Architektur eines konfigurativen Referenzmodellierungswerkzeugs – adapt(x)

Tobias Rieke, Armin Stein

Abstract: *Eine Reihe von Grundsatzentscheidungen beeinflusst die Konstruktion eines konfigurativen Referenzmodellierungswerkzeugs. Hierzu zählen neben der Selektion eines adäquaten Fachkonzepts insbesondere die Auswahl der Implementierungsplattform und die damit einhergehende Softwarearchitektur des Werkzeugs. In diesem Rahmen ist auch zu entscheiden, ob die Konstruktion des Werkzeugs im Rahmen einer Neuentwicklung oder auf Grundlage bestehender Werkzeuge realisiert wird, die entsprechend erweitert werden. Der vorliegende Beitrag erläutert die Anforderungen, die ausgehend von der konfigurativen Referenzmodellierung an ein Modellierungswerkzeug zu stellen sind und leitet daraus Gestaltungsaspekte hinsichtlich der Werkzeugarchitektur ab, welche detailliert beschrieben werden.*

1 Gestaltungsentscheidungen für die Architektur eines konfigurativen Referenzmodellierungswerkzeugs

Bei der Entwicklung der Funktionalitäten, die zur Unterstützung der konfigurativen Referenzmodellierung benötigt werden, muss im Vorfeld entschieden werden, ob eine Neuentwicklung sinnvoll ist, oder ob die Weiterentwicklung vorhandener Werkzeuge als Lösung in Frage kommt. Hierbei ist insbesondere zu beachten in wie weit vorhandene Modellierungswerkzeuge überhaupt erweiterungsfähig sind bzw. ob Neuentwicklungen gegenüber etablierten Werkzeugen überhaupt Chancen eingeräumt werden können, sich am Markt durchzusetzen.

Ausgehend von diesen Überlegungen sind im Rahmen einer Studie Modellierungswerkzeuge untersucht worden, die sich in besonderem Maße für eine Weiterentwicklung eignen – sogenannte Metamodellierungswerkzeuge [DeKn07]. Ob ihrer eingeschränkten Benutzerfreundlichkeit wurde diese Alternative jedoch verworfen. Ebenfalls wurde die Alternative einer

Eigenentwicklung verworfen, da in einem solchen Rahmen nicht vertretbarer Zusatzaufwand bzgl. der Erstellung von Standardfunktionalitäten zu erwarten war, deren Ausgereiftheit von Standardwerkzeugen zudem bei weitem übertroffen werden dürfte.

Aus diesem Grund wurde entschieden, eine Erweiterung zu entwickeln, die unabhängig vom Modellierungswerkzeug ist und dessen Mangel an Konfigurationsfunktionalität ergänzt. Hierzu werden Modelle, die mithilfe einer beliebigen Modellierungssoftware erstellt werden, exportiert, in ein werkzeug- und plattformunabhängiges Format überführt, in der Erweiterung konfiguriert und wiederum als werkzeugabhängige Datei rückimportiert. Die einzelnen Schritte des Im- und Export- sowie des Transformationsprozesses bleiben dabei aus ergonomischen Gründen für den Nutzer transparent.

2 Die Softwarearchitektur von adapt(x)

Eine solche prototypische Implementierung wurde im Rahmen des *RefMod06*-Projekts unter dem Namen *adapt(x)* in Verbindung mit dem Modellierungswerkzeug *ARIS Business Architect 7.0* der *IDS Scheer AG* entwickelt. Dieses Werkzeug zeichnet sich durch die Bereitstellung einer Skriptsprache aus, die es ermöglicht, auf viele Bereiche des Programms Einfluss zu nehmen. Abbildung 1 stellt die aus den Anforderungen entstandene Architektur der Erweiterung dar und wird im Folgenden erläutert.

