

Informationstechnik – Plattform- und programmiersprachenunabhängige Interoperabilität

Information technology –
Platform and programming language independent interoperability

Zur Erstellung einer DIN SPEC können verschiedene Verfahrensweisen herangezogen werden:
Das vorliegende Dokument wurde nach den Verfahrensregeln eines Fachberichts erstellt.

Gesamtumfang 16 Seiten

Inhalt

	Seite
Vorwort	3
1 Anwendungsbereich	3
2 Begriffe	3
3 Plattform- und programmiersprachenunabhängige Interoperabilität	5
4 Middleware basierte Ansätze zur plattform- und programmiersprachenunabhängigen Interoperabilität.....	7
4.1 Proprietäre Ansätze	7
4.1.1 Bridge-Technologie	7
4.1.2 Broker-Technologie	8
4.2 Standardisierter Ansatz	8
5 Transformation des Bytecodes um plattform- und programmiersprachenunabhängigen Interoperabilität zu erreichen	10
6 Diskussion und Bewertung der Ansätze	11
7 Zusammenfassung	12
8 Abkürzungen	13
Anhang A (informativ) Realisierte Implementierungen - Middleware für plattform- und programmiersprachenunabhängigen Interoperabilität.....	14
Literaturhinweise	16

Vorwort

Dieses Dokument wurde vom Arbeitsausschuss NA 043-01-22 „Programmiersprachen“ im Normenausschuss Informationstechnik und Anwendungen erarbeitet.

Einleitung

Über die Jahrzehnte wurde Software in den verschiedensten Programmiersprachen hinweg entwickelt. Dominierte in den 1950ern eine sehr an den Architekturen der seinerzeit populären Großrechner orientierte Programmierung, so setzten sich seit den 1960ern abstraktere Sprachen wie Fortran, Lisp und COBOL durch. Jede Generation hat ihre eigenen Programmiersprachen präferiert, und besonders in größeren Organisationen haben alle diese Sprachen ihre Spuren in Programmen und Programmbibliotheken hinterlassen.

Zudem ist bei der Softwareentwicklung in der letzten Zeit zunehmend der Trend festzustellen, sehr bewusst von den spezifischen Stärken von Programmiersprachen zu profitieren. Statt einer einzigen Sprache wählt man verstärkt eine dem Problem und der Umgebung angemessene Programmiersprache. In der Softwareindustrie muss man sich daher mit dem Zusammenspiel dieser Sprachen, der „Interoperabilität über die Grenzen von Programmiersprachen hinweg“ befassen.

Die oben aufgeführten Szenarien zeigen, dass Interoperabilität sowohl unter wirtschaftlichen als auch technischen Gesichtspunkten ein wichtiges Thema in der Softwareindustrie ist und auch in absehbarer Zukunft bleiben wird.

1 Anwendungsbereich

In dieser DIN SPEC (Fachbericht) soll präzisiert werden, was unter Interoperabilität bei Programmiersprachen zu verstehen ist und wie sie sich gegenüber Portabilität abgrenzt. Es sollen verschiedene genormte und nicht-genormte Ansätze vorgestellt werden, mit denen man heute konkret Interoperabilität über Programmiersprachengrenzen erreichen kann, und ihre Vor- und Nachteile diskutiert werden. Es erfolgt hier bewusst eine Eingrenzung auf Ansätze, die nicht auf Web-Service Technologien beruhen, da diese für viele Situationen, in der es um die Wiederverwendung von Komponenten geht, aus Performance-Gründen nicht geeignet sind. Zur Veranschaulichung werden Beispiele herangezogen, die sich auf konkrete kommerzielle und nicht kommerzielle Softwareprodukte beziehen.

2 Begriffe

Für die Anwendung dieses Dokuments gelten die folgenden Begriffe.

2.1 Bytecode

Befehlssatz einer virtuellen Maschine

ANMERKUNG Programme, die in einer höheren Programmiersprache entwickelt wurden, werden durch einen Compiler in Bytecode übersetzt, der dann von der virtuellen ausgeführt wird. Bytecode wird auch als Zwischencode bezeichnet. Ein standardisierter Bytecode bzw. Zwischencode wird in ISO/IEC DIS 23271 spezifiziert.

2.2 Applikationsplattform

virtuelle Maschine und Compiler für eine oder mehrere höhere Programmiersprachen, die Code für die virtuelle Maschine erzeugen

ANMERKUNG Des weiteren bietet eine Applikationsplattform umfangreiche Klassenbibliotheken und Frameworks für die Entwicklung von grafischen Benutzeroberflächen, Datenbankanwendungen und verteilten, mehrschichtigen Applikationen, sowie Werkzeuge, welche den Entwurf und das Testen unterstützen.

2.3 Bridge

Middleware, die es gestattet Komponenten, die für auf unterschiedlichen Applikationsplattformen entwickelt wurden so zu integrieren, dass Methodenaufrufe über Plattformgrenzen hinweg möglich sind, ohne dass dabei eine Netzwerkkommunikation auftritt

2.4 Broker

agiert als Vermittler zwischen Client und Server und ist damit für die Koordination der Kommunikation zwischen Komponenten zuständig

ANMERKUNG Ein Broker vermittelt den Aufruf einer Methode an den Server, der diese Methode zur Verfügung stellt und leitet, das Ergebnis an den aufrufenden Client zurück.

2.5 Interoperabilität

Programme oder Komponenten, die in unterschiedlichen Programmiersprachen implementiert wurden und eventuell auf verschiedenen Betriebssystemen ablaufen, können gegenseitig alle Funktionalitäten nutzen können, die sie jeweils zur Verfügung stellen

ANMERKUNG Interoperabilität ist in Zusammenhang mit diesem Dokument also vor allem im Kontext der Wiederverwendung von Softwarekomponenten oder kompletten Applikationen zu sehen.

