Der Einsatz von Jini für die Realisierung durchgängiger Steuerungskonzepte in verteilten eingebetteten Systemen*

Stephen Schmitt[1] und Wolfgang Rosenstiel[1,2]

[1] Forschungszentrum Informatik
an der Universität Karlsruhe
Haid-und-Neu-Str. 10-14
D-76131 Karlsruhe
sschmitt@fzi.de,
WWW home page: http://www.fzi.de/sschmitt.html
[2] Eberhard-Karls-Universität Tübingen
Wilhelm-Schickard-Institut für Informatik
Technische Informatik
Sand 13, D-72076 Tübingen
rosenstiel@informatik.uni-tuebingen.de,
WWW home page: http://www-ti.informatik.uni-tuebingen.de/~rosen

Zusammenfassung Mit der zunehmenden Automatisierung von Produktionsprozessen, und den damit einhergehenden komplexeren Anforderungen, haben sich auch die Architekturen der Steuerungs- und Regelungskomponenten solcher Anlagen stark verändert. Vor allem die Vernetzung dieser eingebetteten Systeme hat ein weites Einsatzfeld aufgespannt. Zusätzlich ist durch die Vernetzung auch die Fernwartung, -diagnose und -test von eingebetteten Systemen möglich geworden.
Mit der Einführung von Jini hat sich für die Vernetzung von javafähigen Geräten eine neue Dimension eröffnet. Jini soll es ermöglichen, eine hochdynamische Infrastruktur von vernetzten Objekten zu schaffen, die robust, flexibel und skalierbar gehalten werden kann. Dieser Artikel soll anhand eines einfachen Anwendungsszenarios die Verwendung von Jini für die entfernte Steuerung und Beobachtung eines eingebetteten Systems beleuchten.

1 Einleitung

Betrachtet man die Entwicklung von eingebetteten Systemen im Verlauf der letzten drei Jahrzehnte, so ist festzustellen, daß sich deren Architektur grundlegend verändert hat. In den Anfängen elektronischer Steuerungen wurden einfache Schaltkreise, die meistens aus einigen wenigen Transistoren, Widerständen und Kondensatoren aufgebaut waren, für jede Steuerungsaufgabe neu entwickelt.

* Die Arbeit wurde mit Mitteln des Bundesministeriums für Bildung und Forschung unter dem BMBF Förderkennzeichen 01 M 3035 A im Rahmen des Projektes „Anwendungsbezogener Systementwurf (ABS)“ gefördert.

Daran anschließend wurde mit Hilfe von speicherprogrammierbaren Schaltungen versucht, allgemeinere Steuerungen zu entwickeln, die sich leichter an unterschiedliche Systemumgebungen anpassen ließen. Heute werden für Steuerungsaufgaben moderne Mikrocontroller eingesetzt, die oft über eine moderne 32 Bit RISC-Architektur verfügen. Darauf aufbauend kommen auch verstärkt Echtzeitbetriebssysteme zum Einsatz. Diese sollen helfen, durch einheitliche Systemschnittstellen die Portabilität der Software zu erhöhen, und damit die Kosten für die Softwareentwicklung zu senken.

Solche modernen Echtzeitbetriebssysteme verfügen auch über die Möglichkeit, über eine TCP/IP-Verbindung mit der Umgebung des eingebetteten Systems zu kommunizieren. Über diese Schnittstelle kann das eingebettete System dann an ein Intra- oder das Internet angeschlossen werden. Durch diese Option wird es möglich, von jedem beliebigen Ort der Welt aus, auf das eingebettete System zuzugreifen, wenn ein internetfähiger Browser zur Verfügung steht. Somit kann die Infrastruktur des Internet für die Fernwartung, -diagnose und -test von eingebetteten Systemen genutzt werden. Dies bringt mehrere Vorteile mit sich. Zum einen entfallen teure Reisekosten für den Fall, daß Personal für die Wartung oder zur Fehlersuche anreisen muß. Desweiteren lassen sich lange und teure Ausfallzeiten durch die Früherkennung von Fehlern vermeiden. Letztlich können durch diese ortsungebundene Form der Wartung und Bedienung auch bewegliche Systeme wie z.B. Züge oder Schiffe kontrolliert werden, ohne daß diese spezielle Servicestationen anlaufen müssen.

