Ein RMI-basierter Repository-Server zur Synchronisation der Software-Entwicklung in kooperierenden Unternehmen

E. Ulrich Kriegel und Dirk Kurzmann

Fraunhofer Institut für Software- und Systemtechnik
Mollstraße 1, D-10178 Berlin
{ulrich.kriegel|dirk.kurzmann}@isst.fhg.de

Abstract. Im Rahmen eines Verbundprojektes zur Verbesserung der Softwareproduktion in kleinen und mittelständischen Unternehmen wurde am Fraunhofer ISST ein Repository-Server als Aufsatz zu bestehenden Softwareentwicklungsumgebungen entwickelt. Über diesen Server können mit Hilfe spezieller Werkzeuge Entwicklungsdokumente zwischen verteilten Entwicklern ausgetauscht werden. Der vorliegende Bericht beschreibt den Entwurf und die Implementation des Repository-Servers sowie die Werkzeuge, die auf ihn zugreifen.

1 Einleitung

Im Rahmen des Landesprogrammes IKT des Berliner Senats wurde das Verbundprojekt »SPU: Eine Softwareproduktionsumgebung für kleine und mittelständische Unternehmen« im Zeitraum vom 01.09.1995 bis zum 28.02.1998 gefördert. Die Bearbeitung des Projektes erfolgte durch vier Berliner Unternehmen gemeinsam mit dem Fraunhofer Institut für Software und Systemtechnik, das auch mit der Leitung und Koordination des Projektes betraut war [1].

In der Abschlußphase des Projektes wurde das entwickelte Vorgehensmodell [2] und die prototypische Werkzeugumgebung [3] im Hinblick auf eine Verwendung für die Softwareentwicklung in kooperierenden Unternehmen modifiziert [1].

Ziel dieser Publikation ist die Beschreibung der Werkzeuge, die zum Austausch von Artefakten[1] zwischen den verteilten Entwicklern realisiert wurden. Als Einführung wird in Kapitel 2 die grundlegende Struktur der im SPU-Projekt entwickelten Softwareproduktionsumgebung (SPU), deren Prozesse den Rahmen für die Entwicklung bilden, beschrieben. Daran anschließend werden in Kapitel 3 einige Aussagen zur Implementationstechnologie und zur Implementationsarchitektur des Systems gemacht. Kapitel 4 gibt eine Beschreibung der Realisierung der einzelnen Komponenten. Auf die Berücksichtigung von Sicherheitsanforderungen beim Austausch von Artefakten wird in Kapitel 5 eingegangen. Kapitel 6 gibt eine Zusammenfassung der Erfahrungen bei der Realisierung.

2 Die Struktur der Softwareproduktionsumgebung

Für kleine und mittelständische Unternehmen erweist es sich als günstig, bei der Architektur der Softwareproduktionsumgebung (SPU) von einem einfachen Arbeitsbereich-Repository-Modell auszugehen[2]: Jeder Entwickler besitzt einen separaten Arbeitsbereich, in dem die Entwicklung stattfindet. Alle im Entwicklungsprozeß eingesetzten Werkzeuge arbeiten direkt mit den im Arbeitsbereich abgelegten Dateien. Ein lokales Repository sichert die Revisionierbarkeit der Entwicklung, Änderungen müssen explizit durch die Entwickler in das Repository eingespielt werden.

---

[1] Im Rahmen des SPU-Projektes werden alle im Softwareproduktionsprozeß erzeugten und modifizierten Entitäten wie Dokumente, Images, usw. als Artefakte bezeichnet.

[2] Existierende Softwareentwicklungsumgebungen auf der Basis virtueller Dateisysteme (siehe z.B. Referenzen in [5]), deren Komponenten transparent auf den Inhalt von Repositories zugreifen können, sind für den Einsatz besonders in kleinen Unternehmen sowohl zu komplex als auch zu kostenintensiv.

Zur Synchronisation der Arbeit mehrerer Entwickler werden den Artefakten im Repository Bearbeitungszustände zugewiesen:

- Saved - private Sicherung des Entwicklers;
- Proposed - vom Entwickler getestet und für den Integrationstest freigegebene Revision;
- Published - vom Systemintegrator freigegebene konsolidierte Revision.

