Jini – ein guter Geist für die Gebäudesystemtechnik

Wolfgang Kastner[1], Christopher Krügel[1] und Heinrich Reiter[2]

[1] Technische Universität Wien, Institut für Rechnergestützte Automation
Treitlstraße 1, A-1040 Wien, Österreich
{k,chris}@auto.tuwien.ac.at
[2] European Installation Bus Association (EIBA)
Avenue de la Tanche/Tinklaan 5, B-1160 Brüssel, Belgien
hreiter@eiba.com

Zusammenfassung Die Anforderungen an die Elektroinstallation in Gebäuden und Wohnhäusern ist in den letzten Jahren dramatisch gestiegen. Da konventionelle Elektroinstallationen diesen Ansprüchen nicht mehr ausreichend gerecht werden können, wird der Ruf nach modernen, busfähigen Varianten immer vehementer. Eine dieser Varianten ist der Europäische Installationsbus (EIB). Welche vielfältigen neuen Möglichkeiten der Gebäudesystemtechnik und -automation offen stehen, wenn dieses Feldbussystem Jini-fähig gemacht wird (frei nach dem Motto: *The Network is the Device!*) und wie eine solche Integration funktionieren kann, soll dieser Artikel zeigen.

1 Einleitung

Der Begriff Jini [1] – ob Eigenname oder als Abkürzung für *Java Intelligent Network Infrastructure*, darüber sind sich selbst die Entwickler noch nicht einig – beginnt langsam aber sicher in der mittlerweile sehr großen Java Gemeinde Gestalt anzunehmen. Diskussionen und vereinzelte Fachbeiträge enden allerdings zumeist noch mit jener Zukunftsvision, in der Jini-fähige Drucker die Möglichkeit besitzen, sich, sobald sie an ein Netzwerk angeschlossen wurden, automatisch in einen bestehenden Rechnerverbund integrieren und sodann beginnen, anderen (Jini-fähigen) Komponenten ihre angebotenen Dienste anzubieten. Jini ist aber ein bei weitem mächtigerer Mechanismus, der über reine Druckerkonfiguration hinausgeht. Im Rahmen dieses Artikels soll gezeigt werden, daß in Jini ein in der Gebäudesystemtechnik und -automation lang ersehnter Mechanismus zu finden ist, der es ermöglichen könnte, unabhängige Gewerke zu vereinigen und bei weitem größere Zukunftsvisionen schon bald Wirklichkeit werden zu lassen.

Wir wollen daher zu Beginn dieser Arbeit einen kurzen Einblick in die Jini Technologie geben. Danach werden die vielfältigen Einsatzgebiete, die der Gebäudesystemtechnik und -automation offen stehen, diskutiert, zugleich wird gezeigt, daß es einer neuen Technologie bedarf, die es schafft, alle Bereiche auf einfache Art und Weise zu einem gemeinsam wirkenden Verbund zusammenzuschließen. Um diese Vermutung zu untermauern, greifen wir den Anwendungsbereich der Elektroinstallationen heraus und stellen ein Bussystem vor, dem in

Zukunft die größten Chancen eingeräumt werden, sich marktführend zu behaupten. Gezeigt werden soll, wie dieses Feldbussystem Jini-fähig gemacht werden kann und welche noch größeren Chancen sich für alle Jini-fähigen Geräte ergeben. Abschließend geben wir einen kurzen Ausblick, welche Ziele wir unter Zuhilfenahme von Jini weiterverfolgen werden.

2 Jini

Bei Jini, einem von Sun Microsystems entwickelten Produkt, handelt es sich um ein verteiltes System, das die Aufgabe hat, Leistungen und Ressourcen eines Netzwerks dem Benutzer einfacher als bisher zugänglich zu machen. Die Vorteile eines Netzwerks liegen im sogenannten *Resource Sharing* und der Möglichkeit von Benutzern, nichtlokale Daten und Dienste (*services*) von verschiedenen Punkten aus nützen zu können. Bei traditionellen Netzwerken tritt immer wieder das Problem auf, wie ein Dienstnehmer (*client*) seine gewünschten, von einem Dienstgeber (*server*) angebotenen Leistungen finden kann. Dazu ist es meist notwendig, daß diese Verbindung manuell hergestellt und konfiguriert wird. Jini versucht, diese Einschränkung zu mildern, in dem Protokolle definiert wurden, die es dem Client ermöglichen, automatisch seine benötigten Services zu finden und augenblicklich zu verwenden. Natürlich wird es nicht immer möglich sein, daß eine Applikation vollautomatisch ein passendes Service selektiert. In diesen Fällen kann Jini jedoch eine Vorauswahl treffen und Angebote, die bestimmte Kriterien erfüllen, einem menschlichen Benutzer zur Auswahl anbieten.

