Softwareentwicklung für Mundo Smart Environments

Erwin Aitenbichler und Max Mühlhäuser

Telekooperation, Fachbereich Informatik, TU Darmstadt
(erwin|max)@tk.informatik.tu-darmstadt.de,
WWW home page: http://www.tk.informatik.tu-darmstadt.de/

Zusammenfassung. Das Mundo-Projekt an unserer Arbeitsgruppe befasst sich mit allgemeinen Modellen und Architekturen für Ubiquitous Computing (UC)-Systeme. Dieser Beitrag beschäftigt sich mit der systematisierten Entwicklung von Anwendungen für Smart Environments. Dazu wurde im Mundo-Projekt eine Reihe von Diensten und Werkzeugen entwickelt. Es werden Werkzeuge zur Überwachung, Fehlersuche, Durchführung verteilter Tests, und zum Rapid Prototyping beschrieben und wie diese in den Software-Entwicklungsprozess eingebettet sind. Damit wird es möglich, Anwendungen für Smart Environments systematisiert zu entwickeln.

1 Einleitung

Smart Environments sind Wohn- oder Arbeitsumgebungen, in denen Computertechnik direkt in die Gebäude-Infrastruktur integriert ist. Dabei wird meist das Ziel verfolgt, Interaktionen mit dem Computersystem zu vereinfachen oder Arbeitsabläufe effizienter zu gestalten. Bei der Entwicklung von Anwendungen für Smart Environments müssen Lösungen in den folgenden Problembereichen gefunden werden:

Plattform: Eine verteilte Laufzeitumgebung muss die Kommunikation zwischen den Geräten ermöglichen. Dabei müssen Software-Dienste, die auf unterschiedlichen Plattformen, Betriebssystemen, oder Programmiersprachen basieren, integriert werden. Zusätzlich soll eine spontane Vernetzung mit von Benutzern mitgebrachten mobilen Geräten unterstützt werden.

Weltmodell: In Smart Environments finden Interaktionen nicht mehr stationär statt, sondern vorwiegend in einem frei beweglichen Kontext. Deshalb spielen die Aufenthaltsorte von Benutzern und Objekten in der Umgebung eine Rolle. Ortsfeste Objekte können statisch in einem Weltmodell modelliert werden und bewegliche Objekte können mit Sensoren erfasst werden.

Kontextverarbeitung: Neben dem Ortskontext können noch weitere Sensoren (Lage-, Beschleunigungs- oder Gewichtssensoren, Mikrofone, etc.) oder Dienste (Terminkalender, Raumbuchungssystem, etc.) wertvolle Informationen über Benutzer und deren Tätigkeiten liefern. Diese Rohdaten müssen in für die Anwendungen bedeutsame Kontextinformationen übergeführt werden.

Anwendungslogik: Schließlich muss die Logik der eigentlichen Anwendung entweder direkt programmiert oder mit geeigneten Werkzeugen modellbasiert erstellt werden. Da in der Ubiquitous Computing-Forschung das Erstellen von Prototypen eine wesentliche Rolle spielt, sind hier RAD-Werkzeuge wünschenswert.

In diesem Beitrag werden *Mundo Smart Environments* vorgestellt. Es handelt sich dabei um eine Architektur für Smart Environments, die notwendigen Kerndienste und Entwicklungswerkzeuge, die Lösungen für die beschriebenen vier Problemfelder bieten. Im Weiteren wird ein Software-Entwicklungsprozess für Smart Environment-Anwendungen beschrieben und wie unsere Werkzeuge in diesem Prozess eingesetzt werden.

Dieser Beitrag ist wie folgt gegliedert. In Abschnitt 2 werden verwandte Arbeiten beschrieben. In Abschnitt 3 wird ein kurzer Überblick der Software-Architektur gegeben. Die gemeinsame Softwarebasis bildet die Kommunikations-Middleware MundoCore. Anwendungen bestehen aus einer Reihe von Diensten, die entweder allgemein wiederverwendbare Funktionalität (*common services*), oder anwendungsspezifische Funktionalität implementieren. In Abschnitt 4 wird ein erweiterter Software-Entwicklungsprozess vorgestellt, der auf dem Wasserfall-Modell basiert und die zusätzlichen Entwicklungsschritte für Smart Environment-Anwendungen enthält. Es wird gezeigt, wie unsere Werkzeuge bestimmte Schritte im Prozess unterstützen. Der Beitrag endet in Abschnitt 5 mit einer Zusammenfassung.

2 Verwandte Arbeiten

Werkzeuge zur Erstellung von Umgebungsmodellen sind in kommerziellen LPS-Produkten enthalten, z.B. Ubisense [12] und Elpas [10]. Ubisense enthält keine Werkzeuge zur Anwendungsentwicklung. Die Anwendungsfunktionalität muss daher z.B. in C++ programmiert werden. Elpas arbeitet regelbasiert, wobei der Regeleditor auch für Endnutzer nutzbar ist. Das System ist aber stark auf Anwendungen im Sicherheitsbereich spezialisiert.

