Universität-Gesamthochschule Paderborn

# Projekt NT 2: XML/DXL

# Gruppenkalenderapplikation als Prototyp zur Verwendung des Lotus XML-Toolkit

Office Systeme 2

Prof. Dr. L. Nastansky

Sommersemester 2003

vorgelegt von:

Alexander Lindhorst

Wirtschaftsinformatik

Markus Steckenborn

Wirtschaftsinformatik

## Inhaltsverzeichnis

# 1. Einleitung

Untersuchungsgegenstand des vorliegenden Projektes ist das Lotus XML Toolkit. Dieses wurde bereits evaluiert (vgl. [Lindhorst/Steckenborn 2003]) und soll mit einer prototypischen Applikation eines Gruppenkalenders an Lotus Notes/Domino in der Praxis getestet werden.
Zunächst werden einige grundlegende Aussagen zu der Funktionsweise eines Gruppenkalenders gemacht. Dann führt eine ausführliche Diskussion der Architektur hin zur Beschreibung der implementierten Komponenten.
Die während der Erstellung dieser Arbeit aufgetretenen Probleme führten dazu, dass der geplante Funktionsumfang eines vollständig funktionierenden mobilen Gruppenkalenders reduziert wurde zu einem Proof of Concept.
Installationsbeschreibung, Fehlerbeschreibungen sowie eine Spezifikation der entwickelten Schnittstellen finden sich im Anhang.

# 2. Konzepte

Zur Erstellung einer Gruppenkalenderapplikation werden zunächst grundlegende Konzepte zur Vereinbarung von Terminen, bei denen mehrere Personen beteiligt sind, beschrieben. Dann wird gezeigt, wie dieses bei Lotus Notes/Domino umgesetzt ist.

## *2.1. Terminvereinbarung im Allgemeinen*

Die Vereinbarung von Terminen zwischen mehreren Personen gestaltet sich als ein sehr kompliziertes Verfahren, soweit sich diese Personen nicht zur gleichen Zeit am gleichen Ort befinden. Es muss dann eine Zeit gefunden werden, an der alle Teilnehmer keine weiteren Termine bereits geplant haben. Anschließend wird eine Information an alle Teilnehmer als Einladung gegeben, die dann entweder bestätigt oder abgelehnt wird. Im Falle, dass alle Teilnehmer bestätigt haben, kommt der Termin zu der angegebenen Zeit zu Stande. Ansonsten müssen Prozeduren durchgeführt werden, auf die sich die Teilnehmer im Vorfeld verständigt haben. Eine mögliche Vorgehensweise besteht darin, eine Ungültigkeitsmeldung für den bereits angekündigten Termin mit einem neuen Vorschlag zu schicken, und dann wieder Bestätigungen oder Ablehnungen abzuwarten.

## *2.2. „Scheduling a Meeting“ bei Notes/Domino*

Um die Prozesse bei einer verteilten, asynchronen Terminvereinbarung zu vereinfachen, ist ein Zugriff auf elektronische Terminkalender der gesamten Teilnehmer notwendig. In einer Notes/Domino-Umgebung ist dies möglich durch die Verwendung der persönlichen Maildatenbank der Notes-Benutzer.

Für Benutzer einer Notes-Maildatenbank ist die Vereinbarung von Terminen durch die Funktionalität des Gruppenkalenders bereits vereinfacht. Eine Anzeige mit den übrigen Teilnehmern zeigt die bereits als belegt gekennzeichneten Termine jeden einzelnen Benutzers an. Nachdem ein freier „Timeslot" gefunden wurde, kann ein Termin erstellt, die übrigen Mitglieder als Teilnehmer angegeben und automatisch per E-Mail eingeladen werden. Mit dieser E-Mail wird auch Logik in Form einen eingebetteten Programms geschickt, das zur entsprechend der Benutzer-Reaktion einen Termin in den Kalender einträgt oder eine „Decline"-Nachricht zurücksendet.
Eine identische Übersicht bietet die Funktion „Scheduler" beim Erstellen eines neuen Termins. Hierbei müssen die Mitglieder der Gruppen explizit angegeben werden (vgl. [LotusHilfe]).
Im Hintergrund läuft während dessen eine Anfrage an die Datenbank „Busytime", die Termine der gesamten Domain-Mitglieder zwischenspeichert und von hier aus Anfragen des „Schedulers" und des „Gruppenkalendars" beantwortet. Voraussetzung dafür ist, dass der Anfragende zumindest Leseberechtigung für die Maildatenbanken erhalten hat.
Im Rahmen dieser Arbeit wird die Funktion des Gruppenkalenders bzw. der Schedulers innerhalb der Maildatenbanken durch eine externe Anwendung ersetzt, die per mittels des XML Toolkits Zugriff auf die Notes/Domino Datenbanken erhält und analog der bewährten Prozesse vorgeht. Diese Anwendung wird im Folgenden als Prototyp bezeichnet.

# 3. Architektur des Prototypen

## *3.1. 3-Tier-Architektur*

Bei der Architektur des Prototypen handelt es sich um eine klassische 3-Tier-Architektur. Auf der ersten Ebene der Architektur befindet sich die Notes/Domino-Umgebung. Hier werden alle Daten und Datenbanken gemäß den Notes-spezifischen Vorgehensweisen gepflegt und vorgehalten. Diese Ebene bildet dadurch die Persistenz-Ebene der Anwendung, oftmals wird in einer solchen Konstellation diese Ebene auch als "Backend" bezeichnet.
Auf der untersten Ebene, dem dritten "Tier", steht der Client auf dem mobilen Endgerät; hierbei wird auch vom "Frontend" gesprochen. Dieser Teil der Applikation bildet die Schicht zur Interaktion mit dem Benutzer, sie stellt die Präsentationsebene her. Außer der Interaktion mit dem Benutzer selbst und den damit einhergehenden Plausibilitätsprüfungen bzw. der Ablaufsteuerung der Interaktion verfügt diese Ebene über keine tiefergehende Logik. Sie dient der Darstellung der bereits gespeicherten Daten (aus der Persistenzschicht) sowie zur Entgegennahme der Daten eines eventuellen neuen Kalendereintrags aus der "Feder" des Benutzers.

Die Logik zur Weitergabe der so erstellten bzw. manipulierten Daten an die Persistenzebene ist auf der mittleren Schicht, dem sogenannten Middle Tier, angesiedelt. In der 3-Tier-Architektur wird dies häufig auch als "Business Logic" bezeichnet, da diese Schicht dafür zuständig ist, die relativ bruchstückhaften Einzeldaten der Präsenzschicht unter Berücksichtigung logischer Einschränkungen und problembereichsbezogener Besonderheiten gegenüber der Persistenzschicht in einen Kontext zu bringen und so die eigentliche Dienstleistung erst zu ermöglichen. Dazu treten die Elemente dieser Schicht gegenüber der Präsentationsschicht als Server auf, sie stellen der dritten Ebene Services zur Verfügung, mit denen die Präsentationsschicht ihren Aufgaben nachkommen kann. Gleichzeitig tritt der Middle Tier gegenüber der Persistenzschicht als Client auf. Quasi als Stellvertreter der Präsentationsschicht fordert dieser Tier von der Persistenzschicht gespeicherte Informationen an. In dieser Funktion können die Elemente des Middle Tier als Stellvertreter, als sog. Proxy, bezeichnet werden.