Jini stellt eine Erweiterung von Java dar. Alle beteiligten Komponenten müssen eine Java Virtual Machine (JVM) bereitstellen, um am System teilzuhaben (Abbildung 1). Der Vorteil, der sich aus der Verwendung von Java als zugrundeliegendes Programmiermodell ergibt, liegt zum einen daran, daß Java das Erstellen von verläßlichen Anwendungen durch Sprachmittel unterstützt. Dazu gehört z.B. das strenge Typsystem und eine automatische *Garbage Collection*. Zum anderen wird die Entwicklung von verteilten Applikationen durch *Remote Methode Invocations* (RMI) und das dynamische Nachladen von Klassen (auch über das Netzwerk) komfortabel unterstützt.

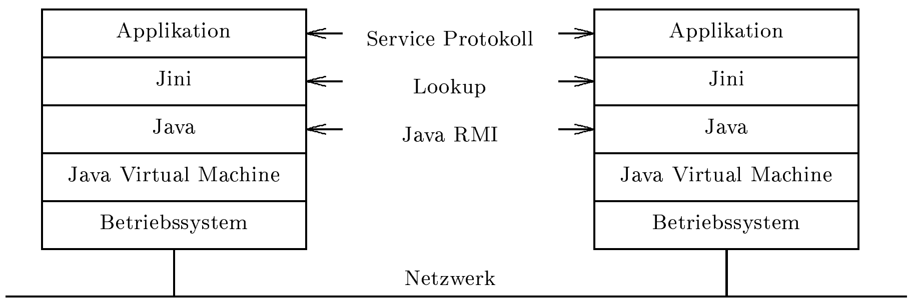

Abbildung1. Jini Modell

Jini, im sogenannten *middleware* Bereich angesiedelt, liegt als logische Schicht über den einzelnen JVMs, die über ein Netzwerk miteinander verbunden sind. Ein logischer Verband von Rechnern, auf denen Jini aufgesetzt ist, wird als *Djinn* bezeichnet. Grundsätzlich gibt es in einem solchen Verband drei unterschiedliche Klassen von Entitäten, die am Djinn teilhaben, nämlich *Services*, *Service User* und das sogenannte *Lookup Service*.

Ein Service ist ein abstrakter Begriff für eine Ressource, die über das Netzwerk in irgendeiner Form genutzt werden kann. Das kann z.B. spezielle Hardware (Sensoren oder Aktuatoren), Software oder auch ein Kommunikationskanal sein. Bei einem Service User handelt es sich um eine Entität, die ein (oder mehrere) Service(s) benützt. Dabei ist das Protokoll (*service protocol*) nicht fest vorgegeben, vielmehr besteht es aus einer Menge von in Java geschriebenen Schnittstellen (*interfaces*). Die tatsächliche Implementierung des Netzwerk-Protokolls und der Kommunikation zwischen Service und Service User kann vom Service selbst gewählt werden. Der dazu benötigte Code wird vom Service User dynamisch geladen. Ein Lookup Service ist letztlich jene Einheit, die zwischen einem Service und einem Service User vermittelt. Stößt ein Service neu zum Djinn, werden durch das *Discovery* Protokoll zunächst die verfügbaren Lookup Services gesucht. Danach meldet das Service seine Dienste mit den dazu passenden Attributen über das *Join* Protokoll an.

Ist nun ein Service User an einer bestimmten Leistung interessiert, so wird vorweg über das Discovery Protokoll eine Verbindung zu einem verfügbaren Lookup Service hergestellt. Daraufhin werden über das Lookup Protokoll und Suchkriterien (Attribute) die passenden Services herausgesucht. Der Service User kann die gewünschten Ressourcen nun direkt aufrufen oder durch die Angabe von weiteren Attributen und einer wiederholten Anfrage beim Lookup Service seine Auswahl verfeinern.

Zusätzlich zu dieser grundlegenden Vermittlungsfunktion, die das dynamischen Hinzufügen und Wegnehmen von Services innerhalb eines Djinns unterstützt, werden von Jini Netzwerk-Probleme wie der Ausfall einzelner Rechner und Netzabschnitte oder Sicherheitsaspekte wie das Übermitteln von Paketen über unsichere Leitungen behandelt. Die dazu verwendeten Konzepte sind *Leasing*, *Transactions*, *Security* und *Events*.

