

Ein LIANTIS Whitepaper

Modellgetriebene Generierung

Inhaltsverzeichnis

1	Überblick	3
2	Modellgetriebene Generierung.....	4
2.1	Model Driven Architecture.....	4
2.2	UML-Codegeneratoren.....	4
2.3	Die Bedeutung der Architektur.....	5
2.4	UML-Modelle ohne technische Details	5
2.5	UML-Modelle mit technischen Details	6
2.6	Das MDA-Toolkit der LIANTIS GmbH.....	8
2.7	MDA für existierende Software-Systeme	9

1 Überblick

Softwareentwicklung ist teuer, zeitaufwändig und riskant. LIANTIS bietet mit der **modellgetriebenen Generierung** ein Verfahren zur Verbesserung Ihrer Softwareentwicklung an.

Die wichtigsten Merkmale einer Softwareentwicklung mit modellgetriebener Generierung sind:

- Software-Systeme werden vor der Implementierung modelliert. Im Fall einer objektorientierten Entwicklung wird dafür ein UML™-Werkzeug verwendet.
- Das Modell dient als „Konstruktionsplan“ für Ihr Softwaresystem. Dieser Plan ist auf einem hohen Abstraktionsniveau und unabhängig von technischen Details wie der Programmiersprache, der verwendeten Datenbank etc.
- Ein Generator verarbeitet dieses Modell und erzeugt den Quellcode des Software-Systems.
- Generatoren für die modellgetriebene Generierung unterscheiden sich deutlich von den marktüblichen Standard-Generatoren. Die hauptsächlichen Unterschiede sind:
 - Das Verfahren ist mit jedem UML Werkzeug anwendbar.
 - Das UML Modell ist auf einem höheren Abstraktionsniveau und dadurch erheblich kompakter, leichter zu verstehen und zu ändern.
 - Es wird ein höherer Automatisierungsgrad erreicht, da der erzeugte Code alle Standard-Funktionalitäten enthält
 - Eine Portierung auf eine anderes technisches Rahmenwerk, eine neue Plattform oder sogar auf eine neue Programmiersprache wird wesentlich erleichtert
 - Sonstige Dateien wie z.B. „makefiles“ werden mitgeneriert

Sie erzielen mit dieser Vorgehensweise folgende Verbesserungen:

- Ein großer Teil der gesamten Anwendung wird generiert – die **Entwicklungskosten** sinken, und die **Projektlaufzeit** verkürzt sich erheblich.
- Durch die Generierung geht die Anzahl der Codierungsfehler signifikant zurück – die **Qualität** steigt, Test- und Korrekturaufwand sinken.
- Das Modell ist der Bauplan Ihres Systems und stimmt immer mit dem Programm-Code überein – die **Wartung** wird einfacher und kostengünstiger.
- **Investitionsschutz:** Vorhandene Software und erprobte Verfahren können in dieses Vorgehen integriert werden. Die Programmiersprache, Datenbank und Benutzeroberfläche kann einfacher ausgetauscht werden.

2 Modellgetriebene Generierung

Die folgenden Abschnitte beschreiben detailliert die Standards und die Verfahren, die LIANTIS im Bereich modellgetriebener Generierung einsetzt. Die Beschreibung ist dadurch notwendigerweise sehr technisch.

2.1 Model Driven Architecture

Die „Model Driven Architecture“ (MDA™) der Object Management Group (OMG) ist eine Initiative für eine neuartige Codegenerierung aus Modellen der „Unified Modeling Language“ (UML™). Die MDA™ möchte das verwirklichen, was Werkzeughersteller bereits seit Jahren versprechen: eine bessere und schnellere Entwicklung von Softwaresystemen mit Hilfe der Unified Modeling Language. Für weitere Informationen über die MDA-Initiative siehe <http://www.omg.org/mda/>.