Innerhalb der Systemarchitektur von modernen *verteilten* eingebetteten Systemen lassen sich verschiedene Ebenen identifizieren, die sich in ihrer Leistungsfähigkeit und Aufgabenverteilung unterscheiden. Die *Führungsebene* nimmt dabei Aufgaben wie Visualisierung, Datenverwaltung oder die Archivierung von Daten wahr, und wird meistens durch PC's oder UNIX Workstations realisiert. Auf der *Steuerungsebene* werden dagegen komplexe Steuerungsalgorithmen ausgeführt, oder die Funktionalität des Gesamtsystems wird überwacht. Hier kommt dann beispielsweise ein Industrie-PC oder ein mit einem leistungsfähigen Mikrocontroller bestücktes Gerät zum Einsatz. Als unterste Ebene kann die *Sensor-* und *Aktorebene* betrachtet werden, die fest umrissene Aufgaben wie Messen, Steuern und Regeln durchführt.

In bisherigen Realisierungen solcher Systeme sind die unterschiedlichen Ebenen weitgehend voneinander abgeschottet, und nur durch proprietäre Protokolle miteinander verbunden. Dies hat negative Auswirkungen auf die Flexibilität, die Erweiterbarkeit und die Wartbarkeit des Gesamtsystems. Es müssen deshalb neue Technologien entwickelt werden, welche die unterschiedlichen Ebenen durchgängig durch ein einheitliches Steuerungskonzept miteinander verbinden, und dabei die Flexibilität und Skalierbarkeit des Gesamtsystems nicht aus den Augen verlieren. In der diesem Artikel zugrundeliegenden Arbeit wurde versucht, als Grundlage für ein solches System das Programmiermodell von Jini zu verwenden. Jini soll durch sein skalierbares und dynamisches Programmiermodell die oben angesprochenen Probleme beim Entwurf verteilter eingebetteter Systeme lösen helfen.

In Kapitel 2 werden zunächst grundlegende Architekturen moderner verteilter eingebetteter Systeme mit dem speziellen Augenmerk auf Fernsteuerung und Fernwartung diskutiert und deren Vor- und Nachteile aufgezeigt. Kapitel 3 stellt dann die Architektur und die wesentlichen Programmierparadigmen von Jini vor, bevor in Kapitel 4 die mit Hilfe von Jini entwickelte Anwendung eines über eine TCP/IP-Verbindung ansteuerbaren CAN-Netzwerkes diskutiert wird. Der Artikel schließt mit einer Zusammenfassung und einem Ausblick.

2 Moderne Architekturen eingebetteter Systeme

Wie oben bereits erwähnt, können moderne eingebettete Systeme hierarchisch aufgebaut werden. Die Funktionalität der einzelnen Hierarchiestufen unterscheidet sich dabei wesentlich. Abbildung 1 zeigt den Aufbau dieser hierarchischen Systemarchitektur.

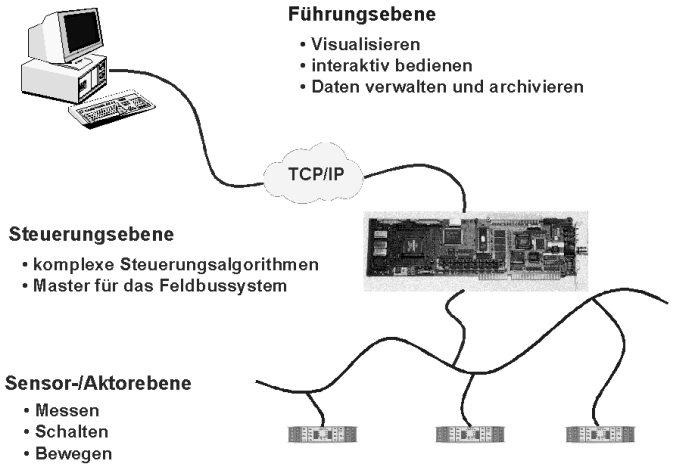

Abbildung1. Hierarchische Systemarchitektur moderner verteilter eingebetteter Systeme

Auf der untersten Ebene, der Sensor-/Aktorebene, kommen einfache Systemkomponenten zum Einsatz, die billig und robust sein müssen, da sie in sehr großer Stückzahl vorkommen, und an exponierten Stellen im System direkt am technischen Prozeß eingesetzt werden. Diese Komponenten können über Feldbussysteme miteinander vernetzt werden, und so Daten untereinander austauschen. Als Beispiel für ein Feldbussystem kann ein CAN-Bus herangezogen werden. CAN, ursprünglich für die Automobilindustrie entwickelt, wird auch in der Automatisierungstechnik mehr und mehr zu einem Standard. Viele Mikrocontroller haben inzwischen schon eine integrierte CAN-Schnittstelle auf dem Chip.

