Systemadministration mit Smartphones
auf Basis einer flexiblen Middleware-Architektur

Michael Skusa

Übersicht

1) Anwendungsszenarien, Beispiele
2) Zielsetzung
3) Anwendungsumgebung
4) Softwarearchitektur
5) Implementierung
6) Sicherheit
7) Erweiterbarkeit, Integration, Ausblick

Übersicht

1) Anwendungsszenarien, Beispiele

2) Zielsetzung

3) Anwendungsumgebung

4) Softwarearchitektur

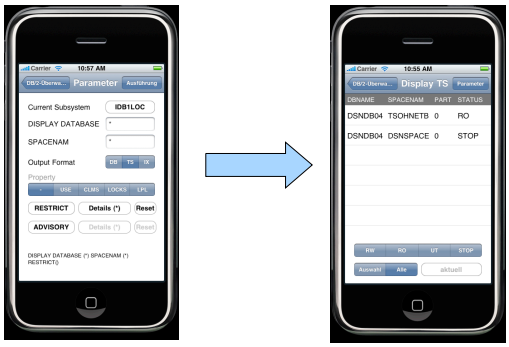
5) Implementierung

6) Sicherheit

7) Erweiterbarkeit, Integration, Ausblick

DB-Überwachung

Aufgabe:
Anzeige von Teilbereichen einer Datenbank, die zur Zeit nur eingeschränkt zugreifbar sind ?



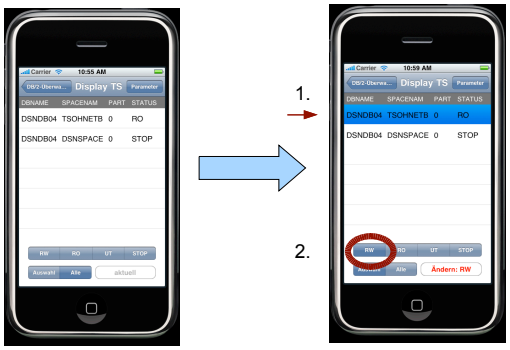
Wichtigkeit für Verfügbarkeit der DB:	sehr hoch
Vorhersehbarkeit:	gering
Häufigkeit:	selten

Links:
Interaktive Parametrisierung eines Datenbankkommandos zur Anzeige von Statusinformationen,
hier: Alle Tablespaces, die sich in einem „Restricted“-Zustand befinden, also nicht im Zustand Read/Write sind.

Rechts:
Das Ergebnis des Befehls - Ein Tablespace ist gestoppt, ein weiterer ist im Zustand „Read Only“.

DB-Steuerung

Aufgabe:
Kurzfristige Behebung eines eingeschränkten Datenbankzustandes.



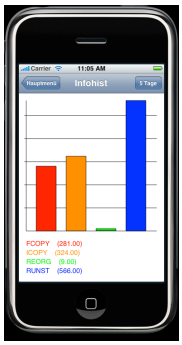
Wichtigkeit für Verfügbarkeit der DB:	sehr hoch
Vorhersehbarkeit:	gering
Häufigkeit:	selten

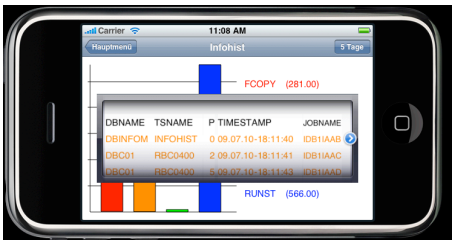
Links:
Die Ergebnis der Statusabfrage nach restricted Tablespaces (vgl. vorige Folie)

Rechts:
1. Anklicken des zu ändernden Tablespace (Pfeil oben links)
2. Auswahl des Zielzustandes „RW“
3. Ausführung der Änderungsoperation (Pfeil unten rechts, „Ändern“-Button)

DB-Überwachung

Aufgabe:
Wie viele und welche Wartungsoperationen wurden in den letzten fünf Tagen ausgeführt ?





Wichtigkeit für Verfügbarkeit der DB:	gering
Vorhersehbarkeit:	hoch
Häufigkeit:	regelmäßig

Links:
Darstellung einer Statusabfrage als Balkendiagramm.