Die einzelnen Schichten kommunizieren über festgelegte Protokolle miteinander. Dabei handelt es sich im Einzelnen bei der Kommunikation zwischen der Persistenzebene (Lotus Notes/Domino) und der Business Logic Ebene um das Lotus XML Toolkit und damit um das Lotus Notes-eigene Kommunikationsprotokoll. Hierbei wird jede Anfrage an das Backend durch eine Darstellung der persistenten Daten in der Domino XML Language (DXL) beantwortet. Aus dieser Antwort filtert die zweite Ebene die im Applikationskontext relevanten Daten und reicht diese weiter an die Präsentationsschicht. Umgekehrt nimmt die zweite Schicht Anfragen und Aufträge der Präsentationsschicht entgegen und generiert daraus nach Durchführung eventueller Änderungen Daten im DXL-Format, die zur Speicherung mit Hilfe des Lotus XML Toolkit an die Persistenzebene überreicht werden.

Für die Kommunikation mit Hilfe des Lotus XML Toolkits bedarf es auf der Business Logic Ebene einer Umgebung, die das Lotus XML Toolkit auch nutzen kann, also entweder einer C++- oder einer Java-Umgebung. Aus noch zu erläuternden Gründen wurde hier eine Java-Umgebung gewählt.

Ein Ziel des Projekts bestand darin, anders als die Lotus-eigene Lösung für mobile Endgeräte möglichst viele Plattformen zu unterstützen. Das "Write Once, Run Anwhere"-Paradigma von Java schien geeignet, diese Anforderung zu erfüllen. Zusätzlich steht mit der Java 2 Mobile Edition eine Technologie zur Verfügung, die mit sehr wenigen Einschränkungen auf Seiten des Quellcodes die spezifischen Anforderungen und Einschränkungen mobiler Endgeräte bedient respektive berücksichtigt. Daher wurde J2ME als technologische Grundlage für die Präsentationsschicht herangezogen.

Innerhalb der J2ME-Technologie werden verschiedene Leistungsklassen von mobilen Endgeräten zu Klassen ähnlicher Geräte zusammengefasst, die durch so genannte

Konfigurationen repräsentiert werden, in denen durch Profile eine feinere Gliederung vorgenommen wird. Für Geräte, die bezüglich Speicher- und Prozessorleistung stark begrenzt sind, aber dabei über irgendeine Form von Vernetzung verfügen wie z.B. Mobiltelefone, steht die Connected Limited Device Configuration (CLDC) mit dem Micro Information Devices Profile (MIDP) zur Verfügung. Dieses Profil stellt einige wenige, relativ beschränkte Elemente zur Interaktion mit dem Benutzer zur Verfügung, Interaktion mit anderen vernetzten Geräten ist auf wenige Protokolle, meistens das Hypertext Transfer Protocol (HTTP), beschränkt. Für leistungsfähigere Geräte mit mehr Speicher und Prozessorleistung kommt die Connected Device Configuration (CDC) zur Anwendung. Sie verfügt über drei Profile, die aufeinander aufbauen: Foundation Profile, Personal Basis Profile und Personal Profile. Alle drei Profile können optional mit Java Remote Method Invocation (RMI) in verteilte Anwendungen eingebettet werden, das Personal Profile unterstützt darüber hinaus das Java Abstract Windowing Toolkit (AWT) fast vollständig und erlaubt damit den Aufbau komplexer grafischer Oberflächen; in seinem Leistungsumfang ist das Personal Profile mit dem Java Development Kit (JDK) 1.1 vergleichbar (vgl. [PersonalProfile]).

Um bei der Umsetzung von komplexen Prozessen auf der graphischen Oberfläche möglichst keinen Beschränkungen unterworfen zu sein, wurde das Personal Profile als Zielumgebung auf den mobilen Endgeräten der Präsentationsschicht ausgewählt. Um ferner bei der Interaktion mit dem Middle-Tier möglichst transparent auf Services und Funktionen des Servers auf der mittleren Ebene zuzugreifen, wurde statt der Interaktion über datenbezogene Protokolle wie HTTP die Interaktion mit der funktionsbezogenen Remote Procedure Call Technologie (RPC) bevorzugt. Für Java-Anwendungen steht mit Java Remote Method Invocation (RMI) eine eigene Ausprägung dieser Technologie zur Verfügung, die – wie bereits erwähnt – auch von allen Profilen der CDC unterstützt wird.

Zur Nutzung dieser Technologie ist es erforderlich, dass die Teilnehmer an beiden "Enden" der Kommunikationsleitung Java-Objekte sind, daher wurde für die Umsetzung der Aufgaben des Middle Tiers ebenfalls eine Java-Lösung als RMI-Server konzipiert. Die genauen Aufgaben dieses RMI-Servers werden an späterer Stelle erläutert; hier sollte entweder eine Java 2 Standard Edition (J2SE) Umgebung mit zusätzlicher RMI-Registry oder sogar eine Java 2 Enterprise Edition (J2EE) Umgebung zum Einsatz kommen, die eine RMI-Registry zwingend enthält.

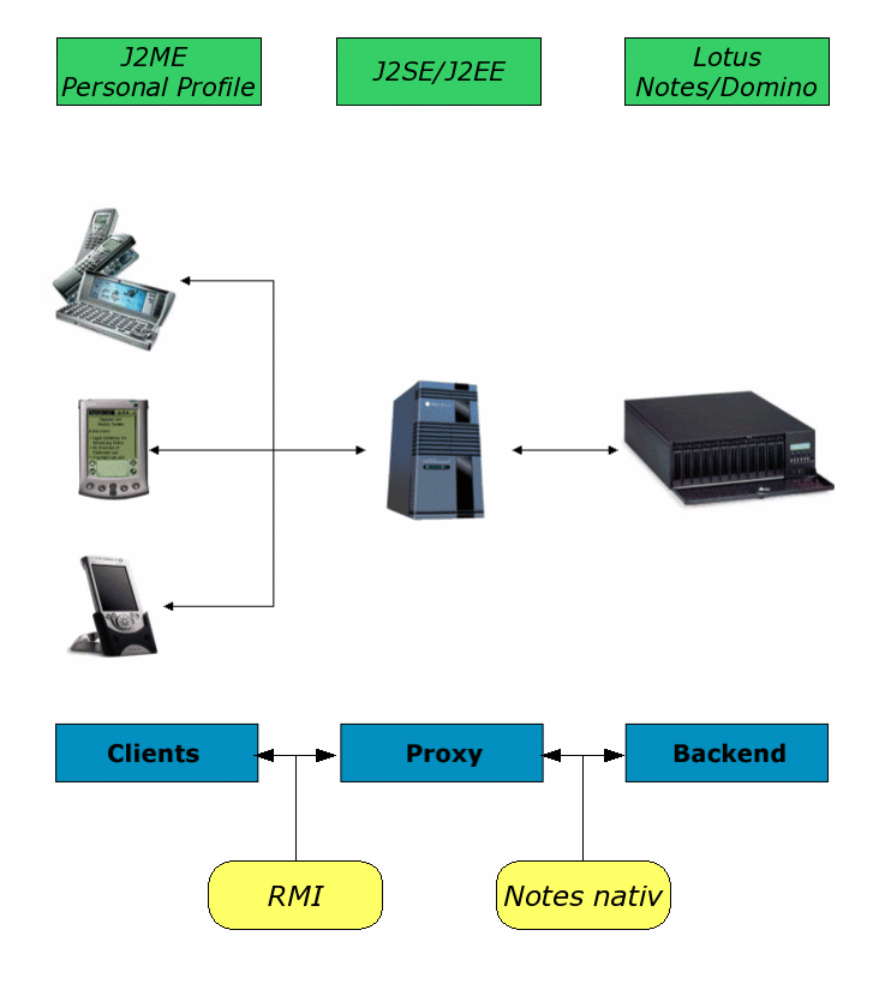



*Abbildung 1: Architektur des Prototypen*

Abbildung 1 stellt die beschriebene Architektur dar.

