Überblick und Vergleich von Technologien zur Realisierung einer Middleware für mobile Informationssyteme

Sven Apel

Institut für Technische und
Betriebliche Informationssysteme
Otto-von-Guericke-Universität Magdeburg
Universitätsplatz 2
39106 Magdeburg
apel@iti.cs.uni-magdeburg.de

Marco Plack

METOP GmbH
An-Institut der
Otto-von-Guericke-Universität Magdeburg
Sandtorstraße 23
39106 Magdeburg
marco.plack@metop-md.de

Abstract: Leistungsfähige mobile Geräte wie z. B. Mobiltelefone, Smartphones oder PDAs (Personal Digital Assistant) sind aus dem täglichen Leben nicht mehr wegzudenken. Moderne Konzepte wie *ubiquitous* und *pervasive computing* beschreiben eine neue Qualität der Verfügbarkeit von Informationen an jedem Ort zu jeder Zeit mit beliebigen mobilen Geräten. Bereits jetzt stehen mit GSM/GPRS und WLAN moderne Kommunikationsverfahren zur Verfügung, die den Zugang zu Informationen nahezu flächendeckend ermöglichen. Das leistungsfähigere breitbandige UMTS wird in Zukunft (2004/2005) GSM/GPRS über kurz oder lang ersetzen. Neben der technischen Möglichkeit zwischen Geräten Informationen auszutauschen, ist jedoch auch eine Integration von Anwendungen und Diensten geräteübergreifend erforderlich. Ein probater Ansatz zu solch einer Integration ist eine Middleware-Plattform, welche alle dafür nötigen Funktionalitäten bereitstellt. Die Middleware ermöglicht die transparente Kommunikation zwischen mobilen Anwendungen bzw. mobilen Informationssystemen. In diesem Beitrag werden Technologien vorgestellt, welche als Grundlage für eine solche Middleware dienen können. Diese Technologien werden anhand ihrer Eigenschaften klassifiziert und ihr Nutzen abgeschätzt. Es werden ein Testszenario sowie gewonnene Erkenntnisse und Ergebnisse vorgestellt.

1 Einleitung

Eine Perspektive der Informationsgesellschaft liegt in der Mobilität der Endgeräte bzw. des Nutzers. Die Vision ist, dass Informationen immer und überall zugänglich sind bzw. bereitgestellt werden können. Eine Vielzahl neuer Endgeräte und die damit verbundenen Möglichkeiten werden zweifellos unser tägliches Leben verändern [Wei93]. Begriffe wie *ubiquitous* oder *pervasive computing* sind derzeit in der Fachwelt in aller Munde. Diese Begriffe spiegeln die Chancen und Möglichkeiten der neuen Mobilität wieder. Mobilität betrifft hierbei den Anwender, die Endgeräte wie PDAs oder Smartphones sowie die Dienste selbst. Wegen des wachsenden Bedarfs von informationszentrierten Anwendungen wird es immer wichtiger, neben herkömmlichen mobilen Anwendungen auch mobile

verteilte Informationssysteme zu etablieren. Zur Datenhaltung und Datenverwaltung der Informationssysteme sind auf den mobilen Geräten und auf stationären Servern DBMS im verteilten System gekoppelt. Ein probater Ansatz zur Integration der überall im System verteilten, von den mobilen und stationären Informationssystemen bereitgestellten, Dienste ist eine Middleware-Plattform. Diese ermöglicht die transparente Kommunikation zwischen herkömmlichen mobilen Anwendungen, mobilen Informationssystemen und standalone DBMS im verteilten System[1]. Dazu muss die Verteilung der Informationen der mobilen DBMS, die transparente Kommunikation der mobilen Informationssysteme und die Zusammenarbeit mit den mobilen verteilten DBMS realisiert werden. Mögliche Anwendungen wie mobile Touristenführer, welche ortsabhängig Informationen liefern, sind auf Basis der derzeitigen Hardware bereits möglich. Die Entwicklung einer grundlegenden Software-Infrastruktur in Form einer Middleware und in Form von mobilen verteilten DBMS muss noch vorangetrieben werden. Dazu müssen viele Probleme im Bereich der Softwaretechnik gelöst sowie auch Verbesserungen im Bereich der Hardware vorgenommen werden.

Die Middleware-Plattform integriert grundlegende Dienste, bietet Standardschnittstellen an und erleichtert damit die Entwicklung von mobilen verteilten Informationssystemen. Die Informationssysteme befinden sich dabei auf verschiedensten mobilen und auch stationären Geräten. Im verteilten System werden die Informationssysteme als Dienstnutzer und Diensterbringer betrachtet. Um für die Kommunikation nötige Details wie z. B. das Kommunikationsprotokoll oder der Synchronisationsmechanismus brauchen sie sich nicht zu kümmern. Auch das Vorhandensein der mobilen verteilten DBMS ist für die Informationssysteme transparent. Die Middleware befindet sich zwischen Betriebssystem und der Anwendungsebene. Sie ist stark verzahnt mit den mobilen verteilen DBMS. In Abbildung 1 ist verdeutlicht, wie die Middleware mit mobilen DBMS zusammenarbeitet. Die DBMS tauschen untereinander, unter Nutzung der Funktionen der Middleware-Infrastruktur, Daten aus und ermöglichen so die transparente Kommunikation zwischen den mobilen Informationssystemen. Weiterhin greift die Middleware auf Funktionalitäten der mobilen DBMS zurück. Alle innerhalb der Middleware anfallenden Daten, sowie die für die mobilen Informationssysteme relevanten Daten werden im mobilen DBMS verwaltet. Der Zugriff der Anwendungslogik eines mobilen Informationssystems auf spezielle Daten anderer Informationssysteme ist transparent. Es spielt dabei keine Rolle ob die Informationssysteme sich auf demselben Gerät befinden oder auf verschiedenen Geräten verteilt sind. Weiterhin kann die Anwendungslogik eines Informationssystems auf entfernte DBMS auf mobilen oder stationären Servern zugreifen.