2.2 UML-Codegeneratoren

Haben Sie für Ihr Softwareprojekt schon mal ein UML-Werkzeug verwendet? Auch einen der dort integrierten Codegeneratoren? Praktisch alle am Markt erhältlichen UML-Werkzeuge sind in der Lage, Code aus Klassendiagrammen zu erzeugen. Aber warum hat diese Technologie bis zum heutigen Tage keine weite Verbreitung gefunden? Es gibt zwei gängige Arten, ein UML-Werkzeug einzusetzen:

- Ein UML-Modell wird als Analyse-Modell erstellt. Die eigentliche Implementierung erfolgt von Hand ohne Generator. Vorteil ist die höhere Abstrahierung des Modells von der Implementierung (das klassische Analyse-Modell). Nachteile sind der höhere Codierungsaufwand und mögliche Inkonsistenzen zwischen Modell und Code. Ist das System einmal fertiggestellt, wird das Analyse-Modell im Alltagsgeschäft der ständigen Erweiterungen und Änderungen vergessen und veraltet.
- Ein UML-Modell wird erstellt und über den integrierten Generator in Quellcode umgewandelt. Eine manuelle Codierung ist dann nicht mehr notwendig, aber das Modell ist quellcode-äquivalent, d. h., jedes Element der Programmiersprache findet sich auch als ein Element im Modell wieder und umgekehrt. Ein generierbares Modell ist im letzteren Fall nur eine grafische Darstellung des Quellcodes. Sobald das Modell eine gewisse Größe erreicht, geht auch die erhoffte Übersichtlichkeit verloren. „Round Trip“ - Funktionalität ermöglicht es, am Werkzeug das System weiterzuentwickeln.

In beiden Fällen kann ein Auseinanderdriften von Modell und System nicht verhindert werden. Der Nutzen beim Einsatz des Werkzeugs ist gering, die Verwendung eine lästige Pflicht. Nur wenn Modellierung Spaß macht und dem Entwickler hilft, Routineaufgaben schneller zu erledigen, hat das Werkzeug langfristig eine Chance.

Was benötigt wird, ist ein Modell, welches sich ähnlich wie ein Analyse-Modell auf einer hohen Abstraktionsebene befindet, aber gleichzeitig als Quelle für die Code-Generierung dient. Um auf dieses Abstraktionsniveau zu gelangen, muss

das Modell von gewissen technischen Details wie der Programmiersprache und der Architektur abstrahieren.

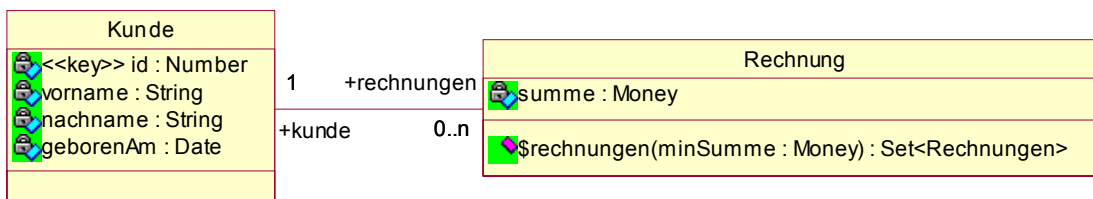
2.3 Die Bedeutung der Architektur

Jedes Software-System, egal ob unter UNIX oder Windows, mit C++, Java oder Fortran programmiert, hat eine **Architektur**. Diese Architektur besteht aus technischen Teilen, z. B. einem Framework für die Benutzeroberfläche und Konventionen, wie dieses Framework zu verwenden ist. Außerdem gibt es Regeln, die beschreiben, wie der Anwendungscode zu strukturieren ist. Wenn zwei Systeme beide eine Oracle-Datenbank und Java Server Pages verwenden, können sie dennoch eine vollständig andere Architektur haben.

Sie können bei einem UML-Generator „von der Stange“ natürlich nicht erwarten, dass dieser Ihre Architektur kennt. Ihnen bleibt nichts anderes übrig, als die Architektur bei Ihrer Modellierung zu berücksichtigen und/oder den generierten Code zu verändern. Dies bedeutet, dass Sie technische Details, wie z. B. die verwendete Programmiersprache, in Ihrem UML-Diagramm sichtbar machen müssen. Bei einer Codeänderung benötigen Sie eine „Round Trip“-Fähigkeit in Ihrem Werkzeug, sonst haben der Code und das Modell bald nicht mehr viel miteinander zu tun.

