

Grid Workflow Execution Service (GWES 2.0) – Tutorial

Andreas Hoheisel

Juni 2009



Fraunhofer Institut
Rechnerarchitektur
und Softwaretechnik



Überblick

- **Architektur:** Workflow- und Ressourcenmanagement
- **Konzept:** Prozessmodellierung mit High-Level-Petrinetzen
- **Workflow-Muster:** Sequentielle und nebenläufige Prozesse
- **Ausführen von Workflows:** Abbildung auf Ressourcen und Scheduling

- **Details zur GWorkflowDL**
- **Details zur Workflow-Ausführung**



Architektur



Komponenten des Grid-Workflow-Management-Systems

XML-Sprachen

GWorkflowDL: Workflow-Beschreibungssprache

D-GRDL: Ressourcen-Beschreibungssprache

Web Services

GWES: Grid Workflow Execution Service

ResourceMatcher: Sucht passende Ressourcen (Software, Hardware, Dienste)

Scheduler: Optimiert Ressourcenauswahl (derzeit Bestandteil vom GWES)

eXist: XML-Datenbank

Linuxtoolbox: Beispiel-Web-Services und Graph-Layout

Servlets/Portlets

GWUI: Grid Workflow User Interface

GWES-Editor: Graphischer Workflow Editor (Prototyp, kommerziell)

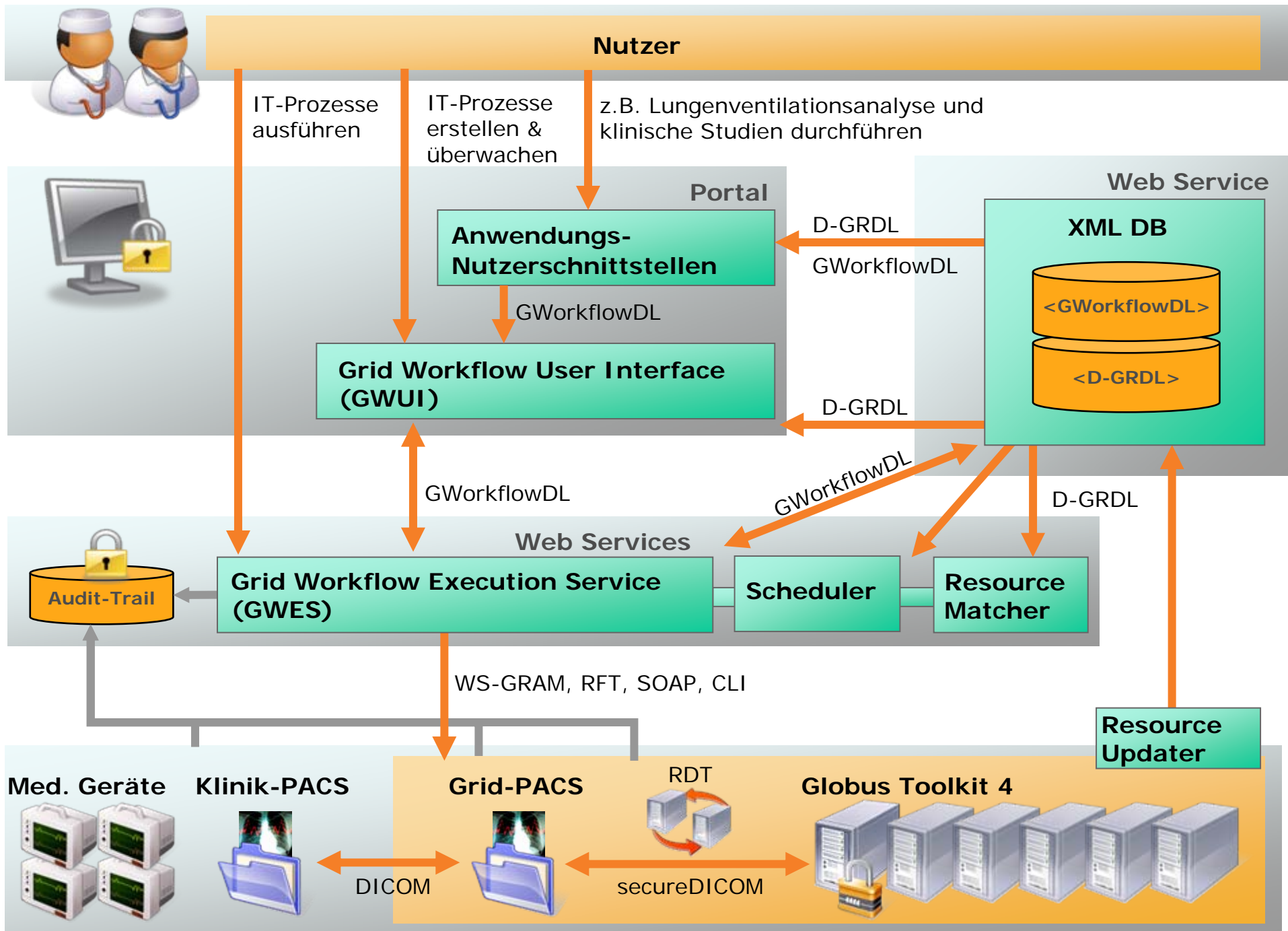
Anwendungsspezifische Portlets (verwenden GWES-Client-API)

Java-Anwendungen

ResourceUpdater: Monitoring von Ressourcen (Daemon auf Ressourcen)

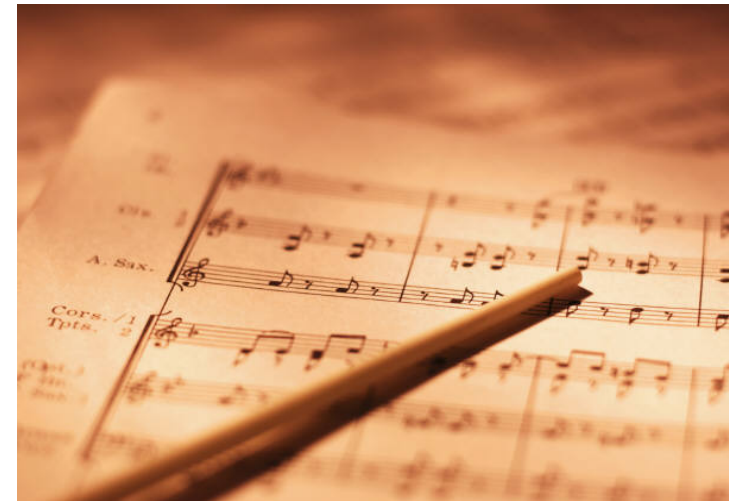
GWESClient: Kommandozeilen-Client für GWES






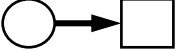
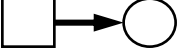



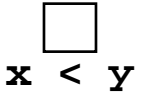





Konzept

Prozessmodellierung mit High-Level-Petrinetzen



GWorkflowDL 2.0 – High-Level-Petrinetz (HLPN)

	Stellen	Repräsentieren Platzhalter für Daten oder Kontrollmarken
 , 	Transitionen	Repräsentieren (abstrakte) Operationen; abzubilden auf Methodenaufrufe, Programmausführungen oder Unterprozesse
	Eingabekanten	Pfeile von Stellen nach Transitionen (Marke wird entfernt)
	Ausgabekanten	Pfeile von Transitionen nach Stellen (Marke wird erstellt)
*neu 	Lesekanten	Gestrichelte Pfeile von Stellen nach Transitionen (Marke wird nicht entfernt)
*neu 	Schreibekanten	Gestrichelte Pfeile von Transitionen nach Stellen (vorhandene Marke wird überschrieben)
	Kapazität	Maximale Anzahl von Marken auf einer Stelle
	Bedingungen	Aktivierte Transitionen treten nur ein, wenn alle Bedingungen erfüllt sind (XPath 1.0)
	Prioritäten	Bei mehreren aktivierten Transitionen treten die mit der höheren Priorität zuerst ein
	Unterscheidbare Marken	Beinhalten reale Daten (Parameter), Referenzen auf Daten oder Zustände (z.B. done, failed, Seiteneffekte)
	Kantenanschrift	Repräsentiert Variablennamen der Operationen (Eingabekanten) oder XPath-Ausdrücke (Ausgabekanten)

Terminologie – High-Level-Petrinetze

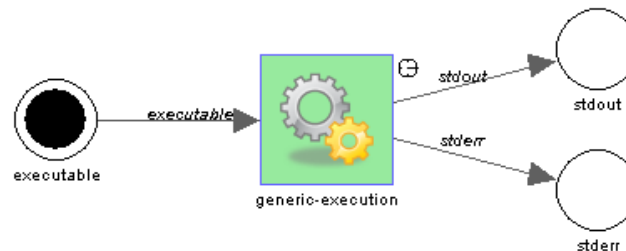
aktiviert (*enabled*)

Eine Transition heißt „aktiviert“ (*enabled*) wenn:

- mindestens eine Marke auf allen Eingabestellen vorhanden ist *und*
- keine der Ausgabestellen ihre Kapazität erreicht hat

eintreten (*occur, fire*)

Aktiviert Transitionen können „eintreten“ (ausgeführt werden, schalten, *occur, fire*) indem sie eine Marke von jeder Eingabestelle nehmen (z.B. Eingabedaten, Vorbedingungen) und eine neue Marke auf jede Ausgabestelle legen (z.B. Ausgabedaten, Seiteneffekte).