Im weiteren Beitrag wird in Abschnitt 2 ein mögliches Anwendungsszenario vorgestellt. Ausgehend von diesem Anwendungsszenario werden in Abschnitt 3 die Zielsysteme festgelegt. In Abschnitt 4 werden dafür verfügbare Softwarelösungen diskutiert. Weiterhin wird in Abschnitt 5 ein Testszenario präsentiert. Neben der verwendeten Hardware und Software wird eine exemplarische Beispielanwendung vorgestellt. Danach werden die aus den Testszenarien resultierenden Erkenntnisse und Ergebnisse diskutiert. Abschließend werden eine Zusammenfassung und ein Ausblick gegeben.

[1] Der Begriff Informationssystem bezeichnet dabei ein DBMS mit Anwendungslogik. Im weiteren Verlauf wird bzgl. der Middleware nur noch von Informationssystemen gesprochen. Die Middleware integriert auch herkömmliche mobile Anwendungen, diese werden aber im weiteren Beitrag nicht explizit Erwähnung finden.

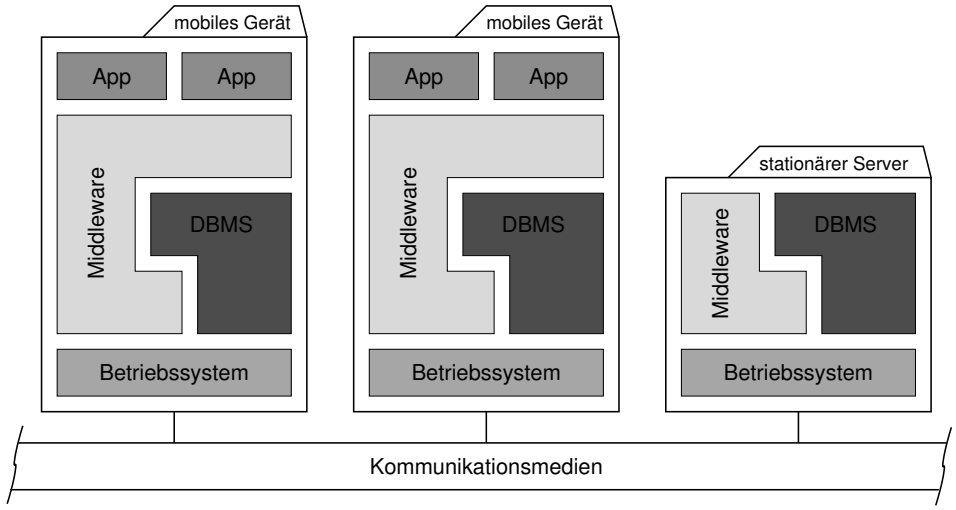

Abbildung 1: Integration von mobilen Anwendungen und mobilen und stationären DBMS durch eine Middleware-Architektur

2 Ein Anwendungsszenario

In diesem Abschnitt wird eine E-Learning Applikation als ein mögliches Anwendungsszenario vorgestellt. Die Idee des E-Learning Szenarios ist, dass Studenten einer Universität die Möglichkeit haben, mittels mobiler und auch stationärer Geräte Informationen auszutauschen. Dazu können Lerngruppen gebildet werden, in denen sich die Studenten anhand ihrer Interessen virtuell zusammenfinden. Neben dem Austausch von aktuellen Lerninhalten zwischen den Studenten mittels mobiler Geräte, sollen auch zentrale Datenbankserver Informationen bereitstellen. Auf dem Campus steht ein Zugangspunkt für die drahtlose Kommunikation bereit. Es wird also ein Infrastruktur-Netzwerk mit drahtlosen sowie drahtgebundenen Übertragungsstrecken aufgebaut. Die E-Learning Anwendung soll natürlich auch funktionieren in dem Falle, dass sich die mobilen Geräte nicht in Reichweite des Netzzugangspunktes befinden. Das System soll beispielsweise auch außerhalb des Campus funktionieren. Hierzu sollen spontan Ad-hoc-Netzwerke zwischen den einzelnen mobilen Geräten aufgebaut werden. In Abbildung 2 ist das E-Learning Szenario dargestellt. Es ist zu beachten, dass ein Infrastruktur-Netzwerk und Ad-Hoc-Netwerke aufgebaut werden.

In Hinblick auf dieses Szenario müssen einige Anforderungen an die Endgeräte sowie an die verwendeten Technologien der Middleware gestellt werden. Die mobilen Geräte müssen eine gewisse Leistungsfähigkeit besitzen und über gewisse Ressourcen verfügen. Die Speicherung, Verarbeitung und Präsentation von Lerninhalten in Form von Textdokumenten, graphisch anspruchsvollen Präsentation bis hin zu Multimediadokumenten verlangt nach einer gewissen Leistungsfähigkeit. Das betrifft nicht nur den Hauptspeicher und die Geschwindigkeit des Prozessors sondern auch die Laufzeit bei Batteriebetrieb,

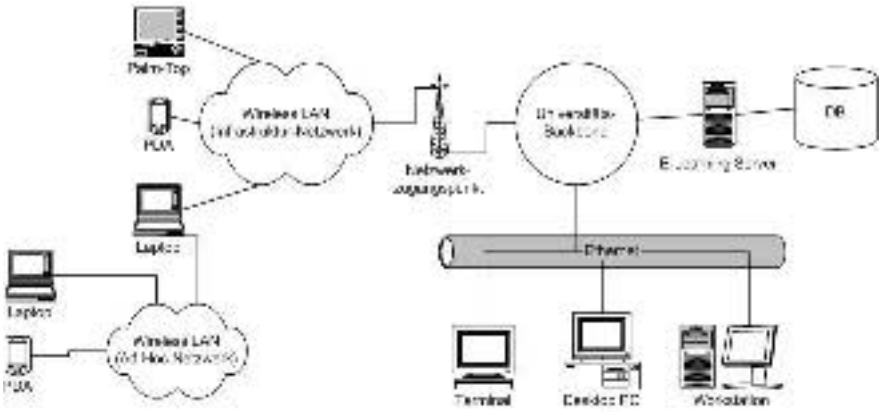

Abbildung 2: Das E-Learning Szenario

das Display sowie das verwendete Kommunikationsmedium und die entsprechenden Protokolle. In diesem Zusammenhang sind auch Ereignisse wie Schwankungen der Bandbreite, häufige Verbindungsabbrüche und Positionswechsel der mobilen Geräte zu beachten. Diese gehören eher zum normalen Verhalten von mobilen verteilten Systemen als das sie als Fehlverhalten interpretiert werden. Die Middleware und insbesondere das Kommunikationprotokoll müssen auf diese Besonderheiten zugeschnitten sein.