Auf der linken Seite sind die innerhalb des Basismodellierungswerkzeugs zu implementierenden Elemente abgebildet, die zur Vorbereitung der Konfiguration dienen. Die von der jeweiligen Modellierungsumgebung unabhängige Komponente *adapt(x)*, welche auf einem klassischen Drei-Schichten-Modell, getrennt in Darstellungs-, Logik- und Datenbankschicht, beruht, ist rechts dargestellt. Weiterhin existiert eine XML-Beschreibung [W3C06] in Form einer XML-Schema-Datei, die als Schnittstelle zur Modellierungsumgebung dient. Auf Seiten der Modellierungsumgebung müssen Möglichkeiten geschaffen werden, um Modelle konfigurierbar zu machen, d. h. es müssen sowohl Konfigurationsterme als auch Konfigurationsattribute an Elementausprägungen und Modelle annotiert werden können (vgl. hierzu den Beitrag von PATRICK DELFMANN und ARMIN STEIN in diesem Band). Die möglichen Ausprägungen der verwendeten Konfigurationsparameter werden in einer MySQL-Datenbank gehalten [MySQ06], auf die auch von *adapt(x)* zugegriffen wird. Hierbei sollte die Kommunikation bidirektional funktionieren, was bedeutet, dass Parameterausprägungen, die in der Modellierungsumgebung (neu) angelegt

werden, so in die Datenbank geschrieben werden, dass sie in *adapt(x)* verfügbar sind, und umgekehrt. Weiterhin ist es wünschenswert, dass ein automatisierter Aufruf von *adapt(x)* aus der Modellierungsumgebung stattfinden kann, um ein benutzerunfreundliches manuelles Exportieren und Importieren zu vermeiden.

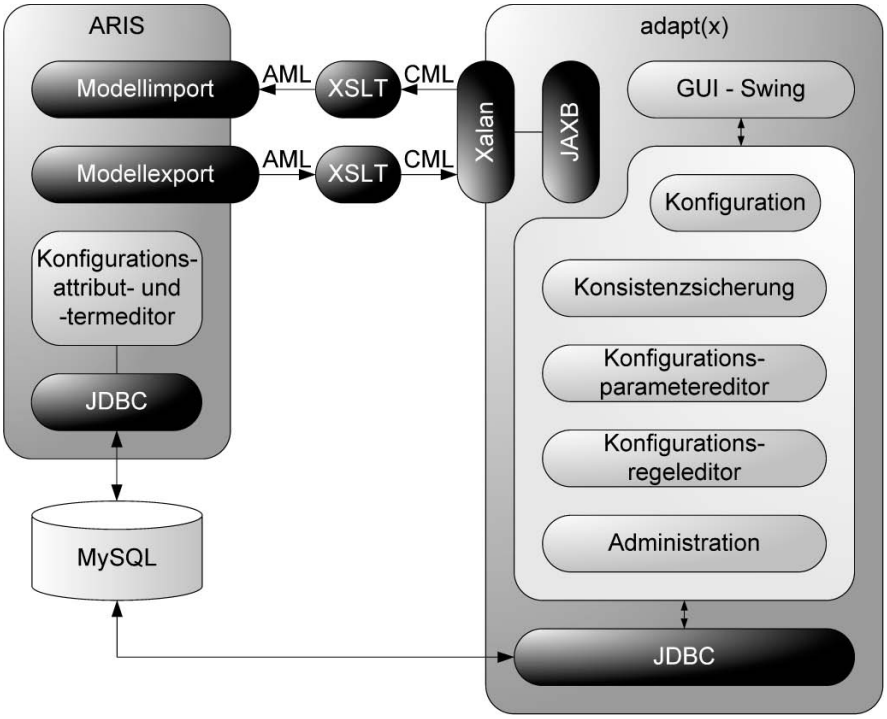

Abbildung 1: Architektur von *adapt(x)*

Als visuelle Schnittstelle für den Benutzer stehen Fenster via *Java Swing* [Sun06a] und *Java AWT* [Sun06b] zur Verfügung, die über Schnittstellen auf Klassen und Methoden innerhalb der Geschäftslogik zugreifen.