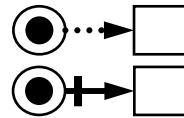


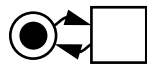
High-Level Petrinetze – Theorie vs. Praxis

Dauer des Schaltens

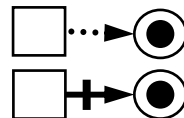
Theorie: Das Eintreten einer Transition erfolgt instantan
Praxis: Mit Transitionen verknüpfte Aktivitäten benötigen eine unbekannte Zeit zur Ausführung
→ Während der Ausführung ist es notwendig die verwendeten Marken auf den Eingabestellen zu sperren und jeweils einen Platz für Marken auf den Ausgabestellen zu reservieren („Reservierungsmarke“)


Leseanten



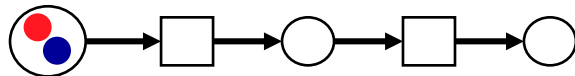
Theorie: Gleichbedeutend mit Schleife 
Praxis: Leseanten ermöglichen nebenläufigen Zugriff auf gemeinsame Daten

Schreibekanten



Theorie: Gleichbedeutend mit Schleife 
Praxis: Schreibekanten reduzieren Datenübertragung und Speicherbedarf

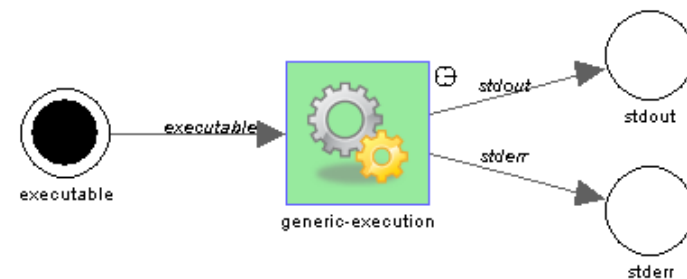
Reihenfolge von Marken



Theorie: Reihenfolge der Marken bleibt erhalten
Praxis: Marken können einander überholen
→ Richtige Gruppierung der Eingabemarken ist wichtig

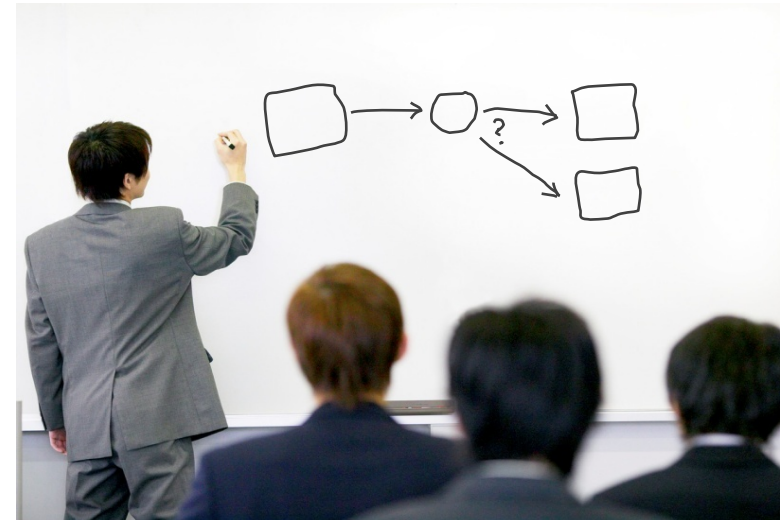
Modellierung von Prozessen mit GWorkflowDL

- Zustand und Aktionen werden modelliert
- Kontroll- und Datenfluss können modelliert werden
- Einfach und ausdrucksstark
- Besonders geeignet zur Beschreibung von verteilten und nebenläufigen Prozessen
- Umfangreiche Theorie verfügbar
- Intuitive Visualisierung möglich
- Kompatibel zu internationalem Standard: ISO/IEC 15909-1
- Konvertierung von anderen Prozessbeschreibungssprachen möglich (z.B. BPEL, ARIS, EPK, PNML, DAGman, SCUFL, ...)



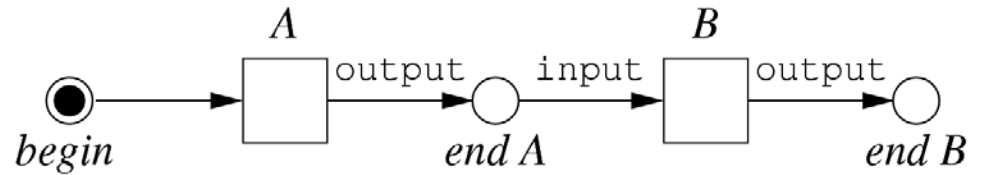
Workflow-Muster

Sequentielle und nebenläufige Prozesse

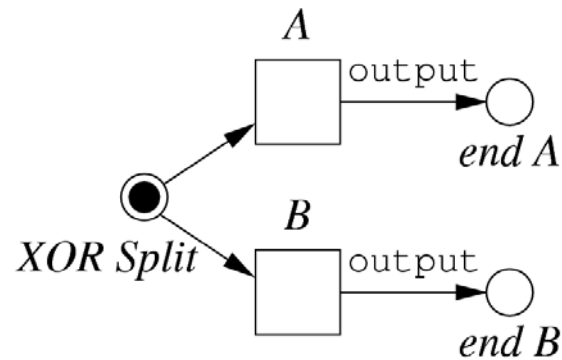


Einfache (sequentielle) Workflow-Muster

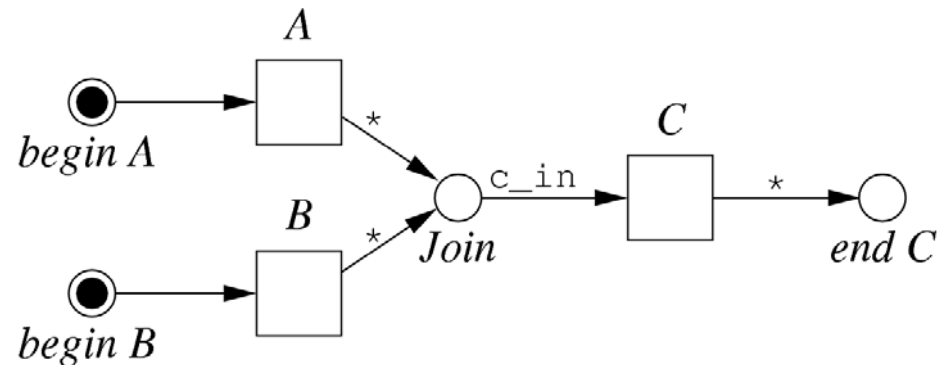
– Sequenz



– Exklusive Auswahl (XOR-Split)

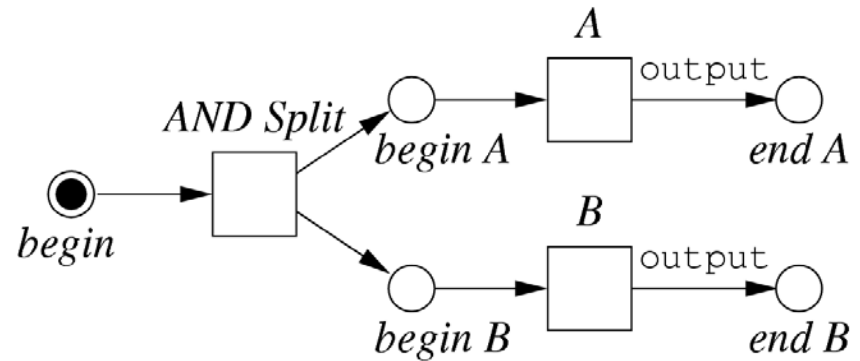


– Mehrfaches Zusammenführen (XOR-Join)

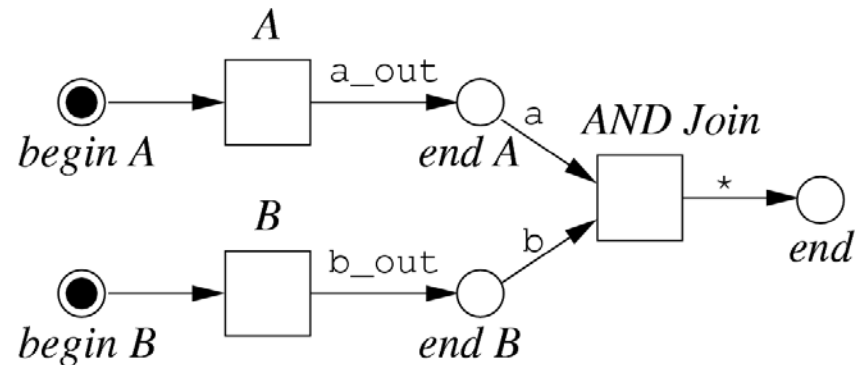


Einfache (nebenläufige) Workflow-Muster

– Parallele Aufteilung (AND-Split)