## 3.2. Vorteile und Einschränkungen

Die vorgestellte Lösung bietet einige Vorteile, die im Folgenden näher erläutert werden.

Ein besonderes Ziel besteht bei verteilten Anwendungen und insbesondere bei der Einbindung von mobilen Endgeräten mit im Vergleich zu „normalen" Desktoprechnern beschränkten Speicher- und CPU-Resourcen darin, die zur Aufgabenbewältigung benötigte Rechenleistung da zu allozieren, wo sie auch wirklich zur Verfügung steht, auch wenn das Ergebnis der Rechenschritte an einem anderen Ort präsentiert wird. Im vorliegenden Fall bedeutet das, dass insbesondere potenziell teure Vorgänge wie das Parsen der in XML/DXL dargestellten Informationen der Persistenzschicht nicht auf einem relativ schwachen PDA durchgeführt werden, auch wenn dieser die Ergebnisse dem Benutzer präsentiert. Stattdessen werden solche Vorgänge auf dem per Design kräftiger zu dimensionierendem Middle Tier durchgeführt und transparent per RMI dem Client auf dem PDA zur Verfügung gestellt in Form eines serialisierten Java-Objekts. Über diese Form der Aufbereitung von Daten in ein applikationsspezifisches Format hinaus werden beim Parsing der in DXL

vorliegenden Informationen auch noch eine Datenfilterung (Ausblenden von für die Applikation irrelevanten Daten) sowie dadurch eine Volumensbegrenzung der an den Client auszuliefernden Daten erwirkt. Dies trägt entsprechend zu einer Entlastung der Präsentationsschicht und damit zu einer schnelleren Interaktion bei.

Darüberhinaus ist der Middle Tier die zentrale Instanz bei jeder Kommunikation zwischen Client und Notes-Datenbanken; dadurch kann er Anfragen und Antworten filtern und eventuell ungültige Werte entsprechend behandeln respektive korrigieren. Die Anwendung wird dadurch fehlertoleranter, was zu erhöhter Zufriedenheit auf der Anwenderseite beiträgt.

Durch Spezifikation eines Profils für die Clientanwendung wird theoretisch eine breite Palette von Geräten auf der Präsentationsebene verwendbar; das Profil stellt sozusagen ein „Versprechen" bezüglich der vorhandenen Fähigkeiten des Gerätes dar und abstrahiert von den spezifischen Eigenschaften. Allerdings gibt es erst wenige Implementierungen des gewählten Personal Profile. Im Rahmen des Projekts wurde die Personal Profile Referenzimplementierung von Sun Microsystems für den Sharp Zaurus (vgl. [CVM]) auf einem Compaq Ipaq H3760 mit Familiar Linux (siehe [Familiar]) und der Oberfläche OPIE (siehe auch [OPIE]) eingesetzt.

Es gibt aber auch einige Einschränkungen, die aus den verwendeten Technologien resultieren. So weist [ToolkitHilfe] darauf hin, dass das Lotus XML Toolkit nur in einer Win32-Umgebung lauffähig ist und darüber hinaus eine Notes- oder eine Dominoumgebung benötigt, da auf einige Funktionalitäten der Laufzeitbibliotheken dieser Umgebungen zugegriffen wird. Im vorliegenden Fall benötigt der RMI-Server sogar eine Domino-Server-Umgebung; daher muss das Backend mit dem funktional davon entkoppelten Middle Tier physisch auf derselben Maschine wie die Persistenzschicht (Domino Server) installiert werden. Die Verwendung eines Benutzernamens bei der Initialisierung einer `DXLSession`, der Grundlage jeder Kommunikation innerhalb des Lotus XML Toolkits, verbietet den Zugriff auf entfernte Datenbanken (vgl. [ToolkitHilfe]). Ohne Benutzernamen lassen sich Datenbanken aber nicht öffnen. Da so also nur auf lokale Datenbanken zugegriffen werden kann und – wie später beschrieben wird – für die Persistenzebene der Zugriff auf Datenbanken eines Domino-Servers notwendig ist, kommt es zur beschriebenen physischen Kopplung dieser zwei Ebenen.

Kritisch zu würdigen ist ferner die durch die Verwendung von RMI relativ enge Kopplung des Middle Tier und der Präsentationsschicht. Zwar wird dadurch aus Sicht der Clientapplikation ein transparenter Zugriff auf die Funktionen des Servers ermöglicht, andererseits wird durch RMI die Verwendung von Java auf beiden Seiten erzwungen.

Wegen der Spezifikation eines Profils ist die Verwendung von J2ME auf dem Client zwar erwünscht, aber das Erzwingen von Java auf dem Server ist zu hinterfragen, insbesondere da mit der Common Object Request Brokerage Architecture (CORBA) ein offener Standard zur Verfügung steht, der bei Beibehaltung der Vorteile (transparenter Zugriff auf Funktionen) auf dem Server eine Vielzahl von Technologien und Plattformen möglich gemacht hätte. Die Verwendung dieses offenen Standards scheiterte jedoch daran, dass es innerhalb der J2ME-Technologie noch kein Profil gibt, das die Verwendung von CORBA unterstützt; d.h. dass es keine Möglichkeit gibt, von einem J2ME-Programm aus per CORBA zu kommunizieren. Selbst die verwendete Java-eigene Lösung RMI steht für das Personal Profile nur als Option zur Verfügung, in der Regel muss diese Option erst auf einem zu verwendenden PDA installiert werden.

# 4. Middle Tier: RMI-Server

Der RMI-Server befindet auf der mittleren Ebene der Architektur. Er tritt gegenüber der Clientapplikation auf dem mobilen Endgerät als Server auf, der die Datengrundlage bereitstellt. Gleichzeitig nimmt er von der Clientapplikation geänderte Daten entgegen. Diese Daten werden dann in gültiges DXL überführt und der Notes-/Dominoumgebung der ersten Ebene zur Speicherung übergeben. Gegenüber der Persistenzschicht tritt der Server also als Client auf und vertritt in dieser Funktion die Clientapplikation, weswegen er als „Proxy“ bezeichnet werden kann.
Der Server ist damit Dreh- und Angelpunkt jeder Kommunikation in der gesamten Anwendung; ihm obliegt es, die Daten der äußeren Schichten füreinander aufzubereiten. Er kapselt dabei die für die Anwendung notwendige Logik, „erweitert“ dadurch die Fähigkeiten der Notes Umgebung für die spezifische Aufgabenstellung und entlastet dabei gleichzeitig die Clientapplikation.

## *4.1. Überlegungen und Designentscheidungen*

Der im vorherigen Kapitel beschriebene Aufbau macht den RMI-Server potenziell zu einem Single Point Of Failure, einem Knotenpunkt, durch dessen Ausfall die gesamte Anwendung unbenutzbar wird. Ein solcher Ausfall kann entweder auf Grund einer Sicherheitslücke oder durch Überlastung herbeigeführt werden. Die Sicherung der Ablaufumgebung des Servers gegen Angriffe ist nicht Gegenstand der vorliegenden Arbeit, diese Sicherung ist von der Verwendung des RMI-Servers auf der entsprechenden Maschine weitestgehend unabhängig. Die Vermeidung von Überlastsituationen ist jedoch zum Teil durch das Design der Anwendung möglich, und so haben beim vorliegenden Prototypen auch entsprechende Überlegungen Eingang in das Design gefunden.

