Info Bridge

Herausgeber: Object Engineering GmbH

Juli 2000 · Nr. 2

Fünf Jahre Object Engineering!

TOP

Fünf Jahre!

Im Frühjahr 1995 wurde die Object Engineering gegründet. Fünf Jahre scheinen eine kurze Zeit, wenn man aber beachtet, was alles in dieser Zeit passiert ist - innerhalb und ausserhalb des Unternehmens -, kommt man nicht umhin, einfach zu staunen.

Wäre die Object Engineering ein Menschenkind (siehe Bild), so wäre sie mit ihren 5 Jahren am Anfang ihres "Lebens" und stark auf Hilfe von aussen angewiesen.

Wie aber sieht dies mit einem Unternehmen im IT-Bereich aus? In diesem Umfeld kommen alle 5 Jahre zwei bis drei neue Technologietrends zum Durchbruch. In den letzten Jahren waren dies Java, XML, CORBA und andere. In diesem Sinn gleicht eine solche Firma nicht einem gut behüteten Kind, sondern vielmehr einem Jungtier, das in einer Herde mitten in der afrikanischen Steppe geboren wurde. Kaum auf der Welt muss es mit den anderen Schritt halten, will es die Risiken des Wetters, der Jäger und der Wildkatzen überleben.

Die *Object Engineering GmbH* ist aus der *Koch System Engineering* entstanden und hat von dieser Firma das Projektgeschäft übernommen. Bereits in den ersten Projekten hat sie sich auf verteilte System-Architekturen unter der Verwendung des defacto-Standards COBRA® und OMA® konzentriert. Durch die erfolgreiche Migration von Altsystemen, welche die ersten Tätigkeiten bestimmten, hat sich die Object Engineering schnell einen guten Namen gemacht.

Damals mussten Series/1-Computer, die bei der PTT Telecom (heute Swisscom AG) im Einsatz standen, abgelöst werden und die bestehenden Software-Komponenten auf neuere Plattformen und neue Technologien migriert werden. Möglich wurde eine Migration dieses komplexen, auf mehreren Plattformen verteilten Systems, durch den

konsequenten Einsatz von CORBA® (Common Object Broker Architecture). Die erste grosse Software-Migration wurde 1997 nach nur 14-monatiger Entwicklungszeit in gutem Teamwork mit dem Kunden abgeschlossen. Der Gewinn des "OMG Object Application Award 1997", den die Swisscom AG für dieses Projekt erhielt, war ein krönender Abschluss der ersten Phase.

Bei der zweiten Phase ging es darum, die gekapselte (wrapping) Midtier-Komponente unter Beibehaltung der CORBA-basierten Schnittstelle zu erweitern und abzulösen. Diese Phase wurde im Herbst 1999 dem Betrieb übergeben. Dabei wurde die Kommunikation nicht nur von der Benutzerstation (PC) zum Midtier-Server, sondern bis hinauf zum Mainframe auf der Basis einer CORBA-Middleware bewerkstelligt. Bemerkenswert bei dieser Arbeit war, dass die Antwortzeiten für den Endbenutzer um 2-3 Sekunden gesenkt und die Wartbarkeit wesentlich erhöht werden konnte. Zwei schlecht wartbare Altkomponenten (SNA-Strecke und in Assembler geschriebene Software) konnten völlig ersetzt werden.

Neben diesem für uns im Mittelpunkt stehenden Projekt haben wir zusätzlich weitere Applikations-Wrapper und generell einsetzbare Komponenten auf der Basis von CORBA-Middleware auf der Serverseite implementiert.

Kürzlich kam ein Projekt der ESA (European Space Agency) dazu, welches für ein schweizerisches Weltraum-Solarexperiment entwickelt wird.

schweizerisches Weltraum-Solarexperiment entwickelt wird

Im E-Commerce Bereich haben wir im letzten Jahr mitgeholfen ein Projekt der Swisscom AG Mobile zu realisieren. Mit NatelGo® wird die Bestellung von Tageskarten im Skigebiet Laax-Valera via Natel SMS möglich. Dabei wurden von uns mehrere CORBA-basierte Server in C++ und Java entwickelt, welche über IIOP (Internet Inter Object Broker Protocol) über das Internet kommunizieren.