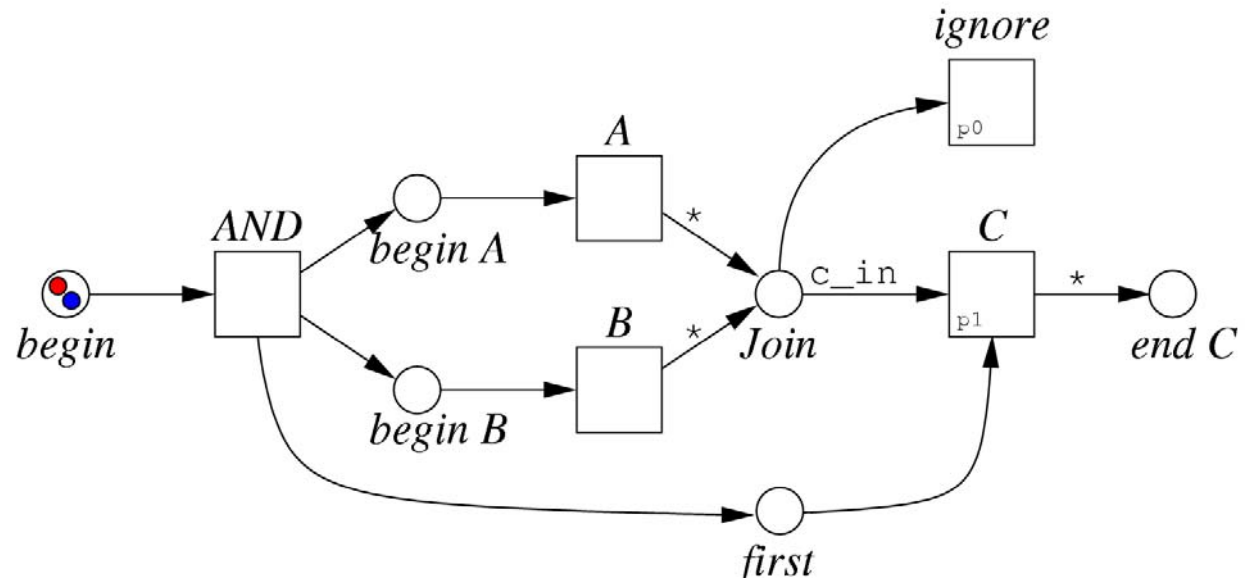


– Synchronisierung (AND-Join)



Komplexere Workflow-Muster

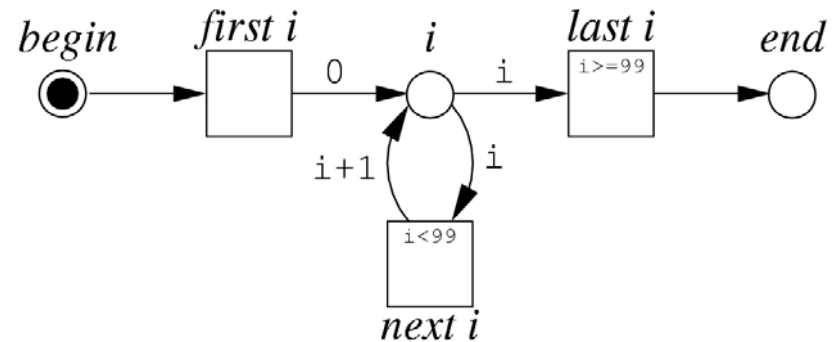
– Einer von M (1-out-of-M)



Iterative (sequentielle) Workflow-Muster

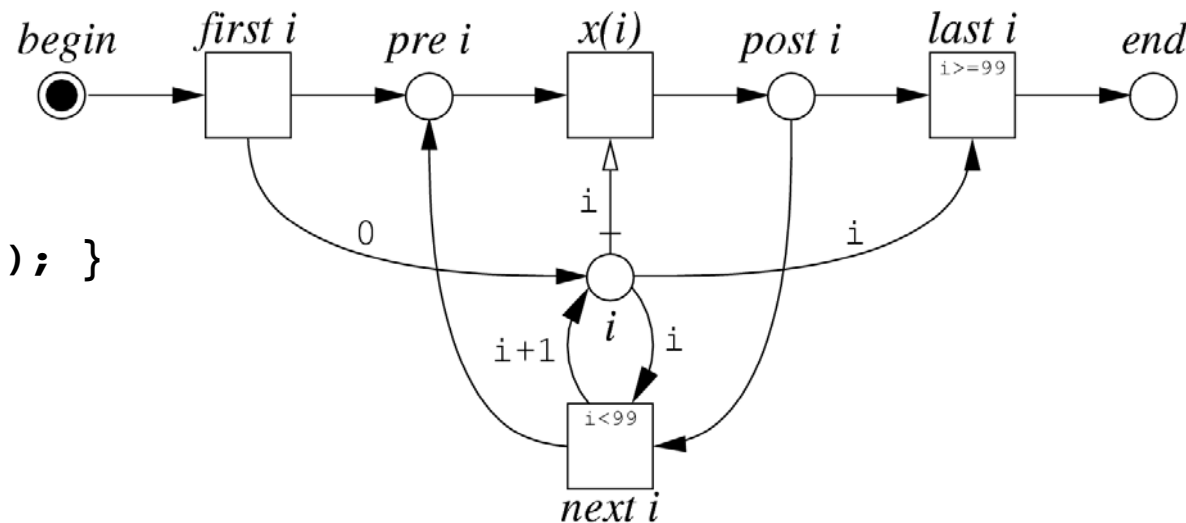
- Strukturierte Schleife:

```
for( i=0; i<99; i++ ) {}
```



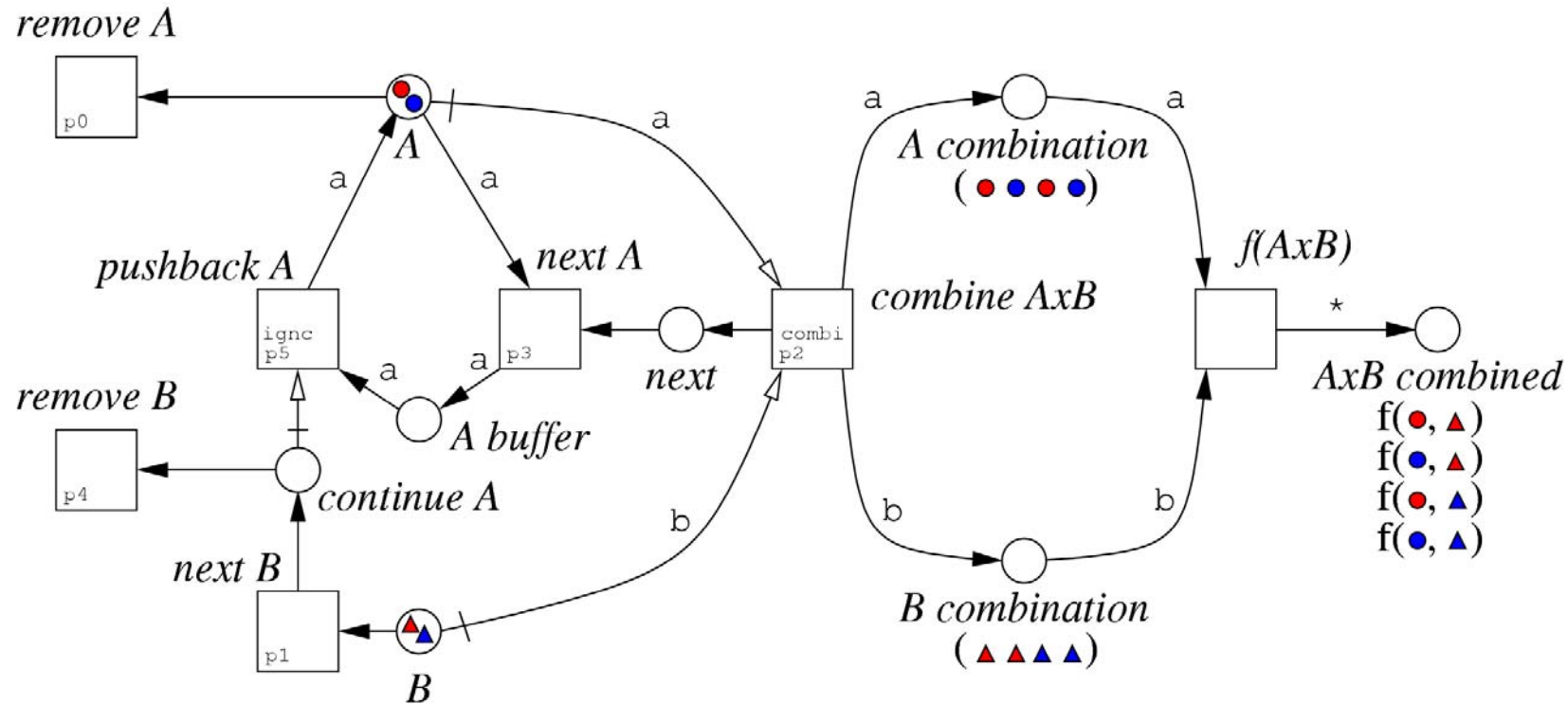
- Strukturierte Schleife mit Aufruf der Funktion $x(i)$

```
for( i=0; i<99; i++ ) { x(i); }
```