Bild 1 zeigt in Form eines Aktivitätsdiagramm das verwendete Prozeßmodell [2]. Parallel zu dem eigentlichen Entwicklungsprozeß, der auf dem »Round Trip Gestalt Engineering Model« von Booch [4] basiert (links), läuft bei jedem Entwickler der rechts dargestellte Konsolidierungsprozeß zur Qualitätssicherung ab.

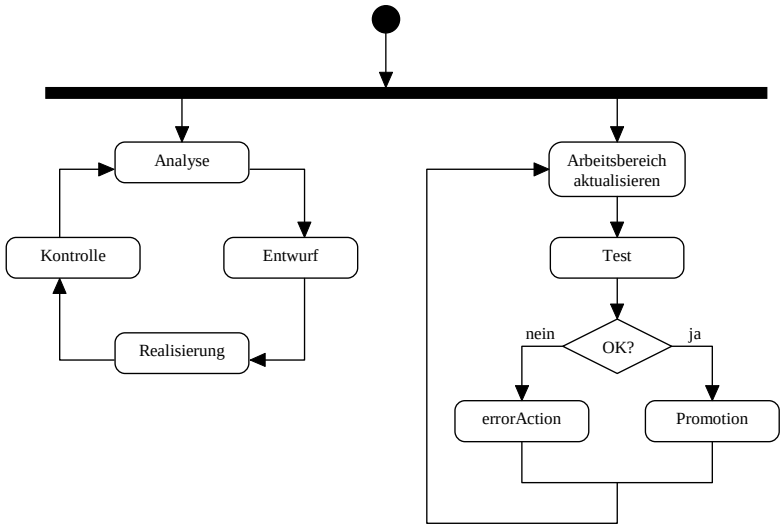

Bild 1. Überblick über das SPU-Prozeßmodell

Ein Entwickler aktualisiert seinen Arbeitsbereich mit den neuesten für den Integrationstest freigegebenen Artefakten seines Teams und mit den neuesten von ihm selbst bearbeiteten Artefakten. Danach werden die Arbeiten gemäß des Prozeßmodells durchgeführt. Sind die Arbeiten beendet, so führt der Entwickler die vorgesehenen Konsistenztests durch. Bei erfolgreichem Ablauf werden die von ihm bearbeiteten Artefakte ins Repository eingespielt, ihr Bearbeitungszustand wird auf die Stufe »Proposed« gesetzt, so daß sie am Integrationstest der anderen Teammitglieder teilnehmen können.

Parallel zu den Prozessen bei den einzelnen Entwicklern läuft pro Lokation ein Konsolidierungsprozeß ab, der alle von den Entwicklern für den Integrationstest freigegebenen Artefakte testet. Bei erfolgreichem Test wird der Bearbeitungszustand der getesteten Artefakte auf den Zustand »Published« befördert.

Da sich die hier beschriebene Vorgehensweise bei der Softwareentwicklung in einzelnen Unternehmen bewährt hat, wurde sie als Grundlage für ein Vorgehensmodell in einem Konsortium kooperierender Unternehmen gewählt. Die lokalen SPU bleiben eigenständig, Entwicklungen finden nur hier durch Mitarbeiter der jeweiligen lokalen Unternehmen statt. Die Synchronisation der verteilten Entwicklung erfolgt durch den Austausch von Artefakten über ein globales Repository. Zusätzliche Werkzeuge, die den Transfer von Artefakten zum und vom globalen Repository bewerkstelligen, werden als »add ons« zu den lokalen Entwicklungsumgebungen bereitgestellt, ohne direkt in diese einzugreifen. Analog zu den oben beschriebenen Konsolidierungsprozessen wird ein Konsolidierungsprozeß definiert, der die Qualität der Artefakte im globalen Repository sichert. Für eine detaillierte Beschreibung der Projektorganisation und der »use cases« beim Austausch von Artefakten muß auf [1] verwiesen werden.

3 Implementationstechnologie und Architektur des Systems

Die Architektur des Systems besteht aus einem globalen Artefaktmanagement (»GAM«) mit einem globalen Repository, dem ein Server mit Zugriff auf das Repository zugeordnet ist, sowie aus Ex- und Import-Werkzeugen, die Mitarbeitern lokaler Unternehmen den Zugriff auf das globale Repository ermöglichen. Bevor wir jedoch weiter auf die Architektur eingehen, folgen einige Bemerkungen zur Auswahl der Entwicklungstechnologie.