2.6 Komponente

entweder eine komplette Applikation / Programm oder eine Softwarebibliothek/Klassenbibliothek, welche für eine Anwendungsdomäne Funktionen/Methoden bereitstellt

2.7 native Komponente

Komponente, die direkt in den Maschinencode einer dedizierten CPU übersetzt wurde

2.8 Middleware

Software, die Applikationen oder Softwarekomponenten über Betriebssystem- und Programmiersprachengrenzen verbindet

ANMERKUNG Middleware ermöglicht dem Softwareentwickler mit dem Programmiermodell der Applikationsplattform zu arbeiten, auch wenn er Komponenten auf einer anderen Applikationsplattform nutzt.

2.9 Portabilität

Eigenschaft des Quellcodes eines Programms, das ohne Änderungen für unterschiedlichen Betriebssysteme bzw. Laufzeitumgebungen übersetzt, gebunden und ausgeführt werden kann

ANMERKUNG Die Begriffe Interoperabilität und Portabilität werden häufig im gleichen Kontext benutzt. Es ist daher wichtig, sie gegeneinander abzugrenzen.

2.9 Virtuelle Maschine VM

Programm, welches eine Menge von genau definierten Instruktionen akzeptiert und diese unabhängig von irgendeiner Programmiersprache ausführt

ANMERKUNG Die Menge von Instruktionen wird auch als Zwischensprache bezeichnet. Eine Instruktion wird deterministisch ausgeführt, basierend auf dem aktuellen Zustand der VM. Eine VM abstrahiert vom Betriebssystem und der Hardware. Ein Compiler übersetzt ein Programm, welches in einer Hochsprache geschrieben wurde, in die Zwischensprache der VM. Die Zwischensprache wird dann entweder interpretiert oder von einem „Just in Time Compiler“ in Maschinensprache übersetzt und dann ausgeführt.

3 Plattform- und programmiersprachenunabhängige Interoperabilität

Von Programmiersprachen-Interoperabilität spricht man, wenn eine Applikation, die in Programmiersprache A entwickelt wurde, eine Komponente nutzt, die in Programmiersprache B geschrieben wurde, beide aber auf dem gleichen Betriebssystem laufen. Beispiele für diese Art der Interoperabilität sind z. B. die Nutzung von Numerikbibliotheken, die in C++ geschrieben wurden, in Java-Programmen. Dies ist speziell in Java über das sog. Java Native Interface (JNI) möglich. JNI ist ein Mechanismus, der es erlaubt, die Java-eigene Laufzeitumgebung, die Java Virtual Machine (JVM), um C++ Klassen oder um C Funktionen zu erweitern, um diese dann in Java-Programmen nutzen zu können. Ein analoger Mechanismus existiert für die Entsprechung der JVM in der .NET-Welt, der .NET Common Language Runtime (CLR), mit dem es möglich ist, native Programmbibliotheken in .NET Programme zu integrieren. Schematisch ist dies in Bild 1 dargestellt. Im Unterschied zur nativen Komponente bietet eine virtuelle Maschine (VM) Programmen eine über Maschinengrenzen hinweg gleiche Ausführungsumgebung mit einem eigenen Zwischencode, der erst durch die VM in echten Maschinencode übersetzt wird (s. u.).

Genauer ist hier ist der Fall dargestellt, dass eine Applikation, die auf in einer VM abläuft, eine native Komponente nutzt. Ein weiteres Beispiel ist die Nutzung nativer C++- / C- oder FORTRAN-Bibliotheken in Python-Programmen. Der umgekehrte Fall, dass eine native Applikation eine Komponente nutzt, die auf einer VM läuft, kommt in der Praxis ebenfalls vor.

Die in Bild 1 dargestellte Interoperabilität ist für den Applikationsentwickler im Allgemeinen relativ einfach realisierbar und benötigt auch keine zusätzliche Middleware, wenn man von dem Code absieht, der für die Integration der Komponenten notwendig ist. Dieser Code lässt sich aber üblicherweise automatisch über sog. Reflexionsmechanismen erzeugen und ist damit für den Softwareentwickler transparent.

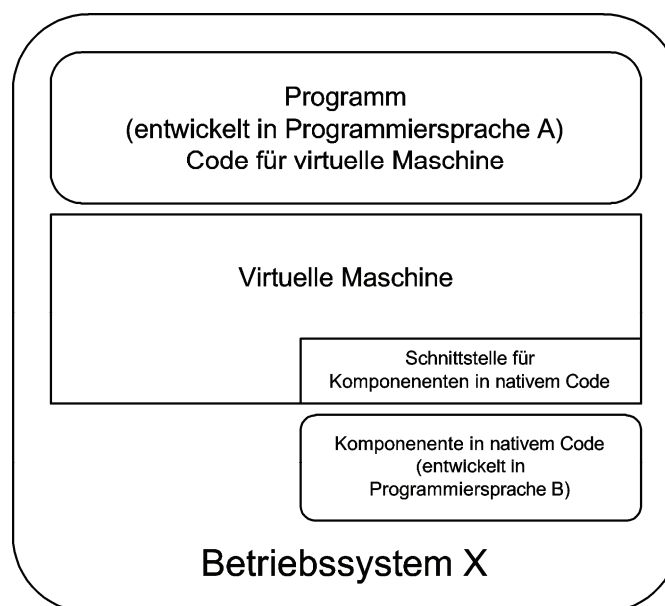


Bild 1 — Sprach-Interoperabilität (1)