3 Zielsysteme

Verschiedene Zielplattformen und Betriebssysteme sind derzeit sehr vielversprechend (Tabelle 1) und könnten in dem beschriebenen E-Learning Szenario eingesetzt werden. Desk-

Endgerät	**Plattform**	**Betriebssystem**
Mobil	ARM, XScale	Microsoft PocketPC 2002, Embedix Linux (Kernel 2.4)
Stationär	Desktop PC, Workstation	Microsoft Windows, Linux (Kernel 2.4), UNIX

Tabelle 1: Überblick über die Zielplattformen

toprechner besitzen heutzutage meist Prozessoren der x86 Familie oder kompatible Modelle. Gängige weitverbreitete Betriebssysteme sind Microsoft Windows und Linux. Im Bereich der PDAs existieren mehrere Varianten. Das Betriebssystem Palm OS ist derzeit am meisten verbreitet, und es existieren Implementierungen für verschiedene Prozessoren. Innerhalb dieses Papiers werden Palm OS Systeme nicht als potentielle Zielsysteme

berücksichtigt, da derzeitige Endgeräte für geplante Anwendungen nicht über genügend Leistungsvermögen und freie Ressourcen verfügen. Aktuelle Palm PDAs verfügen über bis zu 16 MB RAM und Prozessoren mit 33 MHz - 144 MHz. Es stehen in der Regel keine Compact Flash Slots zur Verfügung. Weiterhin ist die Programmierschnittstelle und die angebotenen Betriebssystemfunktionen für geplante Anwendungen sowie für den Einsatz mobiler Informationssysteme, wie das E-Learning Szenario nicht ausreichend. Durch die fehlenden Erweiterungsmöglichkeiten ist die Flexibilität der Geräte außerdem sehr eingeschränkt.

Bei einer weiteren Gruppe von PDA-Systemen werden ARM-Prozessoren und XScale-Prozessoren eingesetzt, und dabei dienen derzeit Embedix Linux oder Microsoft Pocket PC 2002 als Betriebssysteme. Diese Systemkonfigurationen sind derzeit sehr vielversprechend und zeichnen sich durch ein hohes Leistungsvermögen aus. Aktuelle Modelle haben bis zu 64 MB RAM und bis zu 48 MB ROM. Die Prozessoren arbeiten dabei mit bis zu 400 MHz. Sie sind erweiterbar durch SD Slots und CF Slots (z. B. IBM Microdrive 1 GB). Dadurch ist es möglich die Systeme an die Anwendungsfälle anzupassen. Hier sind neben Sekundärspeicher auch Erweiterungen der Kommunikationsfähigkeit gemeint (GSM/GPRS, GPS, WLAN, ...). Ein Vorteil ist, dass verschiedenste Bibliotheken (z. B. QT embedded, MFC, ODBC, ...) auf diesen Systemen zur Verfügung stehen. Weiterhin wird für jedes System eine JVM[2] angeboten. Diese neue Klasse von Geräten ermöglicht die Entwicklung von einer neuen Klasse von mobilen Anwendungen. Es ist möglich nunmehr komplexere Anwendungen auf diesen Geräten zu betreiben. Aus diesen Gründen eigenen sich genannte Geräte als Zielsysteme um komplexe Anwendungen, wie z. B. mobile DBMS oder Multimedia-Anwendungen, zu beherbergen und auszuführen. Alle weiteren nicht erwähnten Systeme, welche weniger Leistungsvermögen und Ressourcen zu Verfügung haben, werden nicht weiter betrachtet.

4 Vorhandene Softwarelösungen

Für die ausgewählten Zielsysteme existiert eine Vielzahl von Softwarelösungen zur Entwicklung von verteilten Systemen. Entfernte Funktions- bzw. Objektaufrufe sind hier die zugrundeliegende Technik. Diverse Klassen- oder Funktionsbibliotheken sind verfügbar, welche eine Programmierschnittstelle zur Implementierung von entfernten Aufrufen erleichtern. Teilweise kommen Generatoren zum Einsatz, welche anhand einer Beschreibungssprache die Kommunikationsbasis eines verteilten Systems fertigen. Wichtige interne Strategien sind die Interprozesskommunikation, verschiedene Synchronisationsmechanismen, Nebenläufigkeit sowie Konsistenzwahrung. Alle diese Aspekte sind für den Nutzer der entfernten Aufrufe im allgemeinen transparent. Dennoch gibt es zum Teil verschiedene Möglichkeiten, interne Strategien wie z. B. den Synchronisationsmechanismus auszuwählen.