Auf der Steuerungsebene, die der Sensor- und Aktorebene übergeordnet werden kann, müssen komplexere Systemfunktionen ausgeführt werden können. Hier

müssen dann auch leistungsfähigere Mikrocontroller eingesetzt werden. Auf dieser Ebene hält der PC in seiner Industrieform als Steuerzentrale Einzug, und wird mit dem Feldbussystem über eine spezielle Schnittstelle verbunden. Im Falle von CAN kann dies beispielsweise über eine PC-ISA basierte Einsteckkarte erfolgen. Auf diesen Rechnern kommen dann auch verstärkt Echtzeitbetriebssysteme zum Einsatz, die über eine TCP/IP-Schnittstelle verfügen.

Durch die zunehmende Verfügbarkeit der Ethernet-Technologie und TCP/IP auf der Steuerungsebene, läßt sich die Führungsebene, als nächsthöhere Systemebene, anbinden. Die Führungsebene ist für die Visualisierung, die interaktive Bedienung und die Datenverwaltung zuständig. Hier können die Grenzen zwischen der Führungsebene und der Steuerungsebene allerdings verschwimmen.

Ein Beispiel einer solchen Architektur ist das CLASS-System von Itschner, Pommerell und Rutishauser [1]. Systemdaten werden hier von Objekten, die die einzelnen Sensoren und Aktoren des Systems repräsentieren, in einer Datenbank abgelegt, die sich auf dem Rechner der Steuerungsebene befindet. Auf diesem Rechner kommt ein Server zum Einsatz, der diese Daten dann auf Anfrage an einen Klienten weiterreicht, der typischerweise als Browser auf einem entfernten Rechner realisiert ist. Das CLASS-System wurde ausschließlich für die Datenerfassung in eingebetteten Systemen entwickelt und kann deshalb nicht steuernd in einen Prozeß eingreifen. Auch sind die Daten nur bedingt aktuell, da sie zunächst in einer Datenbank abgelegt, und dann erst abgerufen werden können.

Ein internet-basiertes System, in welches über einen Browser steuernd eingegriffen werden kann, ist die Applikation von Hergenhan, Weiler, Weiß und Rosenstiel [2]. Hier kommt ein HTTP-Server auf dem eingebetteten System zum Einsatz, von dem ein Applet heruntergeladen werden kann. Über dieses Applet kann dann ein Hochregallager ferngesteuert bedient werden. Die Kommunikation wird durch den Austausch spezieller Signale zwischen dem Applet und einem Server auf dem eingebetteten System realisiert. Auf Führungsebene ist dann eine Workstation mit einem internetfähigen Browser angesiedelt. Diese ist über eine TCP/IP-Verbindung mit einem eigenentwickelten Board verbunden, welches die Steuerungsebene repräsentiert. Das Board steht dann über ein Aktuator-Sensor-Interface [3] mit der Sensor-/Aktorebene in Verbindung.

In beiden Systeme wird die Programmiersprache Java für die Realisierung der graphischen Benutzeroberfläche und für die Kommunikation zwischen dem entfernten Rechner und dem eingebetteten System verwendet. Die Programmiersprache Java eignet sich für diese Aufgaben besonders gut, da sie über umfangreiche Standardbibliotheken, unter anderem in den Bereichen Grafik und Netzwerktechnologie, verfügt.

Mit Jini, einer neuen Systemarchitektur für hochdynamische, vernetzte Systeme, gewinnt die Programmiersprache Java im Bereich eingebetteter Systeme noch mehr an Bedeutung. Deshalb wurde untersucht, in wieweit sich die oben angesprochenen Systeme mit Hilfe von Jini implementieren lassen. Ziel war es, die entfernte Überwachung eines auf CAN basierenden Sensor-/Aktornetzwerkes zu realisieren. Im folgenden Kapitel werden zunächst einige grundlegende Architekturmerkmale von Jini vorgestellt, bevor in Kapitel 4 die daraus resultierende

Systemarchitektur beschrieben wird. Damit ist es möglich, mit Hilfe der Programmiersprache Java ein durchgängiges, flexibles und skalierbares System zu entwickeln.