Da aus den in Kapitel angeführten Gründen RMI-Server und Domino-Server auf derselben Maschine laufen müssen, ist bereits durch den Domino Server von einer erhöhten Last auszugehen. Daher muss die Implementierung des Servers möglichst „schlank" ausfallen.
Prinzipiell liegt bei verteilten Anwendungen immer ein Tradeoff vor zwischen Aufwand zur Übertragung von Daten zwischen Kommunikationsteilnehmern bei Nichtspeicherung eines Zustands und Aufwand zur Verwaltung lokal vorliegender Daten(-kopien) bei Speicherung eines Zustands.
Den o.a. Überlegungen folgend wurde die Kommunikation zwischen Client und Server zustandslos implementiert, es werden auf dem Server zwischen zwei konsekutiven Aufrufen durch denselben Prozess keine Informationen über einen eventuell geänderten Datenbestand gespeichert, der beim zweiten Aufruf wieder zur Verfügung stünde. Dadurch wird zum einen Speicherplatz gespart, der für die Speicherung dieser Informationen anfiele, zum anderen entfällt ein potenziell aufwändiges Mapping von Daten zu Prozessen und der Overhead zur Verwaltung dieser Informationen. Ferner werden auf diese Art keine internen Variablen im Server geführt, die bei verzahnter Ausführung mehrerer Anfragen gegen Zugriffe aus unterschiedlichen Aufrufprozessen durch gegenseitigen Ausschluss abgesichert werden müssten (Synchronisation). Eine solche Absicherung würde die Abarbeitung von Anfragen weiter verzögern und so die Last auf dem Server durch schnelles Wachsen der Warteschlange erhöhen.
Im Umkehrschluss bedeutet dies jedoch, dass ein Client bei jedem Aufruf hinreichend viele Informationen zur Verfügung stellen muss, damit der Server auf Grund dieser Informationen die korrespondierenden Daten aus der Persistenzschicht abfordern kann. Dies impliziert ein höheres Datenvolumen bei der Übertragung der Aufrufe über das Netzwerk. Ferner bedeutet dies auch, dass die benötigten Informationen bei jedem lesenden Aufruf neu aus der Persistenzschicht geholt werden müssen (bei schreibenden Zugriffen wäre dies sowieso der Fall). Der damit verbundene Aufwand wird jedoch gerechtfertigt durch den Wegfall eines Mechanismus zum Abgleich mit den Serverdaten bezüglich der Aktualität, falls die Kommunikation zustandshaft implementiert worden wäre.
Die Antworten auf die Anfragen an die Notes-/Dominoumgebung erfolgen im DXL-Format. Da es Aufgabe des Servers ist, daraus die relevanten Daten zu extrahieren, müssen diese XML-Daten zunächst in den Speicher eingelesen werden. Dieser Vorgang wird als Parsing bezeichnet. Bei diesem Prozess sind grundsätzlich zwei Technologien verfügbar.
Die Document Object Model Technologie (DOM) liest das gesamte XML-Dokument in den Speicher ein und hält es dort komplett in einer Baumstruktur vor, die der Struktur des Ausgangsdokuments entspricht. Da die resultierenden XML-Dokumente häufig mehrere Megabyte umfassen, müsste die Servermaschine bei Anwendung dieser Technik

insbesondere bei gleichzeitiger Abarbeitung mehrerer Anfragen üppig mit Arbeitsspeicher ausgestattet sein. Da nur ein relativ kleiner Teil der Informationen aus der Antwort wirklich relevant ist, ist dies offensichtlich nicht die optimale Technologie für die Aufgabenstellung.
Einen anderen Ansatz verfolgt die Simple Access for XML API (SAX). Bei dieser Technologie wird immer nur der Inhalt eines Dokumentknotens im Speicher gehalten. So genannte Content Handler werden über diesen Inhalt und darüber, zu welchem Element sie gehören, in Kenntnis gesetzt und können sofort entscheiden, ob diese Informationen gespeichert werden müssen oder verworfen werden können. Im Anschluss werden die Informationen – sofern sie nicht gespeichert wurden – „vergessen" und die damit verbundenen Resourcen freigegeben. Auf diese Art und Weise wird relativ wenig Speicher benötigt und gleichzeitig erfolgt das Einlesen sehr schnell. Damit ist diese Technologie klar besser geeignet, der Forderung nach „Schlankheit" nachzukommen, und wurde deswegen für Parsing-Operationen im Server gewählt.
Ein weiterer Aspekt der Performanceverbesserung führte zur Durchbrechung des o.a. Prinzips der Nicht-Verwendung gemeinsamer (innerer) Variablen: Damit der Parsing-Vorgang möglichst schnell abläuft, gibt es für den ganzen Server nur ein einziges Parser-Objekt, das alle Parsing-Vorgänge abwickelt; dadurch wird vermieden, dass mehrere gleichzeitig ablaufende Parsingvorgänge sich gegenseitig die Leistung wegnehmen und beim Umschalten zwischen mehreren solchen Vorgängen potenziell aufwändige Verdrängungsvorgänge von häufig mehreren Megabyte großen Eingabeströmen durchgeführt werden müssen.
Dieser Parser muss entsprechend durch gegenseitigen Ausschluss vor dem gleichzeitigen Zugriff durch mehrere Prozesse geschützt werden, die sich gegenseitig stören würden. Dadurch stellt der Parser ein Bottleneck der Applikation dar; die Alternative mit mehreren sich störenden Parsern hätte aus der Sicht der Autoren aber leicht einen noch größeren Engpass darstellen können, so dass hier das geringere Übel gewählt wurde.
Die Implementierung des gewählten Vorgehens wird im Anhang näher beschreiben.

## *4.2. Probleme*

Im Laufe der Implementierung und beim Testen neu hinzugekommener Funktionalität haben sich zahlreiche Probleme durch die Verwendung des Lotus XML Toolkits ergeben, von denen einige unlösbar blieben. Diese Probleme sind zum Teil so schwerwiegend, dass sie die Anwendung für den Live-Einsatz unbrauchbar machen und diese nur durch einige zusätzlich implementierte Funktionen noch als Proof Of Concept Verwendung finden kann. Der vorliegende Abschnitt beschäftigt sich mit den drei Hauptproblemen und zeigt, welche Maßnahmen dagegen ergriffen wurden, wo dies möglich war.

### 4.2.1. Löschoperation nicht unterstützt

Per Design unterstützt das Lotus-XML Toolkit den Import von XML-Daten in Lotus Notes-Datenbanken sowie den Export von Daten aus solchen Datenbanken in Form von XML bzw. DXL. Allerdings gibt es keinen Befehl, mit dem Daten aus den Datenbanken entfernt werden könnten; dies wäre z.B. denkbar durch Schnittmengenbildung von anzugebenden DXL-formatierten Daten und der zugrundeliegenden Datenbank, aber eine solche Funktion steht nicht zur Verfügung.

Eine denkbare Alternative bestünde darin, einen Datenbankeintrag in DXL zu formatieren und dabei ein Attribut zu setzen, dass den Domino-Server veranlassen würde, den entsprechenden Eintrag demnächst aus der Datenbank zu löschen. Da das DXL-Format nicht hinreichend dokumentiert ist, ist es im Rahmen dieses Projekts nicht gelungen, ein solches Attribut zu finden; daher werden Löschoperationen in diesem Projekt nicht unterstützt.

### 4.2.2. Inakzeptable Antwortzeit des Servers

Beim Testen des DXL-Exports mit größer werdenden Datenbanken hat sich herausgestellt, dass Domino-Server für die Zusammenstellung der DXL-formatierten Antworten übermäßig viel Zeit benötigen. Schwächer dimensionierte Netzleitungen (z.B. Dialup-Verbindungen) scheinen dieses Verhalten noch zu verschlimmern.