[2] Java Virtual Machine

4.1 RPC-Systeme

RPC[3]-Systeme bieten die Möglichkeit zum Entwurf und zur Implementierung von klassischen verteilten Systemen. Ihnen liegt das *request/response*-Kommunikationsmodell zugrunde. Parameter werden an die Zielfunktion übertragen und nach der Abarbeitung wird das Ergebnis zurückübermittelt. Die Kommunikation ist synchron und bidirektional. Dabei existiert nur ein *unicast*-Mechanismus wobei immer genau ein Sender mit einem Empfänger kommuniziert. Weitverbreitet sind z. B. CORBA [Vos97], DCOM [BK98], SUN-RPC [Sri95] und RMI [Eck00]. Werden diese Systeme im mobilen Kontext eingesetzt, treten Probleme auf. Die Systeme enthalten sehr viel Funktionalität und sind sehr mächtig. Viele Features werden jedoch nicht im mobilen Bereich benötigt. Die Ressourcenknappheit und damit mögliche Leistungseinbußen bei mobilen Geräten spielen hier eine große Rolle. Weiterhin ist Portabilität und Interoperabilität gerade in heterogenen Netzwerken enorm wichtig. Sie ist bei den genannten Systemen nicht immer gewährleistet. Die genutzten *Low Level*-Kommunikationsprotokolle tauschen Informationen in binärer Form aus. Sie sind nicht zu jeweils anderen Protokollen kompatibel und es existieren nicht immer offene Standards. Es sind zwar Implementierungen für verschiedene Plattformen verfügbar, aber wenn beispielsweise in einem Knoten im verteilten System DCOM genutzt wird, so müssen alle anderen Knoten mit Windows laufen. Wird CORBA genutzt, so muss jede Anwendung den selben ORB[4] nutzen. Es gibt zwar Fälle in denen CORBA beispielsweise mit DCOM zusammenarbeitet, aber diese Interoperabilität lässt sich nicht auf *High Level*-Dienste wie Transaktionsverwaltung, Authentifizierungsmechanismen oder Naming übertragen. Durch die verbindungsorientierte Übertragung kommt es bei hoher Netzlast und Bandbreitenschwankungen zu Problemen. In Mobilfunknetzwerken tritt dies häufig auf [Rot02] und ist damit nicht zu vernachlässigen. Im Zielszenario sind viele mobile Endgeräte untereinander und mit stationären Geräten im verteilten System gekoppelt. Da bei herkömmlichen für den stationären Bereich konzipierten RPC-Systemen eine starke Bindung zwischen Sender und Empfänger besteht, kann nur schlecht bzw. gar nicht auf teilweise häufige Standortwechsel mobiler Endgeräte reagiert werden. Wie im Abschnitt 2 vorgestellt, spielt die Mobilität der Endgeräte eine wichtige Rolle und sollte daher auch unterstützt werden. Weitherhin birgt der Einsatz von Firewalls und Proxies im verteilten System Probleme, da evtl. wichtige Ports gesperrt sind. Dies behindert einen reibungslosen globalen Einsatz von klassischen RPC-Systemen in massiv heterogenen Netzwerken.

4.2 XML-Nachrichtensysteme

XML[5]-Nachrichtensysteme wie SOAP [Box00] und XMLRPC [Win99] kommunizieren via XML-Nachrichten. Ähnlich wie in RPC-Systemen werden Parameter und Resultate zwischen aufrufender Funktion und Zielfunktion übertragen. Allerdings gibt es meh-

[3] Remote Procedure Call
[4] Object Request Broker
[5] eXtensible Markup Language

rere Interaktionsmuster (1:1, 1:n, n:m) zwischen Sendern und Empfängern[6]. Ein oder mehrere Sender können mit ein oder mehreren Empfängern Nachrichten austauschen. Die Verbindung muss nicht zwingend bidirektional sein. Weiterhin gibt es mehrere Sychronsiationsmethoden. RPC-Systeme können hinsichtlich dieser Funktionalität als echte Untermenge der XML-Nachrichtensysteme bezeichnet werden. Im allgemeinen sind XML-Nachrichtensysteme sehr schlank und leichtgewichtig. Weiterhin sind sie portabel, interoperabel, Plattform- und Programmiersprachunabhängig konzipiert. Das Transportprotokoll ist frei wählbar. Dies erhöht die Flexibilität. Die Kommunikation erfolgt verbindunglos und nachrichtenorientiert. Diese Eigenschaften kommen Problemen wie temporären Verbindungsunterbrechungen und Bandbreitenschwankungen sowie häufigen Standortwechseln entgegen. Im Allgemeinen ist die Kommunikation mit den unterschiedlichen Synchronisationsmechanismen und den verschiedenen Interaktionsmustern sehr flexibel. In Bezug auf die vorgestellte E-Learning Anwendungen ist eine flexible Kommunikation sehr wichtig. Die verschiedenen Iteraktionsmuster der Kommunikation werden in unterschiedlichen Fällen genutzt. Ein Broadcast oder Multicast-Mechanismus ist gerade bei Gruppenkommunication in virtuelle Lerngruppen von enormen Vorteil.

Es existieren aber auch Probleme beim Einsatz von XML-Nachrichtensystemen. Da sehr oft HTTP [FGM+97] als Transportprotokoll eingesetzt wird und Port 80 für die Kommunikationsverbindung genutzt wird, ist es sehr einfach Sicherheitsbarrieren von Firewalls zu umgehen. Da die Nachrichten oftmals in XML-Klartext übertragen werden, ist auch die Möglichkeit des Mithörens Dritter gegeben.

4.3 RPC-Systeme versus XML-Nachrichtensysteme

In Tabelle 2 sind einige Informationen zu RPC-Systemen und XML-Nachrichtensystemen zusammengetragen. Anhand der Tabelle wird deutlich, dass die XML-Nachrichtensysteme gegenüber den klassischen RPC-Systemen hinsichtlich des Einsatzes im mobilen Bereich einige Vorteile haben. Gerade die verbindungslose, nachrichtenorientierte, quasi Proxyfreundliche Kommunikation ist ein wesentlicher Vorteil. Diese Kommunikationeigenschaften sind sehr gut für den Einsatz in der Domäne der potentiellen Zielszenarien geeignet. Ein weiterer wichtiger Grund für den Einsatz der XML-Nachrichtenssysteme ist die, gegenüber den klassischen RPC-Systemen, schlanke und leichtgewichtige Implementierung. Dies ist besonders wichtig beim Einsatz auf den teilweise extrem resourcenbeschränkten mobilen Geräten. Nicht zuletzt spielt auch die Unabhängigkeit von Plattform, Betriebssystem und Programmiersprache gerade im massiv heterogenen Netzwerk eine entscheidene Rolle. Alle diese Gründe veranlassen dazu XML-Nachrichtensysteme zur Realisierung einer Middleware für mobile Informationssysteme zu nutzen.

