Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Projektbericht

André Goldflam

Integration des EventHeaps durch einen XML-Adapter

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

André Goldflam

Thema

Integration des EventHeaps durch einen XML-Adapter

Stichworte

Ubiquitous Computing, Ambient Living, Integration, Living Lab

Kurzzusammenfassung

In diesesm Projektbericht wird die Integration des EventHeaps durch einen XML-Adapter beschrieben. Die Integration wird im Rahmen des Living Place durchgeführt und ermöglicht Anwendungen auf .NET-Basis den Zugriff auf den EventHeap.

Inhaltsverzeichnis

1 Einleitung

Im Rahmen des „Living Place“-Projektes an der Hochschule für Angewandte Wissenschaften soll eine Laborumgebung für Realexperimente für den Bereich der intelligenten Wohnungen geschaffen werden. Dieser „Ambient Living Place“ ist dem wissenschaftlichen Gebiet der Ambient Intelligence zuzuordnen, welches wiederum ein Unterbereich des Ubiquitous Computing darstellt. Für diese Laborumgebung soll die Wohnung von Grund auf saniert und mit der nötigen Technik ausgestattet werden. Zusätzlich findet bereits eine Kooperation mit einer israelischen Universität statt, welche die Innenausstattung der intelligenten Wohnung gestalten und produzieren soll.

2 Motivation

Die Kommunikation zwischen verschiedenen intelligenten Geräten wurde in vorherigen Arbeiten über eine Blackboard-Architektur realisiert. Der bei der Umsetzung eingesetzte zentrale Server ermöglichte allerdings nur die Anbindung von Java-Clients. Matthias Vogt (Vogt (2008)) und Mohammed Ali Rahimi (Rahimi (2008)) arbeiten derzeit an einer auf .NET basierten Bibliothek für Multitouch-Gesten. Multitouch stellt innerhalb des Living Place eine wichtige Art der Eingabe dar, da sie natürlicher ist als eine Eingabe über Maus und Tastatur. In (Witt (2009)) beschäftigt sich Kristoffer Witt mit dem Thema der Spracherkennung sowie der Steuerung von Systemen durch Sprache. Die Implementierung seiner Projektarbeit wurde ebenfalls unter .NET durchgeführt. Die Motivation des Autors liegt nun darin, die Kommunikation zwischen intelligenten Geräten unabhängig von der Implementierungssprache zu ermöglichen.

3 relevante Projekte

Es wurden bereits mehrere Projekte im Rahmen des Living Place erstellt. Meißner (2007), Hollatz (2007) und Urich nutzen den EventHeap als zentrale Kommunikationseinheit in ihrem Projekt „smart:shelf“, welches sich ebenfalls im Bereich einer intelligenten Wohnung positionieren lässt. Sven Tennstedt (2008) und Burka (2008) nutzen ebenfalls den EventHeap zur Kommunikation für ihre Projekte aus dem Bereich der Ambient Awareness. Weitere Projekte, welche noch nicht im Rahmen eines Projektberichtes veröffentlicht worden sind, sind derzeit noch in Entwicklung.

4 Gliederung

Im Folgenden sollen für dieses Projekt relevante Bestandteile und Funktionsweisen des EventHeaps erläutert werden. Anschließend soll der Aufbau eines Events in XML dargestellt und der auf der Basis von .NET implementierte Client vorgestellt werden. Abschließend soll ein Fazit die durchgeführte Arbeit bewerten und einen Ausblick auf zukünftige Verbesserungsmöglichkeiten aufzeigen.

5 Szenario

Als Grundlage dieses Projektes wurde ein Szenario entworfen, welches die Elemente „Multitouch“, „Sprachsteuerung“ und „EventHeap“ vereint. Ein Benutzer soll ein Bild zwischen zwei Multitouch-Bildschirmen hin- und herbewegen können. Die Steuerung der Bewegung soll über Gesten möglich sein. Wenn ein Bild an den Rand des Darstellungsbereiches geschoben wird, soll es automatisch auf dem anderen Multitouch-Display angezeigt werden. Über einen Sprachbefehl soll der Benutzer Bilder aus einem bestimmten Ordner öffnen und zusätzlich eine Abspielsoftware für Musik steuern können. Die Bildschirme sind dabei an zwei unterschiedliche Computer angeschlossen.