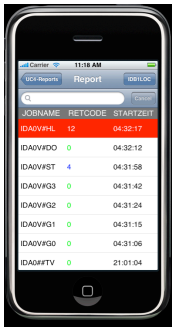
Rechts:
Querformat mit Anzeige zusätzlicher Details zu einem der angezeigten Diagrammelemente, eingefärbt in der Farbe des zugehörigen Diagrammelements.

Mit dem „Pfeil nach rechts“ neben dem aktuell ausgewählten Job kann man den z/OS-Output des Jobs zur Anzeige bringen.

Praktische Nutzung: Routinemäßige Prüfung, ob bestimmte wiederkehrende Aufgaben ausgeführt wurden (z.B. täglich, wöchentlich).

Scheduler-Reporting

Aufgabe:
Überprüfung, welche Jobs im Laufe des letzten Tages gestartet wurden und mit welchem Ergebnis.



Wichtigkeit für Verfügbarkeit der DB:	mittel
Vorhersehbarkeit:	hoch
Häufigkeit:	regelmäßig

Links:
Auswahl des anzuzeigenden Reports anhand des Datums, der „Instanz“ und des „Mandanten“, über die berichtet wird.

Rechts:
Tabellendarstellung des links ausgewählten Reports. Die Tabelle kann durch Eingaben in das Suchfeld gefiltert werden. Fehlerzustände werden farblich hervorgehoben.

Fehlerdiagnose

Aufgabe:
Anzeige der Ausgaben, die ein mit Fehlercode beendeter Job auf dem Host produziert hat.

Wichtigkeit für Verfügbarkeit der DB: hoch

Vorhersehbarkeit: gering

Häufigkeit: selten

Links:
Tabellendarstellung eines Scheduler-Reports.

Rechts:

Im Hochformat:
Die z/OS-DD-Elemente des links ausgewählten Jobs.

Im Querformat:
Der Inhalt eines ausgewählten DD-Elements.

Scheduler-Steuerung

Aufgabe:
Neustart eines abgebrochenen Jobs.

Wichtigkeit für Verfügbarkeit der DB:	mittel
Vorhersehbarkeit:	gering
Häufigkeit:	selten

Links:
Tabellendarstellung der zur Zeit aktivierten bzw. noch nicht erfolgreich beendeten/archivierten Tasks eines Schedulers.
Eine Task ist im Zustand ABENDED (gelb), wurde also abgebrochen. Durch Druck auf den „Wiederanlauf“-Button, wird diese Task erneut gestartet.

Rechts:
Ergebnis des Wiederanlaufs. Liste enthält nach wie vor die abgebrochene Task, aber zusätzlich die durch den Wiederanlauf entstandene neue Task im Zustand RUNNING (grün).

Übersicht

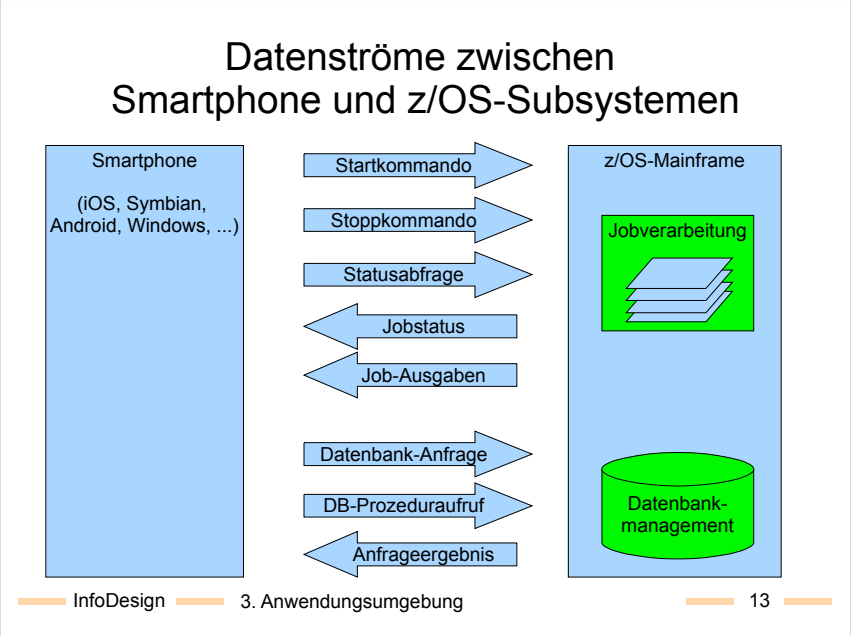
1) Anwendungsszenarien, Beispiele
2) Zielsetzung
3) Anwendungsumgebung
4) Softwarearchitektur
5) Implementierung
6) Sicherheit
7) Erweiterbarkeit, Integration, Ausblick