2.4 UML-Modelle ohne technische Details

Um die Probleme bei der Verwendung der UML zu lösen, schlägt die MDA vor, ein „Platform Independent Model“ (PIM) zu erstellen. Ein PIM ist ein UML-Modell, in dem technische Details weggelassen werden. Dieses Modell befindet sich auf einer höheren Abstraktionsebene und ist unabhängig von technischen Dingen wie der verwendeten Programmiersprache. Das PIM muss aber genug Details besitzen, um durch definierte Abbildungsregeln auf die Ziel-Programmiersprache unter Einhaltung der Architektur-Regeln abgebildet werden zu können. Es ist daher kein klassisches Analyse-Modell.



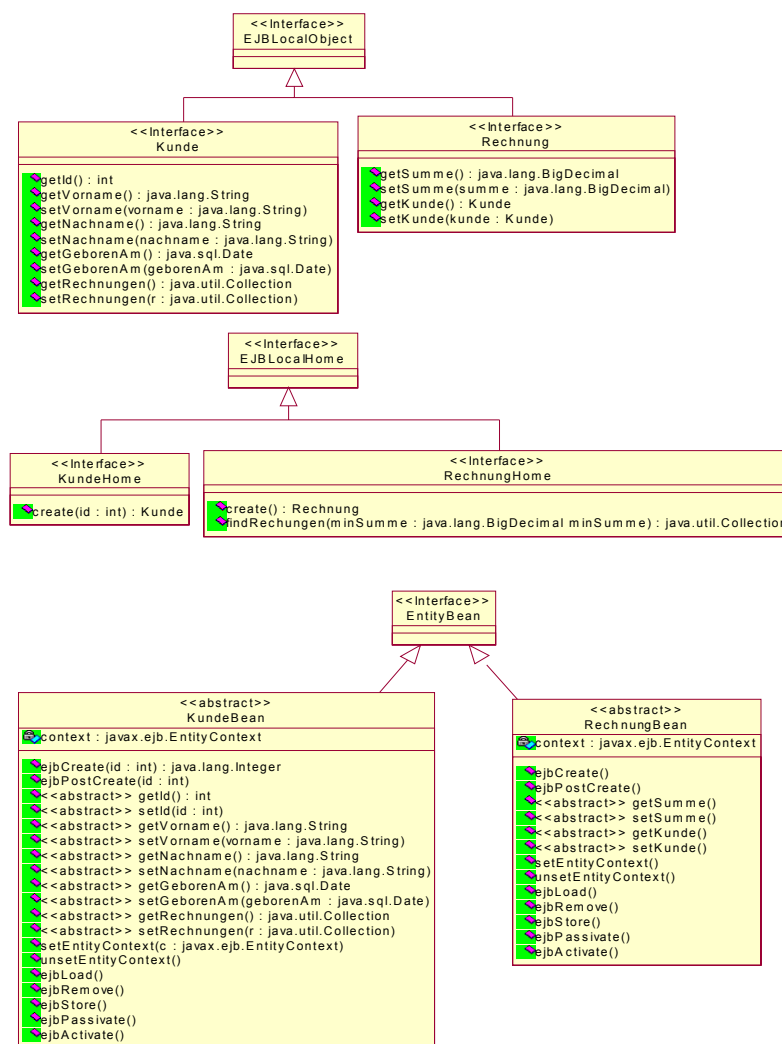
Oben sehen Sie ein PIM-Klassendiagramm, in dem Kunden mit ihren Rechnungen modelliert wurden. Ein Kunde hat einige Detailinformationen wie seinen Nachnamen und wird über eine Kundennummer eindeutig identifiziert. Zu einem Kunden gehören Rechnungen.

Wichtig in einem PIM ist das abstrakte Typ-System. „id : Number“ z. B. bedeutet, dass der Kunde eine Eigenschaft „id“ hat. Jeder mögliche Wert einer „id“ ist eine Zahl. Über den Stereotypen <<key>> wird definiert, dass eine ID einen Kunden eindeutig identifiziert.

