

Entwurf domänenspezifischer Modelle im Web mit Oryx

Matthias Kunze, Mathias Weske

Hasso Plattner Institut an der Universität Potsdam
Prof.-Dr.-Helmert-Straße 2-3, 14482 Potsdam
{matthias.kunze,mathias.weske}@hpi.uni-potsdam.de

Abstract: Die stetige Zunahme der Komplexität von Informationssystemen erschwert den Systementwurf mit herkömmlichen Methoden; universelle Modellierungssprachen wie UML stoßen an ihre Grenzen. Der Einsatz domänenspezifischer Modellierungssprachen verspricht Abhilfe: Der Systementwurf wird auf fachliche Konzepte verlagert, so dass Domänenexperten befähigt werden, aktiv am Systementwurf teilzunehmen.

In diesem Beitrag wird die web-basierte Modellierungsplattform Oryx vorgestellt. Als erweiterbares System ermöglicht Oryx es Benutzern, Modelle mithilfe verschiedener Modellierungssprachen zu entwickeln. Die Autoren erläutern, wie Notation, Syntax und Semantik von Modellierungssprachen auf die Architekturkomponenten Stencil Sets und Plugins von Oryx abgebildet werden und auf diese Weise domänenspezifische Modellierungssprachen in Oryx umgesetzt werden können.

1 Einleitung

Innerhalb der letzten Jahre ist die Komplexität von Informationssystemen enorm gestiegen, so dass universelle Modellierungssprachen häufig an die Grenzen ihrer Aussagekraft stoßen, da sie sich an der Architektursicht oder Implementierung eines Systems orientieren, wie z.B. die Unified Modelling Language (UML, [OMG05]). Oft verlieren sich die fachlichen Anforderungen und Eigenschaften eines derart entworfenen Systems in der Menge der technischen Details. [SVEH07]

Diesem Problem begegnen domänenspezifische Modellierungssprachen (Domain Specific Modelling Language, DSML), indem sie den Systementwurf durch Abstraktion und Formalisierung der Modelle in den eigentlichen Problemraum verlagern. Damit bieten sie Domänenexperten die Möglichkeit, sich bei dem Systementwurf auf das fachliche Problem anstelle dessen Implementierung zu konzentrieren. Anschließend kann das Resultat auf technischer Ebene umgesetzt werden. Dies kann durch Anwendungsentwickler oder durch den Einsatz modellgetriebener Entwicklungsmethoden [CH03] erfolgen. DSML stellen damit ein effektives Mittel für den problemorientierten Systementwurf dar, welches allerdings erst durch geeignete Modellierungswerkzeuge effizient eingesetzt werden kann.

Die web-basierte Modellierungsplattform Oryx ist ein solches Werkzeug. Aufgrund unserer Forschungsausrichtung haben wir uns zunächst auf eine DSML zur Prozessmodellierung konzentriert, die Business Process Modelling Notation (BPMN, [OMG09]). Mit

BPMN kann man Geschäftsprozesse auf fachlicher Ebene erfassen, ohne gleichzeitig deren technische Umsetzung in die Betrachtung einzubeziehen.

Oryx wurde als erweiterbare Webanwendung konzipiert, die sich die Paradigmen des Web 2.0 [Wat07] zu eigen macht und ohne clientseitige Installation auskommt. Diese Entscheidung hat sich als sehr sinnvoll erwiesen, weil häufig mehrere Personen unterschiedlicher Rollen an Projekten zur Prozessmodellierung beteiligt sind, deren Kooperation durch eine web-basierte Lösung signifikant verbessert werden kann. Anstatt Modelldateien per Email zu versenden, werden die Modelle zentral im Web gehalten; alle Beteiligten greifen stets auf aktuelle Daten zu. Diese Eigenschaften der gewählten Architektur könnten sich auch für andere Modellierungsdomänen als sinnvoll erweisen.

Im Folgenden beleuchtet Abschnitt 2 zunächst die Motivation und Anforderungen zur Entwicklung einer erweiterbaren Modellierungsumgebung, bevor deren Umsetzung in Funktionsweise und Design der Plattform erklärt werden. In Abschnitt 3 wird die Realisierung von Modellierungssprachen in Oryx erläutert. Konkret wird gezeigt, wie Notation, Syntax und Semantik einer Modellierungssprache in Oryx spezifiziert werden können. Abschnitt 4 ordnet Oryx in den Kontext modellgetriebener Entwicklungsmethoden und -werkzeuge ein. Abschließend werden eine Zusammenfassung und ein Ausblick gegeben.