Alle unsere Entwicklungen werden mit CASE-Tools (CASE = Computer Aided Software Engineering) entworfen, dokumentiert und auch weiterentwickelt. Die objektorientierte Technologie stand seit Beginn als Basis fest, wobei wir uns nicht auf die Methodik versteifen, sondern diese so einsetzen, dass Resultate erzeugt werden können. Als Werkzeug stehen bei uns C++, Java, CORBA-Middleware und natürlich UNIX (und damit auch Linux) als Plattform im Zentrum.

Wie sieht die Zukunft der Object Engineering GmbH aus?

Die Zukunft sehen wir in der Weiterführung kundenspezifisch entwickelter Lösungen für verteilte Systeme und Anwendungen im Bereich "Engineering". Die Beratung für Architektur und Design von solchen Systemen bildet weiterhin den Einstieg für unsere Kunden in solche Projekte. In unserem Bereich "Standard-Komponenten" werden Applikationskomponenten und Werkzeuge für verteilte Systeme (z.b. Diagnostik Tool in der letzten Info Bridge Nr. 1 vorgestellt) auf der Basis von CORBA und EJB (Enterprise Java Beans) entwickelt und als Halbfabrikate in Projekten eingesetzt.

Jede Firma hat ihre eigene Dynamik, die sowohl von demjenigen, der sie leitet, wie auch der Gruppe tagtäglich von neuem erschaffen wird. Die Initiative jedes Einzelnen, das eingeölte Zusammenarbeiten und das Erreichen der Ziele - die Resultate - machen diese Dynamik aus. Und das ist bei der Object Engineering nicht anders. Nicht die Share Holder Value steht im Zentrum, sondern die Philosophie, dass Know-how weitergegeben werden muss, wenn im Software-Engineering eine gute Zusammenarbeit mit den Kunden und allen beteiligten Personen stattfinden soll. Dazu gehört es, die Nase vorn zu halten und zu wissen, in welche Richtung die Entwicklung der Informatik-Technologie geht. Sich stets weiterzubilden und an den Weiterentwicklungen interessiert zu sein, ist für uns von grösstem Stellenwert. In diesem Sinne bilden wir seit zwei Jahren unseren ersten Informatikerlehrling aus.

Die oben erwähnten Grundgedanken wollen wir beibehalten und in unseren Tätigkeiten und Projekten weiterverfolgen.

Alte Werte erhalten und mit neuen verbinden

Die Object Engineering wird auch weiterhin die "alten" Werte hochhalten, d.h. einerseits auf der technischen Basis erfolgreiche und erhaltenswerte Software erhalten (und nicht ersetzten) und sie durch Migration für die "Neuwelt" zugänglich und verfügbar machen. Dabei dürfen neue Technologien natürlich nicht ausser acht gelassen werden. CORBA, XML und die ganze JAVA-Ära bringen ganz neue Möglichkeiten mit defacto-Standards Lösungen zu erstellen, welche bisher proprietär gelöst wurden. Nicht zu verschweigen Linux, welches die nun bereits 30 Jahre "alte" UNIX-Domäne weiterführt und eine Plattformbasis bietet, welche in Sachen Robustheit und Verfügbarkeit den Plattformen von Rang und Namen eine Nasenlänge voraus ist. Auch hier handelt es sich um althergebrachte, einfache, aber systematische Konzepte und Technologie, die dem Software-Engineer und Enterprise-Architekten das Leben erleichtern können.

Ihre Meinung und Kritik interessiert uns und wir würden und über ein Echo auf unsere Info Bridge freuen. Wir wünschen Ihnen viel Spass beim Lesen.