Das CAMP-System erlaubt das einfache Erstellen von Capture-Anwendungen basierend auf “Magnetic Poetry” [11]. Hull beschreibt ein System zum Erstellen von Medienlandschaften basierend auf Umgebungsmodell und Skripting [6]. Diese Projekte decken nur sehr spezifische Anwendungsfelder ab. Im Umfeld des Nexus-Projekts wurde ein Werkzeug zur Programmierung von intelligenten Umgebungen basierend auf Flussdiagrammen entwickelt [13].

Intelligente Umgebungen werden in zahlreichen weiteren Projekten untersucht, darunter Gaia [8], Aura [5], Nexus [13] und iWork [7].

Im Gegensatz zu den verwandten Arbeiten deckt unser System mit seiner flexiblen Plattform, den wiederverwendbaren Diensten und den Werkzeugen alle Phasen des Entwicklungsprozesses ab. Die Beschreibung des Prozesses wird als weiterer, wichtiger Beitrag angesehen, um die Entwicklung von Anwendungen zu systematisieren.

3 Mundo Smart Environments

Abbildung 1 zeigt einen Überblick der Kerndienste und Werkzeuge unseres Systems und die Datenflüsse auf einem höheren Abstraktionsgrad. Die Kommunikation zwischen Diensten wird von der Kommunikations-Middleware implementiert. Die Teile des Systems werden in den nächsten Abschnitten genauer beschrieben.

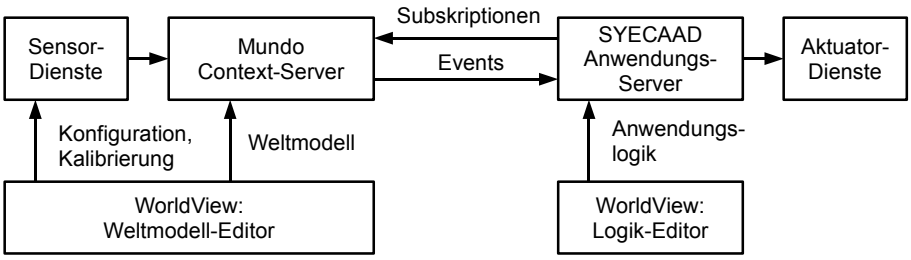

Abb. 1. Datenflüsse zwischen Werkzeugen und Diensten

3.1 MundoCore

Die Kommunikations-Middleware MundoCore [2] wurde speziell für die Anforderungen im Ubiquitous Computing entwickelt. MundoCore basiert auf einem Microkernel-Design, kann dynamisch rekonfiguriert werden und bietet eine einheitliche API für verschiedene Programmiersprachen (Java, C++, Python) auf unterschiedlichsten Geräten an. Das Architekturmodell deckt eine passende Spracheinbindung, verschiedene Kommunikations-Abstraktionen, Transportprotokolle, Aufrufprotokolle, Peer-to-Peer Overlays und automatisches Peer-Discovery ab. Als Integrationsplattform erlaubt MundoCore die Konstruktion von Systemen aus einer Menge heterogener Dienste. Die unterschiedlichen Implementierungen von MundoCore sind auf der Protokoll-Ebene interoperabel. MundoCore setzt einen Precompiler ein, um den Code für Serializer, Client- und Server-Stubs zu erzeugen. Dieser statisch erzeugte Code kann auf allen Plattformen eingesetzt werden, auch wenn diese z.B. keine Reflection unterstützen.

Die C++-Version von MundoCore ist für den Einsatz auf eingebetteten Systemen gut geeignet. Die Middleware wurde für die Plattformen Desktop/Server-Windows, Windows CE, Mac OS/X, SunOS und Linux/ucLinux portiert. Diese Version ist auch wichtig, um andere Systeme in die MundoCore-Infrastruktur integrieren zu können. Die Integrationspunkte mit anderen Systemen stehen oft nur für bestimmte Plattformen und Programmiersprachen zur Verfügung. So erfolgt z.B. die Anbindung an das Ubisense-Ortungssystem über die C++-API von Ubisense auf einem Linux-Server. Der Sprachausgabe-Dienst wird über das C++-API des Microsoft Speech SDK auf einem Windows-Rechner benutzt.

