

# Praktikumsbericht: Anbindung von Soap und Corba an LEAF

## ELCA

Silvan Saxer

ELCA Informatik AG, Schweiz, 2002.

Imput.	Rapport	Version	Date	Auteur(s)	Etat	Visa
6200-646	5900/ISSX	1.0	20.03.02	SSX	Valid	SSX

**I. Inhaltsverzeichnis**

<b>I.</b>	<b>Inhaltsverzeichnis.....</b>	<b>2</b>
<b>1</b>	<b>Organisatorisches .....</b>	<b>3</b>
1.1	Dauer.....	3
1.1.1	Teilzeitanstellung .....	3
1.1.2	Vollzeitanstellung (als ETH Praktikum anrechenbar).....	3
<b>2</b>	<b>Einleitung / Aufgabenstellung.....</b>	<b>4</b>
<b>3</b>	<b>Tätigkeit.....</b>	<b>5</b>
3.1	SOAP Integration.....	5
3.1.1	WSDL .....	5
3.1.2	Shared Context.....	5
3.1.3	Performance Tests.....	6
3.2	CORBA Integration .....	6
3.2.1	Java IDL .....	6
3.2.2	RMI / IIOP .....	6



## **1 Organisatorisches**

Praktikant: Silvan Saxer  
Firma: ELCA  
Betreuer: Philipp Oser

### **1.1 Dauer**

#### **1.1.1 Teilzeitanstellung**

Beginn: 1. August 2001  
Ende: 14. September 2001

#### **1.1.2 Vollzeitanzstellung (als ETH Praktikum anrechenbar)**

Beginn: 22. Oktober 2001  
Ende: 28. Februar 2002

## **2 Einleitung / Aufgabenstellung**

ELCA entwickelt ein Framework, basierend auf Suns Java 2 Enterprise Edition. Ein Teil dieses Frameworks, das LEAF (Lean Extensible Architecture Framework) genannt wird, ist der Service Abstraction Layer, der beliebige Service Infrastrukturen hinter einer dünnen Schicht zu verstecken weiss. Damit ist das Programmierparadigma für alle Service Typen gleich.

Im Rahmen meines Praktikums sollte die Anbindung von SOAP (Simple Object Access Protocol) an LEAF verbessert und ausgebaut werden. Zudem sollte dem Benutzer als weitere Service Infrastruktur CORBA angeboten werden.

Anschliessend sollte ich ein Refactoring eines Teils des Service Abstraction Layers durchzuführen, um die Integration weiterer Infrastrukturen und Service Typen (Message Driven Bean, Entity Bean, etc.) und die Unterstützung verschiedener EJB – Containern eleganter zu unterstützen.

### 3 Tätigkeit

#### 3.1 SOAP Integration

Die Integration von SOAP als Service Typ wurde bereits von einem anderen Praktikanten erledigt. Meine erste Aufgabe bestand nun darin, mich ich das Framework LEAF einzuarbeiten und die Beispiele, die mein Vorgänger entwickelt hatte, zum Laufen zu bringen und anschliessend besser in LEAF zu integrieren. Nach einigen Anläufen, gelang dies auch, die Beispiele liefen stabil und reproduzierbar, bequem zu starten via „ant“ Kommando.

##### 3.1.1 WSDL

Nun ging ich daran, die proprietäre Service Beschreibung, die bis anhin verwendet wurde mit WSDL (Web Services Description Language), dem Standard für die Beschreibung von Web Services, zu ersetzen. Als Referenz galt die W3C Spezifikation, deren Unklarheiten sich durch das Durchforsten einschlägiger Newsgroups aus der Welt schaffen liessen. Ich entschied mich, das Open Source Produkt wsdl4j von IBM zu verwenden, das ein einigermaßen bequemes Lesen und Erstellen von WSDL Dokumenten erlaubte.

Nachdem die Verwendung von WSDL eingebaut war, galt es, die alten Beispiele anzupassen. Zudem sollte sich die Implementierung anhand eines praktischen Beispiels bewähren. Ein einfacher Stockquote Service, der irgendwo im Internet lief, war in kurzer Zeit eingebunden. Nachdem ich noch einige Hilfsprogramme entwickelt hatte, die das Publizieren von Web Services und das Verwenden eben solcher noch weiter vereinfacht, war dieser Teil der Arbeit abgeschlossen.

##### 3.1.2 Shared Context

Ein weiteres Feature sollte eingebaut werden. LEAF unterstützt so genannten „shared context“. Das sind zusätzliche Name-Wert Paare, die bei Aufrufen mitgesendet werden können, jedoch nicht explizit in der Signatur einer Methode erscheinen. Damit soll erreicht werden, dass Eigenschaften wie Sicherheit ein und ausgeschaltet werden können, ohne die Signaturen der Methoden verändern zu müssen. Ein ähnliches Konzept kennt auch CORBA.