Vorteile, Ziele

- Schnelle Reaktionszeit durch Unabhängigkeit des Administrators vom PC-/Terminal-Arbeitsplatz:
  - Überwachung kann in kürzeren Zeitabständen erfolgen,
  - Überwachung ist ortsunabhängig,
  - Entscheidung über Gegenmaßnahmen bei Störungen kann schneller getroffen werden.
- Präsenzzeiten an der Konsole im RZ oder am Arbeitsplatz-PC lassen sich reduzieren (Kostensenkung).
- Hohe Verdichtung der Überwachungsdaten erleichtert Rückmeldung an Management.
- Erforderliche Hardware (Mobiltelefon) in aller Regel bereits vorhanden bzw. auch anderweitig erforderlich.

Übersicht

1) Anwendungsszenarien, Beispiele
2) Zielsetzung
3) Anwendungsumgebung
4) Softwarearchitektur
5) Implementierung
6) Sicherheit
7) Erweiterbarkeit, Integration, Ausblick



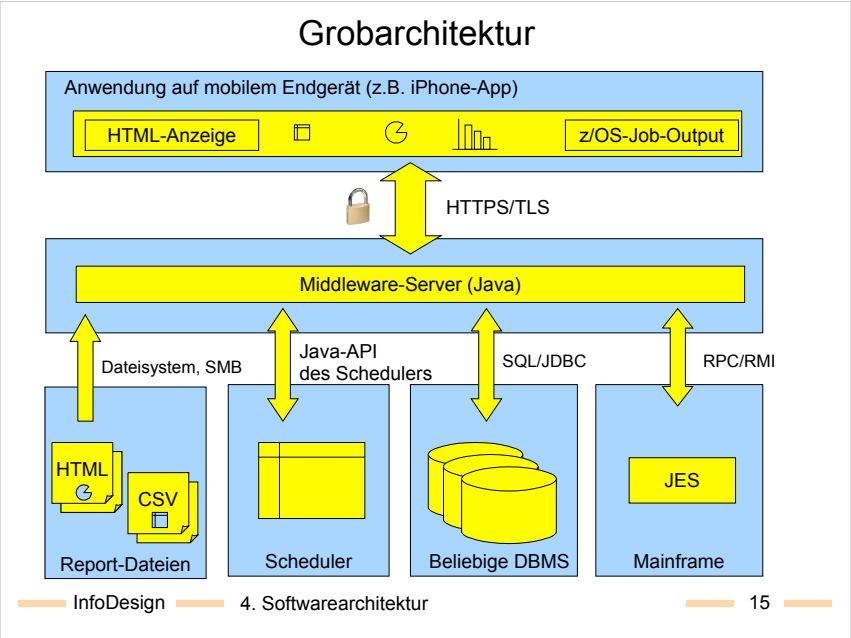
Aus der Sicht der Smartphone-Anwendung sollen Kommandos, Parameter und Statusabfragen an die Jobverwaltung des Mainframes gesendet werden können.

Der Mainframe liefert Statusdaten und Jobausgaben.

An die Datenbanken des Mainframes werden vorgefertigte Anfragen zur Ermittlung oder Änderung des Zustandes der Datenbank geschickt. Diese werden per Smartphone parametrisiert und das Anfrageeregebnis an das Smartphone zurück übertragen.