In den folgenden Abschnitten soll nun ein detaillierterer Blick auf die einzelnen Protokolle, die als Java-Klassen implementiert sind, gegeben werden.

2.1 Discovery Protokoll

Das Discovery Protokoll wird von Services und Service Usern zur Auffindung von Lookup Services verwendet. Dabei gibt es drei verschiedene Ausprägungen:

1. Multicast Request (`class net.jini.discovery.LookupDiscovery`)
2. Multicast Announcement (`class net.jini.discovery.LookupDiscovery`)
3. Unicast (`class net.jini.core.discovery.LookupLocator`)

Beim Multicast Request sendet jene Entität, die ein Lookup Service finden will, ein Multicast Paket an einen well-known Port am lokalen Netzwerk und wartet auf eintreffende Antworten von allen Lookup Services, die auf diesem Netzabschnitt lauschen und antworten dürfen. Um die Lookup Services vorweg einzuschränken, ist es möglich, eine Menge von Attributen (*groups*) anzugeben. Nur wenn diese Menge eine Übermenge der beim Multicast Request verlangten Attribute darstellt, darf ein entsprechendes Lookup Service antworten. Stößt ein neues Lookup Service zum Djinn, kommt das Multicast Announcement Protokoll zur Anwendung. Durch dieses Protokoll meldet der Neuankömmling seine Präsenz im Rechnerverbund an und erlaubt fortan allen interessierten Jini-Teilnehmern, seine Dienste zu verwenden. Da die beiden ersten Protokolle auf der Basis eines beschränkten Broadcasts funktionieren, gibt es zusätzlich die Möglichkeit einer direkten Kontaktaufnahme über das Unicast Discovery Protokoll, bei dem ein bestimmter Rechner, auf dem ein Lookup Service laufen muß, adressiert wird. Dies dient zur Überbrückung von Netzübergängen, die einen Multicast blockieren. Als Ergebnis eines Discovery Vorgangs erhält die suchende Entität ein *Proxy*-Objekt für jedes gefundene Lookup Service (`Interface net.jini.core.lookup.ServiceRegistrar`).

2.2 Join Protokoll

Das Join Protokoll wird von einem Service Provider verwendet, der sich beim Lookup Service registrieren will. Dazu wird über das Proxy-Objekt dem Lookup Service die Klasse, die das Service realisiert und die gewünschte Dauer, für die das Lookup Service die Klasse speichern soll (*Lease*) übergeben. Zusätzlich wird das Service noch mit einer Menge von Attributen versehen, die einem potentiellen Service User ein einfacheres Auffinden gewährleisten. Diese Attribute sind als Menge von Klassen gespeichert, deren Instanzvariablen die Attribute realisieren. Es handelt sich also eigentlich um eine Menge von sogenannten Entrys, die jeweils eine Menge von Attributen speichern. Ein *Entry* ist dabei eine Klasse, die das Interface `Interface net.jini.core.entry.Entry` implementiert und nur aus Instanzvariablen, die serialisierbar sind, bestehen darf.

2.3 Lookup Protokoll

Das Lookup Protokoll stellt das Gegenstück des Service Users zum Join Protokoll dar. Seine Aufgabe ist es, interessierten Jini-Teilnehmern zu ermöglichen, einen Verweis (*handle*) auf ein gewünschtes Service vom Lookup Service zu erhalten. Auch dazu werden Methoden des Proxy-Objekts (des Lookup Services) verwendet. Beim Lookup Prozeß kann die Auswahl des gewünschten Services entweder direkt über die Angabe einer, für jedes Service eindeutigen Servicenummer (`net.jini.core.lookup.ServiceID`) oder durch Attribute erfolgen. Dabei kann die Angabe der Attribute auf zwei verschiedene Arten erfolgen. Eine Variante ist das Festlegen des Types, den die gewünschte Klasse haben soll oder von dem sie abgeleitet sein muß. Die andere besteht in der Angabe von Attributen durch Entrys in der Form von Templates (`net.jini.core.lookup.ServiceTemplate`). Durch einen Matching Algorithmus werden jene Services ausgewählt, deren Attribute (Instanzvariablen der Entrys) mit denen des Templates übereinstimmen.