Nebenläufige Workflow-Muster

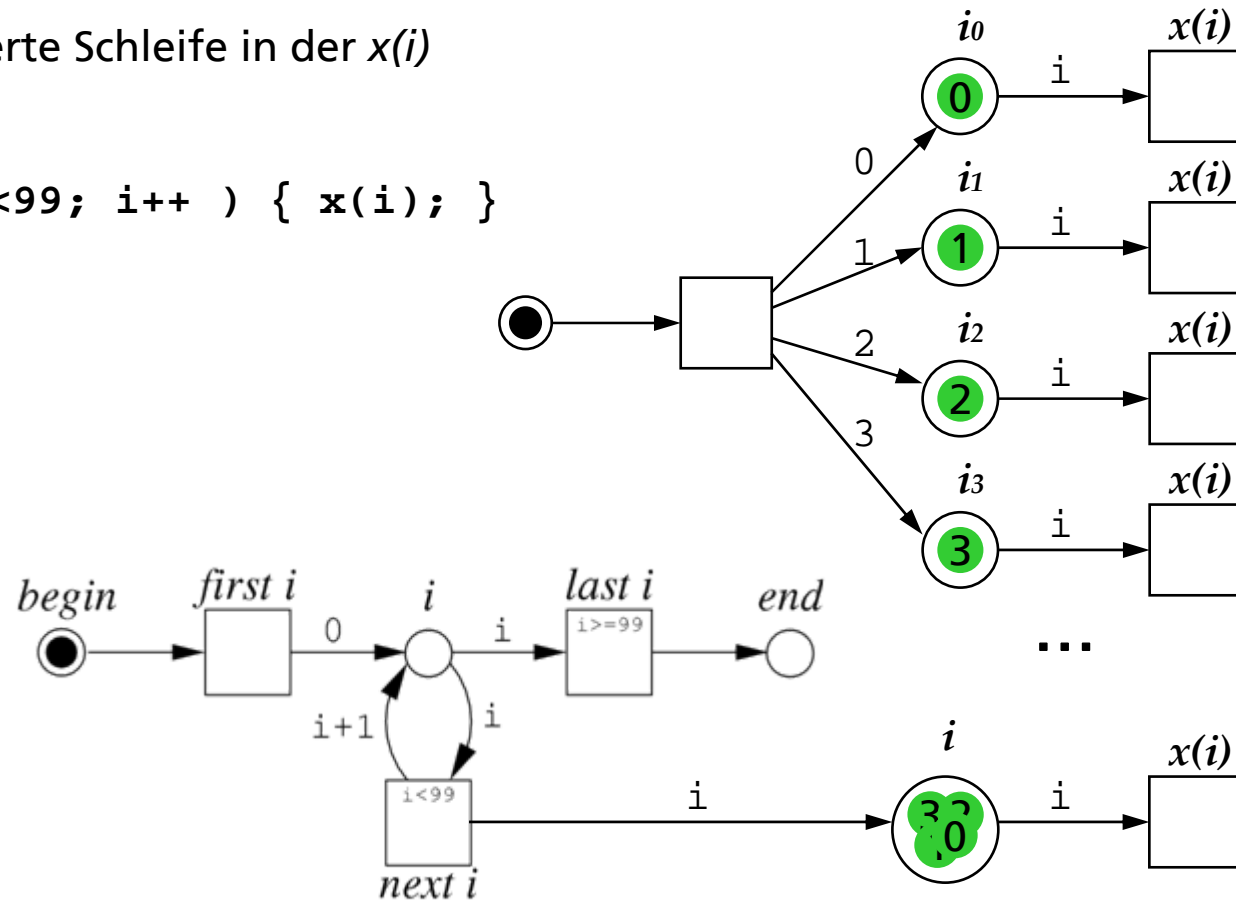
- Berechne $f(A,B)$ für alle Kombinationen der Parameter A und B (aka "Kreuzprodukt")



Nebenläufige Workflow-Muster

- Parallele strukturierte Schleife in der $x(i)$ aufgerufen wird

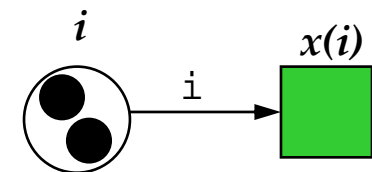
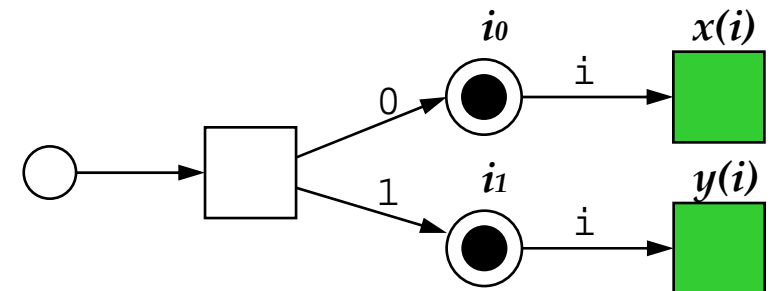
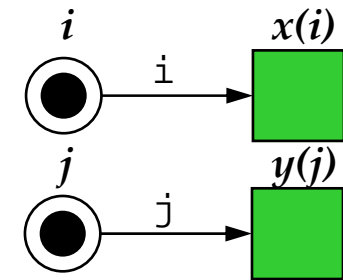
```
parfor( i=0; i<99; i++ ) { x(i); }
```



Nebenläufige Workflow-Muster

Möglichkeiten zur Modellierung von Nebenläufigkeit mit GWorkflowDL:

- Nebenläufige Workflows: Mehrere Workflows werden nebenläufig bearbeitet
- Nebenläufige Transitionen: Mehrere Transitionen, die zur gleichen Zeit aktiviert sind, werden nebenläufig bearbeitet
- Nebenläufige Marken: Mehrere Marken auf den Eingabestellen einer aktivierten Transition lösen mehrere nebenläufige Workflow-Aktivitäten aus (→ „Parameterstudie“)