Übersicht

1) Anwendungsszenarien, Beispiele
2) Zielsetzung
3) Anwendungsumgebung
4) **Softwarearchitektur**
5) Implementierung
6) Sicherheit
7) Erweiterbarkeit, Integration, Ausblick

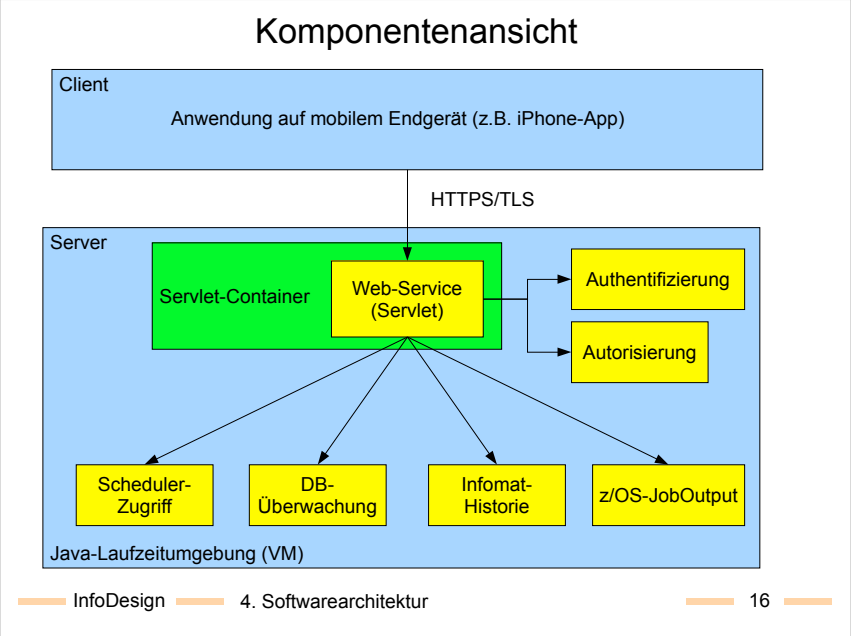


Auf dem Smartphone erfolgt die Auswahl und Parametrisierung der auszuführenden Aktion. Diese Daten werden per HTTP(S) an einen in Java entwickelten Middleware-Server übermittelt und in einem Standardformat (JSON) formatiert.

Der Middleware-Server übersetzt die Daten des Smartphones aus diesem Standardformat in Aufrufe von Schnittstellen, die für den Middleware-Server zugänglich sind, unter anderem:
• JDBC (Datenbankzugriff),
• Java-Schnittstellen spezieller Software (hier: Scheduler),
• RMI (entfernte Methodenaufrufe),
• Standard-Funktionen der Java-Laufzeitumgebung (Dateisystem-Zugriff).

Die über diese Schnittstellen erhaltenen Daten übersetzt der Middleware-Server wieder in ein Standardformat und liefert diese als Antwort an das Smartphone.

Die Visualisierung der Daten auf dem Smartphone wird nativ in der Programmiersprache des Smartphones entwickelt.
Alternativ: Visualisierung wird in der Middleware berechnet und auf dem Smartphone in dessen Standard-Webbrowser dargestellt. (Vorteil: keine Smartphone-seitige Programmierung erforderlich, Nachteil: spezifische Fähigkeiten einzelner Smartphones bleiben ungenutzt, Visualisierung muss trotzdem Smartphone-spezifisch erfolgen (unterschiedliche Display-Größen, unterschiedlich potente Webbrowser))



Die Grafik zeigt die wesentlichen serverseitigen Komponenten der Anwendung (gelbe und grüne Boxen).

Alle Kompoenten des Servers sind unabhängig voneinander entwickelt und funktionsfähig und können durch alternative Komponenten ausgetauscht (z.B. für unterschiedliche Versionen derselben Backendsoftware) oder um zusätzliche Komponenten ergänzt werden.

Scheduler-Zugriff, DB-Überwachung, Infomat-Historie:
Service-Komponenten zur Anbindung unterschiedlicher Backendsysteme, die per Mobilgerät überwacht und gesteuert werden sollen.

Web-Service:
Ein Java-Servlet, das zwei Funktionen erfüllt:
1. Protokolladapter: Vermittlung zwischen den Java-Schnittstellen der Service-Komponenten und der HTTP-Schnittstelle für den mobilen Client.
2. Fassade: Bereitstellung eines einzigen Zugangspunkts, über den alle für den mobilen Client relevanten Service-Komponenten erreichbar sind.