Während die C++-Version verstärkt zur Implementierung von systemnahen Diensten verwendet wird, werden Dienste auf den höheren Ebenen vorwiegend basierend auf der Java-Version implementiert. MundoCore unterstützt mehrere Programmierparadigmen. Verteilte objektorientierte Programmierung mit entfernten Methodenaufrufen sind ein gängiges Konzept zur Implementierung verteilter Systeme. Ereignisbasierte Publish/Subscribe Kommunikation ist gut geeignet zur Verteilung von Kontextinformationen im System, da Multicasting unterstützt wird und eine gute Entkopplung von Sensoren und Konsumenten der Sensordaten erreicht wird. In intelligenten Umgebungen spielt auch zunehmend Streaming von Multimedia- und Sensordatenströmen eine Rolle, z.B. für verteilte Spracherkennung. Die Middleware muss solche Streaming-Anwendungen durch andere Netzwerkprotokolle (z.B. UDP anstatt TCP) und Ratenkontrolle unterstützen, daher müssen Anwendungen gewisse Quality-of-Service Parameter definieren können.

Die Kommunikation zwischen Diensten wird von MundoCore übernommen. Dabei werden üblicherweise auch entfernte Methodenaufrufe basierend auf dem Publish/Subscribe-System verwendet. Dadurch steht dem Client automatisch ein einfacher Namensdienst zum Finden von Objekten im verteilten System zur Verfügung, um Ortstransparenz zu erreichen. Zusätzlich können Dienste an einem beliebigen Ort im System gestartet werden. Die Middleware ermöglicht somit auch Ausführungstransparenz. Dienste werden innerhalb sogenannter *Service Hosts* ausgeführt. Solche Service Hosts sind z.B. der Context-Server oder der Application-Server, die im nächsten Abschnitt vorgestellt werden.

3.2 Context-Server

Aufgabe des Context-Servers ist die Transformation von Sensormessungen zu Kontextinformationen, die bedeutsam für Anwendungen sind. Die Funktionen des Context-Servers sind:

- Interpretieren von Sensordaten und transformieren dieser Daten in einheitliche Darstellungen.
- Vorhalten eines 2D oder 3D-Weltmodells der intelligenten Umgebung, Unterstützung von geometrischen Operationen und Abfragemöglichkeiten.
- Schließen von “höherwertigem” Kontext aus “niederwertigem” Kontext.
- Feststellen von Kontextänderungen und benachrichtigen der Anwendung.
- Speichern von Kontextdaten und Abfragemöglichkeiten über eine Abfragesprache seitens Anwendungen.

Sensordaten werden von den Sensordiensten unter Verwendung von MundoCore an den Context-Server übertragen. Da diese Sensordaten sehr heterogen sind, führt der Context-Server diese daher zunächst in einheitliche Datenrepresentationen über. Zum Beispiel werden die Positions- und Orientierungsdaten unterschiedlicher 3D-Tracking-Systeme in ein gemeinsames Koordinatensystem transformiert. In ähnlicher Weise werden die unterschiedlichen Datenformate und Events von RFID-Lesern, Transponder-Lesern und Infrarot-Badges in einen

einheitlichen ID-Datentyp und Reader-Eventtyp übersetzt. Bis zu diesem Punkt stehen die IDs für Tags, Transponder, oder Badges. Der nächste Schritt in der Verarbeitungskette ist die Übersetzung dieser IDs auf die IDs von Personen oder Objekten, die diese Tags tragen.

Das Weltmodell ist das virtuelle Gegenstück der realen Umgebung in welchem die Anwendung ausgeführt wird. Es handelt sich dabei um ein detailliertes geometrisches Modell, das Wände, Objekte, etc. enthält. Objekte oder räumliche Bereiche im Modell können mit Metadaten annotiert werden. Das Weltmodell dient einerseits zur Abfrage von Metadaten für gegebene Koordinaten und kann zu Visualisierungszwecken eingesetzt werden. Mit Hilfe des Weltmodells ist der Context-Server in der Lage, höherwertigen Kontext aus den Daten von Positions- und Orientierungssensoren, z.B. in welchem Raum sich der Benutzer gerade aufhält, welche Objekte sich in der Nähe des Benutzers befinden, oder welches Objekt der Benutzer gerade anschaut.

Anwendungen können nun Subskriptionen an den Context-Server vornehmen, um ihr Interesse an bestimmten Kontextänderungen auszudrücken. Der Context-Server sendet dann Notifikationen an die Anwendung, wenn sich die entsprechenden Kontextinformationen ändern. Diese Notifikationen beschreiben höherwertigen Kontext, der bereits von den darunterliegenden Sensoren abstrahiert ist. Datenbank-Widgets können verwendet werden, um Kontextdaten für einen gewissen Zeitraum abzuspeichern. Die Anwendung kann auf die gespeicherten Daten dann über eine Abfrageschnittstelle zugreifen.