Bei der Beschreibung dieses Vorgangs fällt auf, daß ein Service User, um ein bestimmtes Service auswählen und verwenden zu können, zur Übersetzungszeit schon detaillierte Angaben über Methoden und Typ des Services benötigt. Da dies nicht immer möglich und gewünscht ist, muß nach Möglichkeiten gesucht werden, diese Einschränkung zu umgehen. Dazu gibt es zwei Wege. Einerseits kann das Service ein Applet anbieten, das die Kommunikation mit dem Benutzer abwickelt. Eine Einschränkung dieser Methode ist, daß ein menschlicher Benutzer über eine Benutzeroberfläche eine Auswahl treffen muß und das für eine völlig automatische Abwicklung ungeeignet ist. Dafür muß er/sie kein Vorwissen über die tatsächlichen Service Methoden besitzen. Die zweite Möglichkeit besteht in der Verwendung der *Reflection API*, die in Java implementiert ist. Dabei handelt es sich um eine Möglichkeit, zur Laufzeit Informationen über Klassen zu erhalten und Methoden aufzurufen, über die zur Übersetzungszeit nichts bekannt war. Dies würde ein automatisches Aufrufen von Services gestatten, ohne eine Benutzeroberfläche und menschliche Interaktion vorauszusetzen.

2.4 Service Protokoll

Ist die Auswahl des gewünschten Service-Objekts einmal abgeschlossen, erhält der Service User einen Handle auf das Service-Objekt. Der Aufruf erfolgt nun einfach wie ein Methodenaufruf. Intern kann dieser als RMI-Call oder über ein beliebiges, anderes Protokoll abgewickelt werden.

3 EIB-Gebäudesystemtechnik

In den letzten Jahren wurden speziell an die Elektroinstallation in Gebäuden immer höhere Anforderungen gestellt. War es früher bei weitem ausreichend, Geräte ein- und ausschalten zu können, ist es heute unumgänglich über Funktionen zum Bedienen, Steuern, Anzeigen, Überwachen und Melden zu verfügen. Auch Energie-Managementfunktionen spielen zusehends eine wichtigere Rolle. Wirtschaftlichkeit und Flexibilität sind weitere bedeutende Schlagworte.

Um die vielen verschiedenen Anwendungsgebiete (wie z.B. Beleuchtungs-, Jalousie- oder Heizungssteuerung, sowie Alarm- und Überwachungsanlagen) in den Griff zu bekommen, gab (und gibt) es "konventionelle" Insellösungen, die einen hohen Verdrahtungsaufwand mit einer Fülle von Leitungen nach sich ziehen. Zudem müssen diese Systeme einzeln installiert, konfiguriert und gewartet werden. Potentielle Nutzungsänderungen von Gebäuden erfordern klarerweise eine Neuverdrahtung und -konfiguration bestehender technischer Komponenten, die üblicherweise nur mit erheblichem Aufwand durchgeführt werden kann.

Zur Lösung dieser Problematik wurde der Europäische Installationsbus (EIB) entworfen [2]. Im Gegensatz zu den zuvor kurz diskutierten Insellösungen der herkömmlichen Elektroinstallation lassen sich mit seiner Hilfe alle betriebstechnischen Abläufe über eine, von allen Komponenten gemeinsam genutzte Leitung erfassen, schalten, steuern und überwachen.

Die Topologie des EIB ist durch eine übersichtliche, speziell für die Gebäudesystemtechnik ausgelegte Struktur gekennzeichnet, die linien-, stern oder baumförmig ausgelegt sein kann. Die EIB-Teilnehmer sind in dieser Struktur in Linien und Bereiche geordnet. Jede Linie ist durch einen Linienkoppler von den anderen Linien getrennt. Bis zu 15 solcher Linien können zu einem Bereich zusammengefaßt werden. Bereichskoppler schließen bis zu 15 weitere Bereiche zu einem Gesamtsystem zusammen. Da eine Linie Platz für maximal 256 Geräte hat, kann ein komplett ausgebautes EIB-System somit theoretisch bis zu 57.600 Teilnehmer umfassen.

Diese Teilnehmer können nun beispielsweise dazu verwendet werden, Außen- und Innenleuchten zu schalten und zu dimmen oder Jalousien bzw. Rolladen zu bewegen. Darüber hinaus sind Heizungs- und Klimaanlagen, sowie Sicherungs- und Sicherheitsinstallationen perfekte Anwendungsgebiete. Geräte der letzten beiden Kategorien haben einerseits dafür zu sorgen, daß immer ein sicherer Systemzustand (also ein Zustand, von dem keine Gefährdung für Personen und Sachen ausgeht) vorliegt, andererseits sind sie dafür zuständig, daß Gebäude gegen unbefugtes Eindringen von Personen gesichert sind. Typische Vertreter solcher Anlagen sind Feuermelder im einen, wie Überwachungsanlagen und Zutrittskontrollen im anderen Fall.