In diesem Projektbericht wird dabei ausschließlich auf die Implementierung des XML-Adapters sowie dem notwendigen .NET-Client eingegangen, welcher für die Kommunikation zwischen den einzelnen Geräten benötigt wird.

6 iROS EvenHeap

Der bereits in der Einleitung genannte zentrale Kommunikationsserver basiert auf dem iROS EventHeap (Brad Johanson (2001)) der Stanford Universität. Der EventHeap ermöglicht es Anwendungen, selbst Events dem EventHeap hinzuzufügen, Events zu suchen, zu löschen oder zu abonnieren. Die Struktur eines Events ist selbst beschreibend, er enthält also alle Informationen, die notwendig sind um die repräsentierten Daten zu rekonstruieren. Im Folgenden soll der Aufbau des EventHeaps näher beschrieben werden. Anschließend soll näher auf den Event selbst eingegangen werden.

6.1 EventHeap

Innerhalb der intelligenten Wohnung soll der EventHeap als Kommunikationsvermittler zum Einsatz kommen. Dabei lassen sich innerhalb der Wohnung zwei Gruppen von Geräten identifizieren: Sensoren und intelligente Geräte. Die Sensoren liefern die Datenbasis, auf der die intelligenten Geräte ihre Entscheidungen fällen können. Die Anwendungen kommunizieren dabei nicht direkt miteinander, sondern nur über den EventHeap.

Ein Sensor sendet seine Daten an den EventHeap und dieser leitet die Informationen an Geräte weiter, welche diese Informationen abonniert haben. Um Events eines bestimmten Typs zu suchen, werden sogenannte TemplateEvents genutzt. Ein TemplateEvent kann als eine Schablone verstanden werden, wobei alle Events bei einer Suchanfrage zurückgeliefert werden, welche bei einem Vergleich mit diesem Muster passen.

Wenn eine kommunizierende Anwendung kurzfristig ausfallen sollte, bedeutet dies keinen zwangsläufigen Zusammenbruch des Kommunikationspartners, da nur dessen Nachrichten ausbleiben. Sollte ein Event nicht konsumiert werden, wird er nach einer gewissen Zeit verworfen.

Weiterhin bietet der EventHeap die Möglichkeit, nach Events zu „snoopen“. Dabei kann eine Anwendung die Kommunikation anderer Anwendung überwachen, ohne in die Kommunikation einzugreifen. Sie beobachtet dabei nur die Events, die zwischen den Anwendungen ausgetauscht werden.

6.2 Event

Der EventHeap basiert auf TSpaces von IBM (P. Wyckoff und Ford), welches eine Implementation von Tuple Spaces ist. Diese wurden erstmals von Nicholas Carriero beschrieben. Ein Event beinhaltet eine Liste aus mehreren Feldern, wobei ein Feld durch fest definierte Attribute beschrieben wird.

Im Folgenden sollen der für den EventHeap spezifische Aufbau der Felder, die vordefinierten Felder sowie die Typen der Wertebelegung eines Feldes vorgestellt werden. Abschließend soll auf das WireBundle eingegangen werden, welches als Transport-Container zur Übermittlung von Events dient.

Feld

Ein Feld besteht aus mehreren Attributen und gilt als selbst beschreibend. Dies bedeutet, dass in einem Feld dessen Name, der Typ und Informationen über die konkreten Werte im Fall eines

Hinzufügens bzw. einer Suche auf dem EventHeap gespeichert sind. Folgende Informationen werden in einem Feld gespeichert:

Type: Ein String, welcher den Typ des Feldes festlegt. Folgende Werte können dabei auftreten:

int, long, float, double: Der Wert des Feldes ist ein numerischer Typ.

string: Das Feld wird im Standard UTF-8-Format gespeichert.

EHJava.Fully qualifed Java class name: Ein Typ einer Java-Klasse, welche nicht den vorherigen Standard-Typen entspricht.

[otherplatform .fully qualified clas name:] Ein Typ einer Klasse einer anderen Plattform, welche nicht den obigen Standard-Typen entspricht.

Name: Der als UTF-8 String gespeicherter Name des Feldes.