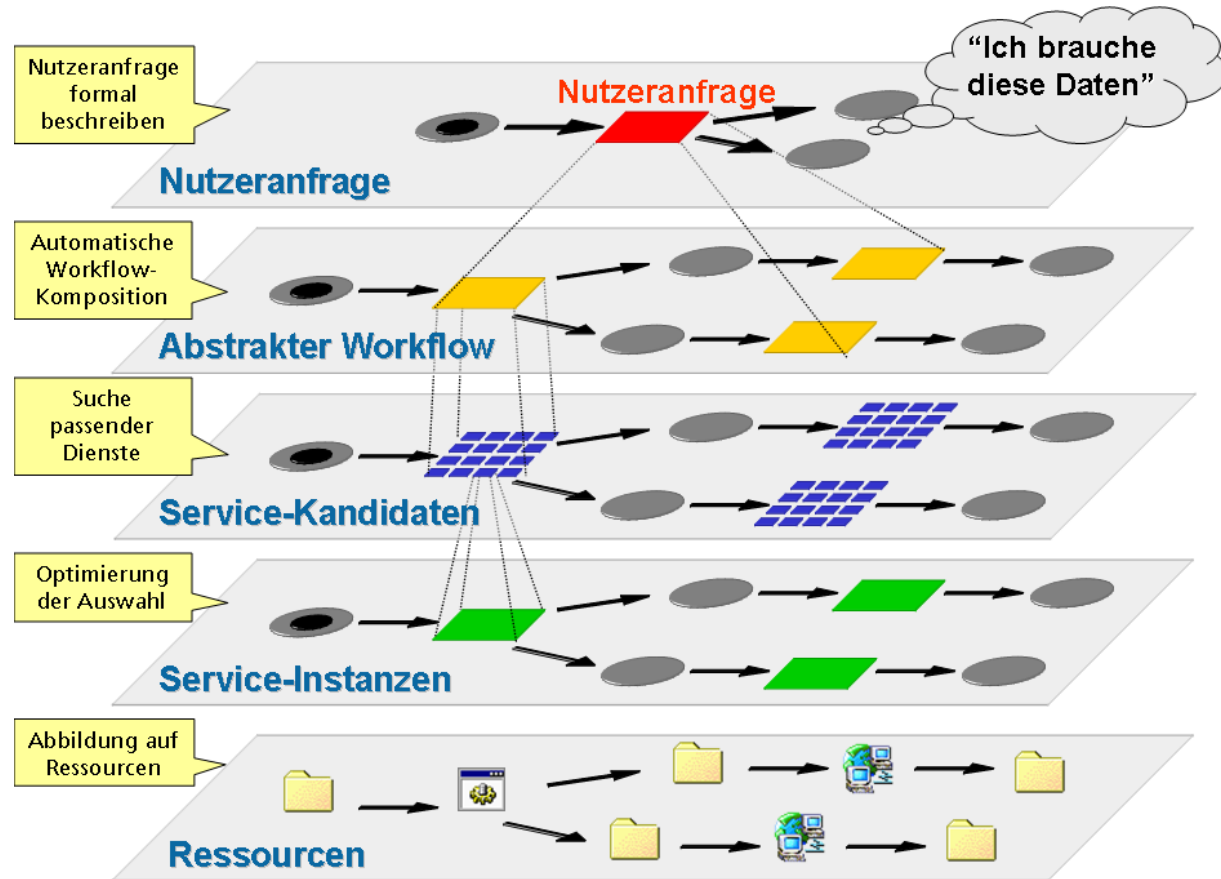
Ausführen von Workflows

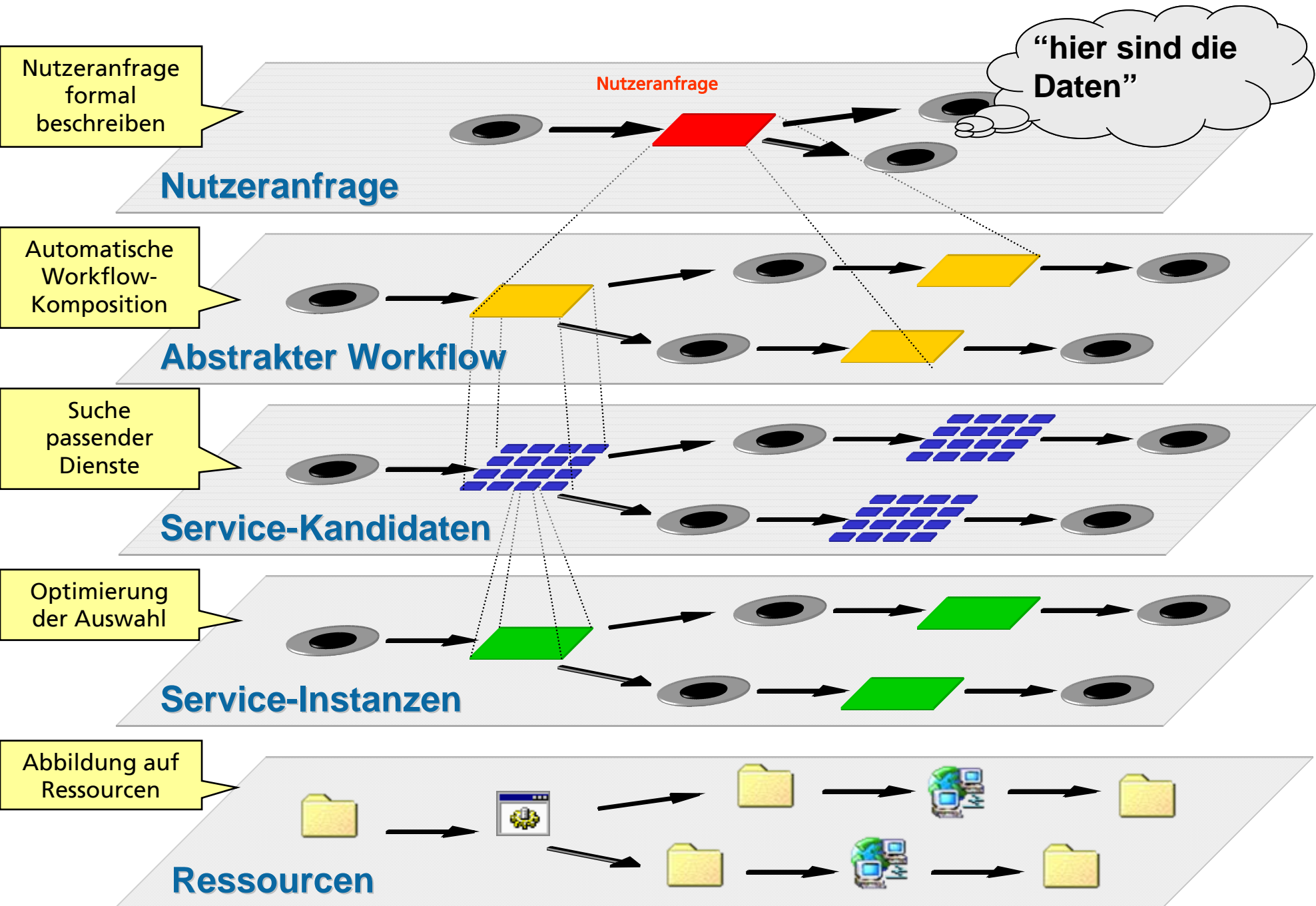
Abbildung auf Ressourcen und Scheduling



Ressourcenplanung und Scheduling

- Ressourcen sind dynamisch: Ressourcen können ausfallen, neue Ressourcen können hinzukommen und aktive Ressourcen können jederzeit entfernt werden
- Deshalb sollten abstrakte Workflows erstellt werden, die unabhängig von der Infrastruktur sind
- Diese abstrakten Workflows werden dann zur Laufzeit auf die verfügbaren Ressourcen abgebildet





GWorkflowDL – Beispiel

```
<workflow xmlns="http://www.gridworkflow.org/gworkflowdl"
  xmlns:oc="http://www.gridworkflow.org/gworkflowdl/operationclass" ID="sortWorkflow">
  <place ID="begin">
    <token><data><file>input1.dat</file></data></token>
    <token><data><file>input2.sat</file></data></token>
  </place>
  <place ID="outputData"/>
  <place ID="hasBeenSorted"/>
  <transition ID="sort">
    <description>sorts strings or numbers</description>
    <input>input</input>
    <output>output</output>
    <outputPlace placeID="hasBeenSorted"/>
    <condition>length($input) > 0</condition>
    <operation>
      <oc:operationClass name="urn:dgrdl:software:sort">
        <oc:operationCandidate type="wsgram" operationName="sort.sh" resourceName="node15"
          quality="0.9" selected="true"/>
        <oc:operationCandidate type="wsgram" operationName="sort.sh" resourceName="node20"
          quality="0.3"/>
      </oc:operationClass>
    </operation>
  </transition>
</workflow>
```

The diagram illustrates the execution of the 'sort' transition. It starts with a 'begin' place containing two tokens. An arrow labeled 'input' leads to the 'sort' transition, which is represented by a green box labeled 'condition'. From the 'sort' transition, an arrow labeled 'output' leads to an 'outputData' place, and another arrow labeled 'hasBeenSorted' leads to a 'hasBeenSorted' place. Three callout boxes are present: 'abstrakte Operation' (orange) points to the <operation> tag, 'ResourceMatcher' (blue) points to the <oc:operationCandidate> tags, and 'Scheduler' (green) points to the <oc:operationCandidate> tags.

ResourceMatcher – Suche passender Ressourcen

Software/Service

```
<resource uri="software:sort-1-0">
  <ofClass uri="urn:dgrdl:software:sort"/>
  <name>sort</name>
  <description>Program that sorts strings</description>
  <simpleProperty ident="executable" type="string" unit="">
    /opt/medigrid/utils/sort.sh
  </simpleProperty>
</resource>
```

Software/Service-Instanz

abstrakte Software/Service-
Klasse (→ GWorkflowDL)

Hardware

```
<resource uri="hardware:server">
  <ofClass uri="urn:dgrdl:hardware"/>
  <name>server</name>
  <description>server host</description>
  <provides>
    <resourceRef uri="software:sort-1-0"/>
  </provides>
  <simpleProperty ident="WSRF.ManagedJobFactoryService" type="uri" unit="">
    https://server:8443/wsrf/services/ManagedJobFactoryService
  </simpleProperty>
</resource>
```