2 Design und Funktionsweise der Modellierungsplattform

Für den Endanwender von Oryx setzt sich die Modellierungsplattform aus zwei Komponenten zusammen: Zum Einen wird den Benutzern mit der Modellierungskomponente – dem *Editor* – ein grafisches Werkzeug zum Bearbeiten von Modellen zur Verfügung gestellt; zum Anderen werden die erzeugten Modelle im *Repository* zentral gespeichert. Dieses Modellregister bietet darüber hinaus Funktionen zum Suchen und Verwalten der Modelle an, unter Anderem zum gemeinsamen Bearbeiten (Sharing), zur Markierung (Tagging) sowie zum Bewerten von Modellen (Rating).

2.1 Motivation und Anforderungen

Oryx basiert auf zwei zentralen Intentionen: Wiederverwendung von Editorfunktionalität für unterschiedliche Modellierungssprachen und zentrale Verwaltung von Modellen für die akademische Forschungsgemeinschaft. Aufgrund der Ausrichtung dieses Beitrags konzentrieren wir uns auf die erste Intention.

Jede graphische Modellierung stellt hohe Anforderungen an Modellierungswerkzeuge. Zunächst müssen unterschiedliche Modellelemente graphisch zugreifbar und editierbar sein. Eine Reihe von Editor-Funktionen muss bereitgestellt werden, um den Modellierern die Arbeit so einfach wie möglich zu machen, etwa das Ausrichten von Modellelementen und das automatische Andocken von Verbindern. Zudem müssen Modelle verwaltet werden, also beispielsweise sicher gespeichert, kopiert und verschoben werden können.

Diese Basisfunktionen müssen in jedem Modellierungswerkzeug vorhanden sein. Hinzu kommen jeweils sprachspezifische Funktionen, etwa bestimmte Verknüpfungsregeln für Modellelemente, die das Werkzeug beachten muss. So kann in einem Petri-Netz beispielsweise eine Transition nur mit Stellen verbunden werden, nicht aber mit anderen Transitionen, während in BPMN Aktivitäten direkt miteinander verbunden werden können. Diese unterschiedlichen syntaktischen Regeln müssen daher in einer Modellierungsplattform formulierbar sein. Im weiteren Verlauf dieses Kapitels erläutern wir, wie diese Aspekte umgesetzt wurden.

2.2 Editor – Modellierungskomponente

Abbildung 1 zeigt die Standardansicht des Editors mit dem BPMN Stencil Set. Durch Laden unterschiedlicher Stencil Sets kann der Oryx Editor unterschiedliche Modellierungssprachen unterstützen (siehe Abschnitt 3.1). Im Zentrum befindet sich die Zeichenfläche, in der Modellelemente hinzugefügt und Modelle bearbeitet werden können.

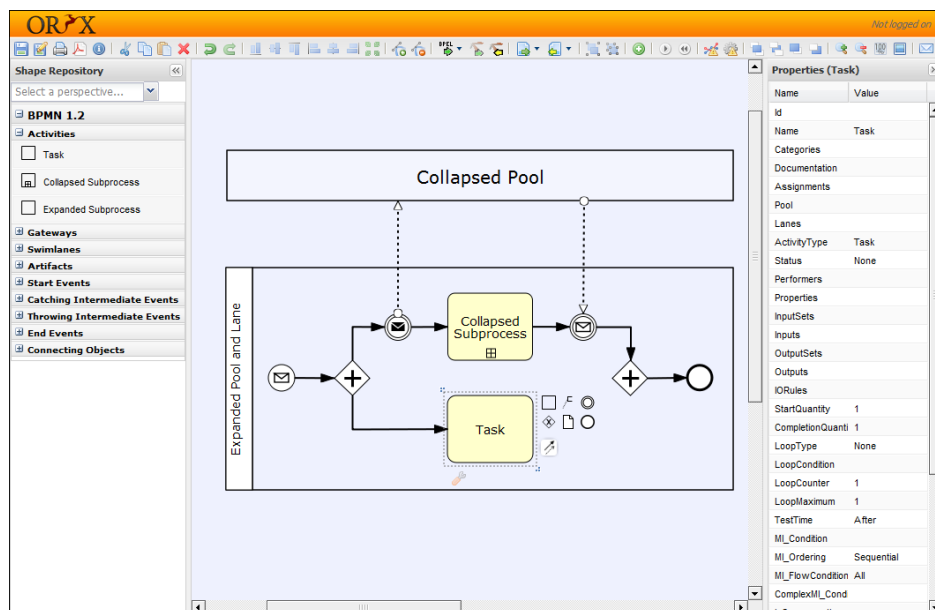


Abbildung 1: Oryx-Modellierungskomponente

Auf der Zeichenfläche wurde ein Modellelement markiert („Task“). Dabei wird das *Shape Menu* angezeigt, welches ausgehend von der Stencil-Set-Spezifikation dynamisch eine Auswahl der Modellelemente anzeigt, welche mit dem gerade markierten Element verknüpft werden können. Das Shape Menu unterstützt die Modellierung durch Darstellen aller an diesem Punkt syntaktisch möglicher Modellelemente.

