Einführung einer neutralen Datenzugriffsschicht als Basis für die Migration einer komplexen Anwendung

Robert Eppig, Werner Teppe
Amadeus Germany GmbH
Siemensstraße 1, 61352 Bad Homburg, Germany
E-Mail: reppig@amadeus.com, werner.teppe@gmx.de

Abstract: *Große Anwendungssysteme werden entworfen, entwickelt, getestet und gehen dann in Produktion. Sind sie erfolgreich, werden sie weiterentwickelt, erhalten neue Funktionen und werden von zusätzlichen Kundengruppen genutzt. Daher stellt sich immer wieder die Frage: Soll die Anwendung angepasst, also migriert werden? Oder ist eine Neuentwicklung auf Basis der neuen Gegebenheiten sinnvoller?*
Amadeus[1] stellte sich mehrfach diese Frage für die Hauptanwendung für Touristische Inhalte der Amadeus Germany GmbH „ASW" („Anwendungssoftware") — und entschied sich jeweils für eine Migration. In dem aktuellen Migrationsprojekt "Solaris2Linux" werden die Anwendungsfunktionen und die Datenbank von Solaris nach Linux migriert. Schon bald wurde bei Amadeus der damit verbundene Austausch der Hardwarearchitektur als eigentliche Herausforderung identifiziert. Verschiedene Lösungsansätze wurden erarbeitet und getestet. Letztendlich scheint der Austausch der Datenzugriffsschicht (Data Access Layer, DAL) das Problem hinreichend zu beseitigen.

1 Einleitung

Große Anwendungssysteme werden entworfen, entwickelt, getestet und gehen dann in Produktion. Über die Jahre hinweg ändern sich Hardware, Betriebssysteme, Softwareentwicklungsumgebungen, Testsysteme und nicht zuletzt die Menschen, die die Systeme weiterentwickeln, testen und einsetzen.[2, 5]
Daher stellt sich immer wieder die Frage: Soll die Anwendung angepasst, also migriert werden? Oder ist eine Neuentwicklung auf Basis der neuen Gegebenheiten sinnvoller?
Amadeus stellte sich mehrfach diese Frage für die Hauptanwendung für Touristische Inhalte der Amadeus Germany GmbH „ASW" — und entschied sich jeweils für eine Migration.[3, 6] In dem aktuellen Migrationsprojekt „Solaris2Linux" werden die Anwendungsfunktionen und die Datenbank von Solaris nach Linux migriert.
Dabei soll auch die bestehende Oracle SPARC Hardwarearchitektur durch die günstigere Intel-X86-Architektur ausgetauscht werden. Als eigentliche Problematik bei der Umstellung nach Linux hat sich dabei die Änderung der Byte-Reihenfolge von Big-Endian auf Little-Endian herausgestellt.

2 Problemstellung Byte-Reihenfolge

Eine Analyse der Quelltexte hat ergeben, dass eine erhebliche Zahl von Fundstellen für Code existiert, der sich auf Rechnern mit verschiedenen Hardwarearchitekturen mit unterschiedlicher Byte-Reihenfolge potenziell unterschiedlich verhält. Das betrifft vor allem den Speicherzugriff über Typumwandlungskonstrukte wie *unions* oder *reinterpret_cast* bei Ganzzahldatentypen und Bitfeldern.

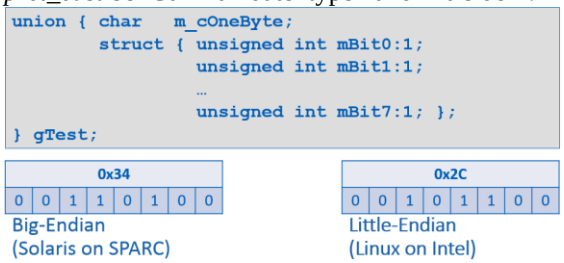

Abbildung 1: Unterschiedliches Verhalten einer *union* aus *char* und einem Bitfeld.

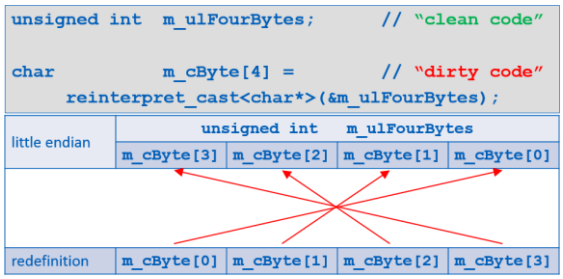

Abbildung 2: Unterschiedliches Verhalten des Typumwandlungskonstrukts *reinterpret_cast* auf eine *unsigned int* Speichervariable.

Ob das geänderte Speicherbild sich auf das Laufzeitverhalten auswirkt, ist nicht alleine durch statische Codeanalyse ermittelbar. Das ist erst mit Hilfe einer vollständigen Datenflussanalyse möglich, die neben dem Quelltext auch alle externen Datenquellen und Daten modifizierende maschinelle sowie manuelle Prozeduren beinhaltet. Der Aufwand für die Erhebung dieser Informationen wurde als zu groß eingeschätzt. Deshalb konnte die Gruppe der

potenziell betroffenen Fundstellen nicht wesentlich auf die wirklich betroffenen reduziert werden.

3 Erster Lösungsansatz: Einführung von Big-Endian-Datentypen

Als erstes wurde untersucht, ob es möglich ist, statt der in C++ eingebauten Datentypen für Ganzzahlen eigene Datentypen zu verwenden, die unabhängig von der Hardwarearchitektur immer „Big-Endian“-Speicherablage verwenden. Diese „BE-Datentypen“ können dann durch eine automatisierte Quelltext-umwandlung die eingebauten Datentypen ersetzen.