Post Value: Der Wert, welcher genutzt wird, wenn der Event zum EventHeap hinzugefügt werden soll.

Template Value: Der Wert, welcher genutzt wird, wenn der Event zur Suche über den EventHeap genutzt werden soll.

Wie sich aus der Struktur des Feldes erkennen lässt, werden die Werte zur Suche und zum Hinzufügen separat gespeichert und beide jedes Mal übertragen, unabhängig davon, ob der Event zum Hinzufügen oder zum Suchen verwendet werden soll. Ein Feld kann entsprechend unterschiedliche Typen und Werte für das Post- und Template-Verhalten besitzen.

Vordefinierte Felder

Die vordefinierten Felder ermöglichen es zum einen, unabhängig von Anwendung und Gerät, die Quelle sowie das Ziel eindeutig zu identifizieren und zum anderen, um systeminterne Parameter (wie z.B. TimeToLive oder SessionID) abbilden zu können.

Notwendige Felder

EventType Legt den Typ des Events fest und grenz ihn somit von den anderen Events ab. Der EventType kann als eindeutiges Merkmal des Events verstanden werden.

Source Eindeutiger Identifikator der Quelle.

Target Die Id des Zieles. Dies kann, muss aber nicht der „Source“-Identifikator des Zieles sein.

SourceApplication Name der Quellanwendung.

TargetApplication Name der Anwendung, für welche der Event bestimmt ist.

SourceDevice Identifikator des Quellgeräts.

TargetDevice Identifiziert das Zielgerät.

TimeToLive Millisekunden, nachdem ein Event vom EventHeap entfernt wird.

Optionale Felder

SourcePerson Die Person, die für das Versenden des Objektes „verantwortlich“ ist.

TargetPerson Zielperson, an welche der Event übermittelt werden soll.

SourceGroup Beschreibt die Quell-Anwendungsgruppe welche diesen Event erzeugt hat.

TargetGroup Definiert die Ziel-Anwendungsgruppe, für welche dieser Event bestimmt ist.

Interne Felder

SessionID Dient zur Sequenzierung von Events.

SequenceNum Dient zur Einordnung und somit ebenso zur Sequenzierung von Events.

EventHeapVersion Dient zur Versionierung und trägt zur Kompatibilität von Events bei.

Typen von Werten

Über das Interface FieldValueTypes kann für ein Feld jeweils der postType sowie der template-Type festgelegt werden.

ACTUAL Zeigt an, dass das Feld einen konkreten Wert enthält.

FORMAL Legt fest, dass das Feld zwar vorhanden sein muss, der konkrete Wert für einen Vergleich bei einer Suche aber irrelevant ist.

VIRTUAL Das Feld ist wird bei einem Vergleich vollständig vernachlässigt.

AUTOSET Zeigt an, dass das Feld automatisch gesetzt und nicht überschrieben werden darf. Dieser Typ darf nur intern durch den EventHeap vergeben werden.

AUTOSET_OVERRIDEABLE Das Feld wird automatisch durch den EventHeap gesetzt, darf aber zukünftig verändert werden.

Die Typen AUTOSET und AUTOSET_OVERRIDEABLE dürfen dabei nur durch den EventHeap vergeben werden.

6.3 WireBundle

Zur Übertragung eines Events wird dieses in ein sogenanntes WireBundle verpackt. Das WireBundle enthält dabei Informationen darüber, wie ein Event verarbeitet und wie auf diesen reagiert werden soll. Das Attribtut destinationTag bestimmt, wie der Server auf das Event reagieren soll, z.B. wird bei destinationTag="'putEvent"' der Server angewiesen, die Events dem EventHeap hinzuzufügen. Wenn ein Client eine Anfrage stellt und der EventHeap ein Event zurückgibt, wird er dessen returnTag in der Antwort als destinationTag setzen. So kann ein Client festlegen, wie andere Clients, welche den Event verarbeiten, auf diesesn reagieren sollen.