Oberhalb der Zeichenfläche befindet sich die *Toolbar*, welche übliche Editorfunktionen, z.B. Speichern, Cut/Copy/Paste und Undo/Redo, enthält. Zusätzlich können weitere Funktionen geladen und in der Toolbar registriert werden. Dazu zählen solche, die abhängig von der aktuell gewählten Modellierungssprache sind, beispielsweise Syntaxprüfung sowie eine Animation der Prozessablauflogik. Diese Funktionen sind als Plugins realisiert; der Plugin-Mechanismus wird in Abschnitt 2.5 beschrieben. Auf der linken Seite findet sich das *Shape Repository*, welches alle verfügbaren Modellelemente des aktiven Stencil Sets enthält. Rechts findet sich schließlich das *Property Window*, welches die Eigenschaften von Modellelementen anzeigt, sowie deren Bearbeitung erlaubt.

2.3 Repository – Modellregister

Der zweite Screenshot (Abbildung 2) zeigt die Benutzeroberfläche des Modellregisters. Ein Benutzer hat dabei Zugriff auf Modelle, für die der momentan angemeldete Benutzer Lese- oder Schreibrechte besitzt. Jedes Modell wird durch eine kleine Vorschau angezeigt, wodurch der Benutzer schnell erkennt, worum es sich bei dem Modell handelt.

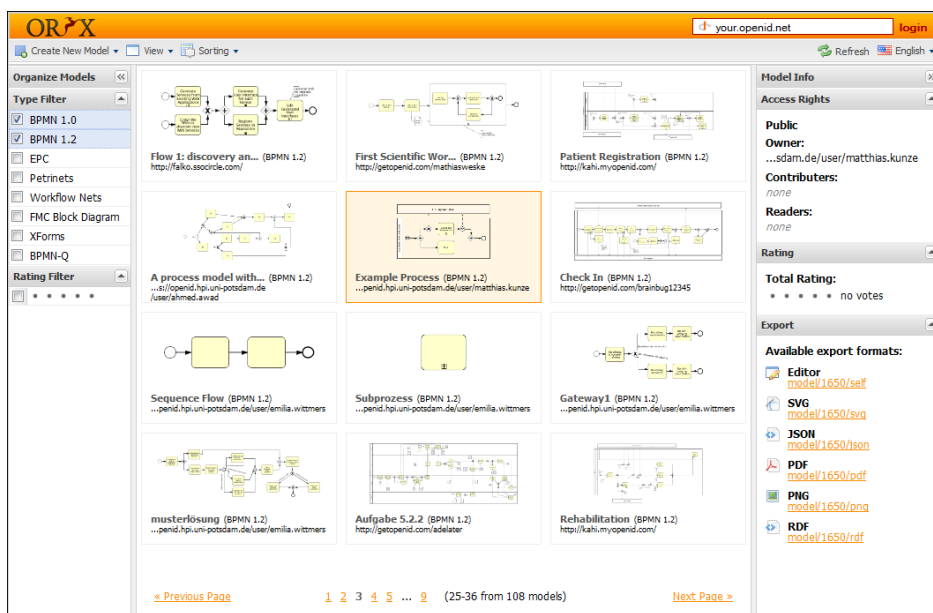


Abbildung 2: Oryx-Modellregister

Das Repository ermöglicht es Benutzern, durch Auswahl eines Modells dessen Eigenschaften, wie Name oder Beschreibung, zu ändern, Tags und Bewertungen hinzuzufügen, weiteren Benutzern Zugriff auf das Modell zu gewähren, sowie es in verschiedene Darstellungsformate zu exportieren, unter anderem nach PNG, SVG und PDF. Außerdem lässt

sich das Modell von hier aus in der Modellierungskomponente öffnen, sowie ein neues Modell in einer ausgewählten Modellierungssprache anlegen.

2.4 Architektur

Die Systemarchitektur von Oryx (Abbildung 3) zeigt die beiden genannten Komponenten Editor und Repository. Beide sind eigenständige Anwendungen, die im Webbrowser des Benutzers ausgeführt werden und über HTTP auf serverseitige Komponenten zugreifen. Technisch besteht jede dieser Komponenten aus einer XHTML-Seite, die benötigte Daten (Anmeldeinformationen, Modelle) und Anwendungslogik (JavaScript) von einem Webserver lädt und dann die Webanwendung ausführt. Dabei wird, wie für RIA typisch, das Document Object Model (DOM) der Seite dynamisch verändert, um eine desktopähnliche Benutzungsschnittstelle zu erzeugen. Zu diesem Zweck wird das Framework *Ext.js*¹ verwendet. Des Weiteren wird *prototype.js*² eingesetzt. Diese Bibliothek stellt Erweiterungen der JavaScript-Typen in Form von Higher Order Functions, sowie ein Klassenkonzept und komfortable AJAX-Kommunikation zur Verfügung.