Die Anwendungsdienste werden typischerweise in Java implementiert. Die Kommunikations-Middleware MundoCore stellt die notwendige Grundlage für die Kommunikation zwischen Diensten im verteilten System bereit. Der Großteil der Verarbeitung von Kontextdaten findet auf dem Context-Server statt. Die Aspekte der Kommunikation und Kontextverarbeitung sind daher weitgehend aus den eigentlichen Anwendungsdiensten ausgelagert. Das vereinfacht die Anwendungsentwicklung.

4 Software-Entwicklung

In der Ubiquitous Computing-Forschung werden zahlreiche Prototypen gebaut. Vor allem sind Anwendungen gefragt, die sich im alltäglichen Einsatz bewähren können. Viele dieser Szenarien sind relativ einfach und geradlinig zu implementieren. Viele Anwendungen werden allerdings nie gebaut, da der gesamte Entwicklungsprozess immer noch relativ kompliziert ist. Die Erstellung einer neuen Anwendung erfordert typischerweise die folgenden Schritte:

- Einrichten der Entwicklungsumgebung: Bevor man mit der Programmierung beginnen kann, muss die IDE gestartet werden. Man muss ein neues Projekt erstellen und abhängig von der Art und Anzahl der Bibliotheken, die verwendet werden, kann dieser Vorgang einige Minuten in Anspruch nehmen. Wenn man außerdem solche Anwendungen nicht täglich programmiert, sind unter Umständen auch Updates der Software oder Dokumentation notwendig.

– Programmierung: Die Implementierung der Anwendungsdienste durch den Programmierer nimmt den Hauptteil der Entwicklungsphase ein.
– Testen: Während der Entwicklung muss das Programm getestet werden. Beim Testen von ortssensitiven Anwendungen ist es oft auch erforderlich, physikalisch Tags in der Umgebung bewegen zu müssen. Dieser Vorgang kann viel Zeit im Anspruch nehmen.
– Deployment: Schließlich müssen die programmierten Anwendungsdienste auf einem Server installiert werden, auf dem sie permanent laufen können. In diesem Schritt müssen Dateien auf den Server kopiert werden und Konfigurationsdateien bearbeitet werden, damit die Dienste gestartet werden.

Um diesen Entwicklungsvorgang zu vereinfachen, haben wir eine Reihe von allgemeinen Diensten und Werkzeugen erstellt, um diesen Prozess so einfach und effizient wie möglich zu gestalten. Ein Ziel war, einer möglichst breiten Benutzergruppe mit technischem Basiswissen, aber nicht notwendigerweise Programmierkenntnissen, solche Anwendungen zu erstellen oder auf ihre Bedürfnisse anpassen zu können.

Abbildung 2 zeigt den erweiterten Software-Entwicklungsprozess für intelligente Umgebungen. In vielen Phasen der Entwicklung reichen die bestehenden Werkzeuge, z.B. eine IDE zur Java-Programmierung, oder JUnit für Unit-Tests, aus. In bestimmten Phasen bietet sich allerdings die Verwendung von zusätzlichen Werkzeugen an.

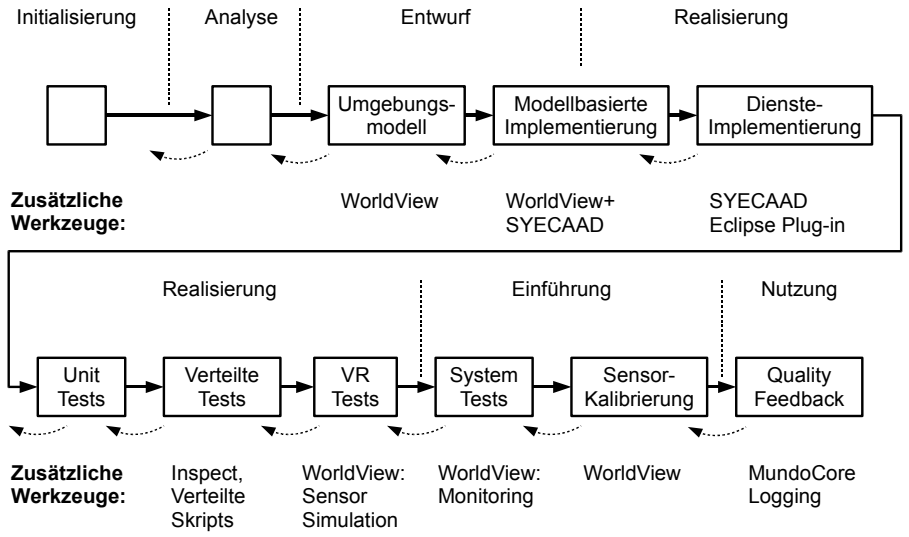

Abb. 2. Werkzeugunterstützung für die unterschiedlichen Phasen der Software-Entwicklung, basierend auf dem Wasserfall-Modell

4.1 Entwurfsphase

WorldView ist ein vielseitiges Werkzeug, das die Phasen Modellierung, Implementierung und Testen unterstützt. In der Modellierungsphase wird WorldView verwendet, um ein räumliches Modell der intelligenten Umgebung zu erstellen. Im Kartenfenster zeigt das Programm einen Gebäudeplan an und bietet damit eine Übersicht der verfügbaren Ressourcen und wo sich diese befinden. Solche Ressourcen sind z.B. Tags, Badges, Sensoren der unterschiedlichen Ortungssysteme, Wandbildschirme, oder elektronische Türschilder (Abbildung 3).