3.1 Auswahl der Entwicklungstechnologie

Die primäre Anforderung an die zu realisierende Architektur war Plattformunabhängigkeit. Zieht man die Tatsache der freien Verfügbarkeit in Betracht, so kam als Implementationssprache nur Java (Version 1.1.x) in Frage. Für die Verwendung von Java sprach weiterhin, daß

- über das »Remote Method Invocation«-Protokoll (RMI) auf einfache Weise eine transparente Kommunikation zwischen Clients und Repository-Server implementiert werden kann;
- über Properties und dynamisches Laden von Klassen das System variabel gestaltet werden kann;
- auf der Basis der »Swing«-Bibliotheken und des durch das Event-Modell unterstützten Observer-Patterns [6] schnell eine komfortable Oberfläche realisiert werden kann;
- das Paket *java.security* und das Werkzeug *javakey* die Erzeugung und Verwaltung von Signaturen und Zertifikaten stark vereinfachen.

Gegen die Verwendung von Java sprachen

- die Nicht-Verfügbarkeit einer komfortablen Entwicklungsumgebung unter Solaris, speziell eines komfortablen Debuggers, und
- die Effizienz bei der Abarbeitung.

Da die Vorteile überwiegen, wurde die Architektur in Java[3] realisiert. Die Entwürfe wurden mit Rational Rose/C++ [7] ausgeführt und dann per Hand kodiert. Als rudimentäre Entwicklungsumgebung wurde der Emacs[4] mit einer Erweiterung zur Arbeit mit Java [8] in Verbindung mit dem Versionsverwaltungssystem CVS [9] eingesetzt.

3.2 Die Implementationsarchitektur des Systems

Bild 2 zeigt in einem Komponentendiagramm die einzelnen Bestandteile der Implementation.

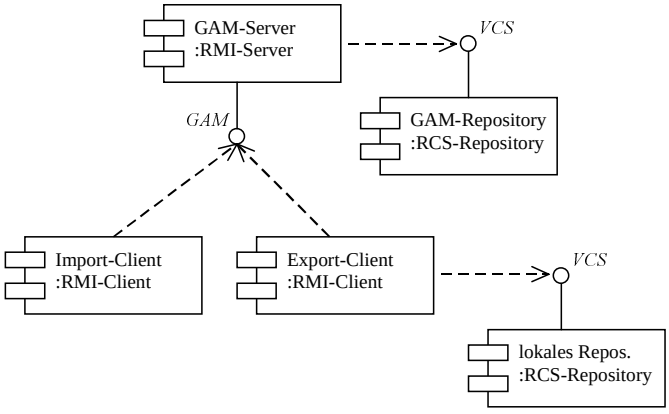

Bild 2. Komponentendiagramm der Bestandteile der Implementation

[3] JDK 1.1.4 und höher.
[4] Gnu Emacs 20.2 [10]

Der Server des globalen Artefaktmanagements (»GAM-Server«) implementiert die Schnittstelle »GAM«, die alle für die Kommunikation notwendigen Operationen deklariert. Die Anbindung der Werkzeuge an das globale Repository erfolgt über eine abstrakte Schnittstelle »VCS«, die für konkrete Revisionsverwaltungssysteme zu implementieren ist. Für das GAM und die lokalen Repositories sind daher unterschiedliche Implementationen einsetzbar.

Im Rahmen des SPU-Projektes wurde für das Revisionsverwaltungssystem RCS [11] ein entsprechender Adapter implementiert, der die Methodenaufrufe an die korrespondierenden RCS-Kommandos weiterleitet.

Um möglichst variabel zu sein, werden die Werte von Initialisierungskonstanten und die Namen der die Interfaces implementierenden Klassen über Property-Dateien eingelesen und dynamisch instantiiert.

Die Kommunikation zwischen Client und Server wird über das »Remote Method Invocation (RMI)«-Protokoll realisiert. Zur Optimierung des Austausches von Artefakten wurden die Konzepte »Subskription« und »Distribution« implementiert.

Eine Subskription beschreibt eine Menge von Artefaktrevisionen, die jeweils durch den Pfad des entsprechenden Artefaktes relativ zur Wurzel eines Arbeitsbereiches und durch die Angabe der Revision gekennzeichnet werden. Zur Beschreibung der Revision kann entweder eine Revisionsnummer oder ein Prädikat der Form »neueste Revision mit ‚einem bestimmten Bearbeitungszustand'« angegeben werden. Bei der Angabe von Revisionsnummern beschreibt eine Subskription eine konkrete Konfiguration von Artefakten. Werden Prädikate zur Beschreibung der Artefaktrevisionen verwendet, so entspricht die Subskription einer »logischen Konfiguration«.