Unternehmensweite und flexible Applikations-Integration mit CORBA-IDL und XML

TOP

Betrachtet man Applikationen mit von aussen zugreifbaren Schnittstellen innerhalb eines Unternehmens, dann kann man dies mit Gebäuden innerhalb einer Stadt vergleichen. Es braucht eine gute Koordination, damit bestehende Applikationen und neue Applikationen zusammenpassen und vor allem auch zukünftig gegenseitig mit wenig Aufwand integriert werden können. Dazu braucht es eine gute Applikations-Architektur, welche - koordiniert durch die Unternehmens-IT-Architektur - im Idealfall beliebige Integrations-Varianten zulässt. Integriert wird über Schnittstellen, welche idealerweise innerhalb eines Unternehmens durch eine zentrale Stelle gegenüber den Unternehmens-Vorgaben überprüft werden.

In den letzten Jahren wurden vielerorts verteilte Lösungen mit CORBA-Middleware (Common Object Broker Architecture) gebaut, die in vielen Unternehmen tagtäglich im intensiven Einsatz stehen. Diese Applikationen haben auf den CORBA-Vorgaben abgestützte Schnittstellen, welche in der CORBA-IDL (Interface Definition Language) definiert und dann in die verschiedenen Implementationssprachen der Zielsysteme übersetzt und implementiert werden. Der grosse Vorteil der CORBA basierten Produkte ist dabei die sprachenübergreifende, plattformübergreifende und für heterogene Systeme geeignete Funktionsweise, welche durch den von der OMG (Object Management Group) definierten Industriestandard normiert wurde.

Oft hat der Mangel an Erfahrung und an Ausbildung dazu geführt, dass dem Design von Schnittstellen zuwenig Aufmerksamkeit geschenkt wurde, resp. zuwenig daran

gedacht wurde, dass Anforderungen und Applikations-Informationsstrukturen während dem Lebenszyklus einer Anwendung ändern können.

Zu Beginn war mancher Entwickler dahin geneigt, die Schnittstellensprache (IDL) dazu zu verwenden, die nach aussen dargestellten Applikations-Strukturen und Funktionen möglichst 1 zu 1 damit abzubilden. Nach dem Ausbreiten und Betriebseinsatz einer mit einer applikations-orientierten Schnittstelle ausgerüsteten Applikation, kam dann die Ernüchterung mit der ersten Struktur- oder Funktions-Anpassung einer solchen Komponente. Einfach gesagt wurde es notwendig, z.B. durch das Wegfallen eines Informationsfeldes oder das Aufspalten eines solchen Feldes in mehrere Teilfelder, dass die Schnittstelle des Services (serverseitige Komponente) angepasst werden musste. Logischerweise muss dann die Client-Applikation ebenfalls angepasst, getestet und neu verteilt werden. Was aber, wenn mehr als eine Applikation die Schnittstelle des Services verwendet, von der unter Umständen die Entwickler nichts wissen? Nun, die Turbulenzen, welche dann bei der Einführung der neuen Version mit der geänderten Schnittstelle entstehen, deuten auf ein mangelhaftes Change-Management hin. Ist dieses intakt, dann könnte man vielleicht schon vorher feststellen, dass es fast nicht realisierbar ist, alle Applikations-Entwicklungs-Teams so zu koordinieren, dass alle Applikationen auf die neue Schnittstelle auf das festgelegte Ausbreitungs-Datum hin geändert und getestet werden können.

Aus unserer Erfahrung mit solchen Änderungsprozessen können folgende Grundregeln aufgestellt werden:

- Auf der Applikations-Service-Seite dürfen nach Möglichkeit nur Anpassungen gemacht werden, welche an bestehenden Schnittstellen nichts verändern.
- Die Verwendung einer Service-Schnittstelle soll für die Client-Seite so einfach wie möglich sein, auch wenn dafür die Server-Seite (Service) möglicherweise komplexer wird. In der Regel gibt es nämlich mehr Client-Applikationen, welche den gleichen Service benutzen, also eine n:1 Beziehung, was dann bedeutet "n mal einfach und 1 mal komplex".
- Schnittstellen sollten immer aufwärtskompatibel sein, damit sowohl neue, wie noch nicht umgestellte Client-Applikationen den Service benutzen können.
- Zu Beginn des Designs die Schnittstelle so auslegen, dass nur die effektiv anstehenden Anforderungen abgedeckt werden. Schnittstellen sollten so schmal wie möglich sein, denn erweitern kann man diese später immer. Unter Erweiterung ist hier das Zufügen von neuen Strukturen (nicht Felder in bestehenden Strukturen oder Parameter in bestehenden Operationen) oder von neuen Operationen gemeint.
- Wo Applikations-Informations-Strukturen regelmässig verändert werden, sollten diese nicht in IDL definiert, sondern auf generische Datenformen abgebildet werden.
- Wo keine oder sehr, sehr wenig Schnittstellen-Änderungen erwartet werden, kann und soll die Schnittstelle vollständig in IDL definiert werden.

Man sieht, dass sich ein Abwägen zwischen typendefinierten und aussagekräftigen IDL-Definitionen und interpretierbaren - und zwar auf Client- und Service-Seite - generischen Daten ergibt. Dies ist natürlich eine Entscheidung, welche vom Applikations-Architekten gemacht werden muss, und diese Entscheidung benötigt ein gutes Verständnis und Erfahrung.

Man kann nun beidseitig den extremen Weg aufzeigen wie:

1. alle relevanten internen Strukturen und Operationen einer Applikation in IDL darstellen und dadurch bei der kleinsten Änderung Schnittstellenanpassungen in Kauf nehmen;
2. in IDL nur eine Operation und eine Schnittstelle (Interface) zu definieren, welche alle Parameter als generischen Datenhaufen bekommt und die Resultate als solchen zurückliefert.

Im Fall a) hat man die "Befriedigung" das CORBA-Konzept möglichst "lückenlos" ausgenutzt zu haben; im anderen Fall hat man das "schlechte Gewissen" CORBA rein als Kommunikationsmedium (Middleware) eingesetzt zu haben.

Die Wahrheit und gangbare Lösung liegt wohl irgendwo dazwischen.

Wenn man alleine auf CORBA-IDL abstützend eine generische Lösung machen will, dann könnte man dies mit dem Datentyp *any* über sogenannte NameValue-Strukturen lösen:

```
struct NameValue
{
   string fKey;
   any fValue;
};
sequence NVList;
```

Eine Operation mit generischen Daten (NVList) könnte beispielsweise so aussehen:

```
interface Order;
interface OrderManager
{
```

```
    Order findOrder(in NVList pIdent);
    void placeOrder (in NVList pOrder, out NVListpResult);
};
```

Diese Schnittstelle ändert nicht, auch wenn irgendein Feld (durch ein NameValue-Objekt dargestellt) dazukommt oder wegfällt. Obiges Beispiel ist bereits ein guter Mittelweg, denn man kann der Schnittstelle ansehen, um welche Funktionen es sich handelt, auch wenn man den Inhalt der NVList-Objekte nicht in IDL beschrieben hat. Eine solche Schnittstelle würde genügend Flexibilität bieten und bei einer total neuen Anforderung käme höchstens eine weitere Operation dazu, was eine bestehende Client-Applikation nicht notwendigerweise negativ beeinflussen würde. Natürlich können wir neben dem einen Extrem aus a) wo wir auch die Order-Parameter mit IDL beschreiben ins andere Extrem gehen, wo wir pro Applikation nur noch eine Schnittstelle haben, die etwa wie folgt aussieht:

```
interface TotalGenericOrderManager
{
    void do(in NVList pParameter, out NVList pResult)
    throws OrderManagerException;
};
```

Hier wäre sogar die Funktion innerhalb der NVList pParameter versteckt, was auf der Server- und Clientseite keine Typenprüfung durch das CORBA-Umfeld bietet. Dafür benötigt es einiges mehr an Interpretations-Aufwand auf Client- und Server-Seite. Also haben wir hier "n x komplex und 1 x komplex". Je nach Applikation muss dies individuell interpretiert resp. programmiert werden. Und dies ist bei der Verwendung vom CORBA-Datentyp *any* (generisch) zudem weder Spass für die Client- noch für die Server-Entwickler. Zur Laufzeit wird dies bei grösseren Datenströmen auch ein Handicap für die Laufzeit, was natürlich so oder so ein gewisser Preis für gewonnene Flexibilität ist.