In WorldView ist es auch möglich, sogenannte areas-of-interest zu definieren. Das sind räumliche Bereiche beliebiger Form, die mit Metadaten versehen werden können. Diese Daten können in das Weltmodell des Context-Servers übertragen werden und dort von Kontext-Widgets verwendet werden. Damit können kontextsensitive Anwendungen Subskriptionen unter Angabe der semantischen Ortsinformation vornehmen, anstatt mit Koordinaten des unterliegenden Sensorsystems arbeiten zu müssen.

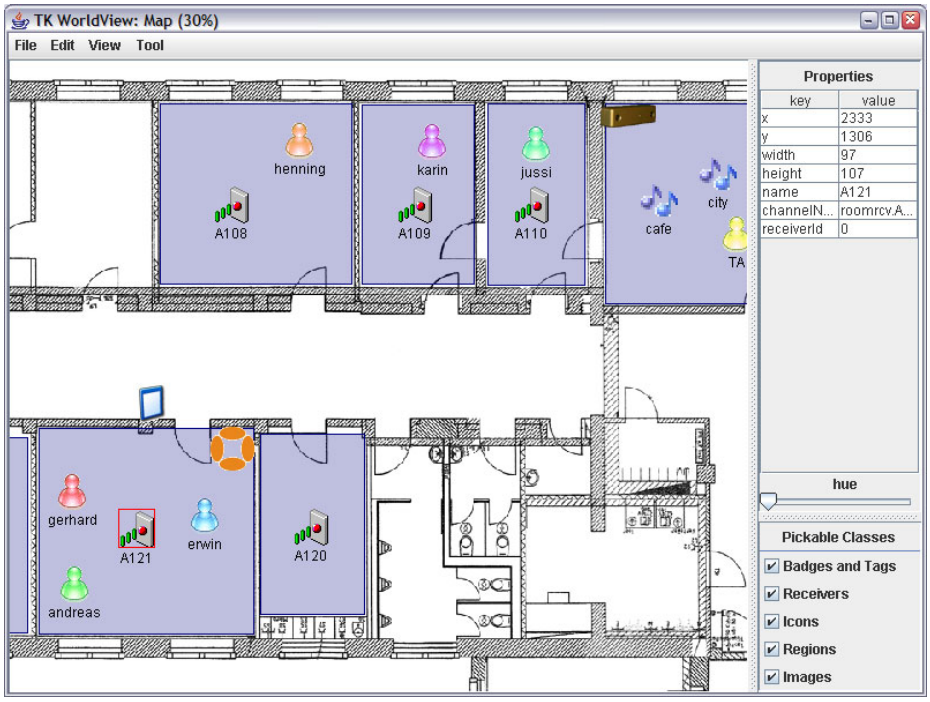

Abb. 3. Editor für das 2D-Umgebungsmodell in WorldView

4.2 Realisierungsphase

Das SYstem for Easy Context Aware Application Development (SYECAAD) [9] ermöglicht die rasche Entwicklung von kontextsensitiven Anwendungen. Einfach strukturierte Anwendungen können vollständig mit diesem Werkzeug realisiert werden, während für komplexere Systeme Prototypen erstellt werden können. Anwendungen werden dabei basierend auf einem graphischen Modell aufgebaut. Die Grundbausteine sind Funktionsblöcke, die beliebig untereinander verbunden werden können.

SYECAAD basiert auf einer Client/Server-Architektur. Die Anwendung wird serverseitig ausgeführt. Clients können sich zum Server verbinden und Anwendungen administrieren, bearbeiten und testen. Der Application Logic Editor in WorldView ist ein solcher Client, um Anwendungen zu bearbeiten. WorldView zusammen mit Context- und Anwendungs-Servern adressieren alle zuvor genannten Aspekte der Anwendungsentwicklung. Der erste Schritt, nämlich das Einrichten der Entwicklungsumgebung entfällt, da die Anwendungslogik zentral auf dem Anwendungsserver abgelegt ist. Die Entwicklungsumgebung ist eine Client-Applikation, die sich auf diesen Server verbindet. Auf diese Weise kann der Editor die Anwendung vom Server laden, anzeigen und bearbeiten. Das Deployment auf dem Anwendungsserver kann durch Klicken eines Buttons bewerkstelligt werden. Zum Testen der Anwendung stellt die Entwicklungsumgebung Funktionen bereit, um Sensoren zu simulieren und das laufende System zu überwachen.