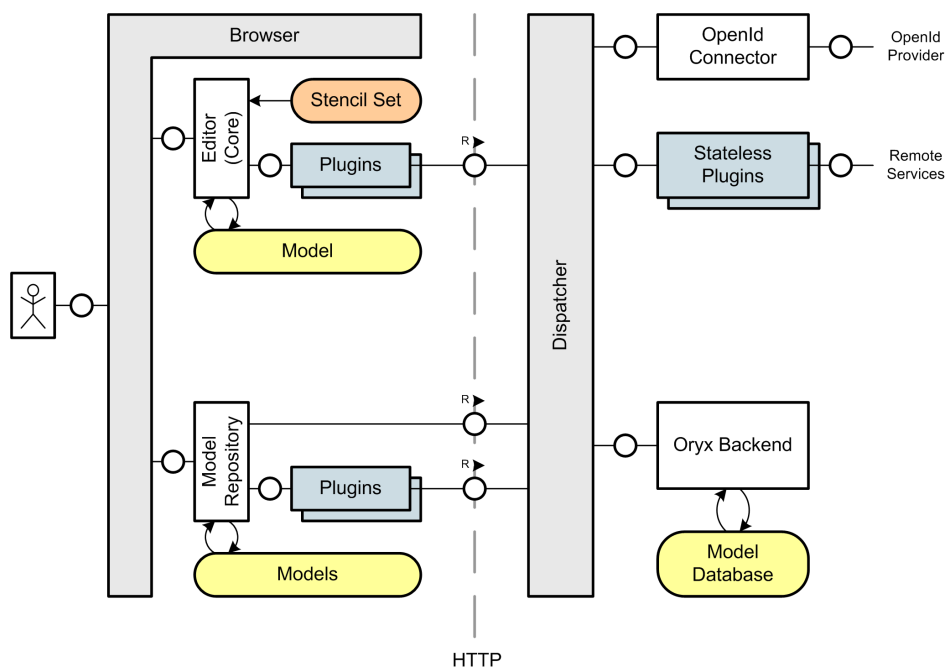


Abbildung 3: Oryx-Systemarchitektur

¹siehe <http://extjs.com>

²siehe <http://prototypejs.org>

Jedes Modell wird durch eine URL identifiziert, und ist mit weiteren URLs assoziiert. Auf diesem Wege ist es möglich, das Modell in verschiedenen Repräsentationen zu betrachten oder zu referenzieren, z.B. als Portable Network Graphic mit dem abschließenden URL-Segment `/png`. URLs die auf `/self` enden, referenzieren das Modell in einer editierbaren Repräsentation, bei deren Aufruf der Editor gestartet wird. Das Modell wird dann in SVG dargestellt und im Editor eingebettet, welches mit der Benutzeroberfläche verknüpft ist und somit interaktiv bearbeitet werden kann.

Serverseitig besteht Oryx hauptsächlich aus drei Komponenten, welche als Servlets in Java EE³ implementiert sind. Der *OpenId-Connector* dient zur Authentifizierung von Benutzern via OpenId [OID07], einem dezentralisierten Authentifizierungssystem: Ein Benutzer hat eine Identität, die bei einem entsprechendem Provider gespeichert und verwaltet wird. Damit kann er sich gegen andere Systeme authentifizieren, indem beim Identitätsprovider die OpenId mit Passwort abgefragt wird. Somit entfällt für Oryx die Notwendigkeit Benutzerkonten zu verwalten. Außerdem können Modelle mit Dritten gemeinsam bearbeitet werden, ohne dass diese sich vorher bei Oryx anmelden müssen, wenn sie eine OpenId haben.

Stateless Plugins beinhalten serverseitig ausgeführte Funktionen, die vom Editor und dem Repository aufgerufen werden können, diese werden im folgenden Abschnitt detailliert diskutiert. *Oryx Backend* bezeichnet den Modellspeicher. Dieser enthält alle gespeicherten Modelle sowie deren Eigenschaften, wie sie in Abschnitt 2.3 erläutert wurden.

2.5 Plugins

Oryx sieht vor, funktionale Erweiterungen durch den Einsatz von Plugins zu realisieren. Dazu wurden zwei Konzepte entwickelt: Zum Einen erlauben clientseitige Plugins, die im Kontext des Webbrowsers ausgeführt werden, das Hinzufügen von Funktionalität zum Editor bzw. dem Repository. Der Kern des Editors („Core“ in Abbildung 3) ist für den Aufbau der Benutzeroberfläche, grundlegende Modellierungsfunktionen, wie das Interpretieren der Stencil Sets, und die Pluginverwaltung zuständig. Die bereits genannten Teile der Benutzungsschnittstelle – Property Window, Shape Repository und die Toolbar – sind selbst als Plugins implementiert. Clientseitige Plugins können sich bei der Benutzungsschnittstelle registrieren und so direkt mit dem Benutzer interagieren; dazu steht diesen unter Anderem das geladene Modell vollständig zur Verfügung.