So wird eine DXL-Antwort im Umfang von ca. 75KB über WLAN in unter einer Sekunde zurückgeliefert, bei Benutzung einer ISDN-Leitungs wurden dafür ca. 10 Sekunden benötigt. Bei einer DXL-Antwort von ca. 12MB vergingen über eine WLAN-Verbindung zwischen drei und vier Minuten, bei Verwendung einer ISDN-Leitung wurden die Versuche nach jeweils etwas mehr als zehn Minuten abgebrochen. Dabei scheint es sich um eine Schwäche der Umsetzung auf dem Server zu handeln, denn in jedem Fall werden der XML-Prolog und die Deklaration der DTD augenblicklich zurückgeliefert, der eigentliche Inhalt der lässt dann aber entsprechend lange auf sich warten.

Da immer wieder Mailboxen abgefragt werden müssen und diese sich realistischerweise eher an 12MB als an 75KB orientieren, reicht dieses Problem aus, um die Anwendung und das verwendete XML Toolkit für den praktischen Einsatz unbrauchbar erscheinen zu lassen.

Da der Quellcode für das XML Toolkit nicht vorliegt, konnte der Ursprung des Problems nicht gefunden und das Problem entsprechend nicht behoben werden.

### 4.2.3. Dynamisches Login unmöglich

Das schwerwiegendste aller Probleme besteht darin, dass es im Rahmen der Dienstleistungserbringung für verschiedene Anwender der RMI-Server sich mit wechselnden

Benutzerdaten dynamisch einloggen muss. Die erste diesbezügliche Einschränkung wird bereits in der [ToolkitHilfe] erwähnt: Um sich in eine Datenbank einzuloggen, muss bei fast allen Datenbanken früher oder später ein Benutzernamen für diesen Zugriff angegeben werden. Die Angabe eines Benutzernamen macht es jedoch unmöglich, auf Datenbanken auf einem entfernten Server zuzugreifen. Damit ist nur noch die Abfrage von lokalen Datenbanken möglich.

Zusätzlich bildet das Lotus-XML Toolkit den Login-Vorgang nicht auf die feingranularen Einstellmöglichkeiten des Pakets `java.security` ab, sondern behilft sich, indem auf der Konsole das Password abgefragt wird. Diese Methode des Logins steht der ganzen Idee einer serverbasierten Anwendung diametral gegenüber, da der Benutzer in der Regel schon auf Grund räumlicher Distanz auf die Konsole keinen Zugriff hat (sonst bräuchte man keinen Server).

Im Rahmen des Projekts wurde zur Umgehung dieser Problematik versucht, die System-eigenen Ein- und Ausgabeströme, die die Konsole benutzen, „umzubiegen" und jeweils die aktuellen Logindaten in die neuen Ströme einzugeben. Anhand einiger Verhaltenstests war nachvollziehbar, dass dieses „Umbiegen" auch gelungen ist. Allerdings kommt hier zum Tragen, dass die Java-Klassen des XML-Toolkits nur Wrapper für das zu Grunde liegende C++-Framework sind (vgl. [Lindhorst/Steckenborn 2003]); anscheinend erfolgt die erwähnte Login-Abfrage durch das Toolkit mit Hilfe des Java Native Interface (JNI) und dabei werden offensichtlich die ursprünglichen Ein- und Ausgabeströme verwendet, auch wenn diese mittlerweile unbrauchbar und geschlossen sind. Der anfragende Prozess wird an dieser Stelle angehalten, ohne jemals eine Fehlermeldung zurückzuerhalten.

Es ist nicht klar, ob dieses Verhalten seine Ursache in der Implementierung des Lotus-XML Toolkits hat oder einen Fehler im von der Virtual Machine implementierten Java Native Interface darstellt; auf jeden Fall macht dieses Verhalten die Anwendung für einen Live-Einsatz unbrauchbar. Gleichzeitig macht diese Form der Login-Abfrage eine klare Aussage über die Eignung bzw. Nicht-Eignung des Lotus XML-Toolkits für serverbasierte Anwendungen.

### 4.2.4. Workaround: Debug-Modus

Um trotz der o.a. angeführten Probleme wenigstens die Konzeption der Anwendung verdeutlichen zu können, wurde der sogenannte Debug-Modus eingeführt. Die Namensgebung rührt daher, dass dieser Modus auf Grund der beschriebenen Probleme die einzige Möglichkeit darstellte, implementierte Funktionalität zu überprüfen.

Dazu wurde händisch mit dem `dxlexport`-Tool des Lotus-XML Toolkits ein Datendump einer Mailbox erzeugt und dieser Dump in die Anwendung integriert. Wenn der Server im Debug-Modus betrieben wird, werden alle Anfragen statt an die „echten" Datenbanken an diesen Datendump geleitet und die Antworten daraus entnommen; das hat allerdings zur Folge, dass alle Mitglieder eines Gruppenkalenders scheinbar dieselben Termine haben, da ja alle Anfragen an die persönlichen Mailboxen an denselben Datenbestand „umgeleitet" werden. Dennoch arbeitet der Server dadurch wenigstens auf „echten" DXL-Daten und die Konzepte, die hinter der Anwendung stehen, können belegt werden.
Der Debug-Modus wird mit Hilfe der Anwendung „Configure Client" (`de.upb.ois.nt2.client.ConfigureClient`) ein- bzw. ausgeschaltet.

# 5. Frontend: Gruppenkalender Client

Der Gruppenkalender Client befindet sich als Frontend in der Dritten Schicht der gewählten Architektur. Hier werden dem Benutzer verfügbare Zeiten aus den Kalendern derjenigen angezeigt, mit denen Termine vereinbart werden sollen. Mit der passenden Java-Umgebung läuft der Client auf einem PDA mit Netzwerkanbindung sowie einem Desktop ohne Lotus Notes/Domino.
Nach der Installation des Clients und dem Einstellen der Settings schickt der Client Informationen über seinen Benutzer an den RMI-Server zurück. Mit diesen Informationen durchsucht der Server nun die Maildatenbank des Benutzers, indem ein DXL-Export dieser Datenbank initiiert, der entstandene Datenstrom geparst und auf die Gruppenkalender-Dokumente untersucht wird. Falls sich der Server im „Debug-Modus" befindet, so wird eine dem Server zur Verfügung stehende XML-Datei als Quelle benutzt.
Aus den zurückgelieferten Namen der Gruppenkalender wählt der Benutzer aus, welcher Kalender als nächstes angezeigt werden soll. Der Server ermittelt dann anhand des „Domino-Directory" die Pfade und Dateinamen der Maildatenbanken der eingetragenen Gruppenkalender-Mitglieder. Die Kalendereinträge dieser Personen werden aus den Datenbanken ausgelesen und in einem Datenobjekt beim RMI-Server abgelegt.


*Abbildung 2: Auswahl des Gruppenkalenders*

Das Datenobjekt wird an den Client übertragen. Dort werden die Termine aller Mitglieder eines Tages jeweils gleichzeitig in einer graphischen Übersicht angezeigt. Durch einen dunkleren Farbton wird dargestellt, wenn sich zwei oder mehrere Termine überlagern.