Subskriptionen werden über *Hashtable*-Objekte realisiert und implementieren das Interface *Serializable*, so daß sie in Form von Dateien[5] lokal beim Bearbeiter verwaltet werden können.

Um den Aufwand bei der Übertragung von Artefakten möglichst zu reduzieren, werden Artefakte nicht als Entitäten vom oder zum GAM-Server übertragen, sondern in Form von Distributionen. Eine Distribution kann man sich als linearisierte und komprimierte Verzeichnisstruktur vorstellen, die jedoch zusätzlich zu Artefakten noch Verwaltungsinformationen enthält, z.B. das entsprechende Subskriptionsobjekt.

Distributionsobjekte enthalten ein Subskriptionsobjekt und ein Feld von Bytes zur Aufnahme der Artefaktrevisionen. Diese werden in einen *ZipOutputStream* geschrieben, der dann zum Zwecke der Serialisierung über ein *ByteArrayOutputStream* in ein Feld von Bytes umgewandelt wird.

4 Beschreibung der realisierten Komponenten

4.1 Das globale Artefaktmanagement

Für das globale Artefaktmanagement wurde ein Server zur Behandlung der Im- und Export-Anfragen realisiert. Der GAM-Server wurde als *UnicastRemoteObject* deklariert und implementiert die Schnittstelle *GAM*

```
public interface GAM extends java.rmi.Remote {
  public ArtefactFolder getRepositoryStructure()
    throws RemoteException;
  public Distribution getDistribution(Subscription subs)
    throws RemoteException;
  ... }
```

[5] Eine Subskription kann mit den Prädikaten gespeichert werden, es ist aber auch möglich, die Prädikate bei der Speicherung aufzulösen und die Subskription mit festen Revisionsnummern zu speichern.

Über den Aufruf der in der Schnittstelle deklarierten Methoden durch die Clients können Informationen aus dem globalen Repository extrahiert oder in das globale Repository transferiert werden.

Der GAM-Server verwaltet in einer Instanz der Klasse *ArtefactFolder* eine Metainformationsstruktur, in der der Aufbau des globalen Repositories und Informationen zu den einzelnen Artefakten wie Revisionen, deren Bearbeitungszustände und -kommentare festgehalten werden. Die Klasse *ArtefactFolder* implementiert das Interface *Serializable* und kann deshalb persistent gespeichert werden, d.h. bei einem Neustart des GAM-Servers kann die Metainformationsstruktur wieder aufgebaut werden.

4.2 Der Import-Client

Für jeden Entwickler steht eine Applikation zum Import der global verwalteten Artefakte zur Verfügung. Dieser Import-Client stellt dazu die im globalen Repository abgelegten Artefakte entsprechend der Verzeichnisstruktur in Form eines Baumes dar. Für jedes Artefakt können Informationen über die vorhandenen Revisionen sowie deren Bearbeitungszustände und Kommentare abgerufen werden. Um die Netzlast zu reduzieren, wird beim Start des Import-Clients zuerst über den Aufruf der Methode *getRepositoryStructure()* des GAM-Servers eine Kopie der im GAM verwalteten Metainformation zum Client transferiert. Während der Laufzeit des Clients kann mit dieser Struktur gearbeitet werden[6], es ist jedoch auch möglich, diesen Cache auszuschalten und ggf. vor jeder Aktualisierung die Metaobjektstruktur zu aktualisieren.

Zur Vorbereitung des Imports von Artefakten werden vom Nutzer zunächst Subskriptionen angelegt und lokal gespeichert, in denen festgehalten wird, welche Artefakte der Nutzer benötigt. Neben einzelnen Artefakten ist es möglich, auch ganze Verzeichnisse zu selektieren; alle darin (auch rekursiv) enthaltenen Artefakte werden dann in die Subskription aufgenommen. Bild 3 zeigt in einem Schnappschuß das Fenster des Import-Client zur Erzeugung von Subskriptionen.