7 XML-Event

Aus dem Aufbau eines Events konnte folgende XML-Definition abgeleitet werden:

```
<Tuple>
  <Field>
    <FieldName/>
    <Type/>
    <FieldValue Type="PostValue">
      <ValueType/>
      <Value/>
    </FieldValue>
    <FieldValue Type="TemplateValue">
      <ValueType/>
      <Value/>
    </FieldValue>
  </Field>
</Tuple>
```

Ein Tuple kann dabei aus mehreren Feldern bestehen, muss aber mindestens die im Abschnitt 6.2 als notwendig bezeichneten Felder enthalten.

Da ein Event an sich noch kein Kommand für den EventHeap enthält, wird es, wie bereits erwähnt, in ein WireBundle verpackt. Die Struktur eines WireBundles sieht in XML folgendermaßen aus:

```
<WireBundle>
  <destinationTag/>
  <returnTag/>
  <TupleList>
```

```
    <Tuple/>
  </TupleList>
</WireBundle>
```

8 .NET-Adapter auf Basis der XML-Events

Im Folgenden soll der Aufbau des in .NET implementierten Clients vorgestellt werden. Dabei soll zuerst auf die Umsetzung der Event-Struktur und anschließend auf die Realisation der Kommunikation eingegangen werden.

8.1 Event

Auf Basis des XML-Events wurde ein .NET-Client implementiert. Der Aufbau der Event-Hierarchie richtet sich dabei nach der Implementation der Events im EventHeap. Da der .NET-Client auf die XML-Serialisierung aufbaut, ist eine deutlich sauberere Implementierung möglich gewesen als in dem ursprünglichen EventHeap. Zur Wahrung der Kompatibilität wurden die Namen der vorgegebenen Felder (im Diagramm die Enums FieldNames und FieldValues innerhalb der Klasse Event) sowie die Typen eines Feldes (Konstanten in der FieldType-Klasse) entsprechend übernommen. Abbildung 8.1 illustriert anhand eines Klassendiagramms den Aufbau der Event-Struktur.

8.2 EventHeapClient

Der Client zur Kommunikation mit dem EvenHeap für .NET ist unabhängig von der internen Struktur des Events, Abbildung 8.2 verdeutlicht den Aufbau des .NET-Clients. Der Aufgabenbereich des EventHeapClients besteht in der Steuerung der Kommunikation zwischen .NET-Anwendung und dem EventHeap, also dem Senden und Empfangen von Events sowie der Entscheidung, welche Operationen für die gesendeten Events auf dem EventHeap ausgeführt werden sollen. Die möglichen Operationen werden in der IEventHeapCore Schnittstelle definiert. Die Basisoperationen werden bereits durch den EventHeapClient abgedeckt: Einfügen (putEvent), Suchen (getEvent), Snoopen (snoopEvent), Löschen (removeEvent) und Abonnieren von Events(registerForEvent) sind implementiert. Derzeit werden noch nicht alle von dem EventHeap möglichen Operationen unterstützt, diese können allerdings bei Bedarf leicht nachimplementiert werden.