Die Verbindung der Netzknoten erfolgt im "klassischen" EIB hauptsächlich über die (für Feldbussysteme typische) geschirmte Zweidrahtleitung. Auch andere Medien, wie Koaxialkabel, Lichtwellenleiter, Netzleitung, sowie Infrarot oder Funk sind vorgesehen bzw. bereits im Einsatz. In jedem EIB-Endgerät ist ein Microprocessor eingebaut, der für die geordnete, dezentral organisierte Kommunikation im EIB-System zuständig ist. Neben diesem Busankoppler bestehen die einzelnen Teilnehmer aus dem jeweiligen Anwendungsmodul (Sensor oder Aktuator).

Geräte können einzeln (physikalisch) oder zu Gruppen zusammengefaßt (logisch) adressiert werden. Die Zuordnung wird einfach per Programmierung (über das EIB-System) hergestellt. Veränderungen an bestehenden Systemen werden auf dieselbe Weise vorgenommen. Wird beispielsweise die Aufteilung eines Raumes (durch Einziehen einer Zwischenwand) geändert und sollen die vorhandenen Leuchten neu geschaltet werden, so ist keine Neuverkabelung notwendig. Die bestehende Installation wird ausschließlich durch die entsprechende Umprogrammierung der beteiligten Komponenten gelöst. Diese Umprogrammierung findet, wie gesagt, über das EIB-System statt.

Das Werkzeug, das für die Projektierung, Inbetriebnahme und Verwaltung der EIB-Endgeräte eingesetzt wird, heißt EIB Tool-Software (ETS). Kern der ETS bildet eine Produktdatenbank, in welcher bei der Installation eines neuen EIB-Produktes alle zum Betrieb notwendigen herstellerspezifischen Daten abgelegt werden.

Unser Projekt verfolgt nun das ehrgeizige Ziel, das EIB-System offen für andere Anwendungsbereiche zu gestalten. Darunter fallen unter anderem Gewerke der Hausautomation, die bereits mit Kommunikationsschnittstellen versehen und daher a priori für die Gebäudeleittechnik interessant sind, wie:

– der eigentliche *Haushaltsbereich* mit klassischen Haushaltsgeräten, also Maschinen wie Tiefkühltruhe, Geschirrspüler und Geräten, die "mobil" sind (Staubsauger und Bügeleisen),
– der *Unterhaltungsbereich* mit typischen Vertretern der Unterhaltungselektronik, die entweder in einigen wenigen Räumen konzentriert sind (Videorecorder, HIFI-Anlage, Spielekonsole) oder im Gebäude verteilt angetroffen werden können (portabler CD-Player),
– der *Informations- und Kommunikationsbereich* bestehend aus alteingesessenen Endgeräten (Telefon, Anrufbeantworter) und modernen informationsverarbeitenden Maschinen (Workstation, PC),
– der zusätzliche *Installationsbereich* (Gas, Wasser) mit dazugehörigen Geräten,
– der bereits zuvor angesprochene *Sicherungs- und Sicherheitsbereich*, sowie
– andere *Gewerke*, die in keine der oberen Anwendungsgebiete eingeordnet werden können (Aufzug).

Punktuell seien nun einige Fallbeispiele herausgegriffen, die sich bei einer Integration all dieser Anwendungsgebiete mit dem EIB-System ergeben könnten.