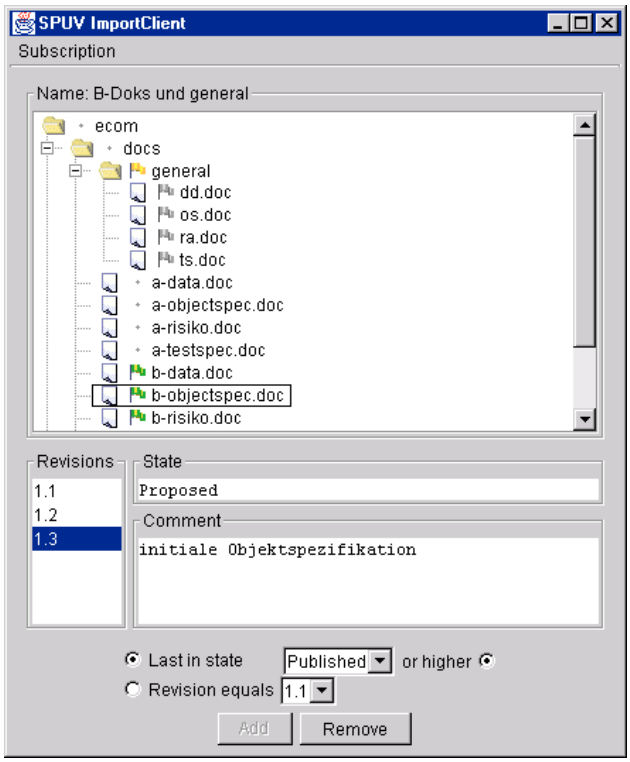

Bild 3. Import-Client zur Erzeugung von Subskriptionen

[6] Im Rahmen des Vorgehensmodells wird davon ausgegangen, daß die Entwickler zu bestimmten Zeitpunkten ihre Arbeitsbereiche aktualisieren, so daß eine Änderung des aktuellen Zustandes des globalen Repositories nicht sofort sichtbar sein muß.

Bei der Auswahl der Revision eines Artefaktes können sowohl konkrete als auch logische Revisionen ausgewählt werden, z.B. »die letzte Revision im Zustand ‚Proposed' oder besser«. Für Verzeichnisse stehen aus naheliegenden Gründen nur logische Revisionen zur Verfügung, die dann für alle enthaltenen Artefakte ausgewertet werden.

Eine Subskription kann eingefroren werden, d.h. alle logischen Revisionen werden in konkrete aufgelöst, so daß zu einem späteren Zeitpunkt eine gewünschte Konfiguration wieder hergestellt werden kann.[7]

Zum eigentlichen Import von Artefakten wird eine Subskription aus der Menge der gespeicherten Subskriptionen ausgewählt und über den Aufruf der Methode *getDistribution(Subscription subs)* des GAM-Servers diesem übergeben. Dort werden die gewünschten Artefakte in einer Distribution zusammengestellt und als Ergebnis des Methodenaufrufs zurückgegeben. Die empfangenen Distributionen werden dann vom Import-Client in einem separaten Verzeichnis abgelegt.

4.3 Distributionsmanager

Die Installation einer Distribution im Arbeitsbereich eines Entwicklers erfolgt durch ein als Distributionsmanager bezeichnetes Werkzeug (siehe Bild 4), mit dem der Inhalt der vorhandenen Distributionen betrachtet und entpackt werden kann.