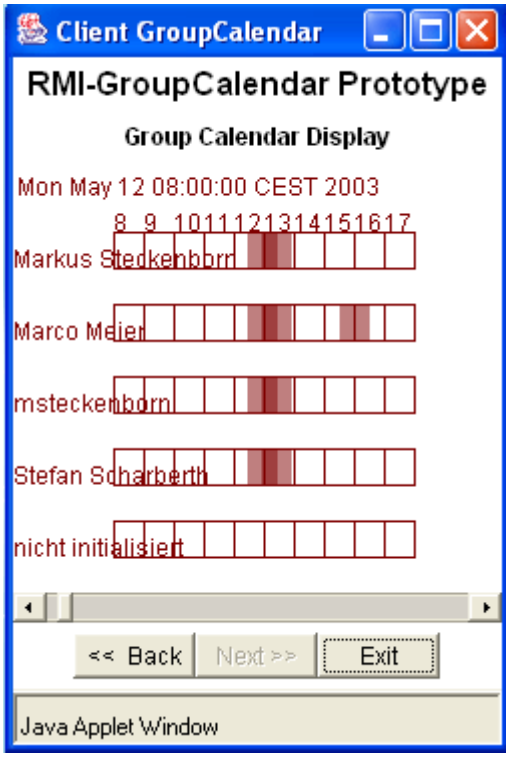



*Abbildung 3: Übersicht über die Termine*

Mit der Scrollbar lässt sich der gewünschte Tag einstellen. Mit Erreichen der Aussenseiten der Scrollbar wird das Anzeigeintervall angepasst.
Mit Mouse-Klick kann sich der Benutzer wahlweise Details zu bestehenden Kalendereinträgen anzeigen lassen oder für die entsprechende Person einen neuen Termin erzeugen
In diesem Prototyp werden alle Kalendereinträge als Typ „Appointment", d.h. ein Termin mit Start- und Endzeit und ohne weitere Teilnehmer, dargestellt. Neue Termine werden damit nur für eine Person erzeugt und in der Tagesübersicht der betroffenen Person dargestellt (siehe Abbildung 3).
Der Server erzeugt in Falle eines neuen Termins eine E-Mail mit Details des Kalendereintrages und schickt sie an den Besitzer, so dass dieser den Termin wie eine Einladung in seinem Posteingang vorfindet.

# 6. Ausblick

Wenn für die kritischen Stellen

- Login
- Löschen von Dokumente
- Reaktionszeit des Domino-Server

alternative Lösungen entwickelt werden oder ein Update des Toolkits veröffentlich wird, so können auf der im Rahmen dieser Arbeit entwickelten Architektur mobile Anwendungen mit Anbindung an Lotus Notes/Domino entwickelt werden.
Eine zusätzliche Vereinfachung/Beschleunigung bei der Abfrage von Terminen entsteht, wenn die in der „busytime.nsf" verschlüsselten belegten Zeiten der Domain-Benutzer decodiert und anstelle der aufwendigen Abfragen des Domino-Directories sowie der einzelenen Mail-Datenbanken der übrigen Teilnehmer verwendet werden.
Ein Ausbau kann mittels des entwickelten RMI-Servers für Termine geschehen. Das Frontend ist um die verschiedenen Kalendereintrags-Typen(Meeting, Reminder, All-Day-Event) zu erweitern.

# 7. Zusammenfassung

Die schwerwiegenden Mängel des XML Toolkits haben dazu geführt, dass die volle Funktionalität eines Gruppenkalenders nicht umgesetzt werden konnte. Durch die Schwächen des Toolkits in Bezug auf die Benutzer-Authentisierung wird eine Anwendung für einen Gruppenkalender unmöglich gemacht.

Eine möglicherweise schwache Unterstützung des DXLExports durch den Domino-Server sowie eine nicht hinreichende DXL-Format Dokumentierung kommen erschwerend hinzu.
Die vorliegende Arbeit zeigt, wie generell ein Zugriff auf die Termine von Lotus Notes/Domino Benutzern aussehen kann. Wenn zu einem späteren Zeitpunkt besagte Probleme behoben bzw. umgangen werden können, so lassen sich die entwickelten Komponenten als Grundlage verwenden für einen funktionierenden Gruppenkalender oder andere mobile Anwendungen.

# 8. Literaturverzeichnis

[CVM] CDC Virtual Machine for the Sharp Zaurus, J2ME Personal Profile Reference Implementation,
http://developer.java.sun.com/developer/earlyaccess/pp4zaurus/

[Familiar] Homepage von „The Familiar Linux Distribution", http://familiar.handhelds.org

[Lindhorst/Steckenborn 2003] Alexander Lindhorst, Markus Steckenborn: „Evaluation des Lotus-XML Toolkits", Projektdokumentation der Projektgruppe NT2 der Veranstaltung Office Informationssysteme I im Wintersemester 2002/2003, Universität Paderborn

[LotusHilfe] Lotus Notes 6 Help

[OPIE] Homepage des Projekts „Open Personal Information Environment", http://www.opie.info

[PersonalProfile] Informationsseite für das J2ME Personal Profile, http://java.sun.com/products/personalprofile

[ToolkitReadme] README zum Lotus XML Toolkit, im Installationsverzeichnis des Toolkits unter ./readme.txt

[ToolkitURL] Lotus Developer Domain - Sandbox - DXL Resources, http://www.lotus.com/xmltoolkit

[ToolkitHilfe] Hilfedokumentation im Lotus XML Toolkit, im Installationsverzeichnis des Toolkits unter ./LotusXML/doc/DXLTools10_html_index.htm

# 9. Glossar

| | |
|---|---|
| Backend | Komponente einer Rechnerlandschaft, die im Hintergrund (ohne dass der Benutzer mit ihr in direkten Kontakt kommt) für andere Komponenten Dienstleistungen erbringt |
| C++ | Programmierhochsprache |
| DXL | Domino XML Language, von Lotus in XML definiertes Dokumentformat |
| Framework | Konsistente Menge von Programmfunktionen zur Lösung von Problemen eines bestimmten Bereiches |
| Java | Programmierhochsprache |
| Parser | Programm zum Einlesen und zur Überführung von XML Dokumenten in eine Speicherstruktur |
| PDA | Personal Digital Assistant, kleiner Computer mit Funkionen zum Management persönlicher Informationen und eventuell weiteren Funktionen |
| Proxy | Programm, das stellvertretend für andere Programme Aufgaben unter Verwendung einer eigenen Logik erledigt |
| RMI | Remote Method Invocation, Entfernter Methodenaufruf |
| SAX | Simple API for XML, ursprünglich eine reine Java-Programmierschnittstelle zur Bearbeitung von XML-Dokumenten, die auf diesem Weg zu einem „de facto" Standard wurde und für andere Sprachen portiert wurde. |
| Toolkit | Sammlung von Programmen für einen bestimmten Aufgabenbereich |
| Win32 | Microsoft Windows™ für 32-Bit Architekturen |
| XML | Extended Markup Language, Sprache zur Definition von Dokumenten |
| XSL | Extended Stylesheet Language, in XML formulierte Sprache zur Erstellung von Stylesheets |
| XSLT | Teilbereich von XSL, der sich mit der Transformation von XML-Dokumenten beschäftigt |

# 10. Anhang

## *10.1. Installationsanleitung*

### 10.1.1. RMI-Server

Für die Installation des RMI-Servers müssen folgende Voraussetzungen erfüllt sein.

- Die Zielmaschine verfügt über eine Win32™-Umgebung.
- Auf der Zielmaschine läuft ein Domino-Server.
- Auf der Zielmaschine gibt es eine J2SE Umgebung ab JDK 1.3, die Applikation `rmiregistry.exe` ist vorhanden. Ferner muss JavaMail auf dem Zielsystem installiert sein.