3 Jini-Grundlagen

Ende Januar hat Sun Microsystems mit der Version 1.0 von Jini [4] eine erste Implementierung eines einfachen Jini-Systems im Rahmen der Java Developer Connection [5] interessierten Entwicklern zur Verfügung gestellt. Mit Hilfe von Jini soll eine gemeinsame Benutzung von Systemkomponenten für Anwender und Geräte in einem globalen Netz stark vereinfacht werden, und mühsame Netzwerkkonfigurationsarbeiten der Vergangenheit angehören. An dieser Stelle wird oft ein Vergleich mit der Telekommunikationstechnik gezogen. Detailkenntnisse über die zugrundeliegende Technologie sind hier nicht notwendig, um als Endverbraucher am weltweiten Telefonnetz teilhaben zu können. Einstecken des Telefons in eine dafür vorgesehene Steckdose genügt hier.

Ein Jini-System kann im wesentlichen aus drei Sichten betrachtet werden, deren Grenzen allerdings verschwimmen. Jini besteht zunächst einmal aus sogenannten *Infrastrukturkomponenten* für die Zusammenführung von einzelnen Teilnehmern eines Jini-Netzes. Ohne diese Infrastrukturkomponenten ist ein Jini-Netzwerk nicht lauffähig. Eine solche Infrastrukturkomponente stellt beispielsweise der *Lookup Service* dar, der zum Auffinden von *Diensten* im Netz verwendet wird. Dienste sind die zweite Sicht, aus denen man ein Jini-System betrachten kann. Ein Dienst kann von einer Festplatte, über einen Drucker, bis hin zu einem elektronischen Wörterbuch alles sein. Genau genommen ist auch der *Lookup Service* ein Dienst, der anderen Teilnehmern das Mitwirken an einem Jini-Verbund ermöglicht. Die dritte Sicht wird durch das *Programmiermodell* von Jini gebildet. Dienste werden dabei durch *Java-Interfaces* repräsentiert. Ein Klient sieht dabei nur die Schnittstelle eines Dienstes und weiß nichts über die zugrundeliegende Implementierung.

Abbildung 2 zeigt prinzipiell ein einfaches Szenario einer Jini-Anwendung. In einem ersten Schritt meldet sich der Dienst bei einem *Lookup Service* an. Der *Lookup Service* kann dabei entweder über das *Unicast* oder *Multicast Discovery Protocol* aufgefunden werden (*Discovery*). Nachdem der *Lookup Service* aufgefunden wurde, überträgt der Dienst in der *Join*-Phase einen Proxy zum *Lookup Service* (1). Der Proxy dient dann später dem Rechner des Klienten als Stellvertreter für den Dienst. Ein Klient identifiziert wiederum über eine Anfrage an den *Lookup Service* einen für ihn in Frage kommenden Dienst (2), und lädt sich dessen Proxy herunter (3). Die Kommunikation zwischen Klient und Dienst wird dann über diesen Proxy abgewickelt (4).

Da sich hinter einem Dienst unterschiedliche Anwendungen verbergen können, sind auch unterschiedliche Formen von Proxies möglich. Ein Festplattengerät könnte zum Beispiel einen proprietären Treiber zum Klienten senden, und mit diesem über ein proprietäres Protokoll kommunizieren. Ein elektronisches Wörter-

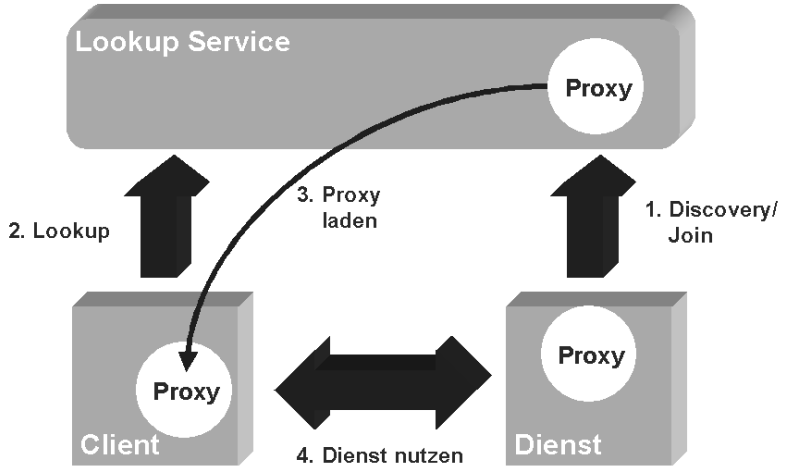

Abbildung2. *Lookup*, *Discovery* und *Join* innerhalb einer Jini-Anwendung

buch würde dagegen nicht seine gesamte Datenbank übertragen, sondern nur ein einfaches GUI als Proxy zur Verfügung stellen.

