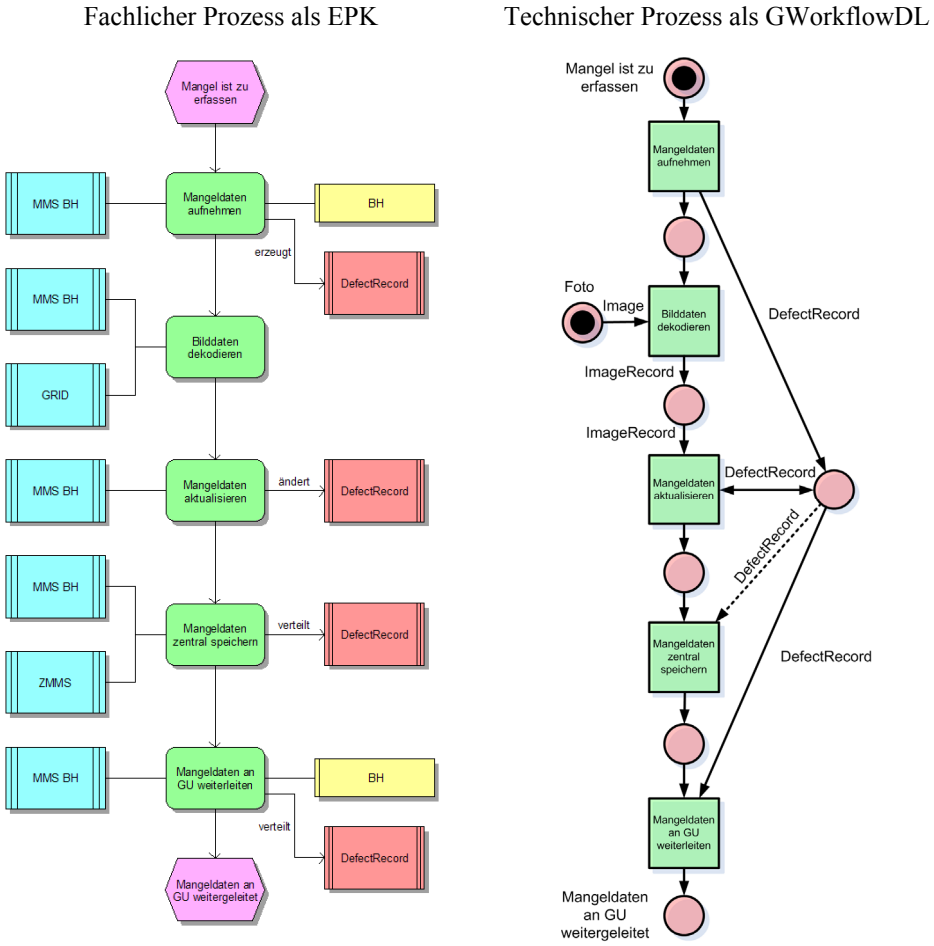

Abbildung 11: Beispiel einer Überführung eines einfachen Geschäftsprozesses aus dem Bereich Mängelmanagement im Bauwesen in einen ausführbaren Grid-Workflow.

Im Rahmen von BauVOGrid ist insbesondere die Transformation von EPML nach GWorkflowDL wichtig, da dies die Schnittstelle zwischen den Geschäftsprozessen und den ausführbaren Grid-Prozessen darstellt. Die Instanziierung eines Prozesses im Sinne der Erzeugung eines konkreten, ablauffähigen Vorgangs entspricht technisch gesehen einem Export eines EPML-Dokuments aus dem Modellierungswerkzeug, einer anschließenden Konvertierung nach GWorkflowDL (vgl. Abschnitt 5) und eines Importes in den GWES.