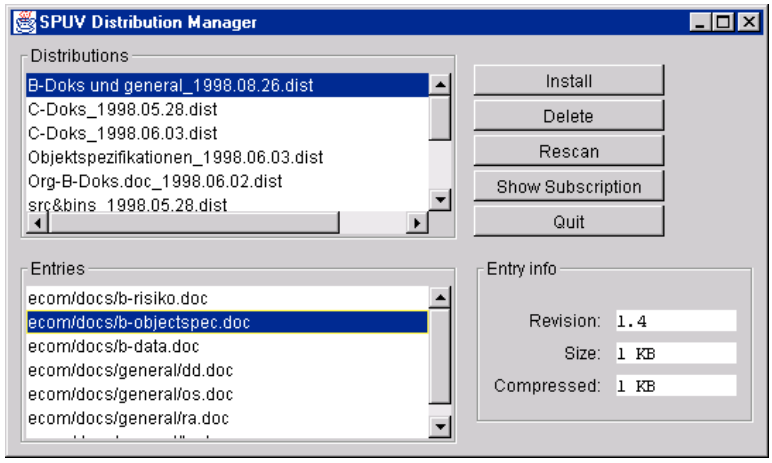

Bild 4. Schnappschuß des Hauptfensters des Distributionsmanagers

4.4 Der Export-Client

Zur Unterstützung des lokalen Export-Managers bei der Erstellung und Versendung von Export-Distributionen steht ein Export-Client zur Verfügung (siehe Bild 5). Ein neuer Export wird in der Regel nach einem der zyklisch durchgeführten Integrationstests vorgenommen. Alle Artefakte, die diese Tests bestanden haben, werden auf den Zustand »Published« befördert und stehen damit für einen Export zum globalen Artefaktmanagement zur Verfügung. Um bereits exportierte Artefakte nicht unnötig erneut zu exportieren, verwaltet der Exportmanager intern die Namen aller bereits exportierten Artefaktrevisionen.

---

[7] Da die Subskriptionen als Dateien gespeichert und auch in einem Repository verwaltet werden können, kann auf der Basis von Subskriptionen ein einfaches Konfigurationsverwaltungssystem realisiert werden.

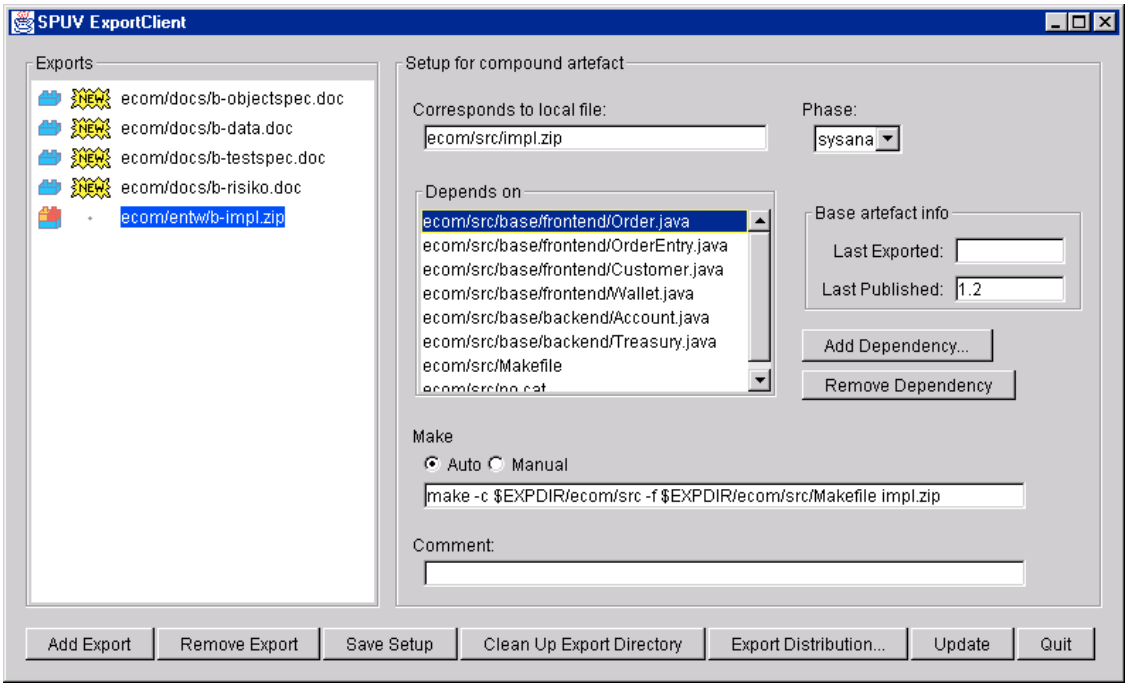

Bild 5. Export-Client

Die in einer Export-Distribution enthaltenen Artefakte werden vom GAM-Server mit dem Zustand »Proposed« in das globale Repository eingecheckt. Nach entsprechenden globalen Konsistenstests wird der Bearbeitungszustand dann auf »Published« erhöht.

5 Sicherheitsaspekte bei der Übertragung von Artefakten

Das verteilte Arbeiten auf der Basis öffentlicher Kommunikationskanäle stellt erhöhte Ansprüche an Datensicherheit, Authentifizierung und Autorisierung.