Es werden drei verschiedene Typen von Funktionsblöcken unterschieden: Sensoren, Operationen und Aktuatoren. Ein Sensorblock empfängt Daten über das Event-System entweder direkt vom Sensor oder vorverarbeitete Daten vom Context-Server. Operationen führen logische oder arithmetische Operationen aus, implementieren Dictionaries, rendern HTML-Seiten, etc. Auf der Ausgabeseite werden Aktuatorblöcke eingesetzt, um die intelligente Umgebung zu steuern. Aktuatoren können über MundoCore Events verschicken oder entfernte Methodenaufrufe durchführen. Dadurch können Applikationen z.B. steuerbare Steckerleisten ein- und ausschalten, Beamer steuern, emails senden, SMS-Kurznachrichten senden, Instant Messenger-Nachrichten senden, oder Informationen auf einem intelligenten Türschild anzeigen. Ein elektronisches Türschild ist ein 8-Zoll Farb-TFT-Display mit einem Touchscreen, das ein herkömmliches Türschild ersetzt.

Functional assemblies dienen als Modularisierungs-Konzept in SYECAAD. Assemblies können unabhängig voneinander installiert, entfernt, neu gestartet, oder bearbeitet werden. In der Praxis wird so die Logik für jeden Raum eines Gebäudes in einer separaten Assembly modelliert. Funktionsblöcke erzeugen ihre Ausgabezustände entweder basierend auf einem ereignisgesteuerten oder zustandsgesteuerten Modell. Wenn sich der Wert den ein Sensor liefert ändert, dann müssen alle davon abhängigen Operationen ihre Zustände neu berechnen. Wird ein Assembly neu gestartet, so können alle Funktionsblöcke unmittelbar ihren Zustand bestimmen.

Wenn die Standard Sensor-, Operations-, und Aktuatortypen nicht ausreichen, so kann das System durch Programmierung von neuen Funktionsblöcken in Java erweitert werden. Es wurde ein Plug-In für die Eclipse-IDE entwickelt, die den Softwareentwickler beim Erstellen neuer Funktionsblöcke unterstützt und ihm verschiedene Code-Templates erstellen kann und gezielt Hilfedokumente bereitstellt.

4.3 Programmierung von Diensten

Um die Entwicklung und das Testen zu unterstützen, hat das MundoCore-Framework einige Erweiterungen. So gibt es Synchronisationsklassen, die eine Deadlock-Erkennung enthalten. Mittels Checkpointing kann festgestellt werden, an welcher Stelle ein Programm blockiert. Das ist hilfreich, wenn z.B. ein Systemaufruf unerwarteterweise blockiert (Beispiel: `new Socket()`). Die C++-Version unterstützt außerdem Heap-Debugging und System Resource Tracking. Diese Mechanismen dienen dazu, um festzustellen, welche Speicherblöcke bzw. Systemressourcen vom Anwendungscode angefordert, aber nicht mehr freigegeben werden. Es wird auch gespeichert, an welcher Stelle im Quellcode diese Ressourcen angefordert wurden, um die Fehlersuche zu vereinfachen.

4.4 Testphase

Um Anwendungen zu testen, kann WorldView verwendet werden, um Sensoren zu simulieren. Zum Beispiel kann der Benutzer unterschiedliche Tracking-Systeme simulieren, in dem er Symbole auf der Karte herumbewegt, anstatt Tags physikalisch herumbewegen zu müssen. WorldView erzeugt dabei die gleichen Events, wie auch das echte Tracking-System erzeugen würde.

Implementierungen von abstrakten Datentypen und kleinere Teile eines Frameworks können gut mit *Unit Tests*, z.B. mit dem Framework JUnit getestet werden. Um allerdings das korrekte Verhalten eines verteilten Systems zu überprüfen, ist es auch notwendig, verteilte Tests durchzuführen.

Verteilter Skript-Interpreter

Um solche verteilten Tests durchführen zu können, haben wir einen verteilten Skript-Interpreter basierend auf MundoCore implementiert. Zunächst müssen Script-Slave-Prozesse auf mehreren Rechnern im Netzwerk gestartet werden. Ein Master-Interpreter steuert diese Slave-Prozesse, verteilt Aufgaben an sie und sammelt die Ergebnisse ein. Da die Interpreter MundoCore zur Kommunikation verwenden, finden sich diese im Netzwerk automatisch, ohne manuelle Konfiguration.