Hardware-Instanz

Software/Service-Instanz

abstrakte Software-Klasse



Hardware-Instanz

Software-Instanz

Hardware-Instanz

Software-Instanz

Hardware-Instanz

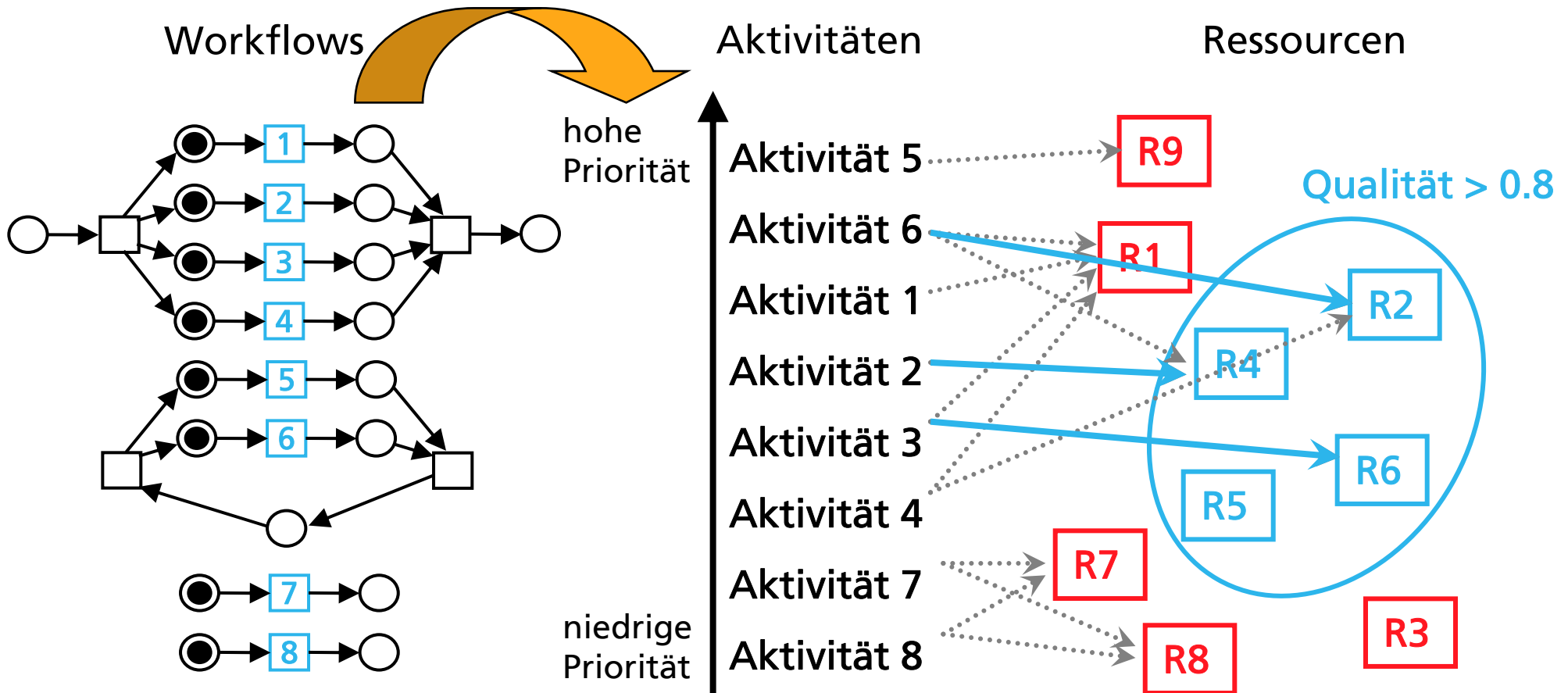
Software-Instanz

Workflow-Scheduler

- Ziel: Einfache und robuste Optimierung der Ressourcenauswahl
- Maximierung der Auslastung verfügbarer Rechenressourcen
- Optimierung der Effizienz (= Durchsatz) für das gesamte System
- „Time-Sharing“ erlaubt (bei Fork)
- Aktuelle F&E: Optimierung der Ausführungsdauer von einzelnen Aktivitäten oder Workflows
- Aktuelle F&E: „Advanced Reservation“
- Konzept: Alle passenden Ressourcen mit einer „Qualität“ größer einem Schwellwert werden (zufallsverteilt) den Workflow-Aktivitäten der Warteschlange zugeordnet, beginnend bei hoher Priorität



Beispiel



Details zur GWorkflowDL

Allgemeine Dokumentation siehe

<http://www.gridworkflow.org/gworkflowdl>

Kontrollmarke | Datenmarke (Dateimarke | Parametermarke | Fehlermarke)

- **Kontrollmarke:** Zur Modellierung und Überwachung von Kontrollflüssen

```
<token>
  [<property.../>]
  <control>true</control>
</token>
```

Ausgabekante ohne Kantenanschrift → true = erfolgreich, false = nicht erfolgreich

- **Datenmarke:** Zur Modellierung und Überwachung von Datenflüssen

```
<token>
  [<property.../>]
  <data><XML.../></data>
</token>
```

- Das XML-Element <data> enthält **ein** Unterelement mit beliebigem (wohlgeformten) XML
 - Bei WS-GRAM-Aktivitäten wird der **Textinhalt** (element.getValue().trim()) von <data> als Wert des Kommandozeilenparameters verwendet (Parametername = edgeExpression)
 - Bei SOAP-Aktivitäten wird das **Unterelement** von <data> entsprechend der edgeExpression umbenannt und als Eingabeparameter in den SOAP-Request kopiert
- **<property>:** Ausgabemarken erben in der Regel die Eigenschaften (properties) der Eingabemarken! (Details zu <property> später)

– Dateimarke:

- Relative Pfadangabe (Warnung: ~ oder `${HOME}` wird von GT4 nicht vollständig unterstützt)

```
<token>  
  <data><file>gsiftp://server/path-relative-to-user-home</file></data>  
</token>
```

- Absolute Pfadangabe

```
<token>  
  <data><file>gsiftp://server//absolute-path</file></data>  
</token>
```

- Der GWES stellt selber fest, ob die Datei bereits auf dem Server liegt, auf dem die Aktivität ausgeführt werden soll und wandelt dann die URL in eine lokale Pfadangabe um:

```
gsiftp://server/relative-path → ${GLOBUS_USER_HOME}/relative-path  
gsiftp://server//absolute-path → /absolute-path
```

- Falls die Datei auf einem anderem Server liegt, wird sie automatisch per FileStageln vor der Ausführung in das lokale Arbeitsverzeichnis übertragen und dann der Aktivität als lokale Pfadangabe übergeben.

– Parametermarke:

- Beispiel 1:

```
<token>
  <data>
    <value1>
      <x>15</x>
      <y>23</y>
    </value1>
  </data>
</token>
```

- Beispiel 2:

```
<token>
  <property name="data.group">dataSet25</property>
  <data><param>-n</param></data>
</token>
```

– Fehlermarke SOAP-Fault:

```
<token>
  <data>
    <soapenv:Fault>
      <faultcode>soapenv:Server.userException</faultcode>
      <faultstring>java.lang.SecurityException: access denied</faultstring>
      <detail>
        <ns1:hostname>clown</ns1:hostname>
      </detail>
    </soapenv:Fault>
  </data>
</token>
```

– Fehlermarke WS-GRAM-Fault:

```
<token>
  <data>
    <soapenv:Fault>
      <soapenv:Code>
        <soapenv:Value>env:Server</soapenv:Value>
      </soapenv:Code>
      <soapenv:Reason>
        <soapenv:Text xml:lang="en">
          Activity 'hoheisel_728f1ab0-567f-11de-9e28-d0a0cbacafa5_0000000004':
          Exception during WS-GRAM invocation
        </soapenv:Text>
      </soapenv:Reason>
      <soapenv:Node>mardschana.zib.de</soapenv:Node>
      <soapenv:Detail>
        operationName=software:test_convert
        executable=/opt/medigrid/convert/test.sh
        factoryEndpoint=https://mardschana.zib.de:8443/wsrp/services/ManagedJobFactoryService
        resourceName=hardware:mardschana.zib.de
        java.net.ConnectException: Connection refused
      </soapenv:Detail>
    </soapenv:Fault>
  </data>
</token>
```

Mit dem XML-Element <property> können spezielle Eigenschaften definiert werden.

<property> ist optionales Unterelement von:

- <workflow>
- <transition>
- <place>
- <token>

– Die Interpretation dieser Eigenschaften hängt von der Implementierung der Workflow-Engine ab.

– Eine ausführliche Liste der aktuellen vom GWES unterstützten „Properties“ siehe:
<http://www.gridworkflow.org/kwfgrid/gwes/docs/faq.html>

Einige wichtige Eigenschaften für <workflow>:

```
<property name="faultManagementPolicy">string</property>
```

- „**abortOnActivityTerminated**“: Workflow bricht ab (abort), wenn eine Aktivität fehlschlägt (Standard)
- „**continueOnActivityTerminated**“: Workflow läuft weiter, wenn eine Aktivität fehlschlägt (z.B. falls Fehlermanagement explizit im Workflow modelliert ist)
- „**suspendOnActivityTerminated**“: Workflow wird unterbrochen (suspend), wenn eine Aktivität fehlschlägt

```
<property name="redistributionOfFailedActivities">boolean</property>
```

- „**true**“: Verwende die Fehlertoleranzmechanismen um fehlgeschlagene Aktivitäten neu zu starten. Erst wenn sich Fehler nicht beheben lässt, greift die „faultManagementPolicy“ (Standard)
- „**false**“: Fehlertoleranz ist ausgeschaltet (z.B. für Tests und Debugging)

```
<property name="workflow.persistence">boolean</property>
```

- „**true**“: Die Historie der Workflows wird in der Datenbank gespeichert (Standard)
- „**false**“: Die Historie der Workflows wird in der Datenbank NICHT gespeichert

Wichtige Eigenschaft für <token>:

`<property name="data.group">string</property>`

- **string**: Datensatz, zu dem die Daten der Marke gehören. Wenn eine Transition mehrere Eingabe- oder Lesekannten mit jeweils mehreren Marken hat, dann werden die Marken entsprechend ihrer „data.group“-Eigenschaft gruppiert und nur Marken des gleichen Datensatzes gemeinsam verarbeitet.
- Falls eine Marke keine „data.group“-Eigenschaften besitzt, kann sie mit beliebigen anderen Datensätzen zusammen verwendet werden (Joker).

Einige wichtige Eigenschaften für <transition>:

`<property name="breakpoint"/>`: Kann ohne Wert gesetzt werden, damit der Workflow jedes mal, wenn die Transition ausgeführt werden soll, unterbricht (suspend)

`<property name="combine.data.groups"/>`: Die Transition erstellt Ausgabemarken mit einer „data.group“-Eigenschaft, welche eine Kombination aus den „data.group“-Eigenschaften der Eingabemarken ist.