In dem Projekt wurde exemplarisch die Überführung von Geschäftsprozessen aus dem Bereich Mängelmanagement im Bauwesen auf Dienste der D-Grid-Infrastruktur durchgeführt. Abbildung 11 zeigt einen einfachen Beispielprozess, der die Schritte von der Aufnahme eines Baumangels durch den Bauherrn bis hin zur Übergabe der Mangeldaten an den Generalunternehmer zum Inhalt hat. Zunächst werden bei einer Baustellenbegehung verschiedene Baumängel durch den Bauherrn (BH) erfasst und mit einem mobilen Gerät (z. B. PDA) in das System eingegeben. Anschließend werden vorhandene Fotos, welche zusätzlich mit einem anderen Gerät von den Mängeln zur Dokumentation erstellt wurden, den Mängeln zugeordnet. Die Zuordnung erfolgt über die Dekodierung von im Foto enthaltenen Datamatrix-Codes (2D-Barcodes), die beim Fotografieren mit ins Bild gehalten wurden. Da die Datamatrix-Codes nur einen sehr kleinen Bereich der Fotografien ausmachen und erst nach mehreren Bildbearbeitungsschritten deutlich zu erkennen sind, ist dieser Prozessschritt sehr rechenintensiv und wird in der Regel auf Rechen-Clustern im Grid ausgeführt. Nachdem die Fotos den Mängeln zugeordnet wurden, müssen die Mängeldaten aktualisiert, zentral gespeichert und an den Generalunternehmer (GU) weitergeleitet werden, welcher für die Beseitigung der Mängel zuständig ist.

Die Notwendigkeit einer Automatisierung eines solchen Prozesses wird deutlich, wenn man beachtet, dass bei größeren Baustellen – wie zum Beispiel den Bau eines Flughafens – durchaus etwa 200GB an Fotos und 40.000 Mängel anfallen können, deren Beseitigung bis zu 10% der gesamten Bausumme machen kann. Für jeden Mangel muss dabei eine neue Instanz des entsprechenden technischen Prozesses ausgeführt werden. Durch den hier vorgeschlagenen Ansatz können die ausführbaren Prozesse direkt aus den entsprechenden EPK-Modellen generiert und auf leistungsfähigen Grid-Ressourcen ausgeführt werden, ohne dass sich die Nutzer mit den technischen Details der Infrastruktur auseinandersetzen müssen. Durch die strukturell sehr einfache und direkte Abbildung der Geschäftsprozessmodelle auf ausführbare IT-Prozesse können zudem dynamische Änderungen in den Geschäftsprozessabläufen, welche z. B. durch eine Havarie auf der Baustelle oder durch den Wechsel von Unterauftragnehmern (Insolvenz) ausgelöst werden können, schnell auf modifizierte IT-Prozesse abgebildet werden. Im Gegenzug wird eine direkte Überwachung der IT-Prozesse in der Geschäftsprozesssicht ermöglicht, sodass zum Beispiel auf Verzögerungen im Prozessablauf aufgrund von Engpässen in der IT (Datenspeicher voll) direkt auf Ebene der Geschäftsprozesse reagiert werden kann.

7 Schlussfolgerungen und Ausblick

Im vorliegenden Beitrag wurde aufgezeigt, wie EPK-Modelle durch einen Transformationsansatz in Grid- und Cloud-Computing-Umgebungen zur Ausführung gebracht werden können. Besondere Merkmale dieses Ansatzes sind zum Einen die vergleichsweise einfachen Transformationsregeln, die eine direkte und robuste Überführung von EPML-basierten Prozessbeschreibungen in GWorkflowDL-basierte Prozessbeschreibungen gestatten. Zum Anderen werden die Aspekte Skalierung, Ressourcenmanagement und Technologieunabhängigkeit explizit berücksichtigt, womit wesentliche Herausforderungen im Bereich des Grid- und Cloud-Computing adressiert werden.