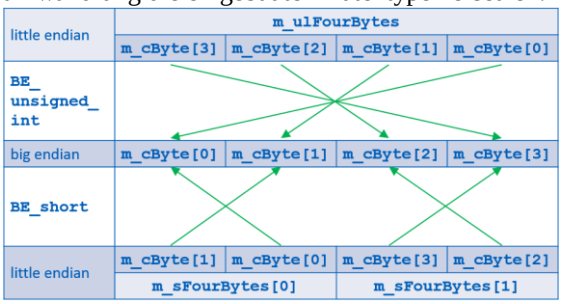

Abbildung 3: Einsatz „BE_*“ Datentype für eine *union* aus *unsigned int* und *short*.

Es konnte gezeigt werden, dass dieser Lösungsansatz realisierbar ist und die Problematik erfolgreich löst. Allerdings wurden dabei einige Nachteile festgestellt. Die wichtigsten sind:

- Aufrufe von System- und nicht im Quelltext vorliegenden Bibliotheken müssen manuell behandelt werden.
- Nicht alle Literale und Konstanten können automatisch erfasst werden.
- Die Initialisierungsreihenfolge und damit das Programmverhalten kann sich ändern.
- Die Wartbarkeit wird deutlich verschlechtert.
- Die Laufzeit wird deutlich verschlechtert.

Diese Nachteile haben dazu geführt, dass dieser Lösungsansatz verworfen wurde.[7]

4 Lösung durch Einführung einer Datenzugriffsschicht

Die überwiegende Zahl der potenziellen Problemstellen befindet sich in Bereichen, die mit persistierten Daten arbeiten. Im Quelltext ist dieser Zugriff bereits über eine einheitliche Schnittstelle, den „Q-Funktionen“, vereinheitlicht. An dieser Stelle wurde eine neue Zugriffsschicht, der „Data Access Layer (DAL)“, eingeführt.

Zuerst wurden die verwendeten Datenstrukturen einheitlich in einem XML-Format beschrieben. Als Quelle dienten die Strukturbeschreibungen aus dem Quelltext in *cpp*- und *hpp*-Dateien. Da diese Beschreibung nicht nur nicht normiert, sondern auch widersprüchlich war, musste dieser Schritt für die etwa 600 Satzstrukturen manuell durchgeführt werden. Daraus erzeugt danach ein Codegenerator die Zugriffsfunktionen, die alle Daten so liest und schreibt, dass sie unabhängig von der Hardwarearchitektur immer korrekte Werte liefert.

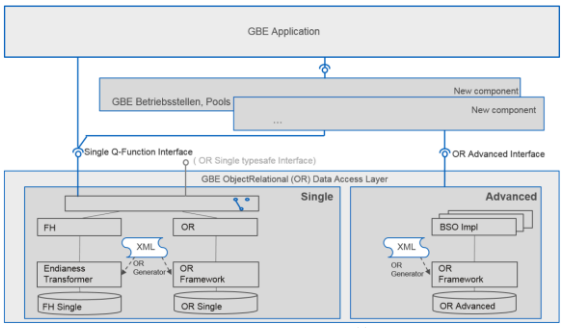

Abbildung 4: Architektur der Datenzugriffsschicht DAL

5 Aktueller Stand der Migration

Die Bereinigung der Datensatzbeschreibungen konnte erfolgreich durchgeführt werden und hat bereits eine Qualitätsverbesserung des Quelltextes zur Folge. Allerdings war dieser Aufwand größer als ursprünglich angenommen und höher als der für die Entwicklung des Generators für die DAL. Mit dessen Hilfe konnten bereits 85 % der Datentabellen und wesentliche Teile der Anwendung umgesetzt werden. Sie laufen schon unter Linux auf x86-Architektur. Dabei hat sich gezeigt, dass neben der Einführung der DAL nur in sehr wenigen Fällen manuelle Anpassungen nötig sind, um die Problematik der Byte-Reihenfolge zu lösen.

6 Ausblick

Die Applikation befindet sich in ständiger Weiterentwicklung mit etwa einer neuen Version pro Monat, die parallel für Solaris und Linux eingeführt wird.[8] Daneben stehen noch umfangreichere Tests aus.[3] Die Inbetriebnahme wird danach in einem Folgeprojekt durchgeführt und bis Ende 2018 abgeschlossen.

Literatur

[1] http://www.amadeus.com
[2] Werner Teppe: Redesign der START Amadeus Anwendungssoftware, Softwaretechnik-Trends 23(2) (2003)
[3] Werner Teppe: The ARNO Project: Challenges and Experiences in a Large-Scale Industrial Software Migration Project; Proceedings European, Conference on Software Maintenance and Reengineering (CSMR), pp. 149-158, 2009
[4] Werner Teppe: Teststrategien in komplexen Migrationsprojekten, Softwaretechnik-Trends 29 (2009)
[5] Werner Teppe: Migrationen — (K)eine Alternative für Langlebige Softwaresysteme?, Softwaretechnik-Trends 33(2) (2013)
[6] Uwe Erdmenger: SPL-Sprachkonvertierung im Rahmen einer BS2000 Migration, Softwaretechnik-Trends 26(2) (2006)
[7] Data Reengineering, Evolution and Migration to Prepare a Legacy Application Platform Migration, Softwaretechnik-Trends 35(2) (2015)
[8] Werner Teppe: Software Migrationsprojekte in der Zeit von "Cloud Computing" und "Agile Software Development", Softwaretechnik-Trends 34(2) (2016)