[6] Mehrere mögliche Interaktionsmuster sind nur beim SOAP-Protkoll vorhanden.

	RPC-Systeme	XML-Nachrichtensysteme
Vertreter	CORBA, Sun-RPC, DCOM, RMI	XMLRPC, SOAP
allgemein	∗ Hoher Funktionsumfang ∗ Sehr mächtig ∗ Weit verbreitet	∗ Schlank, leichtgewichtig ∗ Portabel, Interoperabel ∗ Plattform-, Sprachunabhängig ∗ Transportprotokoll frsei wählbar
Kommunikation	∗ Verbindungsorientiert ∗ Synchron ∗ Bidirektional ∗ 1:1 Interaktion	∗ Verbindungslos ∗ Nachrichtenorientiert ∗ Synchron & Asynchron ∗ Bidirektional & Unidirektional ∗ 1:1, 1:n, n:m (SOAP)
Übertragungsformat	∗ binär	∗ XML
Probleme	∗ Mangelnde Interoperabilität ∗ Standortwechsel ∗ Einsatz von Proxies & Firewalls ∗ Große Datenmengen	∗ Sicherheitsprobleme (Port 80) ∗ Evtl. Klartextübertragung ∗ Nachteil bei Performance

Tabelle 2: Technologievergleich

4.4 XMLRPC versus SOAP

XMLRPC und SOAP sind zwei bereits relativ weit verbreitete XML-Nachrichtensysteme. Trotz Gemeinsamkeiten, bestehen dennoch wichtige Unterschiede. XMLRPC ist ein reines *request/response*-Protokoll, d. h. die Kommunikation wird immer von einem Client initiiert. Der Server wartet auf solche Anfragen und antwortet. Durch die sehr geringe Komplexität ist das XMLRPC-Protokoll leicht zu erlernen und zu bedienen. Es existieren viele Implementierungen, die sehr schlank entworfen und realisiert sind. Allerdings können nur Grunddatentypen wie Skalare, Felder und namenlose Strukturen übertragen werden. Das Protokoll ist dahingehend nicht erweiterbar und wenig flexibel. Derzeit existiert keine Möglichkeit Transaktionen zu nutzen, Daten zu verschlüsseln oder Authentifizierungsmechanismen einzubeziehen. Diese komplexeren Mechanismen werden jedoch in Szenarion wie der E-Learning Anwendung dringend benötigt.

SOAP ist ebenfalls ein schlankes System. Neben dem *request/response*-Mechanismus wird auch *one-way*- und *multicast*- Kommunikation unterstützt. Im Gegensatz zu XMLRPC können getypte Parameter anhand ihres Namens übertragen werden. Neben den Grunddatentypen können anwenderspezifische Datentypen und Namespaces frei definiert werden. Das Protokoll ist erweiterbar und anpassbar. Für die Kommunikation sind mehrere Parameter wählbar, wie das darunterliegende Transportprotokoll oder der Kodierungsstandard. Weiterhin existiert die Möglichkeit das Protokoll um Transaktions- Authentifizierungs- und Verschlüsslungsmechanismen zu erweitern. Ein Vorteil ist auch, dass derzeit verschiedene Adapter zu anderen Protokollen wie DCOM oder CORBA verfügbar oder in Entwicklung sind. Ein weiterer Grund SOAP zu nutzen ist die mögliche Anbindung an die heutzutage immer beliebteren Web Service Plattformen [AAB+02].

	XMLRPC	SOAP
Vorteile	∗ Geringe Komplexität ∗ Effizienz ∗ Hohe Geschwindigkeit ∗ Große Bandbreite ∗ Einfache Wartung & Fehlersuche	∗ Flexible Nachrichtendienste (1:1,1:n,n:m) ∗ Umfassende Beschr. der Kommunikation ∗ Anwenderspezifische Datentypen ∗ Erweiterbares, Anpassbares Protokoll ∗ Transaktionsunterstützung ∗ Authentifizierungsmechanismen ∗ Verschlüsselung ∗ Anbindung an Web Services Plattformen
Nachteile	∗ Nur Grunddatentypen ∗ Nicht erweiterbar ∗ Wenig flexibel ∗ 1:1 Interaktion (request/response) ∗ Keine Transaktionen ∗ Keine Authentifizierung ∗ Keine Verschlüsselung	∗ Höhere Komplexität ∗ Gewisser Overhead ∗ Schwer erlernbar ∗ Durchsetzung des Standards (W_3C)

Tabelle 3: Vorteile von SOAP und XMLRPC

Da die Middleware möglichst flexibel, erweiterbar, skalierbar und komfortabel gestaltet sein soll, ergibt sich aus Tabelle 3, dass sich der Einsatz von SOAP anbietet. Gerade hinsichtlich der Realisierung von komplexen Anwendungen sowie des massiven Einsatzes von softwaretechnischen Strategien zur Konfigurierbarkeit und Erweiterbarkeit sind die Eigenschaften von SOAP von großem Nutzen. SOAP ist modular aufgebaut, leicht erweiterbar und es ermöglicht eine umfassende Beschreibung der Daten und der Kommunikation. Von vielen Anwendungen benötigte Mechanismen, wie Verschlüsselung von Daten, Transaktionsverwaltung oder sichere Authentifizierung können innerhalb von SOAP realisiert werden. Eine weitere Beispielanwendung ist ein Online-Parkscheinautomat. Mittels mobiler Endgeräte könnte es ermöglicht werden Parkscheine innerhalb einer Stadt online anzufordern und auch gleich zu bezahlen. Die Anwendung benötigt die genannten Mechanismen, welche SOAP bereitstellt. Neue komplexere Mechanismen können dem Protokoll einfach hinzugefügt werden. Aufgrund der vielfältigen Vorteile werden die Nachteile bei Effizienz oder Wartbarkeit gegenüber XMLRPC in Kauf genommen.