Zu den Infrastrukturkomponenten zählen auch verteilte Ereignisse. Diese besitzen zusätzlich zu ihrer Ereignisnummer eine Sequenznummer, über die ein Klient die Reihenfolge auflösen kann, in der ein bestimmter Ereignistyp aufgetreten ist. Im nächsten Kapitel wird aufgezeigt, wie mit Hilfe von Proxies und verteilten Ereignissen eine einfache Steuerung und Überwachung eines CAN-Netzes über eine Ethernet-Verbindung realisiert werden kann.

4 Jini-Ansteuerung von CAN-Netzwerken

4.1 Zugrundeliegende Systemarchitektur

Für die Anbindung eines Jini-Systems an ein CAN-Netzwerk wurde eine Systemarchitektur gewählt, wie sie in Kapitel 2 vorgestellt wurde. Abbildung 3 zeigt schematisch den Aufbau der Implementierung.

Für die Führungsebene können sowohl PC-basierte, als auch UNIX-basierte Lösungen verwendet werden, da die Benutzerschnittstelle komplett in Java implementiert wurde. Der dort verwendete Rechner wird dann über TCP/IP mit einem PC verbunden, der die Steuerungsebene repräsentiert. Für die CAN-Schnittstelle dieses PC's wird eine ISA-Einsteckkarte verwendet, die die Verbindung zu drei Boards der Sensor-/Aktorebene herstellt.

Bei den Boards handelt es sich um eine Entwicklungsplattform für eingebettete Systeme, welche am Forschungszentrum Informatik entwickelt wurde. Dieses Board verfügt über einen leistungsstarken Embedded PowerPC 403 Mikrocontroller sowie mehrere E/A-Schnittstellen. Neben der für die hier beschriebenen

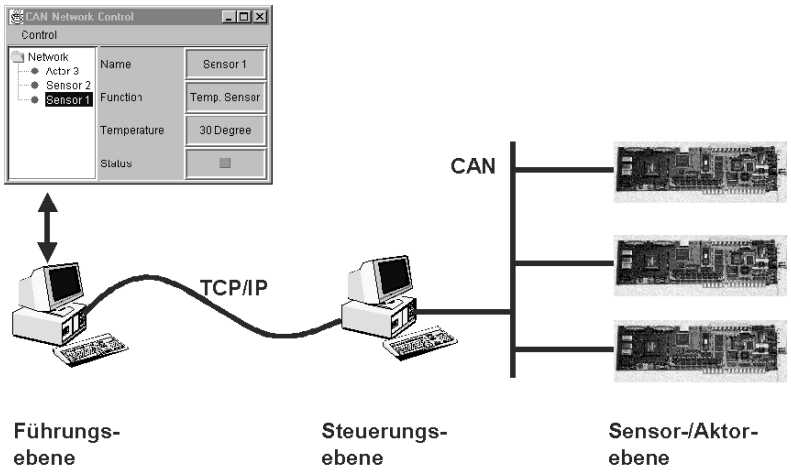

Abbildung3. Systemarchitektur des Jini/CAN-Systems

Anwendung notwendigen CAN-Schnittstelle besitzt das Board auch eine Ethernetschnittstelle. Auf dem Board steht VxWorks als Echtzeitbetriebssystem zur Verfügung. Es wird in der in Kapitel 2 beschriebenen Realisierung einer Hochregalansteuerung als Rechner der Steuerungsebene eingesetzt. Dies war in der hier entwickelten Anwendung nicht möglich, da auf dem Board keine JVM zur Verfügung stand, welche für die Verwendung von Jini unbedingt erforderlich ist.