In manchen Fällen reichen die Fähigkeiten von Webbrowsern jedoch nicht aus, um gewünschte Funktionen zu realisieren, beispielsweise beim Export in das PNG-Format. Andererseits ist es wünschenswert, Oryx mit weiteren Systemen zu verbinden, im Rahmen von Prozessmodellen beispielsweise mit einer Ausführungseingine (vgl. [KDW08]).

In solchen Fällen besteht die Möglichkeit, clientseitige Plugins durch serverseitige Plugins zu ergänzen. Dies erlaubt es auch, auf existierende Services im Web zuzugreifen. In jedem Fall muss ein clientseitiges Plugin im Editor geladen sein, welches Nutzerinteraktion und AJAX-Kommunikation mit seinem serverseitigen Pendant realisiert.

³siehe <http://java.sun.com/javase/>

Serverseitige Plugins sind isolierte Komponenten. Aus Gründen der Sicherheit und Stabilität des Backends haben diese keinen Zugriff auf das Modellrepository und können keinen Interaktionszustand speichern. Daher werden sie auch *Stateless Plugins* genannt.

3 Domänenspezifische Modellierungssprachen in Oryx

Im Gegensatz zu informellen Diagrammen besitzen Modellierungssprachen per Definition bestimmte Eigenschaften, die Aussehen, Korrektheit und die Bedeutung von Modellen auszeichnen. Entsprechend [JS06, TR03] lassen sich DSML mit folgenden Konzepten ausführen:

Notation Konzepte (Begriffe) der Domäne sowie deren Repräsentation in einer Sprachspezifikation, beispielsweise durch Symbole

Syntax Regeln und Einschränkungen hinsichtlich der Komposition dieser Konzepte, die den Modellierungsprozess steuern

Semantik Abbildungen dieser Konzepte auf verschiedene Bedeutungen

Oryx unterstützt diese drei Konzepte mithilfe von *Stencil Sets* und *Plugins*, wie im Folgenden erläutert wird.

3.1 Stencil Sets – Notation und Syntax von Modellierungssprachen

Ein Stencil Set ist eine textuell in JSON (JavaScript Object Notation) spezifizierte Beschreibung der Domänenkonzepte und enthält das gesamte Repertoire der verfügbaren Modellelementtypen (Shapes). Ein Modellelementtyp entspricht dabei genau einem Konzept der Domäne und ist dem Typ Knoten oder Kante zugeordnet, hat eine eindeutige Identität, einen Namen und eine Beschreibung, sowie eine Definition von Eigenschaften, die ein Modellelement näher spezifizieren bzw. konfigurieren. Zur grafischen Repräsentation existiert für jeden Modellelementtyp eine SVG-Spezifikation für dessen Darstellung auf der Zeichenfläche sowie ein Symbol im PNG-Format, welches dieses im Shape Repository und Shape Menu (vgl. Abbildung 1) repräsentiert.

Listing 1 ist ein Ausschnitt der Spezifikation für einen Task in BPMN. Man erkennt die oben genannten Charakteristika sowie Pfadangaben, die einerseits seine Repräsentation in SVG, andererseits das PNG-Symbol referenzieren.

Neben der Notation spezifiziert ein Stencil Set aber auch die Syntax der Modellierungssprache, also die Regeln, die die Korrektheit eines Modells bestimmen. Schon während des Modellierens wird sichergestellt, dass diese Regeln nicht verletzt werden. Kompositionsregeln im Stencil Set legen fest, mit welchen anderen Elementen ein Element verbunden werden darf oder welche Elemente in anderen Elementen enthalten sein dürfen. In BPMN ist es beispielsweise nicht möglich, Tasks direkt in Pools unterzubringen, da letztere immer

nur eine Organisationseinheit, nicht aber eine ausführende Rolle beschreiben. Daher müssen in einem Pool erst Rollen in Form von Lanes modelliert werden, in denen die Tasks dann untergebracht werden können. Außerdem kann man die Anzahl von Kanten pro Knoten auf Modellebene beschränken. In BPMN ist es beispielsweise nicht erlaubt, mehr als eine eingehende Kontrollflusskante in ein End-Ereignis fließen zu lassen; Start-Ereignisse dürfen keine eingehende Kontrollflusskanten besitzen, etc.

```
{
  "type": "node",
  "id": "Task",
  "title": "Task",
  "groups": ["Activities"],
  "description": "An atomic activity.",
  "view": "activity/node.task.svg",
  "icon": "new_task.png",
  "properties": [
    ...
  ]
}
```

Listing 1: JSON-Spezifikation einer BPMN-Aktivität im Stencil Set

In manchen Fällen ist es nötig, komplexere Regeln zu definieren, die über Verbindungs- oder Enthaltenseinrelationen zweier Modellelemente hinausgehen. Dies ist mit Hilfe von Stencil Set Extensions möglich. Mit diesen können Stencil Sets erweitert werden, indem sie sich bei bestimmten Shapes registrieren und deren Verhalten beim Modellieren beeinflussen.