Beispiel: Eingabemarken mit data.group="a" und data.group="b" → Ausgabemarke mit data.group="a x b"

`<property name="ignore.data.groups">string</property>`

- „read“: Ignoriere die „data.group“-Eigenschaften von Lesemarken
- „control“: Ignoriere die „data.group“-Eigenschaften von Kontrollmarken

`<property name="priority">int</property>`

- int: Bei gleichzeitig aktivierten Transitionen tritt die Transition mit der höheren Priorität zuerst ein

`<property name="resource.allocation.group">string</property>`

- string: Mehrere direkt oder indirekt aufeinanderfolgende Aktivitäten werden auf derselben Ressource ausgeführt, wenn die entsprechenden Transitionen derselben „resource-allocation-group“ angehören (für räumliches Co-Scheduling). Der String darf keine Leerzeichen enthalten!

`<property name="timeout.running">long</property>`

- long: Zeitdauer in Millisekunden, nach der eine Aktivität im Status „RUNNING“ oder „ACTIVE“ abgebrochen wird (mit Berücksichtigung von Wartezeiten).

`<property name="timeout.active">long</property>`

- long: Zeitdauer in Millisekunden, nach der eine Aktivität im Status „ACTIVE“ abgebrochen wird (ohne Berücksichtigung von Wartezeiten)

XML-Attribut „edgeExpression“ (Kantenanschrift) (1/3)

Kommandozeilenprogramme (WS-GRAM):

- edgeExpression = Name des Kommandozeilenparameters
- Reservierte edgeExpressions: **stdin**, **stdout**, **stderr**

```
<transition ID="cat1">
  <inputPlace placeID="d25-file" edgeExpression="input1" />
  <inputPlace placeID="d26-file" edgeExpression="input2" />
  <outputPlace placeID="out-file" edgeExpression="output" />
  <outputPlace placeID="stdout-file" edgeExpression="stdout" />
  <operation>
    <oc:operationClass name="urn:dgrdl:software:cat">
      <oc:operationCandidate type="wsgram" ... />
    </oc:operationClass>
  </operation>
</transition>
```

```
→ cat.sh -input1 /tmp/d25.dat -input2 /tmp/d26.dat \
  -output output.$ACTIVITY_ID.dat > stdout.$ACTIVITY_ID.dat
```

XML-Attribut „edgeExpression“ (Kantenanschrift) (2/3)

Web-Services (SOAP):

- Eingabekanten: Benennen von SOAP-Request-Parametern und ggf. Hinzufügen von Elternelementen
 - ➔ edgeExpression ist identisch mit entsprechendem „message part name“ (RPC-style) bzw. „element name“ (document-style) der WSDL
- Ausgabekanten: Filtern oder Verarbeiten von SOAP-Rückgabedokumenten (XPath 1.0)
- „*“ steht für alle XML-Elemente, insbesondere hilfreich für Rückgabewerte!

```
<transition ID="StreamingEditor">
  <property name="xmlns:tns">http://sedit.textgrid.de</property>
  <property name="xmlns:tgc">http://textgrid.info/TGCrudService</property>
  <readPlace placeID="xsl" edgeExpression="tns:stylesheet" />
  <inputPlace placeID="input" edgeExpression="input/*" />
  <outputPlace placeID="metadata" edgeExpression="tgc:tgObjectMetadata" />
  <outputPlace placeID="SEall" edgeExpression="*" />
  <outputPlace placeID="fault" edgeExpression="soapenv:Fault/soapenv:Detail" />
</transition>
  <operation>
    <oc:operationClass>
      <oc:operationCandidate type="soap" ... />
    </oc:operationClass>
  </operation>
</transition>
```

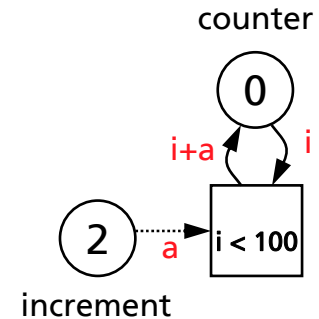
XML-Attribut „edgeExpression“ (Kantenanschrift) (3/3)

Rechnen mit Kantenanschriften – Algorithmen als Kontrollfluss

- Eingehende Kanten: Variablenzuweisung
- Ausgehende Kanten: Verarbeitungsanweisung (XPath 1.0)
- Beispiel:

```
<transition ID="i_plus_a">  
  <description>i+=a</description>  
  <readPlace placeID="increment" edgeExpression="a" />  
  <inputPlace placeID="counter" edgeExpression="i" />  
  <outputPlace placeID="counter" edgeExpression="i + a" />  
  <condition>$i &lt; 100</condition>  
</transition>
```

→ for (i=0; i<100; i+=a) { }



- Aktivierte Transitionen treten nur ein, wenn alle Bedingungen der Transition erfüllt sind.
- Spezifikation: Boolesche Ausdrücke gemäß XPath 1.0 (siehe <http://www.w3.org/TR/xpath>).
- Der Kontext der Bedingung ist das gesamte GWorkflowDL-Dokument im aktuellen Zustand.
- `<$edgeExpression>` repräsentiert die Marke, welche über die entsprechende Kante mit der Transition verbunden ist und für das Eintreten der Transition verwendet wird. Beispiel:

```
<place ID="input"><token><data><i>1</i></data></token></place>
<transition ID="tr">
  <inputPlace placeID="input" edgeExpression="i" />
  <condition>$i = 1</condition>
</transition>
```

→ `$i` wird ersetzt durch `gwdl:place[@ID="input"]/gwdl:token[1]`
- Zusätzliche, über XPath 1.0 hinausgehende Funktionen:
 - `gwdl:current-dateTimeMs()`: Aktuelle Zeit in Millisekunden seit 1970
 - `gwdl:dateTime2ms()`: Konvertiert dateTime-Objekt in Millisekunden seit 1970
 - `gwdl:ms2dateTime()`: Konvertiert Millisekunden seit 1970 in dateTime-Objekt
 - `gwdl:ms2year()`, `gwdl:ms2month()`, `gwdl:ms2day()`, `gwdl:ms2hour()`, `gwdl:ms2minute()`, `gwdl:ms2second()`: Konvertiert Millisekunden seit 1970 in Jahr, Monat, Tag, Stunde, Minute und Sekunde (jeweils als int).

Beispiele

```
<condition>$exitcode = 'true'</condition>
<condition>normalize-space($x) = 'AB'</condition>
<condition>$i < 2</condition>
<condition>$i >= 2</condition>
<condition>$found != 'Found'</condition>
<condition>gwdl:ms2second(gwdl:current-dateTimeMs()) > 30</condition>
<condition>string-length($value) > 0</condition>
<condition>starts-with($value, 'sort')</condition>
<condition>$params/gwdl:data/*/@xsi:type = 'xsd:string'</condition>
<condition>count($fault/gwdl:data/soapenv:Fault) = 1</condition>
<condition>contains($solution, 'failure')</condition>
<condition>substring-before(substring-after($solution, 'lower="'), '') <
  substring-before(substring-after($solution, 'upper="'), '')</condition>
```

- Mittels operationCandidate KANN (zum Beispiel für Tests) ein konkretes Software/Hardware-Paar zur Ausführung vorgegeben bzw. vorgeschlagen werden.
- Wenn **selected=„true“** gesetzt ist, findet KEIN Scheduling (Lastverteilung) statt
- Falls operationCandidate NICHT gesetzt ist, werden automatisch vom ResourceMatcher passende Kandidaten anhand des Namens der OperationClass gesucht

→ Im Produktivbetrieb sollte <operationCandidate> NICHT manuell vorgegeben werden, da dann in der Regel keine Fehlertoleranz und Lastverteilung erfolgt!

Kommandozeilenprogramme (WS-GRAM)

– Aufruf mit Software-Eintrag in der D-GRDL-Datenbank

```
<oc:operationCandidate type="wsgram"  
  operationName="software:test"  
  resourceName="hardware:serv"  
  selected="true"/>
```

- operationName verweist auf die URI der Software-Beschreibung in der D-GRDL-Datenbank
- resourceName verweist auf die URI der Hardware-Beschreibung in der D-GRDL-Datenbank

– Aufruf ohne Software-Eintrag in der D-GRDL-Datenbank

```
<oc:operationCandidate type="wsgram"  
  operationName="/opt/medigrid/test/test.sh@software:test PBS"  
  resourceName="https://serv:8443/wsrp/services/ManagedJobFactoryService@hardware:serv"  
  selected="true"/>
```

WebServices (SOAP):

– Aufruf mit Service-Eintrag in der D-GRDL-Datenbank

```
<oc:operationCandidate type="soap"  
  operationName="service:transfer-file"  
  resourceName="hardware:serv"  
  selected="true"/>
```

- operationName verweist auf die URI der Service-Beschreibung in der D-GRDL-Datenbank
- resourceName verweist auf die URI der Hardware-Beschreibung in der D-GRDL-Datenbank

– Aufruf ohne Service-Eintrag in der D-GRDL-Datenbank

```
<oc:operationCandidate type="soap"  
  operationName="transferFile"  
  resourceName="http://serv:8080/gwes/services/FileTransfer?wsdl"  
  selected="true"/>
```