Im Rahmen der Implementation wird die Frage der Sicherheit unter zwei Aspekten berücksichtigt: Integrität und Authentizität. Auf die explizite Integration von Verschlüsselungstechniken bei der Übertragung von Artefakten wurde im Rahmen der Implementation verzichtet. Einerseits standen für asymmetrische Verfahren [12][13] keine frei verfügbaren Implementationen in Java zur Verfügung, andererseits sollte der Prototyp so einfach und kostengünstig wie möglich gestaltet werden. Es ist jedoch problemlos möglich, eine Verschlüsselung der Datenübertragung nachträglich zu integrieren.

Zur Sicherstellung von Integrität und Authentizität werden in der vorliegenden Implementation digitale Signaturen verwendet. Die Realisierung erfolgte auf der Basis des *java.security*-Pakets, das Implementationen für Schlüssel- und Signaturenobjekte sowie für Zertifikate anbietet. Die Erzeugung und Verwaltung von Entitäten, Schlüsseln und Zertifikaten wird mit dem Werkzeug *javakey* durchgeführt, wobei für jeden Nutzer, Export-Manager und für das globale Artefaktmanagement separat eine private Datei *identitydb.obj* angelegt wird [14].

Bei der Übertragung von Subskriptionen oder Distributionen wird neben der Signatur ein Zertifikat mit dem öffentlichen Schlüssel des Signierers übermittelt. Der Zertifizierer garantiert darin durch Signierung des Zertifikats, daß der in der Signatur angegebene Schlüssel der öffentliche Schlüssel des Signierers ist. Zusätzlich zur Validierung der Signatur muß bei dieser Vorgehensweise die Vertrauenswürdigkeit des Zertifizierers geprüft werden.

Im Rahmen der Artefaktverwaltung wird eine hierarchische Vorgehensweise bei der Zertifizierung gewählt. Das globale Artefaktmanagement zertifiziert die lokalen Export-Managements, jedes Export-Management stellt Zertifikate für die lokalen Clients aus.

Das Sicherheitskonzept ist so entworfen, daß die reine Applikationslogik des globalen Artefaktmanagements nicht berührt wird, d.h. für die angebotenen Dienste (Import und Export von Artefakten) werden die notwendigen Sicherheitsüberprüfungen völlig transparent durchgeführt. Dies wurde durch die Einführung einer zusätzlichen Sicherheitsschicht (siehe Bild 6) in die bestehende Architektur für den Methodenaufruf über RMI erreicht. Die Einbindung dieser Schicht kann durch Setzten einer Eigenschaft (*spuv.secure=true*) beim Start des GAM-Servers aktiviert werden.[8]

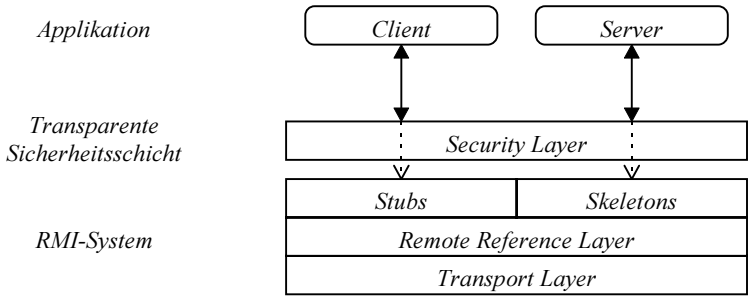

Bild 6. Ergänzung der RMI-Architektur für das Sicherheitskonzepts

6 Zusammenfassung

Im Vergleich mit publizierten Lösungen, bei denen auf zentrale Repositories mit speziellen Revisionsverwaltungssystemen über das WWW [15] oder über ftp [16] zugegriffen wird, ist die hier vorgestellte Lösung unabhängig von einem speziellen Revisionsverwaltungssystem und erfüllt damit die im Projekt definierte Anforderung, daß die Werkzeuge zum Austausch von Artefakten unabhängig von den lokalen Softwareentwicklungsumgebungen sein müssen.

RCS als Werkzeug zur Verwaltung des globalen Repositories kann durch andere Revisionsverwaltungssysteme ersetzt werden, indem man Adapter implementiert, die die Schnittstelle VCS implementieren.