Wir können nun eine weitere Komponente dazunehmen (böse Zungen behaupten es sei einfach nur ein Hype), was wir in diesem Fall mit XML (eXtended Markup Language) tun. XML wird vom W3-Consortium (siehe www.w3.org oder auch www.xml.org) definiert und stellt eine von SGML (Structured images Markup Language) abgeleitete und mit HTML (HyperText Markup Language) verwandte Texttag-orientierte Darstellungs-Sprache dar. XML ist speziell für die Darstellung von Daten gedacht und ist strenger als HTML, aber weniger komplex als SGML. Ohne allzu tief auf XML einzugehen, kann man sagen, es ist endlich eine normierte Sprache, um Daten-Strukturen (z.B. Daten-Message-Strukturen) darstellen zu können. Da es eine Sprache ist, können damit natürlich wieder viele Arten von Strukturen definiert werden, welche vertikal, aber durch verschiedene Konsortien geschäftsprozessspezifisch enger definiert werden.

Weiter können zu XML-Datendarstellungen auch Überprüfungs-Templates (.dtd = Document Type Definition) und für die vielseitige Darstellung Style-Sheets oder Schemas definiert werden. Dank diesen Spezifikationen - und das ist der grosse Vorteil - kann man XML-fähige Werkzeuge einkaufen oder frei vom Internet herunterladen. Diese XML-Datenströme können ohne grossen Aufwand für den Entwickler automatisch verarbeitet werden.

Für unser Thema hier kann ein XML-Parser als Bibliothek (Java oder C++, siehe z.B.:

www.alphaworks.ibm.com/tech/xml4j

www.alphaworks.ibm.com/tech/XML4C

www.java.sun.com/developer/earlyAccess/xml/

eingesetzt werden, welcher einen eingehenden Datenstrom in einen Objekt-Baum umsetzt und dann für das Programm direkt verarbeitbar macht.

Der geneigte Leser sieht den Ansatz schnell für unseren nächsten Designtrick, unser OrderManager. Mit Verwendung von XML statt dem CORBA-Datentyp könnte die besprochene Schnittstelle wie folgt aussehen:

```
typedef sequence XMLList;
interface Order;
interface OrderManager
{
    Order findOrder(in XMLList pIdent);
    void placeOrder (in XMLList pOrder, out XMLList pResult);
};
```

Wenn der Applikations-Service wie die Client-Applikation über einen XML-Parser verfügen, dann können die übermittelten XML-Daten schneller und vor allem einfacher als eine *any*-Interpretation in einen dynamischen Objekt-Baum umgewandelt werden, den die Programme wieder direkt verwenden können. Kleine Nachteile - wie Typenprüfung durch CORBA-Typen - können vor allem für kommerzielle Applikationen in Kauf genommen werden, da diese in der Regel sowieso in Textform übermittelt werden.

Zugegeben, CORBA-Fanatiker werden mir dafür alles Böse wünschen, doch

pragmatische Ansätze haben schon immer mehr gebracht, als methodisch fanatische Verfechtungen. Architektur und Design ist dann gut, wenn man vorhandene Technologien optimal kombiniert und darauf weitere Konzepte aufbauen kann.

CORBA-Middleware durch seine sehr gute Unterstützung von heterogenen Plattformen als Basis und XML als bereits breit angewendete Datenspezifikations-Sprache richtig angewendet, bilden ein praktisches und flexibel einsetzbares Gespann, welches zudem noch das Attribut nicht proprietär tragen darf.