2.5 UML-Modelle mit technischen Details

In einem zweiten Schritt wird das PIM durch einen MDA-Generator auf ein plattformspezifisches Modell abgebildet. Die MDA bezeichnet dieses Modell als PSM („platform specific model“). Mit der Unterscheidung zwischen PIM und PSM definiert die MDA einen Prozess für die Entwicklung von Software-Systemen: Modelliere das PIM, bilde es auf ein PSM ab, codiere das PSM. Die Werkzeuge von LIANTIS helfen dabei, sowohl die Abbildung auf das PSM als auch die Codierung zu automatisieren. Erst dadurch kann das Potenzial der MDA ausgenutzt werden.

Im Folgenden werden beispielhaft verschiedene Abbildungen auf denkbare Zielumgebungen dargestellt. Damit wird verdeutlicht, wie unterschiedlich die plattformspezifischen Modelle sein können. Man erkennt auch, wie aufwändig eine manuelle Erstellung des plattformspezifischen Modells sein würde.



Im ersten Beispiel wird dieses PIM auf Java-Klassen und Interfaces abgebildet, die den Regeln der „Enterprise Java Beans“(EJB™)-Architektur genügen. Um diesen Schritt zu verdeutlichen, ist oben das Resultat der Transformation als Java-UML-Klassendiagramm dargestellt. Dieses Modell ist ein PSM (platform specific model), da es alle Informationen besitzt, um 1 zu 1 auf Code abgebildet zu werden.

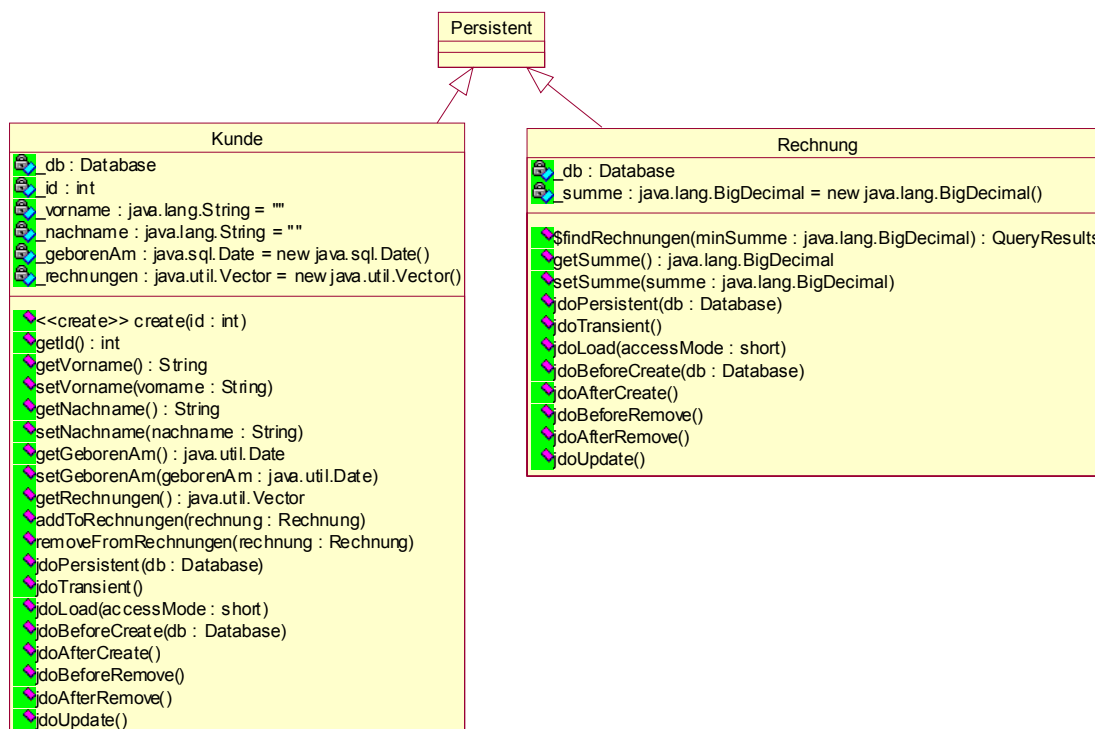
Dieses UML-Diagramm müssten Sie erstellen, um mit einem normalen Java-Generator Ihre Java Beans zu erzeugen! Wenn Sie lieber das erste Modell erstellen, verstehen und warten wollen, kommen Sie um eine **architektur-spezifische** Generierung nicht herum.