Durch die Verwendung von Java als Implementationssprache konnte trotz der »puristischen« Entwicklungsumgebung schon in kurzer Zeit ein funktionsfähiger Prototyp erstellt werden, der dann iterativ verfeinert wurde. Die in den unterschiedlichen Paketen implementierten Klassen zur Realisierung verteilter Objekte, von Oberflächen und von Signaturen/Zertifikaten haben die Arbeit wesentlich beschleunigt. Teilweise machte sich das Fehlen eines komfortablen Debuggers bemerkbar. Problematisch sind die relativ langen Startzeiten der Werkzeuge unter JDK1.1.6. Die Erzeugung von Signaturen bzw. deren Validierung ist besonders bei großen Objekten zeitaufwendig. Wir erwarten jedoch, daß sich die Performanz bei der Erzeugung von Signaturen bzw. bei deren Validierung mit dem Einsatz einer verbesserten virtuellen Maschine auf der Basis der HotSpot-Technologie [17] stark verbessert.

Alle Werkzeuge wurden unter UNIX/Solaris 2.51 entwickelt und dann unter Windows/NT 4.0 getestet. Portabilitätsprobleme konnten für die in Java realisierten Werkzeuge nicht festgestellt werden, es traten jedoch Portabilitätsprobleme durch unterschiedliche Implementationen von RCS unter UNIX und Windows/NT 4.0 auf.

[8] Das zum Implementationszeitpunkt nur als Beta-Version zur Verfügung stehende JDK 1.2 bietet eine weitere Möglichkeit der transparenten und dynamischen Einbindung von Sicherheitsmechanismen. Die standardmäßig vorgegebene TCP-Implementation der Transportschicht kann hier durch andere Implementationen ersetzt werden. Damit ist es z.B. möglich, das RMI-Protokoll auf eine SSL-Implementation aufsetzen zu lassen.

Zusammenfassend kann man feststellen, daß sich die Implementation wesentlich einfacher gestaltete als vergleichbare Projekte unter Verwendung von Sprachen wie z.B. C++. Aus persönlicher Erfahrung muß jedoch auch gesagt werden, daß die Qualität und Effizienz der Entwicklung in Java noch nicht mit der in dynamischen objektorientierten Sprachen vergleichbar ist, z.B. mit Entwicklungen in CommonLisp, wo integrierte, in der Sprache selbst geschriebene Entwicklungssysteme mit inkrementellen Compilern zur Verfügung stehen, die die Entwicklung stark beschleunigen können.

Literatur

[1] E. U. Kriegel, D. Kurzmann, J. Altenhein, W. Handke, M. Löw,»Eine Softwareproduktionsumgebung für kleine und mittelständische Unternehmen - Kooperation in virtuellen Unternehmen« , Fraunhofer ISST, 47/98

[2] E. U. Kriegel, J. Altenhein, M. Schlösser-Fassbender, P. Thierse,»Eine Softwareproduktionsumgebung für kleine und mittelständische Unternehmen«, Fraunhofer ISST, 36/96

[3] E. U. Kriegel, D. Kurzmann, J. Altenhein, M. Löw, P. Thierse, »Eine Softwareproduktionsumgebung für kleine und mittelständische Unternehmen - 2. Teil«, Fraunhofer ISST, 42/97

[4] G. Booch, »Object-Oriented Design with Applications«, Benjamin Cummings, 1991

[5] W.F.Tichy (Ed),»Configuration Management«, Wiley 1994

[6] E. Gamma, R. Helm, R. Johnson, J. Vlissides, »Design Patterns - Elements of Reusable Object-Oriented Software«, Addison-Wesley, 1995

[7] http://www.rational.com

[8] http://sunsite.auc.dk/jde/

[9] http://www.cyclic.com

[10] http://www.gnu.org/software/emacs/emacs.html

[11] W.F. Tichy, »RCS - A System for Version Control«, Software -- Practice & Experience 15, 7, 637 - 654.

[12] B. Schneier, »Applied Cryptography: Protocols, Algorithms and Source Code in C«, Wiley & Sons, 1996

[13] Man Young Rhee, »Cryptography and Secure Data Communication«, McGraw-Hill, 1994

[14] »Java Cryptography Architecture: API Specification and Reference« , Sun Microsystems, 1997

[15] J. Reuter, S. Hänßgen, J. Hunt, W. Tichy,»Distributed Revision Control Via the World Wide Web« in »Proceedings of the 6th International Workshop on Software Configuration Managemen«, Lecture Notes in Computer Scieence Nr. 1167, Springer, 1996

[16] http://www.silver.com/cvs-via-ftp

[17] http://java.sun.com/products/hotspot/index.html