Authentifizierung, Autorisierung: Werden zur Zeit vom Web-Service genutzt, um festzustellen, ob ein Nutzer bekannt und berechtigt ist, den Web-Service zu nutzen.

Servlet-Container: Eine spezielle Laufzeitumgebung für Web-Anwendungen (z.B. Apache Tomcat, Oracle OC4J), die zur Ausführung von Servlets (hier: Web-Service) erforderlich ist.

Im Beispiel laufen alle Komponenten innerhalb derselben Laufzeitumgebung bzw. innerhalb derselben Java-Maschine, einschließlich des Servlet-Containers und des Web-Service-Servlets. Jede Komponente könnte auch in ihrer eigenen Java-Umgebung, sogar jeweils auf einem eigenen physischen Rechner laufen.

Aktuelle Implementierung

- Datenquellen bzw. zu überwachende Systeme:
  - per JDBC erreichbare relationale Datenbanken,
  - per Dateisystem oder SMB erreichbare CSV-Dateien,
  - per Java-API nutzbare Scheduler und Host-Prozesse.
- Middleware (Java):
  - Komponenten – JMX-Beans mit RMI-Schnittstellen,
  - als JMX-Bean eingebetteter Servlet-Container (Apache Tomcat),
  - Servlet für die Vermittlung zwischen HTTP und Komponentenschnittstellen,
  - JSON als Austauschdatenformat zwischen Client und Middleware.
- Client/Frontend
  - Native iPhone App (Objective-C)

Anmerkung zu Objective-C:
Von Middleware abweichende Programmiersprache bedeutet zunächst höheren Einarbeitungsaufwand, spielt für die mittel- bis langfristige Entwicklung aber keine Rolle.

- Ein Java-Programmierer würde vermutlich Java-basierte Endgeräte wie Android oder Blackberry bevorzugen.
- Ein .NET-Programmierer würde vermutlich Windows-Mobile-oder Wondows-Phone-Endgeräte bevorzugen.
- Ein C- oder Objective-C-Programmierer würde bevorzugt für das iPhone entwickeln.

Eine Kommunikation mit der hier vorgestellten Middleware ist mit **jedem** dieser Endgeräte möglich. Die verwendete Programmiersprache ist dabei eher nebensächlich.

Übersicht

1) Anwendungsszenarien, Beispiele
2) Zielsetzung
3) Anwendungsumgebung
4) Softwarearchitektur
5) Implementierung
6) Sicherheit
7) Erweiterbarkeit, Integration, Ausblick

Sicherheit

- Nutzer von Überwachungs-/Steuerungssoftware (Administratoren) besitzen in der Regel sehr weitreichende Zugriffsberechtigungen.
- Wird die Nutzung dieser Rechte über ein Mobilgerät ermöglicht, sind entsprechend weitreichende Sicherheitsüberlegungen nötig.

Mögliche Risiken:

- Diebstahl des Endgeräts,
- Fehlbedienung
  (Endgerät suggeriert Einfachheit bzw. Harmlosigkeit von Operationen),
- Angriff auf Application-Server (HTTP- oder RMI-Ports),
- Abhören des Verkehrs zwischen Mobilgerät und Middleware,
- Angriff auf Backend-Systeme (Host, Datenbank, Scheduler),

Stichworte zu Risiken:

- Diebstahl: Falls Zugangsdaten fest verdrahtet oder abgespeichert wären, könnte ein Dieb die Anwendung mit Administratorrechten weiternutzen.
- Fehlbedienung: Administratoreingriffe können unternehmenskritische Systeme nicht nur überwachen oder in Betrieb nehmen, sondern theoretisch auch zum Stillstand bringen und erfordern daher entsprechende Sicherheitsabfragen oder Beschränkungen auf bestimmte Geräte oder Nutzer.
- Angriff auf Application-Server: Ist die Zugriffskontrolle nicht Teil einer Komponente, sondern ein optionaler Dienst, darf die Komponente nicht über einen Kanal ohne Zugriffskontrolle nutzbar sein.
- Die Datenübertragung zwischen Client und Server darf nicht im Klartext erfolgen.
- Läuft die Middleware auf demselben physischen Server wie die Backend-Software, werden u.U. versehentlich Backend-Services im Internet sichtbar (Beispiel: Firewall wurde versehentlich „zu offen“ konfiguriert und lässt nicht nur Verkehr für Port der Middleware, sondern für alle Ports des Backend-Servers durch)