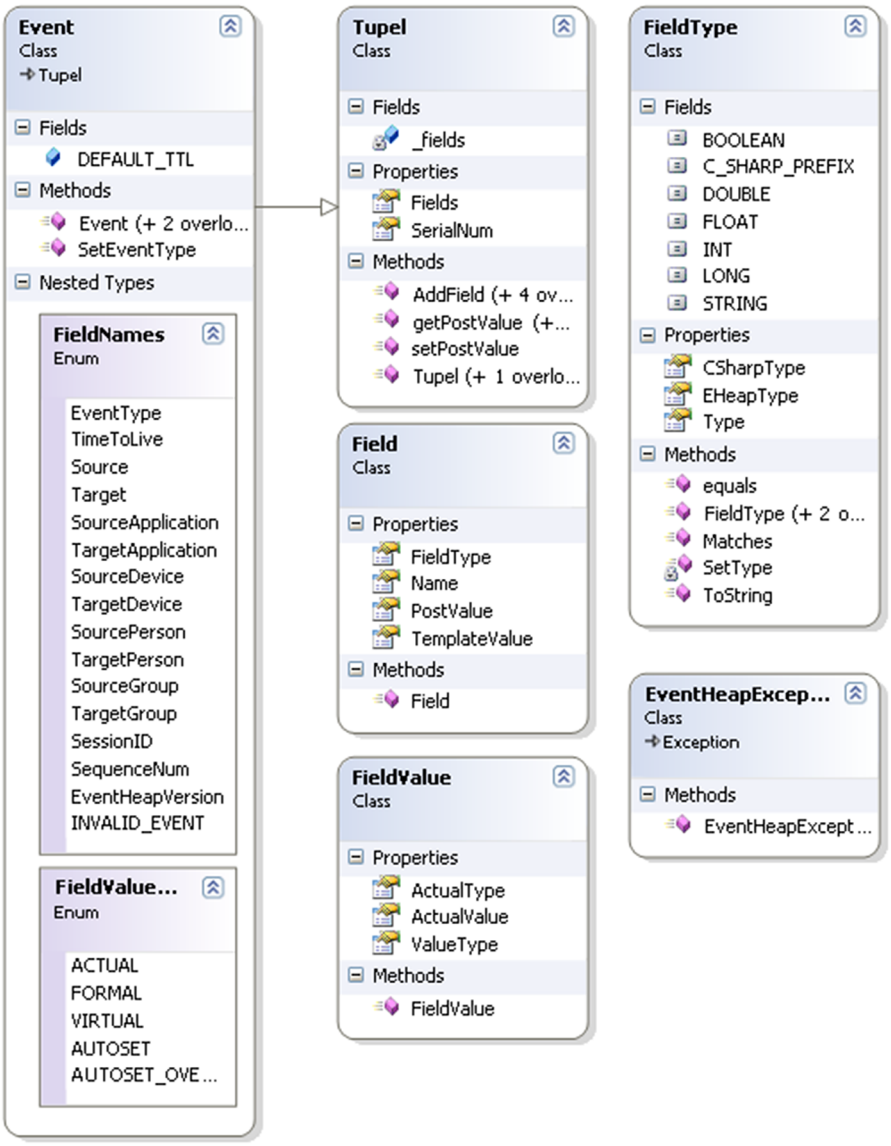

Abbildung 1: Event-Klassenübersicht

Das Senden und Empfangen wird jeweils über eine BlockingQueue realisiert. So können mehrere Operationen hintereinander abgewickelt werden, ohne dass die Anwendung auf die erfolgreiche Übermittlung warten muss. Sämtliche Kommunikation ist asynchron.

Die Serialisierung der WireBundle- sowie der Event-Objekte wurde über die XmlDocument-

Klasse aus der System.Xml-Bibliothek realisiert. Der Versuch, die Objekte automatisch über Attribute serialisieren zu lassen, scheiterte, da sich die Struktur eines Events als zu kompliziert herausstellte.

Der EventHeapClient kann dabei leicht in eigene Anwendungen integriert werden, da über den EventReceived-Delegate einfach die empfangenen Events verarbeitet werden können.