Um einen Test durchzuführen, wird der Dateiname einer XML-Skript-Datei an den Master-Interpreter übergeben. Wenn der Master-Interpreter auf das Kommando `execRemote` stößt, dann wird auf einem entfernten Interpreter ein neuer Prozess gestartet und das im Tag enthaltene Teilskript dort ausgeführt. Der Slave-Interpreter führt das Skript aus und gibt die Ausgaben von standard-out und standard-error an den Master-Interpreter zurück. Der Master-Interpreter

schreibt alle Ausgaben in eine Log-Datei. Im Weiteren geben die Shell-Return-Codes an, ob ein Skript erfolgreich abgelaufen ist, oder nicht. Ein Test-Skript kann die folgenden Kommandos enthalten:

- `send`: Sendet eine Nachricht über das Publish/Subscribe-System. Damit ist es auch möglich, beliebige Methoden von MundoCore-Diensten aufzurufen.
- `expect`: Erwartet eine bestimmte Nachricht vom Publish/Subscribe-System. Hier kann ein Filterausdruck spezifiziert werden, der von der ankommenden Nachricht erfüllt werden muss.
- `exec`: Führt einen lokalen Prozess aus. Es kann sich dabei um ein weiteres Skript handeln oder um ein beliebiges anderes Programm. Einfache Tests, z.B. zum Test, ob Nachrichten richtig über das Publish/Subscribe-System übertragen werden, können in Skripts definiert werden. Komplexere Tests müssen z.B. in Java programmiert werden. Der Skript-Interpreter wird in diesem Fall zusätzlich zum Verteilen der Java-Programme im Netzwerk eingesetzt.
- `execRemote`: Startet ein Skript auf einem entfernten Slave-Interpreter.

Ein Skript kann beliebig viele Prozesse auf einem Rechner parallel starten. So ist es z.B. möglich, ein Netzwerk von 16 Testrechnern auf 4 Rechnern zu emulieren.

Um die vom Master-Interpreter ausgegebene Log-Datei aufzubereiten, haben wir ein weiteres Tool ("Log Distiller") entwickelt. Durch Angeben von Filtern ist es möglich, nur bestimmte Log-Einträge weiter auszuwerten. Der Distiller berechnet die Offsets der Systemuhren der Slave-Interpreter, bringt die Ereignisse in die richtige zeitliche Abfolge und stellt die Ausgaben der parallel ablaufenden Prozesse nebeneinander in mehreren Spalten dar.

Inspect

Das MundoCore Inspect-Werkzeug wird verwendet, um das laufende System zu überwachen. Diese Anwendung verbindet sich mit dem `DebugService` eines beliebigen lokalen oder entfernten MundoCore-Prozesses und erlaubt es, diesen zu untersuchen. Es ist möglich, die Transportverbindungen, Routing-Tabellen des Publish/Subscribe-Systems, und Nachrichten anzusehen. Dienste können administriert, konfiguriert und die Schnittstellen inspiziert werden. Es können auch direkt Nachrichten in das System gesendet werden oder entfernte Methodenaufrufe über einen generischen Client ausgeführt werden.

VR-Tests

Zum Test von Anwendungen kann WorldView benutzt werden, um Sensoren zu simulieren. Um ein Tracking-System zu simulieren, werden auf der Kartenansicht die entsprechenden Symbole mit der Maus bewegt. WorldView erzeugt dann die gleichen Nachrichten, wie es auch das echte Tracking-System erzeugen würde. Dazu unterstützt WorldView die folgenden Sensortypen:

- Mundo Badge System: In der Kartenansicht werden die Standorte von IR-Empfängern und deren Empfangsbereiche modelliert. Wird ein Tag-Symbol

in den Empfangsbereich eines Empfängers geschoben, dann beginnt WorldView, die entsprechenden Events zu erzeugen.
- IRIS: IRIS (InfraRed Indoor Scout) ist ein präzises IR-Tracking System, das auf Stereo-Sicht beruht [3]. Auch hier können die entsprechenden Tags in der Kartenansicht bewegt werden und WorldView erzeugt die entsprechenden Nachrichten.
- Talking Assistant: Der Talking Assistant [4,1] verfügt über einen Orientierungssensor. Auch dieser Sensor kann simuliert werden.

4.5 Einführungsphase

Wenn die Ergebnisse des VR-Tests zufriedenstellend sind, kann die Anwendung zusammen mit den echten Sensoren getestet werden. Hier kann WorldView die Sensordaten der zuvor beschriebenen Systeme darstellen. Dazu können z.B. die Daten der unterschiedlichen Tracking-Systeme auf der Karte visualisiert werden. Wenn ein Tag physikalisch bewegt wird, so folgt auch das entsprechende Symbol auf der Karte in Echtzeit. Zusätzlich werden die folgenden Funktionen unterstützt.