Aktuelle Sicherheitsmaßnahmen

Gegen

- Diebstahl des Endgeräts:
  Keine Speicherung von Zugangsdaten zu Backend-Systemen auf dem Endgerät:
  - explizite Abfrage vor erstmaliger Nutzung der entsprechenden Funktion,
  - erneute Abfrage nach jedem Neustart der Anwendung,
  - abhängig vom Endgerät: automatisches Sperren/Beenden der Anwendung bei Wechsel in Stand-By-Modus oder beim Wechsel der Anwendung.
- Fehlbedienung
  (Endgerät suggeriert Einfachheit bzw. Harmlosigkeit von Operationen):
  - Middleware bietet keine Skripting-Schnittstelle o.ä. an, sondern nur einen (kleinen) Satz von fachlich sinnvollen/erwünschten Operationen.
  - Endgerät kann nicht direkt auf Backend-System zugreifen, sondern nur über die von der Middleware angebotenen Funktionen.
  - Potentiell gefährliche Middleware-Komponenten können einzeln abgeschaltet werden.

Aktuelle Sicherheitsmaßnahmen

Gegen

- Angriff auf Application-Server (HTTP- oder RMI-Ports):
  Nur der HTTP-Port der Middleware ist aus dem Internet durch die Firewall sichtbar. Die Middleware selber muss auch die Backend-Systeme „sehen“, muss sich also im Intranet der Firma (z.B. in der DMZ) befinden.
- Abhören des Verkehrs zwischen Mobilgerät und Middleware:
  SSL/TLS – Verbindung kommt nur zustande, falls Endgerät das Zertifikat des Servers erfolgreich prüfen kann. Danach handeln beide eine Verschlüsselung des Datenverkehrs aus.
- Angriff auf Backend-Systeme (Host, Datenbank, Scheduler):
  Middleware ist auf physisch getrenntem Rechner installiert. Durch Firewall ist nur der Middleware-Server und nur dessen HTTP-Port erreichbar. Backend-Zugriffsdaten werden von der Middleware nicht gespeichert, sondern nur an Backend-Systeme durchgereicht.
- Die Zugriffskontrollen der Backend-Systeme bleiben vollständig wirksam. Ein Nutzer einer Mobilfunkanwendung hat auf dem Backend-System keine weiterreichenden Rechte, als er sie auch an seinem Arbeitsplatz hätte.

Übersicht

1) Anwendungsszenarien, Beispiele
2) Zielsetzung
3) Anwendungsumgebung
4) Softwarearchitektur
5) Implementierung
6) Sicherheit
7) Erweiterbarkeit, Integration, Ausblick

Aktueller Funktionsumfang

- Server:
  - Überwachung von DB/2 auf z/OS
    (Datenbanken, Tablespaces, DDF, Log-Archivierung, Threads, Utilities)
  - Überwachung von Oracle
  - Überwachung Infomat-basierter Aktivitäten (Historie)
  - Scheduler-Überwachung (zur Zeit für UC4)
    - Anzeige/Aufbereitung von Reportdateien des Schedulers,
    - Auswertung der internen Datenbank des Schedulers (per SQL), z.B. für Zugriff auf nicht oder noch nicht per Report-Datei zugängliche Daten,
    - Steuerung des Schedulers
      - Anzeige laufender Tasks,
      - Anhalten/Fortsetzen, Abbrechen/Wiederanlauf von Tasks,
      - Setzen/Entfernen von Haltepunkten.
- Endgerät:
  - Apple iOS (iPod Touch, iPhone)