3 XML-basierter Austausch von Modelldaten: CML

Die Kommunikationsschicht zum Modellierungswerkzeug wird durch eine MySQL-Datenbank und ein XML-Schema umgesetzt. Die Datenbank einerseits enthält hierbei die Konfigurationsparameter, deren Ausprägungen, Konfigurationsregeln, Benutzerinformationen, Sprachdefinitionen und Anwendungsregeln der verwendeten Modellierungsumgebung. Um eine Kon-

figuration unabhängig von einer speziellen Modellierungsumgebung vornehmen zu können, wurde andererseits ein XML-Format entwickelt, dessen Struktur sich nach den für die Konfiguration relevanten und benötigten Informationen richtet. Angelehnt an die Anforderungen des „*co*nfigurative *in*formation modeling“ wurde die Bezeichnung *CML* für „coin markup language“ gewählt. Dieses Schema umfasst allgemeine (*MiscArea*), definitions- (*DefArea*) und modellspezifische (*ModelArea*) Bereiche (vgl. Abbildung 2 links, unterer Ast). Die zwei übergeordneten Attribute Tool und RefModID speichern einerseits eine Bezeichnung der Modellierungsumgebung, andererseits den Namen des Referenzmodells, das exportiert wurde. Anhand des Namens des Werkzeugs werden in der Datenbank die entsprechenden Metamodellregeln und Konsistenzsicherungsalgorithmen ausgewählt, die werkzeugspezifisch sind.

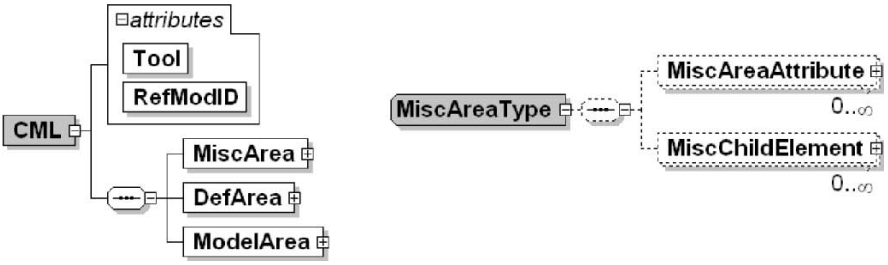

Abbildung 2: Grundstruktur der CML, Aufbau der MiscArea

Elemente, die für die Konfiguration nicht benötigt werden (*MiscChildElement*), können im Bereich *MiscArea* in verschiedenen Containern durch den Konfigurationsprozess hindurchgereicht werden. Die Bezeichnung *ChildElement* bezieht sich im Folgenden immer auf Elemente, die nicht für eine Konfiguration relevant sind, jedoch für die weitere Verwendung in der Modellierungsumgebung zwischengespeichert werden müssen. Beispiele für solche Elemente sind verwendete Schriftarten, Sprachkonventionen oder Datenbankeinstellungen des Modellierungswerkzeugs.

Im Bereich *DefArea* (vgl. Abbildung 3) werden Definitionen von Elementen (*DefType*), die innerhalb des Referenzmodells ausgeprägt werden, gespeichert. Da Definitionsausprägungen in unterschiedlichen Modellen vertreten sein können, werden die Typen unabhängig von den jeweils konkreten Modellen in diesem gesonderten Bereich abgelegt. Definitionen können wiederum werkzeugspezifische Informationen enthalten, die für eine Konfiguration nicht relevant sind – Beispiele hierfür sind der Name des Erstellers, Zeitstempel der letzten Änderungen oder intern vergebene IDs. Diese werden als *ChildElement* in der entsprechenden Rubrik abgelegt (*Def-ChildElement*). Notwendig für eine Konfiguration ist die Hinter-

legung des Namens (*DefLabel*), welche Ausprägungen eine Definition besitzt (*DefOcc*), welche Kanten ein-, und welche ausgehen (*InDefCxn*, *OutDefCxn*) sowie Verbindungen zu verknüpften Modellen, wenn ein Element als Ausgangspunkt für eine Verfeinerung dient (*LinkedModel*). Innerhalb der Attribute (Bereich *attributes*) werden Informationen über den Elementtyp (*DefType*), die eindeutige ID der Definition (*DefID*) sowie ein zugeordnetes Symbol (*DefSymbol*) gespeichert. Zusätzliche, tiefer geschachtelte Attribute können in dem Container *DefAttribute* gespeichert werden. Mit diesen Angaben kann eine Elementtypselektion durchgeführt werden, indem jedes Element, das einem gewählten Typ entspricht, aus der CML entfernt wird.