Durch den vorliegenden Ansatz sollen zukünftig dynamische Prozesse ermöglicht werden, die eine enge bidirektionale Kopplung zwischen der Geschäftsprozessmodellierung einerseits und der ausführenden Plattform andererseits voraussetzen. Ad-hoc Änderungen der Geschäftsprozesse resultieren dadurch direkt in geänderten IT-Prozessen. Im Gegenzug kann auf Unterschiede in der Verfügbarkeit oder Zuverlässigkeit von IT-Ressourcen direkt durch Modifikationen im Geschäftsprozess reagiert werden.

Zukünftiger Forschungsbedarf besteht hinsichtlich der expliziten Abbildung der Informations- beziehungsweise Datenflüsse. Ein mögliches Vorgehen zur Generierung der hierfür notwendigen Datenmarken und Kantenanschriften im GWorkflowDL-Netz aus EPK-Attributen und EPK-Informationsobjekten wurde in Abschnitt 5 nur sehr grob skizziert und muss weiter konkretisiert werden.

8 Literaturverzeichnis

[Aa99] van der Aalst, W.M.P.: Formalization and verification of event-driven process chains. In: Information and Software Technology, Volume 41, Issue 10, Elsevier, 1999; S. 639-650

[AH02] van der Aalst, W.M.P.; ter Hofstede, A.H.M.: Workflow Patterns: On the Expressive Power of (Petri-net-based) Workflow Languages. Department of Technology Management, Eindhoven University of Technology P.O. Box 513, NL-5600 MB, Eindhoven, The Netherlands. 2002.

[Ar05] Arroyo S, Cimpian E, Domingue J, Feier C, Fensel D, König-Ries B, Lausen H, Polleres A, Stollberg M (2005) Web Service Modeling Ontology Primer : W3C Member Submission 3 June 2005. Innsbruck, DERI. – URL http://www.w3.org/Submission/WSMO-primer/ [Zugriffsdatum 20.07.2006]

[Al06] Alt M., Gorlatch S., Hoheisel A., Pohl H.-W.: Using High-Level Petri Nets for Hierarchical Grid Workflows. 2nd IEEE International Conference on e-Science and Grid Computing (e-Science 2006), Amsterdam, Netherlands, IEEE, 2006.

[Al06b] Alt M., Hoheisel A., Pohl H.-W., Gorlatch S.: A Grid Workflow Language Using High-Level Petri Nets. PPAM05, LNCS 3911, Springer, 2006; S. 715-722

[Ca04] Cabral L, Domingue J, Motta E, Payne TR, Hakimpour F (2004) Approaches to Semantic Web Services: an Overview and Comparisons. In: Bussler C, Davies J, Fensel D, Studer R (Hrsg) The Semantic Web: Research and Applications: Proceedings of the First European Semantic Web Symposium, ESWS 2004 Heraklion, Crete, Greece, May 10-12, 2004. Berlin, Springer, S. 225-239

[Cl99] Clark, J.: XSL Transformations (XSLT) Version 1.0. W3C Recommendation, W3C, 1999; http://www.w3.org/TR/xslt [Zugriffsdatum 20.07.2009]

[CD99] Clark, J.; DeRose, S.: XML Path Language (XPath) Version 1.0. W3C Recommendation, W3C, 1999; http://www.w3.org/TR/xpath [Zugriffsdatum 20.09.2009]

[CS05] Cardoso J, Sheth AP (2005) Introduction to Semantic Web Services and Web Process Composition. In: Semantic Web Services and Web Process Composition: First International Workshop, SWSWPC 2004, San Diego, CA, USA, July 6, 2004.

[Do09] Dollmann, T.; Fellmann, M.; Thomas, O.; Loos, P.; Hoheisel, A.; Katranuschkov, P.; Scherer, R.: Process-Oriented Collaboration in Grid-Environments: A Case Study in the Construction Industry. In Proceedings of the 15th Americas Conference on Information Systems: August 6-9, San Francisco, California, USA, 2009.

[GH07] Gubala T.; Hoheisel A.: Highly Dynamic Workflow Orchestration for Scientific Applications. CoreGRID Technical Report Number TR-0101, CoreGRID, 2007.