Wird ein derart erweitertes Modellelement im Modell verwendet, kann die Extension zur Laufzeit entscheiden, welche Kompositionen korrekt sind. Dies ermöglicht die Definition kombinatorischer Regeln, welche sich aus der Konfiguration verschiedener Elemente des gesamten Modells ergeben. In BPMN wäre eine solche kombinatorische Regel, dass eine Message-Flow-Kante zwischen zwei Tasks nicht innerhalb des selben Pools auftreten darf. Dabei beeinflussen nicht nur die verbindende Kante und die verbundenen Knoten die Entscheidung über korrekte Syntax, sondern auch deren Beziehung im Gesamtmodell.

3.2 Plugins – Hinzufügen von Semantik zu Modellierungssprachen

Semantik bezeichnet die Bedeutung von wohlstrukturierten, d.h. syntaktisch korrekten, Modellen [tHP97]. Erst durch das Hinzufügen von Abbildungen der Modellsyntax auf eine solche Bedeutung entsteht ein Mehrwert für Modelle, der über dokumentarische Erfassung hinausgeht. Dabei existiert kein einheitliches Verständnis darüber, wie Semantik für Modellierungssprachen zu formalisieren ist [CK07].

Im Kontext von modellgetriebener Softwareentwicklung meint die Zuordnung von Semantik häufig die Spezifikation von Generatoren, welche ausgehend von einem abstrakten

Modell ein konkreteres Modell erzeugen (translational Semantics). Im Bereich Prozessausführung sind besonders Ausführungssemantiken⁴ interessant, mit deren Hilfe man Modelle direkt in einem entsprechend konfigurierten Interpreter ausführen lassen kann (operational Semantics). Sehr häufig werden auch Modelle einer domänenspezifischen Sprache wie BPMN in domänenübergreifende Sprachen wie Petrinetze übersetzt (denotational Semantics), beispielsweise um für diese formale Validierungen auf abstrakter Ebene durchführen zu können (axiomatic Semantics).

Wie schon erwähnt wurde, finden Semantiken in diesem Sinne keine Berücksichtigung im Stencil Set, können aber als Plugins implementiert werden. Dabei ist es möglich, diese clientseitig oder serverseitig als Stateless Plugins auszuführen, wie bereits in Abschnitt 2.5 vorgestellt wurde. Dazu zählen Transformationen in verschiedene Formate, z.B. BPMN nach BPEL (vgl. [PDKL07]), Verifikations- und Validierungsmechanismen, z.B. EPK-Soundness-Checks (vgl. [Men07]) und andere Modellanalysen.

Neben dem grafischen Export von Modellen, z.B. nach PNG und PDF, lassen sich diese auch in strukturierte, maschineninterpretierbare Datenformate exportieren. Auf diese Weise können Modelle, die in Oryx entworfen wurden, auch in anderen Werkzeugen verwendet werden. Ein solches ist beispielsweise das Prozessanalysewerkzeug ProM [vdAvDG⁺07]. Diese Formate wurden zumeist eigens zur Serialisierung domänenspezifischer Modelle definiert, wie die XML Process Definition Language (XPDL, [WfM08]); es werden aber auch generische Formate unterstützt. So existiert für Modelle, unabhängig von deren Typ, die Möglichkeit, sie nach RDF [W3C04] zu konvertieren. Für RDF existieren wiederum Parser für verschiedene Programmiersprachen, die entsprechende Dokumente in Objektmodelle umsetzen, auf denen dann diverse Operationen ausgeführt werden. Zudem können mit XML-Transformationen andere Formate aus RDF erzeugt werden.

4 Verwandte Arbeiten

Das Thema domänenspezifische Modellierungssprachen wird oft im Zusammenhang mit modellgetriebener Entwicklung referenziert, da sich der Mehrwert dieser Sprachen eben aus der Abbildung auf Sprachen höherer Konkretisierung ergibt (translational Semantics). Typischerweise enthalten Werkzeuge zur Modellierung von DSML selbst Funktionalität zur Erzeugung der Sprache, z.B. Eclipse GMF⁵ oder die Generic Modeling Environment⁶. Eine Auswahl dieser wird in [AFR06] vorgestellt.

Dabei liegt der Fokus auf der Entwicklung eines expliziten, formalen Metamodells, aus dem dann die Konfiguration eines Werkzeugs generiert wird. Im Umkehrschluss entstehen aus diesem Zusammenhang hybride Werkzeuge, die DSML nicht nur verwenden, sondern eben auch erzeugen und aktualisieren können, bis hin zur Unterstützung durchgängiger Techniken zur modellgetriebenen Softwareentwicklung.