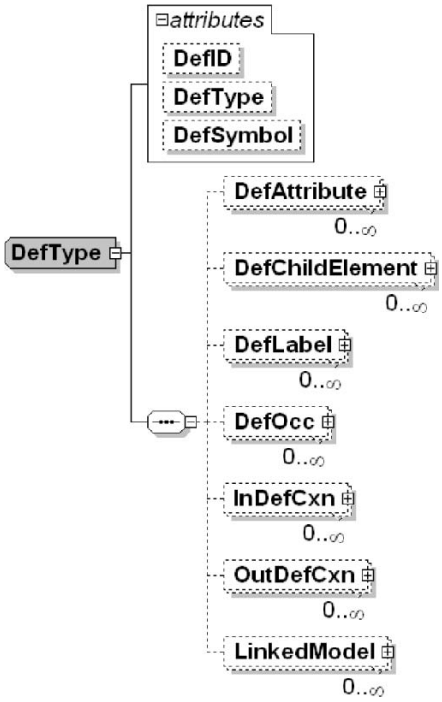

Abbildung 3: DefArea mit Darstellung der untergeordneten Elemente

Durch die Aufzählung sämtlicher Ausprägungen (*Occurrences*) der Definition innerhalb des XML-Elements *DefOcc* können diese und die damit im Zusammenhang stehenden Kanten beim Entfernen einer Definition ebenfalls gelöscht werden.

Modelle sind immer jeweils genau einer Technik zugeordnet und werden als *ModType*-Element im Bereich *ModelArea* geführt (vgl. Abbildung 4). Der Typ eines Modells (*attributes: ModType*) kann – im Falle von

ARIS – bspw. „MT_EEPC“ sein, ein Code, der für „ModelType Enhanced EventProcessChain“ steht. Damit *adapt(x)* werkzeugunabhängig erkennen kann, dass es sich bei diesem Kürzel um das werkzeugspezifische für „EPK“ handelt, muss in der Datenbank im Vorfeld ein Namespace für das Modellierungswerkzeug angelegt werden, anhand dessen die Übersetzung vorgenommen werden kann.

Durch die optionalen Elemente *COINAttribut* und *COINTerm* können auch attributierte oder mit Termen versehene Modelle – unabhängig von ihrem Typ – mittels einer Modellselektion entfernt werden. Ansonsten entsprechen die Elemente des *ModTyp*“ denen des *DefType*; *attributes: ModID* ist eine eindeutige ID des Modells, *attributes: ModName* eine Bezeichnung. *ModLabel* kann Modellbezeichnungen in unterschiedlichen Sprachen aufnehmen, *ModChildElement* umfasst wiederum nicht für die Konfiguration benötigte Elemente, *ModAttribute* komplexere, nicht benötigte Attributstrukturen.

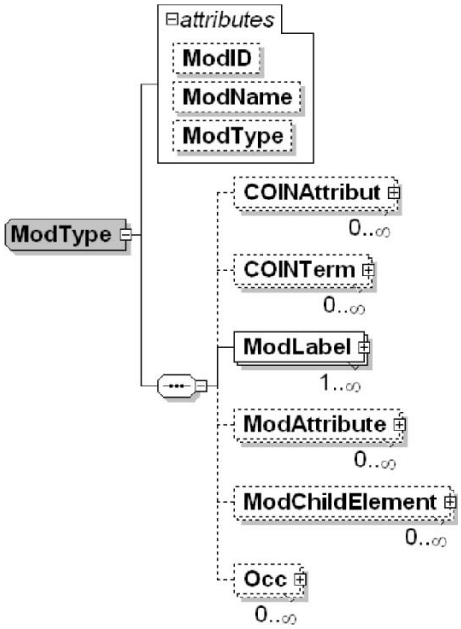

Abbildung 4: Modellelement mit Darstellung der untergeordneten Elemente

Die in Modellen gebündelten Ausprägungen sind unter dem Element *Occ* abgelegt (vgl. nochmals Abbildung 4). In den *attributes* auf oberster Ebene werden die für die Konfiguration zusätzlich relevanten Informationen gespeichert, während die Konfigurationsattribute und -terme – wie schon bei

den Modellen – in zusätzlichen Elementen gespeichert werden: *COINAttribut* und *COINTerm* (vgl. Abbildung 5).