Ihr UML-Generator muss also nicht nur die Zielsprache, sondern auch die **Ziel-Architektur** kennen.

Die Hauptunterschiede zwischen beiden Modellen:

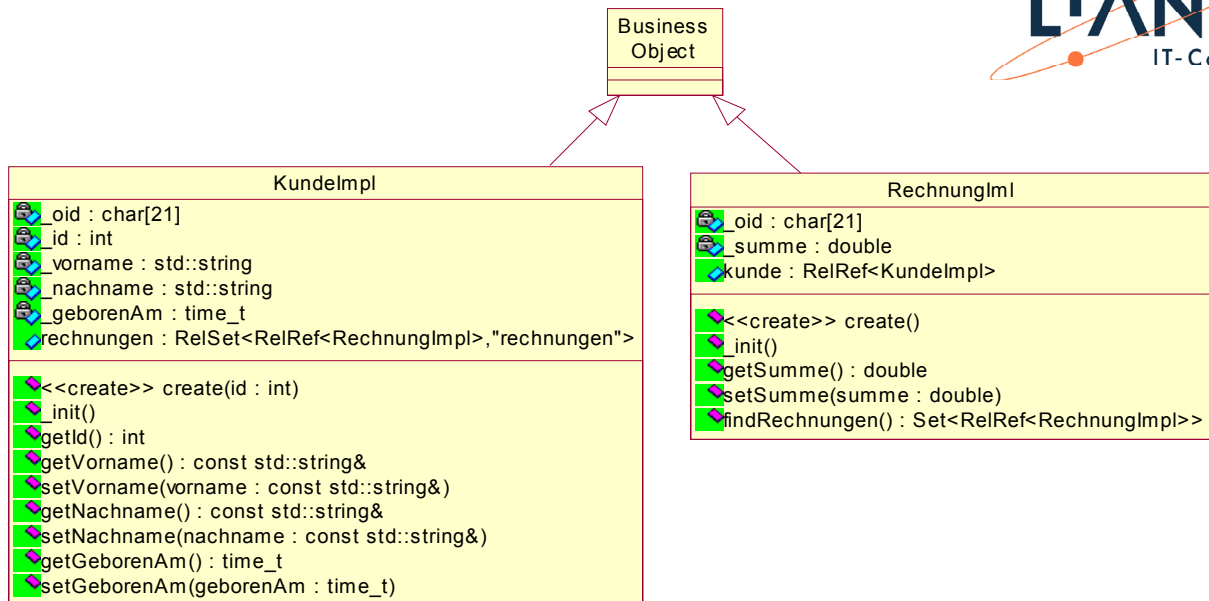
- Im PIM arbeiten Sie mit dem Konzept „Kunde“, während Sie im EJB-Modell EJB-Klassen und Schnittstellen modellieren müssen.
- Im PIM arbeiten Sie mit Attributen, im EJB-Modell mit abstrakten Zugriffsoperationen.
- Beziehungen sind im EJB-Modell nicht sichtbar, sondern müssen als Zugriffsoperationen modelliert werden.
- Im EJB-Modell sind Java-Typen sichtbar (z. B. `java.util.Collection`), das PIM arbeitet mit einem abstrakten Typ-System (z. B. `Set<Rechnung>`, d. h. eine Menge von Rechnungen).

Sollten Sie in Zukunft statt EJB den konkurrierenden Standard „Java Data Objects“ (JDO) bevorzugen, können Sie Ihr PIM wie folgt auf Java abbilden:



Dieses PSM verwendet die gleiche Programmiersprache Java, hat aber eine vollkommen andere Struktur. Dies macht deutlich, dass die Ziel-Architektur einen wesentlichen Einfluss auf den Generierungsprozess haben muss.