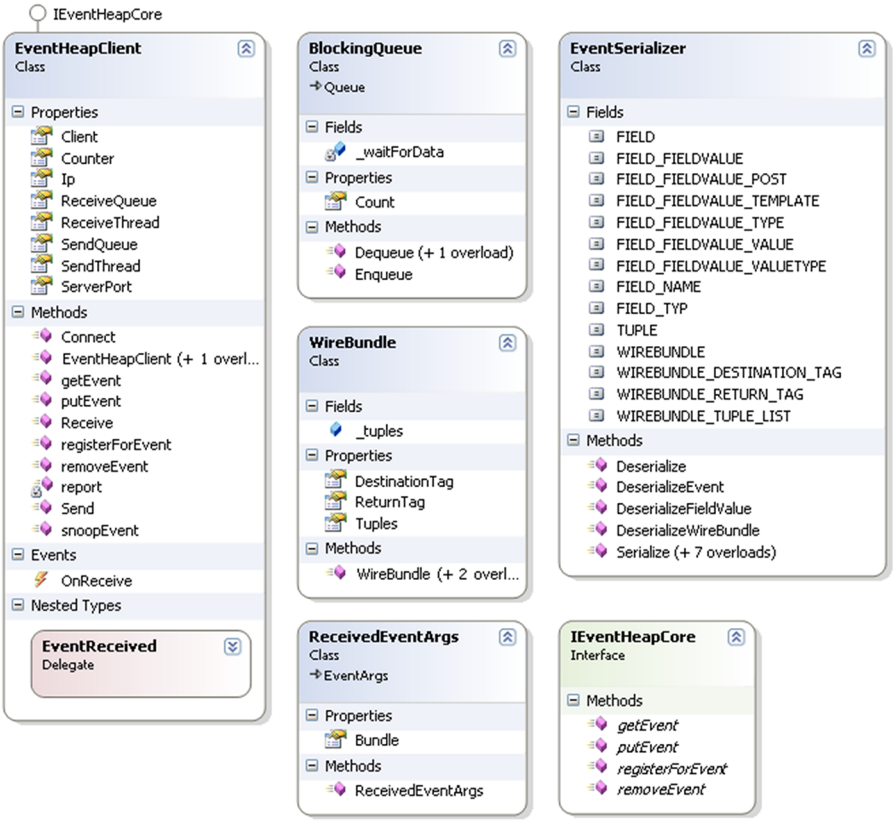

Abbildung 2: Klassenübersicht des .NET Event-Clients sowie der Serialisierungs-Klasse für ein Event

8.3 Erweiterungen des EventHeaps

An dieser Stelle sollen die notwendigen Erweiterungen an dem EventHeap vorgestellt werden, welche notwendig sind, um eine Serialisierung über XML zu ermöglichen. Unabhängig von der Serialisierung musste zuerst der Core angepasst werden, da die vorliegenden Cores nicht automatisch die SequenceNum und SessionId vergaben und daher Fehler auftraten, sobald

komplexere Operationen auf dem EventHeap durchgeführt wurden. Der Fehler konnte einfach umgangen werden, indem der bestehende Core kopiert und die notwendigen Felder automatisch hinzugefügt wurden.

Die De- bzw. Serialisierung von Events ist in der Klasse WireBundle gekapselt. Entsprechend musste diese Stelle den Bedürfnissen der der neuen Serialisierung über XML angepasst werden. Die Serialisierung an sich wurde durch die Klasse XmlEventHeapSerializer realisiert. Sie stellt das Gegenstück zur Serialisierungsklasse in .NET dar.

9 Fazit

Die Implementierung eines XML-Adapters sowie die Erweiterung des EventHeaps konnten erfolgreich durchgeführt werden. Die für das Szenario implementierten Anwendungen konnten über den EventHeap kommunizieren und die Bilder von einem Multitouch-Bildschirm zum anderen verschieben. Der derzeitige Stand dient somit als Basis für zukünftige Anwendungen, die den EventHeap über XML ansprechen müssen, da sie nicht die bisher notwendige Java-Serialisierung unterstützen. Erweiterungen erscheinen zu diesem Zeitpunkt unproblematisch. Der implementierte Client sollte zusätzlich ausreichend getestet werden, unter anderem zur Überprüfung der Performance und Stabilität.

Literatur

[Brad Johanson 2001] BRAD JOHANSON, Pat Hanrahan Terry W.: The Event Heap: An Enabling Infrastructure for Interactive Workspaces. (2001)

[Burka 2008] BURKA, Florian: *Ambient Awareness - Federfarbener Tunnel*. 2008

[Hollatz 2007] HOLLATZ, Dennis: *smart:shelf*. 2007

[Meißner 2007] MEISSNER, Stefan: *smart:shelf*. 2007

[P. Wyckoff und Ford] P. WYCKOFF, T. J. L. ; FORD, D. A.: TSpaces. In: *IBM Systems Journal 37(3)*. – URL http://www.almaden.ibm.com/cs/TSpaces

[Rahimi 2008] RAHIMI, Mohammed A.: *Multitouch: Out of the shelf. Gestenbasierte Interaktion für verfügbare Applikationen*. 12 2008

[Sven Tennstedt 2008] SVEN TENNSTEDT, Julia P.: *Emotional Tent - Ambient Awareness*. 2008

[Urich] URICH, Jaroslaw: *smart:shelf*

[Vogt 2008] VOGT, Matthias: *Berühren durch Begreifen*. 12 2008

[Witt 2009] WITT, Kristoffer: *Untersuchung der Eignung von Spracherkennung zur Transkription von Radiospots*. 01 2009