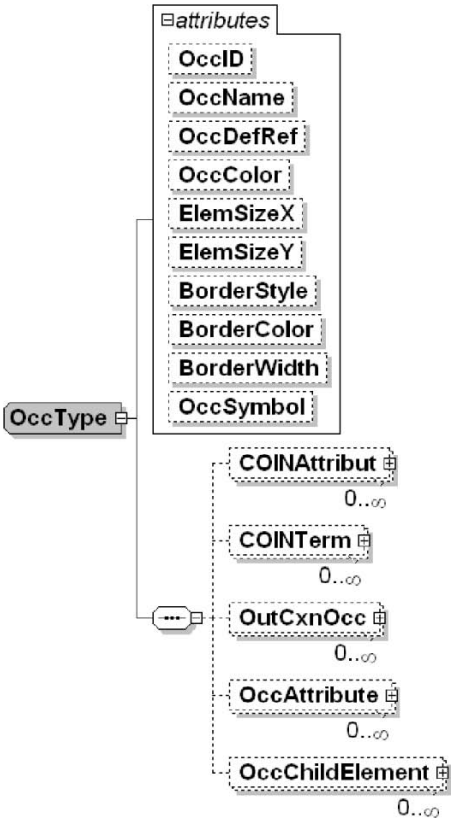

Abbildung 5: Ausprägungselement mit Darstellung der untergeordneten Elemente

Die *attributes* des Elements dienen in erster Linie der Ermöglichung einer Darstellungsvariation, indem die Farbe (*OccColor*), Größe (*ElemSizeX*, *ElemSizeY*) und Rahmenart (*BorderStyle*, *BorderColor*, *BorderWidth*) hinterlegt wird. Für eine Darstellungsvariation der Symbole ist ein Attribut *OccSymbol* reserviert, das bei *ARIS* jedoch nicht geändert werden kann, da dort eine 1:1-Zuordnung zwischen Elementtyp und Darstellung in der internen Datenbank festgelegt ist. Eine Ausprägung ist eindeutig über eine *attributes:OccID* identifizierbar und trägt eine Bezeichnung (*attributes: OccName*). Weiterhin wird auf die ID der Definition der Ausprägung referenziert (*attributes:OccDefRef*). Innerhalb eines *Occurrence*-Elements werden außerdem die in den Definitionen enthaltenen Kanten ausgeprägt

(*OutCxnOcc*). *OccChildElement* und *OccAttribute* erlauben wieder für jeden Bereich das Speichern von modellabhängigen, für die Konfiguration irrelevanten Informationen.

4 Ablauf einer Konfiguration innerhalb von adapt(x)

Um schließlich ein Referenzmodell innerhalb von *adapt(x)* konfigurieren zu können, wird aus der CML-Datei durch *JAXB* [Sun06c], einem Java-XML-Objekt-Mapper, ein Java-Objekt erzeugt, dessen Elemente durch eigens für CML geschaffene *JAXB*-Klassen manipuliert werden können. Diese Geschäftslogik vereint die oben angeführten in den Werkzeugen fehlenden Funktionen. Hierzu gehören neben den Mechanismen der Konfiguration selbst auch die Überprüfung der Modellkonsistenz, eine Benutzerverwaltung sowie Klassen zur Termerzeugung und -auswertung.

adapt(x) nutzt *XALAN* [Apac05] als XSLT-Transformator, um die verschiedenen modellierungsumgebungsabhängigen XML-Sprachvarianten in CML zu übersetzen. Hierfür ist es notwendig, dass eine entsprechende XSLT-Datei für die jeweilige Modellierungsanwendung erstellt wird.