- Personalisierung: Der Talking Assistant kann mittels einer Smartcard personalisiert werden, die den Benutzer identifiziert. Wird ein TA personalisiert, so wird in der Kartenansicht der Name der Person angezeigt, anstatt der Identifizierung des Gerätes.
- Context-Server-Client: Diese Funktion erlaubt es, interaktiv Subskriptionen beim Context-Server zu definieren und die Ergebnisse zu visualisieren. Der Filter einer Subskription wird in Form einer Python-Funktion beschrieben.

4.6 Nutzungsphase

WorldView kann verwendet werden, um das laufende System zu überwachen. Dazu können z.B. die Daten der unterschiedlichen Tracking-Systeme auf der Karte visualisiert werden. Wenn ein Tag physikalisch bewegt wird, so folgt auch das entsprechende Symbol auf der Karte in Echtzeit.

Im Weiteren bringt das MundoCore-Framework Unterstützung für Logging mit. Die entsprechenden Klassen sind auf allen Plattformen verfügbar. Es ist somit auch Logging auf den CDC und CLDC Java-Profilen möglich, deren Standard-Frameworks keine Logging-Klassen enthalten. Auf manchen Geräten ist das Lesen von Logs nur schwer möglich. Ein Programm, das auf einem Mobiltelefon läuft, kann z.B. Log-Ausgaben schreiben und man kann sich mit dem Inspect-Tool auf dieses Gerät verbinden und die Log-Ausgaben auslesen.

5 Zusammenfassung

Die rasche Entwicklung von Prototypen spielt eine wichtige Rolle in der Ubiquitous Computing-Forschung. Das Mundo Smart Environments-System mit seiner flexiblen Plattform, den wiederverwendbaren Diensten und den Werkzeugen

stellt eine leistungsfähige Basis für solche Anwendungen dar. Der beschriebene Prozess ist ein weiterer Beitrag, um die Entwicklung von Anwendungen für Smart Environments zu systematisieren.

Literaturverzeichnis

1. Erwin Aitenbichler, Jussi Kangasharju, and Max Mühlhäuser. Talking Assistant Headset: A Smart Digital Identity for Ubiquitous Computing. In *Advances in Pervasive Computing*, pages 279–284. Austrian Computer Society (OCG), 2004.
2. Erwin Aitenbichler, Jussi Kangasharju, and Max Mühlhäuser. Experiences with MundoCore. In *Third IEEE Conference on Pervasive Computing and Communications (PerCom'05) Workshops*, pages 168–172. IEEE Computer Society, March 2005.
3. Erwin Aitenbichler and Max Mühlhäuser. An IR Local Positioning System for Smart Items and Devices. In *Proceedings of the* 23^{rd} *IEEE International Conference on Distributed Computing Systems Workshops (IWSAWC03)*, pages 334–339. IEEE Computer Society, May 2003.
4. Erwin Aitenbichler and Max Mühlhäuser. Audiobasierte Endgeräte für Ubiquitous Computing und geeignete Infrastrukturen. In *HMD - Praxis der Wirtschaftsinformatik: Ubiquitous Computing*, pages 68–80. dpunkt Verlag Heidelberg, 2003.
5. David Garlan, Daniel P. Siewiorek, Asim Smailagic, and Peter Steenkiste. Project Aura: Toward Distraction-Free Pervasive Computing. *Pervasive Magazine*, 1(2):22–31, April 2002.
6. Richard Hull, Ben Clayton, and Tom Melamed. Rapid Authoring of Mediascapes. In *Ubicomp 2004*, 2004.
7. Brad Johanson, Armando Fox, and Terry Winograd. The Interactive Workspaces Project: Experiences with Ubiquitous Computing Rooms. *Pervasive Magazine*, 1(2):71–78, April 2002.
8. Manuel Román, Christopher Hess, Renato Cerqueira, Anand Ranganathan, Roy H. Campbell, and Klara Nahrstedt. A Middleware Infrastructure for Active Spaces. *Pervasive Magazine*, 1(4):74–83, October 2002.
9. Jean Schütz. SYECAAD: Ein System zur einfachen Erzeugung kontextsensitiver Applikationen. Master's thesis, Technische Universität Darmstadt, 2005.
10. Visonic Technologies. Elpas - Local Positioning Systems. http://www.visonictech.de/elpas.html (last visited: 22.09.2006), 2006.
11. Khai N. Truong, Elaine M. Huang, and Gregory D. Abowd. CAMP: A Magnetic Poetry Interface for End-User Programming of Capture Applications for the Home. In *Ubicomp 2004*, 2004.
12. Ubisense. Homepage. The Smart Space Company. http://www.ubisense.net/ (last visited: 22.09.2006), 2006.
13. Torben Weis, Marcus Handte, Mirko Knoll, and Christian Becker. Customizable Pervasive Applications. In *PerCom 2006*, 2006.