Als gewünschte Programmierschnittstellen wurden C++ und Java ausgewählt, da sie sich hervorragend für die Realisierung von großen Softwareprojekten eignen. Das umgesetzte objektorientierte Paradigma ist dabei sehr hilfreich und erlaubt modular, skalierbar, erweiterbar und komfortabel zu programmieren. Außerdem existieren viele nützliche Bibliotheken und Schnittstellen, wie Datenbankanbindungen oder Bibliotheken für Standardprobleme sowie nützliche Algorithmen.

5 Testszenario

In diesem Abschnitt wird ein Testszenario betrachtet. Zuerst werden die Endgeräte ausgewählt und eine Beispielanwendung vorgestellt. Die Beispielanwendung ist sehr einfach gehalten. Dennoch sind die Ergebnisse in Hinsicht Realisierbarkeit und in Hinsicht der Evaluierung eingesetzter Software und Hardware verwendbar. Die Ergebnisse und gewonnene Erkenntnisse werden im weiteren diskutiert.

5.1 Endgeräte

Als Testgeräte dienen verschiedene PDAs und Desktoprechner. Sie wurden anhand der in Abschnitt 3 festgelegten Zielsysteme ausgewählt. Als Kommunikationsmedium dient Wireless LAN [CWKS97]. Prinzipiell wären auch andere Übertragungsstandards möglich. In Tabelle 4 sind die ausgewählten Testgeräte zu finden. Sie entsprechen den Vorgaben, die durch die Definition der Zielsysteme gegeben sind.

Endgerät	**Plattform**	**Betriebssystem**
Toshiba e740 WiFi	Intel XScale	Microsoft PocketPC 2002
Sharp Zaurus SL-5500G	ARM	Embedix Linux (Kernel 2.4)
Desktopsysteme	x86	Microsoft Windows 2000, Linux (Kernel 2.4)

Tabelle 4: Überblick über die ausgewählten Testsysteme

Jedes der Testgeräte wird die Rolle des Clients sowie des Servers übernehmen. Die Abbildungen 3 und 4 enthalten die möglichen Konstellationen. Es ist zu beachten, dass im Fall der Kommunikation zweier PDAs ein *ad-hoc*-Netzwerk und im Fall der Kommunikation zwischen PDA und Desktoprechner ein Infrastruktur-Netzwerk aufgebaut wird. Beim Infrastruktur-Netzwerk wird ein *Wireless Access Point* als Kopplung zwischen WLAN und LAN genutzt.

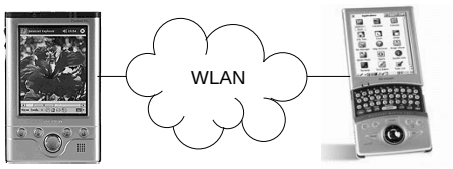

Abbildung 3: *Ad-hoc*-Netzwerk zwischen PDAs

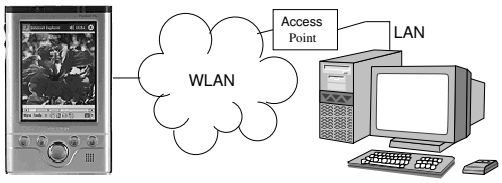

Abbildung 4: Infrastruktur-Netzwerk zwischen PDA und Desktoprechner

5.2 Beispielanwendung - Der Warenhaus-Dienst

Da die Realisierung des in Abschnitt 2 vorgestellten Szenarios zu komplex ist wurden zur Evaluierung der Technologien ein anderes Szenario gewählt.

Als Beispielanwendung wird ein elektronisches Warenhaus implementiert. Diese ist eine einfache Anwendung und hat in der implementierten Form keine praktische Relevanz. Dennoch eignet sich diese Anwendung, um Aussagen für praktisch relevante Anwendungen hinsichtlich der Realisierbarkeit zu machen. Durch die geringe Komplexität der Beispielanwendung können keine verlässlichen Aussagen über die Performance getätigt werden. Die Beispielanwendung kann aber als Grundlage für reale Anwendungen dienen.

Das elektronische Warenhaus verwaltet in einer Datenbank verschiedenste Artikel. Es werden die eindeutige Artikelnummer und die vorhandene Stückzahl verwaltet. Der Warenhaus-Dienst wartet auf Anfragen von Klienten. Der Klient übergibt bei einer Anfrage die Artikel-Nummer und die gewünschte Anzahl des Artikels. Der Warenhaus-Dienst sucht in der Datenbank Informationen zum angeforderten Artikel und anhand dieser wird der Lieferstatus (komplett verfügbar / partiell verfügbar / nicht verfügbar) sowie die Anzahl des verfügbaren Artikels als Resultat zurückgegeben.

Der Warenhaus-Dienst soll als nebenläufiger Server implementiert werden und an einem festgelegten Port lauschen. Der Dienst wird als eine Funktion implementiert, d. h. Anfragen werden mittels entferntem Funktionsaufruf getätigt.

5.3 Ergebnisse

Die gewählte Sprachschnittstelle und gleichzeitig die Programmiersprache für die Testanwendung ist vorerst C++. Der Warenhaus-Dienst wurde mit mehreren C++ basierten SOAP-Implementierungen getestet[7]. Allerdings ließen sich nicht alle SOAP-Implementierungen auf allen gewünschten Testplattformen nutzen. In Tabelle 5 sind alle getesteten Implementierungen dargestellt. Neben den unterstützten Plattformen sind verschiedene Anmerkungen aufgeführt. Diese beziehen sich in erster Linie auf Probleme mit den

[7] Es wurden nur C++ Implementierungen getestet. Ein Test von Java Implementierungen sowie ein Vergleich mit den C++ Implementierungen steht noch aus und konnte deswegen nicht mit in dieses Papier genommen werden (siehe Abschnitt 6).

Name	Win 2000	Pocket PC	Linux	Anmerkungen
eSOAP	√	-	√	nutzt Standard C-Bibliothek
PocketSOAP	√	√	-	COM/DCOM-Anbindung
gSOAP	√	√	√	Stub- und Skeletoncompiler
Ms SOAP Toolkit	√	-	-	nutzt ATL, COM
XSOAP++	√	-	√	nutzt Ausnahmen
easySOAP++	√	-	√	nutzt Standard C-Bibliothek
WhiteMesa	√	-	-	nutzt COM
SimpleSOAP	√	-	-	nur Visual C++
WASP-C++	√	√	√	komplex (≈300 Klassen)

Tabelle 5: C++-basierte SOAP-Implementierungen (einzelne Implementierungen sind unter `http://www.soapware.org/` zu finden)

einzelnen Testplattformen.