4.2 Software-Architektur der Anwendung

Die Implementierung einer Jini-fähigen Netzwerkanwendung erfordert die Bereitstellung einer gewissen Infrastruktur. Zu diesen Infrastrukturkomponenten zählt unter anderem der *Lookup Service*. Bei ihm können sich Dienste registrieren lassen. Klienten wiederum erfragen beim *Lookup Service* mögliche zur Verfügung stehende Dienste. Für die Implementierung des *Lookup Service* wurde von Sun Microsystems eine Spezifikation herausgegeben, die von Jini-Implementierungen eingehalten werden muß. Der Version 1.0 von Jini, die Ende Januar von Sun freigegeben wurde, ist eine Beispielimplementierung eines *Lookup Service* beigefügt, die den Namen *Reggie* trägt. Reggie wurde in der hier beschriebenen Anwendung als *Lookup Service* genutzt, und kommt auf dem PC der Steuerungsebene zum Einsatz.

Ein auf dem Steuerungsrechner angesiedelter Geräteverwalter meldet die einzelnen Sensoren und Aktoren des CAN-Netzwerkes beim *Lookup Service* an. Er überwacht dazu das CAN-Netz und identifiziert neue Knoten anhand einer speziellen ID, mit der sich CAN-Knoten beim Anschluß an das Netz identifizieren. Diese ID gibt Auskunft über die Funktionalität des CAN-Knotens. Der Gerätemanager erzeugt dann für jeden identifizierten CAN-Knoten ein entsprechendes Stellvertreterobjekt, welches dann beim *Lookup Service* angemeldet

wird. Die Kommunikation mit dem CAN-Knoten wird dann über dieses Stellvertreterobjekt realisiert.

Da sich die Funktionalität der einzelnen CAN-Knoten unterscheiden kann, existiert für jeden möglichen Knoten ein spezielles Objekt, welches dem Geräteverwalter bekannt sein muß. Einem solchen Objekt ist eine grafische Benutzeroberfläche, *Event Handler* und Code zugeordnet. Die grafische Benutzeroberfläche wird für die entfernte Beobachtung und Steuerung des CAN-Knotens verwendet. In Abbildung 3 ist die grafische Benutzeroberfläche eines Sensors im rechten Teil der Anwendungsoberfläche zu sehen. Das GUI eines Aktors könnte dagegen beispielsweise einen Schieberegler beinhalten, über den ein anderer Temperaturwert eingestellt werden kann.

Sowohl die grafische Benutzeroberfläche als auch zum CAN-Knoten gehörige *Event Handler* werden dem *Lookup Service* als *Entry Items* zusätzlich zum Proxyobjekt übergeben. Ein Eintrag im *Lookup Service* besteht deshalb aus dem Proxy des CAN-Knotens und Bytecode für das GUI und *Event Handler*, die auf entfernte Ereignisse reagieren können. Da die Behandlung entfernter Ereignisse innerhalb eines Jini-Systems für die entfernte Steuerung und Kontrolle von besonderer Bedeutung ist, wird dieser Punkt im nächsten Abschnitt noch gesondert behandelt.

Wird auf dem Rechner der Führungsebene die Anwendung für die entfernte Beobachtung und Steuerung eines CAN-Netzes gestartet, muß zunächst die Adresse des Rechners der Steuerungsebene, auf dem der *Lookup Service* abläuft, angegeben werden. Zum Auffinden des *Lookup Service* wurde hier das *Unicast Discovery Protocol* verwendet. Dies ist ausreichend, da einer Firma in der Regel bekannt ist, wo ihre Produkte stehen, und so die Rechnernamen bekannt sind. Grundsätzlich wäre es aber auch möglich, über das *Multicast Discovery Protocol* den *Lookup Service* zu lokalisieren. Wurde der *Lookup Service* gefunden, wird dort nach gemeldeten Sensoren und Aktoren nachgefragt.

Abbildung 3 zeigt ein Szenario, in welchem drei CAN-Knoten gefunden wurden. Diese werden auf der linken Seite der Anwendungsoberfläche angezeigt. Die grafische Oberfläche des gerade angewählten CAN-Knotens erscheint zusätzlich im rechten Teil des Fensters. Dort ist exemplarisch die Schnittstelle eines Temperatursensors dargestellt.