⁴Für BPMN ist bis zur Version 1.2 keine Ausführungssemantik spezifiziert, weshalb entsprechende Modelle erst in entsprechende Sprachen übersetzt werden, z.B. YAWL [vdAtH05] oder BPEL [OAS07]. Die BPMN Spezifikation ab der Version 2.0 wird hingegen eine explizite Ausführungssemantik enthalten.

⁵siehe <http://www.eclipse.org/modeling/gmf/>

⁶siehe <http://www.isis.vanderbilt.edu/Projects/gme/>

Im Gegensatz zu diesen Werkzeugen wurde Oryx aber nicht mit dem vorrangigen Ziel entwickelt, modellgetriebene Techniken zu unterstützen, sondern eine erweiterbare Modellierungsplattform zu errichten, welche die Zusammenarbeit mit mehreren Personen aktiv unterstützt. Aus diesem Grund findet sich in Oryx konzeptionell auch kein formales Metamodell bzw. keine formale Sprachspezifikation. Wie allerdings in Abschnitt 3 erläutert wurde, lassen sich DSML durch Stencil Sets (Notation und Syntax) und Plugins (Semantik) umsetzen.

Ein Metamodell lässt sich bei Oryx allenfalls in der textuellen Repräsentation des Stencil Sets wiederfinden. Eine Konformität zu MOF [OMG02] und die damit einhergehenden Vorteile, wie automatisches Model Checking auf Grundlage des MOF Metamodells oder werkzeugunabhängige Interoperabilität basierend auf XMI [OMG07], sind nicht vorhanden. Das ist zum Einen darin begründet, dass Oryx einen einfachen Ansatz bezüglich Modellierungssprachen verfolgt und zum Anderen, dass verwendete Modellierungssprachen nicht zwangsläufig MOF-konform sind, dazu zählen BPMN bis zur Version 1.2 [OMG09] und EPK [KNS92]. Im Gegensatz zu formalen Metamodellen lässt sich in den Stencil Sets von Oryx auch keine abstrakte Syntax [tHP97] definieren, da hier die Syntax einer Modellierungssprache an seine Repräsentation (Notation) gebunden ist.

5 Zusammenfassung

In dieser Arbeit wurde Oryx im Kontext domänenspezifischer Modellierung vorgestellt. In Hinblick auf Design und Einsatz der Plattform wurde gezeigt, wie sich die Anforderungen von Oryx an den Eigenschaften einer erweiterbaren Modellierungsplattform ausrichten und wie diese umgesetzt wurden. Oryx umfasst dabei nicht nur einen Modelleditor sondern auch ein Modellrepository, in dem Modelle zentral gespeichert und verwaltet werden können. Von dort lassen sie sich in verschiedene Formate exportieren und stehen so weiteren Analysemethoden und -werkzeugen zur Verfügung.

Obwohl Oryx nicht als Werkzeug zur Unterstützung domänenspezifischer Modellierungssprachen konzipiert wurde, eignet es sich für diesen Zweck. Die Definition einer Modellierungssprache besteht aus der Spezifikation von Notation, Syntax und Semantik. Es wurde gezeigt, wie diese Konzepte in Oryx mit Hilfe von Stencil Sets (Notation und Syntax) und Plugins (Semantik) umgesetzt werden können. Im Rahmen modellgetriebener Entwicklung eignet sich Oryx bedingt, da formale Metamodelle und umfassende Techniken, wie die automatische Generierung konkreter Modelle durch die Plattform nicht hinreichend unterstützt werden.

Oryx hat sich in der Praxis, auch gegenüber anderen Werkzeugen, erfolgreich bewährt. Da die Plattform als erweiterbare Webanwendung umgesetzt wurde, fördert sie die Kooperation mehrerer, an der Modellierung beteiligter Personen. Da sowohl Modellierungswerkzeug als auch Modelle zentral im Web gespeichert und angeboten werden, haben alle Beteiligten stets Zugriff auf aktuelle Daten, auch über Organisationsgrenzen hinweg. Die gewählte Architektur und die resultierenden Eigenschaften können sich auch für andere Domänen als sinnvoll erweisen. Daher wendet sich das Projekt um Oryx gezielt an

Forschung und Lehre in Bereichen in denen Modellierung zum Einsatz kommt. Die Modellierungsplattform ist unter der MIT Open Source Lizenz [MIT98] veröffentlicht und steht allen Interessenten unter <http://oryx-project.org> zur Verfügung.

Danksagung. Oryx ist eine Initiative, zu der zahlreiche Personen beigetragen haben – studentische und wissenschaftliche Mitarbeiter des Hasso Plattner Instituts und anderer Hochschulinstitute, sowie Beitragende unserer Community. Die Autoren danken dem Oryx-Team und allen Beteiligten für die umfangreichen Entwicklungen.