Zwischen einzelnen SOAP-Implementierungen existieren erhebliche Unterschiede (Tabelle 5) bzgl. der Unterstützung einzelner Plattformen. Die meisten Probleme gibt es bei Geräten, welche das Betriebssystem Microsoft Pocket PC 2002 nutzen. Die verschiedenen Implementierungen konnten nicht unter Pocket PC 2002 übersetzt werden, da keine vollständige C++ Standard Template Library (STL) verfügbar ist. Die STL nutzt unter anderem massiv Ausnahmen und Ausnahmebehandlungen, welche nicht von Microsoft Pocket PC unterstützt werden. Es gibt zwar einige unvollständige Versionen[8], die aber wichtige, von den SOAP-Implementierungen benötigte Funktionen nicht enthalten. Werden Ausnahmen verwendet, ist die Anwendung meist nicht möglich. Weiterhin sind nicht alle benötigten C-Headerdateien bzw. die nötigen Bibliotheksdateien vorhanden (Bsp. `time.h`). Durch die unvollständige STL und die fehlenden C-Header fallen eine große Zahl von Implementierungen heraus und können somit nicht genutzt werden. Ein weiteres Problem ist die Nutzung bzw. Anbindung von COM-Komponenten, der Active Template Library (ATL) oder anderen windowsspezifischen Headerdateien und Bibliotheken. Da diese nicht für Linuxsysteme verfügbar sind, fallen die betreffenden SOAP-Implementierungen heraus. Nur die Implementierungen *WASP-C++*[9] und *gSOAP*[10] waren auf allen Plattformen verfügbar bzw. ließen sich für diese übersetzen. WASP-C++ bietet einen nebenläufigen Server und eine Klassenbibliothek für die Klienten [Rot02]. Weiterhin stehen einige WSDL-Werkzeuge zur Beschreibung der Dienste zur Verfügung [CCMW01]. Das Problem an WASP-C++ ist die hohe Komplexität. Es existieren ca. 300 C++ Klassen für den Anwendungsprogrammierer. Große Teile werden zur Implementierung der Middleware-Plattform nicht benötigt und verkomplizieren die Nutzung unnötig. gSOAP ist im Gegensatz zu WASP-C++ keine reine Programmierschnittstelle [Rot02]. gSOAP liefert einige komfortable Generatorwerkzeuge. Der Generator erhält als Eingabe eine C++ Headerdatei, welche die Schnittstelle des Dienstes beschreibt. Daraus werden *client stubs* und *service skeletons* erzeugt. Der Anwender verwendet diese zur Implementierung von Klient und Dienst. Der generierte Quelltext ist sehr effizient. Es wird ein moderner *XML Pull Parser* verwendet, so dass im Gegensatz zu einem *DOM Parser* kein Speicherplatz

[8] http://users.libero.it/g.govi/index.html
[9] http://www.systinet.com/
[10] http://gsoap2.sourceforge.net/

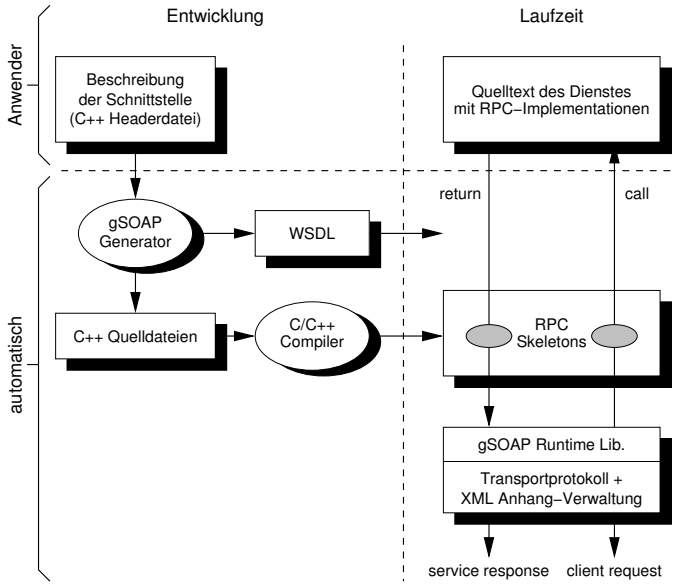

Abbildung 5: Aufbau und Funktion eines gSOAP-Dienstes

verschwendet wird. Der erzeugte Dienst kann iterativ oder nebenläufig arbeiten und als *standalone* oder CGI basierter Server in Betrieb genommen werden.

Wegen der Portabilität, des Komforts und der Effizienz bietet sich die gSOAP-Implementierung als Grundlage der Middleware-Plattform an. In Abbildung 5 ist der Aufbau eines Dienstes dargestellt. Es sind die Teile, die der Anwendungsentwickler und die der Generator erzeugen, zu erkennen. Aus der C++ Schnittstellenbeschreibung erzeugt der Generator eine WSDL Beschreibung und C++ Quelltext. Der erzeugte Quelltext wird vom Anwender mit Funktionalität angereichert. Die erzeugten *Skeletons* (de-)serialisieren übergebene Parameter und nutzen die gSOAP Runtime Library zur Kommunikation mit den Klienten.

Im Falle eines Klienten (Abb. 6) wird anhand einer Dienstbeschreibung (WSDL) C++ Quelltext generiert. Die vom C++ Compiler erzeugten Stub-Routinen werden dann vom Klienten zur Kommunikation genutzt und serialisieren bzw. deserialisieren die übergebenen Parameter. Die Stub-Routinen nutzen ihrerseits die Funktionen der Runtime Library und treten so mit dem Dienst in Verbindung.