Andres Koch
El. Ing. HTL, M. Math.

Weitere Infos finden Sie unter:

www.objeng.ch/whitepapers/IDLDesign.pdf

www.ibm.com/developer/xml/

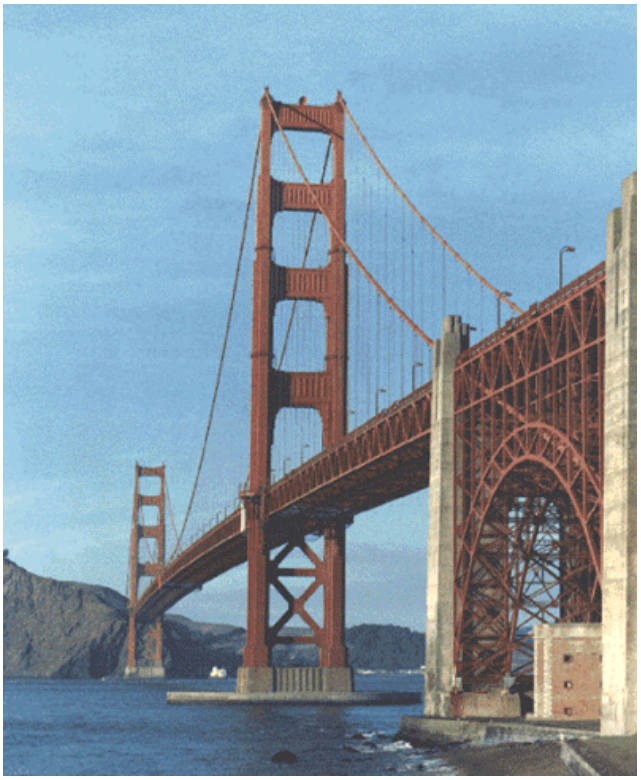
www.xml.org

www.w3.org

Wettbewerb

TOP

Wie heisst dieses Meisterwerk der Baukunst?



Es ist ein Bild der Golden Gate Bridge, von der San Francisco Seite aufgenommen.

Editorial

Liebe Leserin
Lieber Leser

Wer im Projektgeschäft Erfolg haben will, muss flexibel sein. Läuft ein Projekt an, sind es in der Regel wenige Personen, die sich mit Architektur und Design der Lösung

es in der Regel wenige Personen, die sich mit Architektur und Design der Lösung beschäftigen. Sobald aber die Detail-Design- und Implementations-Phase beginnt, wird die Sache sehr ressourcenintensiv.

Die Object Engineering ist in ihren 5 Jahren nur moderat gewachsen, auch wenn nicht kleine Aufgaben zu bewältigen waren. Bereits seit Beginn stand darum die "Virtuelle Kooperation" im Vordergrund. Wir haben uns mit Partnerfirmen zur *Solution Network Group* zusammengeschlossen, die es uns erlaubt, bei Bedarf auf spezialisierte Skills und Ressourcen zurückzugreifen, ohne dabei unsere Flexibilität und Wendigkeit zu verlieren. Seit kurzem werden auch die Akquisitionsaktivitäten innerhalb dieser Kooperation durchgeführt. Dies hat den Vorteil, dass die Object Engineering weiterhin flexibel bleibt und sich voll auf die Engineering-Tätigkeit konzentrieren kann.

Bei jeder Zusammenarbeit stehen Kommunikation und persönliche Beziehung im Vordergrund, damit auf einer guten Vertrauensbasis auch professionelle Leistungen erzielt werden können.

Das war in den ersten fünf Jahren der Fall und wird auch weiterhin so bleiben. Wir möchten Ihnen an dieser Stelle danken für diese gute Kooperation, das Vertrauen und die gute menschliche Beziehung.

Mit freundlichen Grüssen

Andres Koch
Dipl. El. Ing. HTL, M. Math
Geschäftsführer
Object Engineering GmbH