- operationName muss identisch mit dem in der WSDL definierten „operation name“ sein.
- resourceName enthält die URL der WSDL

In Kürze geplant: Sub-Workflows

- Aufruf mit Workflow-Eintrag in der D-GRDL-Datenbank

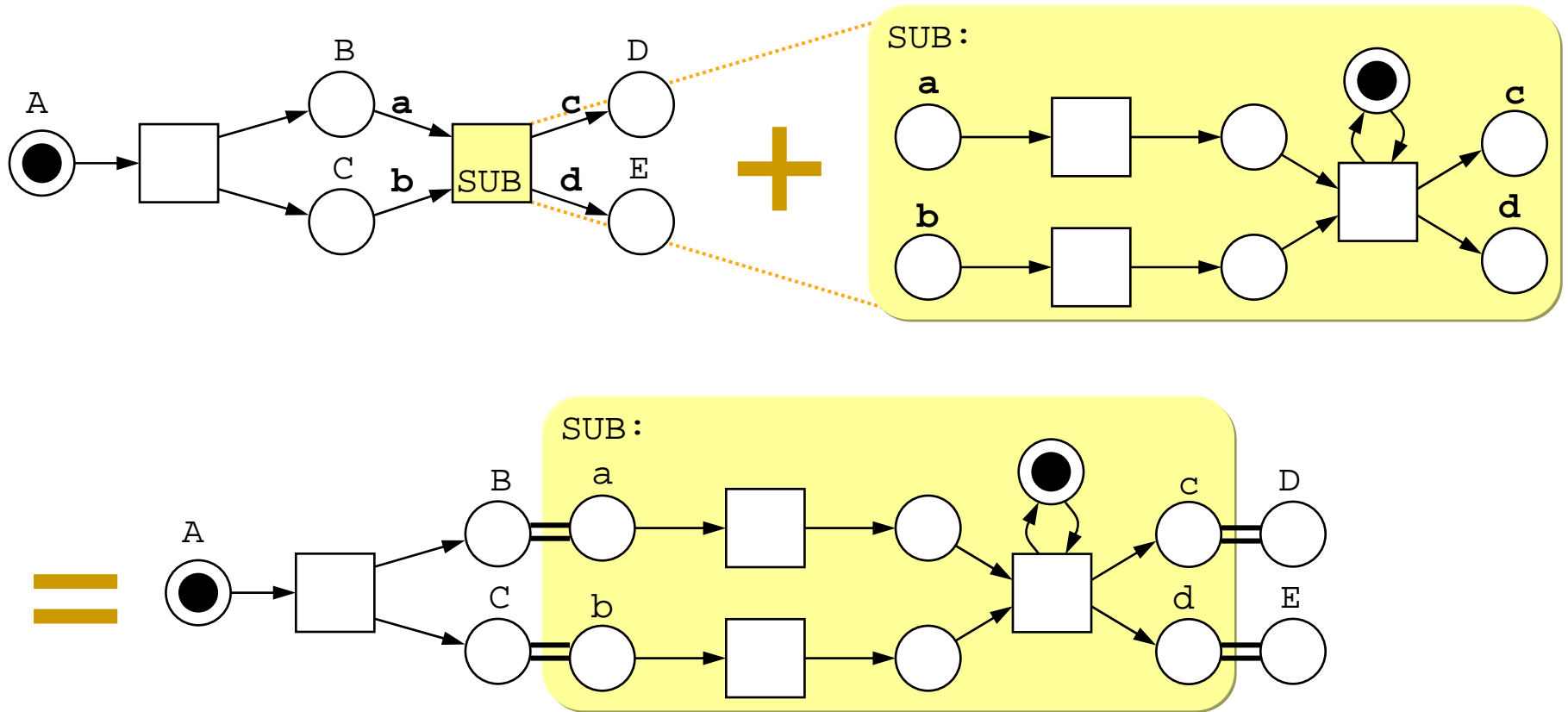
```
<oc:operationCandidate type="workflow"  
  operationName="workflow:sub-task"  
  resourceName="service:gwes-serv"  
  selected="true"/>
```

- Aufruf ohne Workflow-Eintrag in der D-GRDL-Datenbank

```
<oc:operationCandidate type="workflow"  
  operationName="file:///tmp/sub-task.gwdl"  
  resourceName="http://serv:8080/gwes/services/GWES?wsdl"  
  selected="true"/>
```

- Jedes EINTRETEN der Transition erzeugt eine neue Instanz des Sub-Workflows.
- Die für das Eintreten der Transition relevanten Marken aus dem Workflow werden entsprechend der Kantenanschriften (edgeExpressions) auf die Stellen (places) des Sub-Workflows kopiert.
- Die Transition, welche den Sub-Workflow erzeugt, schaltet erst dann, wenn der Sub-Workflow beendet wurde.

Sub-Workflows



Details zur Workflow- Ausführung

Wie wird ausgeführt

Kommandozeilenprogramme per Globus Toolkit 4.0.x (type=„wsgram“)

Portal	GWES → GRAMActivity → modifizierte CoG-Java-API (GramJob)
Grid-Knoten (Linux)	→ GT4 JobManager → fork/qsub
Cluster-Knoten (Linux)	→ gwes-command-line-operation.sh → Kommandozeilenprogramm

SOAP-Methodenaufrufe (type=„soap“)

Portal	GWES → WSActivity → Apache Axis
Grid-Knoten	→ Apache Axis → Java (→ ...)
SOA-Knoten	→ gSOAP → C/C++/Fortran → .NET → C# → ...

Zusätzlich geplant: Kommandozeilenprogramme per Condor (type=„condor“)

Portal	GWES → CondorActivity → condor_submit → condor_schedd
Cluster-Knoten (Windows/Linux)	→ condor_startd → Kommandozeilenprogramm

Eigenheiten von WS-GRAM

- (bisher) keine „Advanced Reservation“ möglich (auch wenn das die Batch-Queue-Systeme oftmals unterstützen)
- Wichtige Umgebungsvariablen sind nur selten gesetzt, z.B. ist oftmals kein „PATH“ vorhanden → Möglichst absolute Pfade verwenden (die leider auf jeder Ressource anders sein können)!
- WICHTIG: Bereitstellung von standardisierten (Nagios-kompatiblen) Test-Skripten zum Testen der installierten Anwendung durch die Anwendungsentwickler und Überwachungsprogramme!
- WS-GRAM ist nicht sehr zuverlässig; Status von Jobs ist manchmal nicht mehr ermittelbar! → timeout in den Transitionen definieren!

Fehlersuche – WS-GRAM-Aktivitäten

- Workflow-Properties: warn.x, error.x
- Fehlermarken
- Der GWES legt vor der Ausführung jeder Aktivität ein temporäres Arbeitsverzeichnis auf dem Grid-Knoten an, in MediGRID unter: `/opt/medigrid/tmp/$ACTIVITY_ID`
(`$ACTIVITY_ID = $WORKFLOW_ID_XXXXXXXXXX` mit `XXXXXXXXXX` = Aktivitätszähler)
- Dort sind diverse, fürs Debugging hilfreiche Dateien zu finden:
 - `exec.log`: Enthält die den kompletten Kommandozeilenbefehl, welcher zur Ausführung verwendet wurde, sowie einen Zeitstempel und, falls die Ausführung nicht abgebrochen wurde, den exit-code.
 - `stdout.$ACTIVITY_ID.dat`: stdout der Anwendung
 - `stderr.$ACTIVITY_ID.dat`: stderr der Anwendung
- Die Dateien sind in der Regel nur für den lesbar, dessen Zertifikat für die Ausführung des Workflows verwendet wurde
- Jeder Nutzer der MediGRID-VO kann sich per `gsissh -p 2222 <servername>` auf den Grid-Knoten einloggen und sich die temporären Daten anschauen
- Die Verzeichnisse werden in der Regel automatisch nach ca. einem Monat gelöscht!

Fehlersuche – Einige Exit-Codes

- 0: OK
- 1: (Nagios test script) WARNING
- 2: (Nagios test script) CRITICAL
- 3: (Nagios test script) UNKNOWN
- 96: (GWES cli start script) The stdin is not available or not readable
- 97: (GWES cli start script) The executable is not available or not executable
- 98: (GWES cli start script) The executable has not been specified
- 99: (GWES cli start script) Working directory has not been specified
- 127: (WS-GRAM:JobManager.pm) A child error code with an exit code of 127 indicates that the application could not be run
- 128+: (Linux) Program has been interrupted by signal X-128, e.g., 143-128 = 15 = SIGTERM.
- 153: (PBS:qstat) Unknown Job Id
- 271: (PBS) Abbruch des Jobs, oftmals Hinweis darauf, dass Anforderungen an Queue überschritten wurden (z.B. Walltime oder Speicherbedarf)

Vielen Dank für Ihre Aufmerksamkeit!

Kontakt

andreas.hoheisel@first.fraunhofer.de

<http://www.first.fraunhofer.de/>

<http://www.andreas-hoheisel.de/>

Download Grid Workflow Execution Service:

<http://www.gridworkflow.org/gwes/>

Bug-Reports, Wünsche, Anforderungen, Aufträge:

Bitte per E-Mail an

andreas.hoheisel@first.fraunhofer.de