Die Datenbankschnittstelle ist mittels einer JDBC-Klasse [Sun06d] im Singleton-Verfahren realisiert, die dafür sorgt, dass lediglich eine Datenverbindung während der Session benötigt wird. Die Schnittstellen zur Geschäftslogik sind derart implementiert, dass von der zugrunde liegenden Datenbankarchitektur abstrahiert wird, was es ermöglicht, einerseits unterschiedliche Datenbanktypen zu verwenden, andererseits beliebige Zugriffsmethoden – wie bspw. *Hibernate* [Elli04] – einzubinden.

Schließlich wird während der Konfiguration ein Audit-Trail erstellt, der es ermöglicht, die Konfiguration einerseits menschenlesbar auszuwerten, andererseits aufgrund eindeutig definierter Symbole eine maschinelle Bearbeitung erlaubt. Der Audit-Trail ist insbesondere für das Controlling von Konfigurationsprozessen relevant, wie sie im Beitrag von CHRISTIAN SEEL und PETER LOOS in diesem Band thematisiert werden.

5 Vorbereitung der Konfiguration in ARIS mit Makros

Damit ein Modell von *adapt(x)* konfiguriert werden kann, muss es im Vorfeld mit Konfigurationsregeln angereichert werden. Dies bedeutet im Einzelnen, dass die für die Konfiguration relevanten Elemente – seien es Objekte oder Modelle – attributiert bzw. mit den entsprechenden Termen versehen werden. Hierzu ist es erforderlich, dass das Attributieren für den Be-

nutzer möglichst intuitiv vorgenommen werden kann. Realisiert wurde dies durch ein in Aris-Script entwickeltes Plugin innerhalb von *ARIS* in Form von Makros (vgl. Abbildung 1). Dieses besteht aus Dialogen, die den Anwender bei der Auswahl der Parameter anleiten und die Erstellung von Konfigurationsregeln unterstützen (vgl. zur Nutzung dieses Plugins den folgenden Beitrag von PATRICK DELFMANN, TOBIAS RIEKE und ARMIN STEIN). Um zu vermeiden, dass Parameter redundant angelegt werden und somit unter Umständen Inkonsistenzen auftreten, wird auf die mit *adapt(x)* gemeinsam genutzte Datenbank zugegriffen.

Mit der vorgestellten getrennten Architektur ist es möglich, mittels der Ergänzung der unterschiedlichen Modellierungswerkzeuge durch eine Anleitung zur Attributierung der Elemente nahezu jedes Modell, das als XML exportiert werden kann, zu konfigurieren.

Literaturverzeichnis

[Apac05] The Apache Software Foundation: Xalan-Java Version 2.7.0. http://xml.apache.org/xalan-j. 2005.

[DeKn07] Delfmann, P.; Knackstedt, R.: Konfiguration von Informationsmodellen. Untersuchungen zu Bedarf und Werkzeugunterstützung. In: Oberweis, A.; Weinhardt, C.; Gimpel, H.; Koschmider, A.; Pankratius, V.; Schnizler, B. (Hrsg.): eOrganisation: Service-, Prozess-, Market-Engineering. Proceedings der 8. Internationalen Tagung Wirtschaftsinformatik. Band 2. Karlsruhe 2007, S. 127-144.

[Elli04] Elliott, J.: Hibernate: a developer's notebook. Cambridge 2004.

[Sun06a] Sun Microsystems: Creating a GUI with JFC/Swing. http://java.sun.com/docs/books/tutorial/uiswing. 2006.

[Sun06b] Sun Microsystems: The AWT in 1.0 and 1.1. http://java.sun.com/products/jdk/awt. 2006.

[Sun06c] Sun Microsystems: Java Architecture for XML Binding (JAXB). http://java.sun.com/webservices/jaxb. 2006.

[Sun06d] Sun Microsystems: Java SE - Java Database Connectivity (JDBC). https://java.sun.com/javase/technologies/database/index.jsp. 2006.

[MySQ06] MySQL AB: My SQL – Die populärste Open-Source-Datenbank der Welt. http://www.mysql.de. 2006.

[W3C06] World Wide Web Consortium: Extensible Markup Language (XML). http://www.w3.org/XML. 2006.