Vorgehen:

1. Entpacken Sie die Datei `server.zip` in ein Verzeichnis ihrer Wahl.
2. Öffnen Sie die Kommandozeile (`cmd32`)
3. Wechseln Sie in das Verzeichnis, in das Sie das Zip-Archiv entpackt haben.
4. Wenn die Lotus Notes-Umgebung noch nicht in die PATH-Variable aufgenommen wurde, nehmen Sie sie in die Variable auf:

   ```
   set %PATH%=%PATH%;C:\Lotus\Notes
   ```

5. Starten sie die Applikation `rmiregistry.exe`, diese finden Sie üblicherweise im Unterverzeichnis `\bin` des Java-Installationsverzeichnisses.
6. Starten Sie den Server: `java -classpath .;DXLTools.jar de.upb.ois.nt2.server.NotesDelegateImpl`
7. Tragen Sie im erscheinenden Dialog die für den RMI-Betrieb notwendigen Daten ein:


8. Wählen Sie als „Codebase" das Verzeichnis, in das Sie das Zip-Archiv entpackt haben, als Policy-Datei wählen Sie die Datei `server.policy` in diesem Verzeichnis aus. Unter „Server

Name“ vermerken Sie den Namen der Zielmaschine, unter dem sie per Domain Name Service erreicht werden kann.

Der RMI-Server ist damit gestartet. Führen Sie als nächstes die Installationsschritte unter „Configure Client“ durch.

Hinweis: Die eingegebenen Daten werden gespeichert, sie müssen bei einem Neustart nicht eingegeben werden; die Eingabe der Befehle kann unterbunden werden, indem an das Startkommando der Schalter „-nogui“ angehängt wird.

### 10.1.2. ConfigureClient

Hinweis: Auf der Maschine, auf der dieses Programm laufen soll, muss eine J2SE-Umgebung ab Version JDK 1.3 vorhanden sein.

Vorgehen:

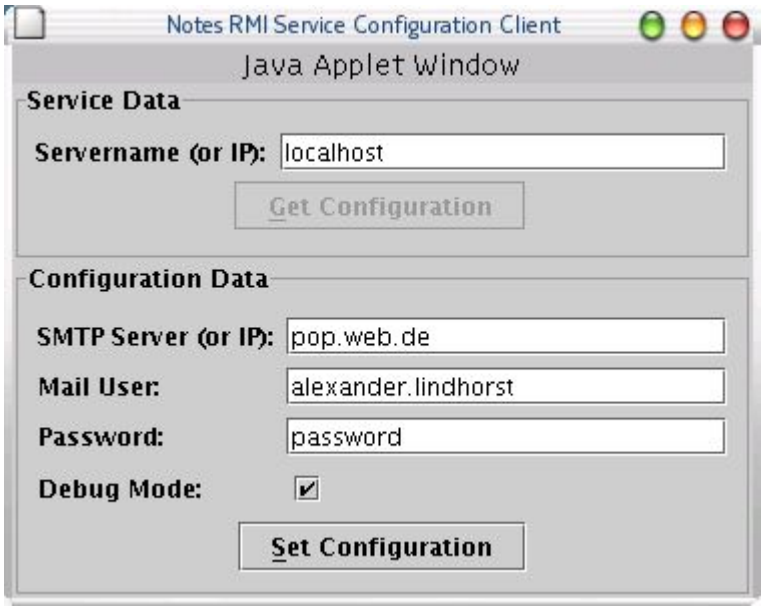
1. Entpacken Sie das Archiv `configure.zip` in ein Verzeichnis Ihrer Wahl.
2. Öffnen Sie die Kommandozeile und wechseln Sie in das Verzeichnis, in das sie das Archiv entpackt haben.
3. Starten Sie den Client:

   ```
   java -classpath .;configure.jar -Djava.security.policy=client.policy

   de.upb.ois.nt2.client.ConfigureClient
   ```

4. Geben Sie im Dialog als Servernamen den Namen ein, der dem RMI-Server beim Start als Servername angegeben wurde.
5. Klicken Sie auf „Get Configuration“.
6. Geben Sie nun im unteren Dialogbereich die benötigten Daten ein.
7. Klicken Sie auf „Set Configuration“

Hinweis: Um eine andere Maschine konfigurieren, geben Sie den neuen Namen im Feld




„Servername" ein und klicken danach erneut auf „Get Configuration".

### 10.1.3. Gruppenkalender Client

Hinweis: Auf der Maschine, auf der dieses Programm laufen soll, muss mindestens ein J2ME Personal Profile vorhanden sein.

Vorgehen:

1. Entpacken Sie das Archiv `prototyp.zip` in ein Verzeichnis Ihrer Wahl.
2. Starten Sie den RMI-Server sowie die dazu notwendige RMI-Registrierung.
3. Öffnen Sie die Kommandozeile und wechseln Sie in das Verzeichnis, in das sie das Archiv entpackt haben.
4. Starten Sie den Gruppenkalender Client:

   ```
   java -classpath .;prototyp.jar

   -Djava.security.policy=client.policy de.upb.ois.nt2.prototyp.Prototyp
   ```

5. Sie werden wird nun aufgefordert, im Settingsdialog die IP-Adresse bzw. Servernamen des RMI-Servers, Name, Passwort, Maildatenbank anzugeben (siehe Abbildung 5).

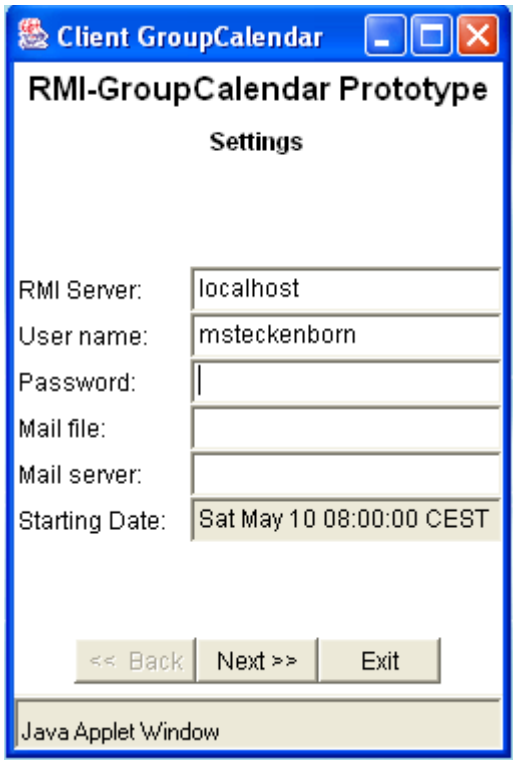



**Abbildung 5 Settings-Dialog**

6. Wählen Sie einen Gruppenkalender aus.

## *10.2. Troubleshooting*

Was tun, wenn Sie Java-Fehlermeldungen erhalten?

Überprüfen Sie zuerst, ob alle Komponenten laufen und ob die Verbindung per TCP/IP zwischen den Komponenten möglich ist. Deaktvieren Sie Ihre ggf. Ihre Firewall-Programme. Bedenken Sie auch, das die RMI-Registrierung zuerst gestartet und die Policy-Files als Startoption mit übergeben werden müssen.

Nach dem Start des Gruppenkalender Client erscheint eine Dialogbox mit folgender Fehlermeldung:

java.rmi.ConnectException: Connection refused to host: 13.213.21.22; nested exception is:
java.net.ConnectException: Connection timed out: connect

Der angegebene Server konnte nicht gefunden werden.

Behebung: Kontrollieren Sie die Server-Adresse in Settings und starten Sie den RMI-Server falls noch nicht geschehen

Nach dem Start des Gruppenkalender Client erscheint eine Dialogbox mit folgender Fehlermeldung:

java.rmi.ConnectException: Connection refused to host: localhost; nested exception is:
java.net.ConnectException: Connection refused: connect

Behebung: Starten Sie die RMI-Registry und den RMI-Server

Nach dem Start des Gruppenkalender Client erscheint eine Dialogbox mit folgender Fehlermeldung:

java.rmi.NotBoundException: NotesDelegate at sun.rmi.registry.RegistryImpl.lookup (RegistryImpl.java:106) at sun.rmi.registry.RegistryImpl_Skel.dispatch(Unknown Source)

Behebung: Starten Sie den RMI-Server.