Integration in bestehende Umgebung

- Der vom Mobilgerät genutzte Java-Web-Server kann
  - als eigenständiger Server-Prozess oder
  - als Teil einer bestehenden Web-Server-Installation genutzt werden.
- Report-Dateien können
  - Auf lokalen Platten des Rechner liegen, auf dem der Java-Server läuft,
  - auf entfernten Platten liegen, die sich per NFS, SMB o.ä. in das Dateisystem des Java-Servers einhängen lassen.
- Der Rechner, auf dem der Java-Server laufen soll, benötigt lediglich eine funktionsfähige Java-Laufzeitumgebung. Das Betriebssystem des Server-Rechners spielt keine Rolle (derzeitige Implementierung funktioniert u.a. unter Windows, Linux, MacOS und z/OS).
- Diejenigen Rechner, die vom Java-Server aus überwacht werden sollen, müssen aus dem Netzwerk, in dem der Java-Server läuft, sichtbar und zugreifbar sein. Gegenüber den überwachten Datenbanken, Schedulern und Hosts verhält sich der Java-Server wie ein gewöhnliches Client-Programm.
- Der HTTPS-Port des Java-Servers muss für das Mobilgerät aus dem Internet (oder aus einem berechtigten WLAN oder VPN) erreichbar sein.

Erweiterbarkeit

- Über JDBC ist die Überwachung jeder Datenbank möglich,
  - für die ein JDBC-Treiber verfügbar ist (DB/2, Oracle, MS-SQL-Server, SAP-DB, …) und
  - die per SQL abfragbare Überwachungsdaten liefert (z.B. aus ihrem Data Dictionary)
- Alle DB-Administrationsaufgaben, die per Stored Procedure gestartet werden, können als Aktionen auf dem mobilen Gerät verfügbar gemacht werden.
- Alle Aktionen, die über eine dokumentierte Java-Schnittstelle einer zu überwachenden Software (Scheduler, Application-Server, Message Oriented Middleware, Datenbank) zugänglich sind, können auf dem mobilen Gerät verfügbar gemacht werden.
- Report-Dateien, die in einem hinreichend strukturierten und dokumentierten Format vorliegen (XML, CSV) können für die Anzeige auf dem Mobilgerät aufbereitet werden.

Alternative Endgeräte

- Zur Zeit wird das Apple iPhone als „Frontend“ für den Java-Server genutzt.
- Der Java-Server kann von **nativen Anwendungen** (Apps) jedes Endgeräts genutzt werden (z.B. Android, Blackberry, Windows Mobile), für das Bibliotheken für den HTTPS-Zugriff verfügbar sind.
- Die Entwicklung eines **Desktop-Client** (z.B. auf Basis von .NET, Java, Mac OS) für den Java-Server für den Einsatz auf Notebooks, Netbooks oder Desktop-PCs ist ebenso möglich.
- Ist die Entwicklung einer nativen Anwendung nicht möglich oder erwünscht, lässt sich auch ein **Web-Client** z.B. mit einer HTML- , Flash- oder Silverlight-Oberfläche entwickeln.
- Die gesamte Kommunikation mit den zu überwachenden Servern und die Verdichtung der Daten findet bereits auf dem Java-Server statt.
- Für ein alternatives Endgerät muss für eine native Anwendung „nur“ die Oberfläche zur Eingabe von Parametern und zur Anzeige von Ergebnissen neu entwickelt werden (Aufwand abhängig vom Entwicklungsstand des Endgeräts, seiner Entwicklungswerkzeuge und der zu nutzenden Funktionen des Java-Servers).

Ausblick – künftige Entwicklung

- Server:
  - Zusätzliche Überwachungsfunktionen für bestehende Dienste (DB/2, UC4)
  - Dienste für spezielle Datenbanken (DB/2 LUW, Oracle, MS-SQL usw.)
  - Dienste für spezielle Scheduler
  - Zugriff auf vorhandene Überwachungsprozesse (z.B. z/OS HealthChecker)
  - Zugriff auf vorhandene Administrations- und Fernwartungssoftware (Nutzung des Mobiltelefons als „schmalbandigen“ Client zur Fernwartung)
  - Integration der Überwachung von Werkzeugen unterschiedlicher Hersteller unter einer einzigen, an die Arbeitsabläufe des Kunden optimal angepassten Oberfläche.

- Endgerät:
  - Portierung des Frontends auf andere Mobilplattformen (Android, Blackberry, …) oder serverseitig verwaltete Frontends (HTML, Flash)
  - Angepasste Visualisierungen und GUI-Elemente für spezielle Funktionen (2D, 3D, OpenGL)