Die Aufgabe bestand nun darin, „shared context“ auch in SOAP zur Verfügung zu haben. Ein Grund dazu ist, dass momentan ein ähnliches Framework wie LEAF auf Basis von .NET geplant ist. Es wäre nun natürlich wünschenswert, wenn diese beiden Frameworks miteinander in SOAP sprechen könnten und der „shared context“ über diese Verbindung nicht verloren ginge.

Ein aufmerksames Lesen der SOAP Spezifikation zeigte dann, dass sich der Header der SOAP Message dafür eignen würde. Da die Spezifikation jedoch noch nicht völlig ausgereift ist, entschieden wir uns für einen generelleren Ansatz. Die Verwendung des Headers sollte nur eine mögliche Implementierung sein. Ich entwickelte daher einen Callback-Mechanismus, dessen Handler einfach ausgewechselt werden können. Jede Implementierung dieses Handlers hat sowohl den Request von LEAF als auch den generierten Request an SOAP zur Verfügung. So ist eine beliebige Konvertierung des „shared context“ möglich.

Als „out of the box“ Lösung entwickelte ich drei Implementierungen. Nach einigen Tests mit den bereits vorhandenen Beispielen wurde auch diese Entwicklungsphase abgeschlossen.

### 3.1.3 Performance Tests

Im Anschluss wurden einige Performance Analysen gemacht. Leider wurde der grösste Teil der Zeit innerhalb der Klassen des verwendeten Apache SOAP verbraucht, so dass präzise Tests etwas umständlich geworden wären. Ein Profiling mit hprof zeigte jedoch keine einfach zu bewältigen Performance Probleme in Apache SOAP.

Wir vermuteten, dass das Problem bei dem schlechten Scheduling von Windows liegen könnte. Und siehe da: als ich die Tests auf Linux laufen liess, liefen sie doppelt so schnell!

## 3.2 CORBA Integration

### 3.2.1 Java IDL

Meine nächste grössere Aufgabe war die Integration von CORBA. Ich verwendete Orbix 2000 bzw. Orbix E2A als ORB und versuchte mit einigen Beispielen von Java auf diesen ORB zuzugreifen. Allerdings ziemlich ergebnislos. Die Nachforschung in diversen CORBA Newsgroups erhärtete die Vermutung, dass der in JDK 1.3.1 mitgelieferte ORB mehr Bugs als funktionierende Zeilen Code hat. Eine Anfrage bei IONAs Support brachte eine ähnliche Antwort. Allerdings erklärten sie, dass die Interoperabilität mit JDK 1.4 wesentlich besser sei. So spielte ich meine Beispiele mit der aktuellen Betaversion des JDK 1.4 durch. Falls alles klappte, einzig void Methoden und Methoden ohne Eingabeparameter funktionierten nicht. Dieser Fehler war dann wiederum in einer Newsgroup beschrieben.

Wir entschieden uns, die Integration trotzdem vorzunehmen, obwohl JDK 1.4 noch nicht fertig ist. Schliesslich kann man auch Orbix auf der LEAF Seite verwenden. So implementierte ich die CORBA Anbindung so, dass sie mit verschiedensten ORBs betrieben werden kann. Um den ORB zu wechseln muss man nun bloss einige Konfigurationsinformationen anpassen. Ich entwickelte auch hier einige Beispiele um die Verwendung zu demonstrieren.

### 3.2.2 RMI / IIOP

Selbstverständlich wäre es auch wünschenswert, von CORBA aus, LEAF Services ansprechen zu können. Dazu entwickelten IBM und Sun ein Protokoll, das sie RMI over IIOP nannten. Dabei werden alle RMI (Remote Method Invokation) Aufrufe über das IIOP Protokoll von CORBA betrieben. Es sollte daher ein Kinderspiel sein, von einem CORBA Client auf einen EJB-Container, der RMI/IIOP verwendet, zuzugreifen. Sollte, denn die Realität sah etwas anders aus. Der verwendete WebLogic Server generierte eine IDL (Interface Definition Language) Datei, die Orbix 2000 partout nicht richtig interpretieren konnte. Allerdings ist eine CORBA Interaktion ohne IDL nicht sinnvoll machbar. Bea erklärte, dass Orbix 2000 Bugs in ihrem IDL Compiler habe, IONA erklärte, dass wohl WebLogic Bugs in seinem IDL Generator habe. Alles klar: niemand wollte „schuld“ sein. Fazit: RMI/IIOP funktioniert nicht. Wir verzichteten daher auf weitere Untersuchungen zu diesem Thema.