Nach dem Start des Gruppenkalender Clients erscheint eine Dialogbox mit folgender Fehlermeldung:

java.security.AccessControlException: access denied (java.net.SocketPermission cvfS resolve) at java.security.AccessControlContext.checkPermission(AccessControlContext.java:270)

Starten Sie den Client erneut unter Angabe des Policy-Files

Nach dem Start der RMI-Registry, des RMI-Server und des Gruppenkalender Client erscheint eine Dialogbox mit folgender Fehlermeldung:

java.lang.NullPointerException
at de.upb.ois.nt2.server.NotesDelegateImpl.getCalendarNames(NotesDelegateImpl.java:110)

Es konnte keine Datenquelle zum Auslesen der Gruppenkalendarnamen geöffnet werden. Es wurde keine Maildatenbank gefunden oder eine Verbindung zum Domino-Server in Verbindung mit dem angegebenen Benutzer und dem Passwort war nicht möglich.
Überprüfen Sie die Settings oder wechseln Sie in den „Debug-Modus“, um mit einem XML-Datendump weiterzuarbeiten

Nach dem Start des ConfigurationClient und Click auf GetOptions erscheint eine Dialogbox mit folgender Fehlermeldung:

Java.rmi.NotBoundException:NotesDelegate

Starten Sie den RMI-Server und ggf. den ConfigurationClient erneut.

Nach dem Start des ConfigurationClient und Click auf GetOptions erscheint eine Dialogbox mit folgender Fehlermeldung:

Java.rmi.ConnectException: Connection refused to host: 131.234.78.179; …

Starten Sie den RMI-Server und drücken Sie erneut „Get Configuration“

## 10.3. Funktionen des Servers / Schnittstellenbeschreibung

Der Server implementiert seine Remote-Schnittstelle (die Schnittstelle, die an Clients exportiert wird, so dass sie die nutzbaren Funktionen kennen) gemäß sechs Funktionen, die im Folgenden vorgestellt werden.

### 10.3.1. Gruppen-Kalender-bezogene Funktionen

#### 10.3.1.1. getCalendarNames

Die Methode `getCalendarNames` erhält als Parameter ein Objekt der Klasse `de.upb.ois.nt2.data.Person`, aus der alle für die Verbindungsaufnahme mit dem Server notwendigen Daten abgeleitet werden. Daraufhin wird die Mailbox der entsprechenden Person vom Domino-Server im DXL-Format abgefragt und beim Parsing die Namen aller Gruppenkalender der aktuellen Person extrahiert. Diese Namen werden als String-Array zurückgeliefert.

#### 10.3.1.2. getCalendar

Die Methode benutzt das übergebene Objekt der Klasse `de.upb.ois.nt2.data.Person`, um die Informationen zur Verbindung mit dem Domino-Server zu erhalten. Daraufhin wird die Mailbox der aktuellen Person abgefragt und aus der DXL-formatierten Antwort in einem Schritt alle Termine der aktuellen Person sowie Informationen über die weiteren Teilnehmer des Gruppenkalenders gewonnen. Mit den gewonnenen Terminen wird ein Objekt der Klasse `de.upb.ois.nt2data.Calendar` instanziert. In einem zweiten Schritt wird die Datenbank „`names.nsf`" abgefragt und dabei die Informationen über die weiteren Teilnehmer vervollständigt.
Im dritten Schritt wird für jeden einzelnen weiteren Teilnehmer dessen Mailbox geparst und daraus seine Termine extrahiert, mit denen dann ebenfalls `Calendar`-Objekte instanziert werden. Im letzten Schritt werden alle `Calendar`-Objekte zu einem `Calendar`-Objekt zusammengeführt; das Ergebnis ist ein Gruppenkalendar mit den Termininformationen aller beteiligten Mitglieder. Dieser Kalendar wird als Ergebnis der Anfrage zurückgeliefert.
Bei der Implementierung dieser Methode wurde zunächst versucht, die benötigten Informationen in einem Schritt durch Abfragen der Datei „busytime.nsf" zu gewinnen; in der DXL-formatierten Antwort der entsprechenden Anfrage sind die Informationen allerdings

verschlüsselt, so dass diese Informationen nicht zur Aufgabenlösung herangezogen werden konnten.

#### 10.3.1.3. addCalendarEntry

Aufgabe der `addCalendarEntry`Methode ist es, Informationen im Auftrag der Clientapplikationen in DXL aufzubereiten und sie dem Backend zur persistenten Speicherung zu übergeben. Dabei unterscheiden sich die Abläufe, je nachdem ob es sich um einen einfachen Termin (Appointment) oder oder um einen Gruppentermin (Meeting) handelt.

Bei einem Appointment werden die Daten des Termins in DXL überführt, die als Parameter angegebene Person wird als „Owner" des Termins deklariert und die Daten per XML Toolkit in die Mailbox der entsprechenden Person geschrieben. Ferner erhält die aktuelle Person eine Email, in der auf den Vorgang hingewiesen wird.

Handelt es sich bei dem Termin um ein Meeting, werden alle anderen betroffenen Personen ebenfalls per Email von dem Vorgang in Kenntnis gesetzt mit der Bitte, den Termin (händisch) in ihren Kalender aufzunehmen. Hier wurde darauf verzichtet, die Daten direkt in den jeweiligen Kalender per DXL und Lotus XML Toolkit zu schreiben, um erstens Schwierigkeiten mit der Rechtesituation bezüglich einzelner Datenbanken zu vermeiden und zweitens sicherzustellen, dass die Beteiligten aktiv von dem Vorgang Notiz nehmen.

#### 10.3.1.4. removeCalendarEntry

Ziel dieser Methode war es ursprünglich, Einträge aus einem Kalender zu entfernen. Allerdings wurde keine Möglichkeit zur Implementierung eines solchen Vorgangs mit dem Lotus XML Toolkit gefunden. Kapitel 4.2 geht darauf genauer ein.

Die Methode ist daher „leer" implementiert.

### 10.3.2. Konfigurations-bezogene Funktionen

Für die korrekte Funktionsweise des Servers bedarf es einiger Informationen, auf die in den o.a. Kalender-bezogenen Funktionen lesend zugegriffen wird. Die Methoden zur Behandlung dieser Informationen werden im vorliegenden Abschnitt behandelt.

Dem Prototyp liegt mit der Klasse `de.upb.ois.nt2client.ConfigureClient` ein Konfigurationsprogramm bei, das auf die hier angeführten Methoden zugreift.

#### 10.3.2.1. getConfiguration

Der Aufruf dieser Methode liefert ein Objekt der Klasse `de.upb.ois.nt2data.ConfigurationData` zurück, aus dem die derzeitigen Einstellungen des Servers bezüglich zu verwendendem Mailserver, -account und -password ausgelesen werden können. Ferner kann dieser Datenstruktur entnommen werden, ob sich der Server derzeitig im Debug-Mode befindet. Zur Bedeutung des Debug-Mode sei auf Kapitel  verwiesen.

#### 10.3.2.2. setConfiguration

Diese Methode nimmt ein Objekt der Klasse `de.upb.ois.nt2data.ConfigurationData` und übernimmt die Einstellungen daraus für alle weiteren Aktionen. Diese Daten werden gespeichert und stehen nach einem Neustart des Servers sofort zur Verfügung; eine Neueingabe ist nicht erforderlich.