4.3 Behandlung verteilter Ereignisse

Der grafischen Oberfläche des Temperatursensors ist ein *Remote Event Listener* zugeordnet, der auf Temperaturänderungen am Sensor reagiert. Über ein entferntes Ereignis wird die Ausgabe der aktuellen Temperatur gesteuert. Zusätzlich läßt das entfernte Ereignis ein rotes Kästchen aufleuchten, wenn eine kritische Schwellentemperatur überschritten wurde.

Für die Implementierung verteilter Ereignisse sind mehrere Objekte notwendig. Ein *Event Generator* ist für die Erzeugung eines entfernten Ereignisses und die Benachrichtigung der entsprechenden *Remote Event Listener* zuständig, die sich bei ihm angemeldet haben. Der *Event Generator* wurde ebenfalls auf dem Rechner der Steuerungsebene implementiert, und wird vom Geräteverwalter bei

der Erzeugung des Systems beim *Lookup Service* angemeldet. Für Sensoren und Aktoren können unterschiedliche *Remote Event Listener* implementiert werden, die alle die Schnittstelle `RemoteEventListener` und dessen `notify`-Methode implementieren müssen. Dieser *Remote Event Listener* wird dann zusammen mit dem Sensorobjekt und dessen GUI beim *Lookup Service* angemeldet und vom Klient abgeholt. Der Klient erzeugt dann eine Instanz des *Remote Event Listeners*, und meldet diese beim *Event Generator* an. Eine Referenz auf den *Event Generator* erhält der Klient dabei ebenfalls vom *Lookup Service*. Der *Remote Event Listener* beinhaltet eine Referenz auf das ihm zugeordnete GUI und kann bei einem Ereignis, den er vom *Event Generator* erhält, entsprechend das GUI neu setzen.

5 Zusammenfassung

Die Architektur von eingebetteten Systemen hat sich im Laufe der letzten Jahre stark verändert. Inzwischen sind vernetzte eingebettete Systeme keine Seltenheit mehr. Innerhalb dieser Systemarchitekturen lassen sich verschiedene Hierarchiestufen mit unterschiedlicher Funktionalität ausmachen. Auch unterscheiden sich die Kommunikationsprotokolle zwischen den Hierarchien. Auf der oberen Ebene kommt zunehmend TCP/IP zum Einsatz, während auf den unteren Ebenen Feldbussysteme wie ASI oder CAN dominieren. Eine solche Systemarchitektur hatte die Verwendung von proprietären Protokollen zwischen den einzelnen Rechnern des Systems zur Folge.

Um in Zukunft die Entwicklungskosten für solche Systeme zu senken, müssen deshalb flexible, skalierbare und standardisierte Technologien für die Konzeption und Realisierung verteilter eingebetteter Systeme entwickelt werden, die diese neuen Strukturen berücksichtigen. In der hier vorgestellten Anwendung wurde deshalb der Einsatz von Jini für die entfernte Beobachtung und Steuerung eines CAN-Netzes untersucht. Jini ist aufgrund seiner dynamischen und skalierbaren Architektur für diese Aufgabe durchaus geeignet. Sollte die Java Technologie auch auf der untersten Sensor-/Aktorebene Einzug halten, lassen sich mit Jini sicherlich durchgängig in Java realisierte modulare Anwendungen entwickeln.

Aufgrund der noch relativ jungen Jini-Technologie ist zu vermuten, daß die Menge der möglichen Anwendungen noch lange nicht ausgereizt ist. Gegenstand zukünftiger Arbeiten am Forschungszentrum Informatik wird deshalb sein, mögliche Anwendungsszenarien von Jini für die entfernte Beobachtung und Steuerung eingebetteter Systeme zu untersuchen, und dafür geeignete Systemarchitekturen zu entwickeln.

Literatur

1. Itschner, R., Pommerell, C., Rutishauser, M.: CLASS: Remote Monitoring of Embedded Systems in Power Engineering. IEEE Internet Computing, **2(3)** (May/June 1998) 46–52

2. Hergenhan, A., Weiler, C., Weiß, K., Rosenstiel, W.: Internet-basierte eingebettete Systeme in der industriellen Automation. Tagungsband Java und Eingebettete Systeme. (1998) 19–31
3. Kriesel, W., Madelung, O.: Das Aktuator-Sensor-Interface für die Automation. Hanser Verlag.
4. Sun Microsystems. Jini. http://www.sun.com/jini/index.html (1999)
5. Java Developer Connection. http://developer.java.sun.com/ (1999)