Oder Sie verwenden, wie hier im abschließenden Beispiel, C++ mit einer objektorientierten Datenbank gemäß dem ODMG-Standard:



Analog könnte das Model auf Smalltalk, Object COBOL, C# oder eine beliebige andere objektorientierte Sprache abgebildet werden. Eine Abbildung auf prozedurale Sprachen wie C oder COBOL ist etwas aufwändiger, aber dennoch leicht möglich.

Mit diesen Beispielen ist der Vorteil des MDA-Ansatzes zu erkennen:

- Das PIM befindet sich auf einem höheren Abstraktionsniveau, d. h., es ist kompakter und leichter zu verstehen.
- Das PIM ist nicht an eine bestimmte Programmiersprache gebunden.
- Das PIM ist nicht an eine bestimmte Architektur gebunden.
- Abgeleitete, triviale Operationen zum Verwalten von Objekten und deren Attributwerten sind im PIM nicht sichtbar.

Damit ist es für ein Projekt sogar möglich, die verwendete Programmiersprache zu wechseln. Dies ist zwar mit einem hohen Aufwand verbunden, da der individuell erstellte Code umgeschrieben werden muss. Es liegt aber im Bereich des Möglichen.

Wenn Sie heutzutage einen UML-Codegenerator verwenden, sind Sie an das Werkzeug, an die Programmiersprache und andere technische Infrastrukturen wie z. B. die verwendete Datenbank gebunden. Die MDA-Generatoren der LIANTIS GmbH sind eine Befreiung von diesen Fesseln.

Die MDA-Generatoren der LIANTIS sind mit jedem UML-Werkzeug einsetzbar. Durch ein flexibles Baukasten-System kann schnell und einfach jede beliebige Programmiersprache für jede Architektur generiert werden, auch für Ihre Zielumgebung.

2.6 Das MDA-Toolkit der LIANTIS GmbH

LIANTIS verwendet für die Erstellung eines Generators ein spezielles Toolkit. Dieses besteht aus wiederverwendbaren Komponenten und speziellen Implementierungstechniken, die über mehrere Jahre erfolgreich in sehr großen bis kleinen Projekten eingesetzt wurden.

Die LIANTIS erstellt Generatoren, die folgende Merkmale aufweisen:

- Das Modell ist ein PIM. Es ist abstrakter als der generierte Code. Dadurch wird Ihr Modell einfacher zu warten, zu verstehen und anzupassen.
- Das Modell und der Code passen immer zusammen – über den gesamten Lebenszyklus Ihres Software-Produkts.
- Der Generator erzeugt Ihre gewünschte Programmiersprache und hält sich an Ihre Programmier-Regeln.
- Der Generator erstellt nicht nur Methodenrumpfe, sondern – sofern möglich – auch eine partielle oder vollständig Implementierung. Typische Kandidaten für eine automatische Generierung sind beispielsweise Methoden zum Setzen und Abfragen von Attributen und zum Verwalten von Beziehungen.
- Das Modell kann geändert und erneut generiert werden, ohne individuell erstellten Code zu verlieren.
- Generierte Dateien werden automatisch erstellt oder im Verzeichnisbaum verschoben. Wenn eine Datei nicht mehr benötigt wird, wird sie automatisch gelöscht.
- Dateien, die sich nicht geändert haben, werden vom Generator nicht neu geschrieben. Damit müssen sie in der IDE nicht neu kompiliert werden.
- Der Generator ist leicht in Ihr UML-Modellierungswerkzeug, Ihre IDE und Ihr Versionskontroll-System integrierbar.
- Der Generator kann neben dem eigentlichen Quellcode auch Hilfsdateien wie Makefiles, Projektdateien, EJB-Deployment-Deskriptoren und Dokumentation erzeugen.

2.7 MDA für existierende Software-Systeme

Auch für existierende Systeme ist die MDA einsetzbar. Dazu wird Ihr Quellcode durch einen einmaligen „Reverse Engineering“-Schritt in ein PIM verwandelt. Dieser Schritt ist normalerweise nicht vollständig zu automatisieren, das Modell muss von Hand überarbeitet werden. Anschließend wird Ihr System durch den Generator MDA-tauglich gemacht. Die LIANTIS unterstützt Sie bei der Durchführung des „Reverse Engineering“ für alle objekt-orientierten Programmiersprachen.