So komfortabel ein Generator auch sein mag, so existiert doch ein Nachteil. Da aus einer Schnittstellenbeschreibung der Dienst und der Klient erzeugt werden, ist es schwer auf diesen im Sinne von Programmfamilien [Par79] aufzubauen. Jede Änderung der Schnittstelle würde eine Änderung in der Hierarchie darauf aufbauender Funktionen hervorrufen. So ist es schwer, ein möglichst erweiterbares, skalierbares und wiederverwendbares System zu konstruieren.

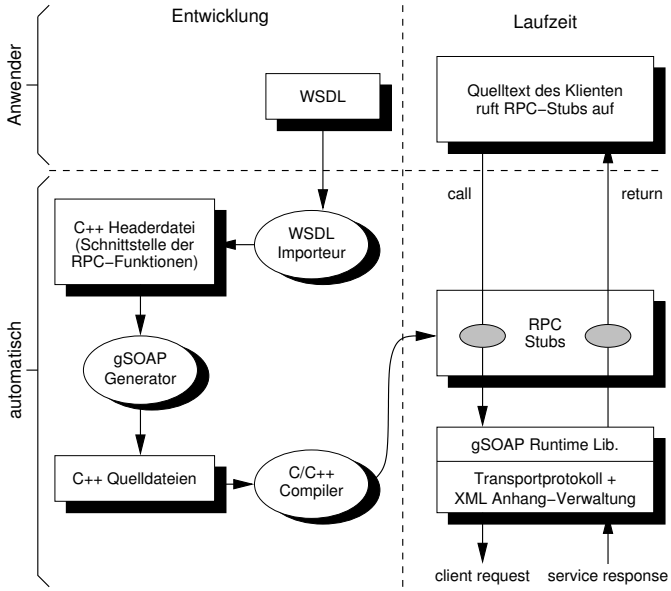

Abbildung 6: Aufbau und Funktion eines gSOAP-Klienten

6 Zusammenfassung und Ausblick

Es existieren verschiedene Technologien zum Entwerfen und Implementieren von verteilten Systemen. Die Gruppe der klassischen RPC-Systeme eignet sich nicht für die massiv heterogenen, hoch dynamischen Netzwerke, die entstehen, indem mobile Endgeräte integriert werden. Die Gruppe der XML-Nachrichtensysteme eignet sich hier besser, da sie sich durch ihre Portabilität und die schlanke Implementierung auszeichnet. Sie sind prädestiniert für den Einsatz in verteilten Systemen aus mobilen Endgeräten. SOAP ist ein schlankes, komfortables und erweiterbares Protokoll und ist in einigen wichtigen Punkten XMLRPC überlegen. Der Vergleich einiger C++ basierter SOAP-Implementierungen ergab, dass nur zwei SOAP-Implementierungen für alle Testplattformen verfügbar sind. Als geeignete Variante stellte sich gSOAP heraus.

Als nächster Schritt steht ein Überblick und der Vergleich mehrerer Java-basierter Implementierungen sowie die Realisierung der hier besprochenden Beispielanwendung an. Anhand der Beispielanwendung wird entschieden, welche Java-basierten Implementierungen sich eignen. Es bleibt zu prüfen, welchen Funktionsumfang die speziellen virtuellen Maschinen auf den Zielsystemen haben, und ob dieser den Ansprüchen der SOAP-Implementierungen genügt.

Dann werden alle geeigneten C++-basierten und Java basierten Implementierungen hinsichtlich Komfort, Performance, Interoperabilität und Portabilität verglichen. Auch die zur Verfügung stehenden Bibliotheken und Schnittstellen werden mit einbezogen. Anhand die-

ser Ergebnisse ist es möglich eine Auswahl der zu nutzenden Programmiersprache zur Implementierung einer Middleware-Plattform für mobile verteilte Informationssysteme und Anwendungen zu treffen.

Literatur

[AAB+02] K. Apshankar, D. Ayala, C. Browne, V. Chopra, T. McAllister, and P. Sarang. *Professional Open Source Web Services*. Wrox Press Ltd, 2002.

[BK98] N. Brown and C. Kindel. Distributed Component Object Model protocol: DCOM/1.0, January 1998.

[Box00] D. Box. Simple Object Access Protocol (SOAP) 1.1, W_3C Note 08 May 2000.

[CCMW01] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web Services Description Language (WSDL) 1.1, W_3C Note 15 March 2001.

[CWKS97] B. P. Crow, I. Widjaja, J. G. Kim, and P. T. Sakai. IEEE 802.11: Wireless Local Area Networks. *IEEE Communications Magazine*, 35(9):116–126, 09 1997.

[Eck00] Bruce Eckel. *Thinking in JAVA*. Prentice Hall, 2nd edition, 2000.

[FGM+97] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, and T. Berners-Lee. IETF RFC 2068: Hypertext Transfer Protocol – HTTP/1.1, January 1997. Available at: http://www.ietf.org/rfc/rfc2068.txt (Informational).

[Par79] D. L. Parnas. Designing Software for Ease of Extension and Contraction. *IEEE Transaction on Software Engineering*, SE-5(2), 1979.

[Rot02] J. Roth. *Mobile Computing: Grundlagen, Technik, Konzepte*. dpunkt-Verlag, 1. edition, 2002.

[Sri95] R. Srinivasan. RFC 1831: RPC: Remote Procedure Call Protocol Specification Version 2, August 1995. Status: PROPOSED STANDARD., ftp://ftp.internic.net/rfc/rfc1831.txt, ftp://ftp.math.utah.edu/pub/rfc/rfc1831.txt.

[Vos97] G. Vossen. The CORBA Specification for Cooperation in Heterogeneous Information Systems. In P. Kandzia and M. Klusch, editors, *Proceedings of the First International Workshop on Cooperative Information Agents*, volume 1202 of *LNAI*, pages 101–115, Berlin, February 26–28 1997. Springer.

[Wei93] M. Weiser. Hot Topics: Ubiquitous Computing. *IEEE Computer*, October 1993.

[Win99] D. Winer. XML-RPC Specification, 1999. http://www.xmlrpc.com/spec.