Literatur

- [AFR06] Daniel Amyot, Hanna Farah und Jean-François Roy. Evaluation of Development Tools for Domain-Specific Modeling Languages. In Reinhard Gotzhein und Rick Reed, Hrsg., *SAM*, Jgg. 4320 of *Lecture Notes in Computer Science*, Seiten 183–197. Springer, 2006.
- [CH03] Krzysztof Czarnecki und Simon Helsen. Classification of Model Transformation Approaches. In *OOPSLA'03 Workshop on Generative Techniques in the Context of Model-Driven Architecture*, 2003.
- [CK07] Thomas Cleenewerck und Ivan Kurtev. Separation of concerns in translational semantics for DSLs in model engineering. In *SAC '07: Proceedings of the 2007 ACM symposium on Applied computing*, Seiten 985–992, New York, NY, USA, 2007. ACM.
- [JS06] Ethan K. Jackson und Janos Sztipanovits. Towards a formal foundation for domain specific modeling languages. In *EMSOFT '06: Proceedings of the 6th ACM & IEEE International conference on Embedded software*, Seiten 53–62, New York, NY, USA, 2006. ACM.
- [KDW08] Stefan Krumnow, Gero Decker und Mathias Weske. Modellierung von EPKs im Web mit Oryx. Bericht, Hasso-Plattner-Institut an der Universität Potsdam, 2008.
- [KNS92] G. Keller, M. Nüttgens und A.-W. Scheer. Semantische Prozessmodellierung auf der Grundlage “Ereignisgesteuerter Prozessketten (EPK)”, 1992.
- [Men07] Jan Mendling. *Detection and Prediction of Errors in EPC Business Process Models*. Dissertation, Vienna University of Economics and Business Administration, 2007.
- [MIT98] MIT. MIT Open Source License. <http://www.opensource.org/licenses/mit-license.php>, 1998.
- [OAS07] OASIS. Business Process Execution Language (BPEL) Specification, Version 2.0. <http://www.omg.org/spec/UML/>, Nov 2007.
- [OID07] OIDF. OpenID Specification. <http://openid.net/developers/specs/>, Dec 2007.
- [OMG02] OMG. Meta Object Facility (MOF) Specification, Version 2.0. <http://www.omg.org/docs/formal/02-04-03.pdf>, Apr 2002.
- [OMG05] OMG. Unified Modeling Language (UML) Specification, Version 2.0. <http://www.omg.org/spec/UML/>, Jul 2005.

- [OMG07] OMG. XML Metadata Interchange (XMI) Mapping Specification, Version 2.1.11. <http://www.omg.org/technology/documents/formal/xmi.htm>, Dec 2007.
- [OMG09] OMG. Business Process Modelling Notation (BPMN) Specification, Version 1.2. <http://www.bpmn.org/>, Feb 2009.
- [PDKL07] Kerstin Pfitzner, Gero Decker, Oliver Kopp und Frank Leymann. Web Service Choreography Configurations for BPMN. In *WESOA 2007*, LNCS, Vienna, Austria, 2007. Springer.
- [SVEH07] Thomas Stahl, Markus Völter, Sven Efftinge und Arno Haase. *Modellgetriebene Softwareentwicklung: Techniken, Engineering, Management*. Dpunkt Verlag, 2007.
- [tHP97] A. H.M. ter Hofstede und H.A. Proper. How to Formalize It? Formalization Principles for Information System Development Methods. Bericht, University of Queensland, Australia, 1997.
- [TR03] Juha-Pekka Tolvanen und Matti Rossi. MetaEdit+: defining and using domain-specific modeling languages and code generators. In *OOPSLA '03: Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, Seiten 92–93, New York, NY, USA, 2003. ACM.
- [vdAtH05] W. M. P. van der Aalst und A. H. M. ter Hofstede. YAWL: yet another workflow language. *Inf. Syst.*, 30(4):245–275, 2005.
- [vdAvDG⁺07] Wil M. P. van der Aalst, Boudewijn F. van Dongen, Christian W. Günther, R. S. Mans, Ana Karla Alves de Medeiros, Anne Rozinat, Vladimir Rubin, Minseok Song, H. M. W. (Eric) Verbeek und A. J. M. M. Weijters. ProM 4.0: Comprehensive Support for *eal* Process Analysis. In *ICATPN*, Seiten 484–494, 2007.
- [W3C04] W3C. Resource Description Framework (RDF) Specifications. <http://www.w3.org/RDF/>, 2004.
- [Wat07] Stephen Watt. Mashups – The evolution of the SOA, Part 1: Web 2.0 and foundational concepts. <http://www.ibm.com/developerworks/webservices/library/ws-soa-mashups/index.html>, Oct 2007.
- [WfM08] WfMC. Process Definition Interface – XML Process Definition Language, Version 2.1a, Oct 2008.