[HA06] Hoheisel A.; Alt M.: Petri Nets. In (Taylor I.J., Gannon D., Deelman E., Shields M.S. Hrsg.): Workflows for e-Science – Scientific Workflows for Grids, Springer, 2006.

[Ho06] Hoheisel, A.: User Tools and Languages for Graph-based Grid Workflows. In: Special Issue of Concurrency and Computation: Practice and Experience, Wiley, 2006.

[Ho08] Hoheisel, A.: Grid-Workflow-Management. In (Weisbecker, A.; Pfreundt, F.-J.; Linden, J.; Unger, S. Hrsg.): Fraunhofer Enterprise Grids – Software. Fraunhofer IRB Verlag, Stuttgart, 2008. ISBN: 978-3-8167-7804-2

[Ho09] Hoheisel, A.: XML-Schema der Grid Workflow Description Language (GWorkflowDL) – Version 2.0. Fraunhofer FIRST, 2009; http://www.gridworkflow.org/kwfgrid/src/xsd/-gworkflowdl_2_0.xsd [Zugriffsdatum 20.09.2009]

[Is04] ISO/IEC 15909-1: High-level Petri nets – Part 1: Concepts, definitions and graphical notation. 2004.

[GKS08] Gehre, A.; Katranuschkov, P; Scherer, R.J.: Semantic Support for Construction Process Management in Virtual Organisation Environments. In: (Zarli, A.; Scherer, R. Hrsg.): ECPPM 2008 – eWork and eBusiness in Architecture, Engineering and Construction, Taylor & Francis, London, 2008, S. 85–92

[KS92] Keller, G.; Nüttgens, M.; Scheer, A.-W.: Semantische Prozeßmodellierung auf der Grundlage "Ereignisgesteuerter Prozeßketten (EPK)". In: Scheer A-W (Hrsg): Veröffentlichungen des Instituts für Wirtschaftsinformatik, Nr. 89, Saarbrücken : Universität des Saarlandes. 1992; http://www.iwi.uni-sb.de/Download/iwihefte/heft89.pdf

[Ma04] Martin, D.; Burstein, M.; Lassila, O.; Paolucci, M.; Payne, T.; McIlraith, S.A.: Describing Web Services using OWL-S and WSDL. Arlington, VA, BBN Rosslyn office. 2004; http://www.daml.org/services/owl-s/1.1/owl-s-wsdl.html [Zugriffsdatum 21.11.2005]

[MN04] Mendling, J.; Nüttgens, M.: Exchanging EPC Business Process Models with EPML. In (Nüttgens, M.; Mendling, J. Hrgs.) Proc. of the 1st GI Workshop XML4BPM - XML Interchange Formats for Business Process Management; Modellierung 2004, Marburg Germany, 2004, S. 61-79.

[My09] GWorkflowDL-Registry des myExperiment-Projektes: http://www.myexperiment.org/workflows/search?query=GWorkflowDL [Zugriffsdatum 20.09.2009]

[Oa07] OASIS: Web Services Business Process Execution Language Version 2.0. OASIS, 2007; http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html [Zugriffsdatum 20.09.2009]

[Ob09] Object Management Group: Business Process Modeling Notation (BPMN) – Version 1.2. Object Management Group, 2009; http://www.omg.org/spec/BPMN/1.2/ [Zugriffsdatum 20.09.2009]

[Vo08] Vossberg, M.; Hoheisel, A.; Tolxdorff, T.; Krefting, D.: A Workflow-based Approach for Fault-tolerant Medical Image Transfer in Health Grids. In: Future Generation Computer Systems, Elsevier, 2008.

[WK02] Weber, M.; Kindler, E.: The Petri Net Markup Language. In (Ehrig, H.; Reisig, W.; Rozenberg, G.; Weber, H. Hrgs.) : Petri Net Technology for Communication Based Systems. Berlin, Heidelberg; Springer, 2002.