– Mit Hilfe der Fernbedienung des TV Gerätes kann die Heizung geregelt oder Rolläden gesteuert werden. Die Benutzerführung erfolgt über das Fernsehgerät, auch Rückmeldungen der EIB-Geräte werden auf diesem ausgegeben.
– Wird das TV Gerät eingeschaltet, sollen die Leuchten im entsprechenden Zimmer gedimmt, bei Tageslicht gänzlich abgeschaltet werden.
– Sobald eine Person das Haus (oder für längere Zeit einen Teilbereich eines größeren Gebäudes) betreten hat, reagiert die Heizungssteuerung (natürlich in Abhängigkeit jener Werte, die von EIB-Wettersensoren aufgezeichnet wurden).
– Falls eine Herdplatte zu überhitzen droht, gehen alle Leuchten im Haus in kurzen Abständen an und aus, um auf die potentielle Gefahr aufmerksam zu machen. Reagiert der Hausbewohner nicht, wird nach gewisser Zeit automatisch ein Feueralarm ausgelöst.
– EIB-Bewegungsmelder schalten auf Wunsch in einem leerstehenden Haus in willkürlichen Räumen für kurze Zeit Unterhaltungselektronik-Geräte ein, um vorzutäuschen, daß sich jemand im Gebäude befindet.
– Ein Alarmtaster im Lift eines Gebäudes bewirkt ein Klingeln in jenem Appartment, in dem Personen vermutet werden, die zur Hilfe eilen können.
– Alle EIB-Funktionen, die lokal über entsprechende Bedienelemente eingestellt werden, sind auch über Internet (und somit von den entferntesten Orten) von autorisierten Personen steuerbar.
– usw.

Wünschenswert wäre nun, daß sich EIB-Komponenten, sobald sie an das Bussystem angeschlossen wurden, automatisch initialisieren, Jini-fähige Geräte suchen und im Bedarfsfall eine Verbindung zu ihnen öffnen, bzw. umgekehrt Geräte des Djinns via Jini Zugriff auf EIB-Komponenten erlangen können. Wie das vor sich gehen kann, zeigt das anschließende Kapitel.

4 Jini und EIB

Wir wollen nun eine Systemarchitektur vorstellen, die es ermöglicht, ein EIB-System Jini-fähig zu gestalten. Unser Ziel ist, beide Systeme so zu vereinigen, daß sie problemlos nebeneinander koexistieren und bei Bedarf die Services des jeweiligen anderen Systems nutzen können. Für den Benutzer soll der Eindruck entstehen, er hätte es mit einem homogenen Angebot an Diensten zu tun. Geräte, die Jini unterstützen, können Sensoren des EIB verwenden und EIB-Aktuatoren beeinflussen, EIB-Komponenten wiederum sollen bei Bedarf von allen Services, die im Djinn verfügbar sind, profitieren. Die grundsätzliche Idee besteht darin, daß Teile des EIB, die logisch zu einer Gruppe zusammengefaßt sind (z.B. Beleuchtung, Jalousien), über einen Group-Manager (GM) gesteuert werden. Dieser Group-Manager macht speziell jene Leistungen, für die er ausgelegt worden ist, über Jini verfügbar. Jeder Group-Manager besteht aus drei Teilen:

1. Djinn-Kommunikationsmodul,
2. EIB-Kommunikationsmodul und
3. Jini-Kontrollmodul.

Das Kommunikationsmodul zum Djinn dient dazu, die eigenen Services am Lookup Service zu registrieren und gewünschte Dienste zu suchen beziehungsweise diese aufzurufen. Das Kommunikationsmodul zum EIB übernimmt die Kontrolle der Feldbus-Logik (Abfragen von Sensoren, Steuerung von Aktuatoren). Nach der Installation einer (oder mehrerer) neuer EIB Komponenten im EIB-System übernimmt eine Steuerlogik (Jini-Kontrollmodul) die Aufgabe, die neuen Geräte über ihr EIB-Kommunikationsmodul zu konfigurieren und eine Anmeldung am Djinn über die Jini-Schnittstelle abzuwickeln. Die Steuerlogik wird in der ersten Ausbaustufe als eigenständiges Modul entwickelt, das unter anderem auch Zugriff auf die EIB-Produkt- und Projektdatenbank hat bzw. aktuelle Produktdaten auch aus dem Internet abfragen kann. In weiterer Folge ist eine Integration in die ETS-Software (z.B. als Plugin) vorstellbar und vorteilhaft.

Im laufenden Betrieb werden danach alle kommenden Anfragen aus dem Djinn via Jini-Kontrollmodul in Steuerbefehle für den EIB umgelegt; vom EIB kommende Anforderungen werden durch das jeweilige Steuermodul als Service Calls im Djinn durchgeführt. Da EIB-Anfragen über den Djinn abgewickelt werden, ist es auch denkbar, daß ein Group-Manager die Services eines anderen verwendet und damit die Konfiguration (ggf. auch die Kommunikation) von Devices am selben Bus oder über EIB-Busgrenzen hinweg transparent in Jini Service Aufrufe umgewandelt werden. Man beachte: auf diese Weise können EIB-Geräte unterschiedlicher Technologie miteinander in Kontakt treten, sodaß Group-Manager auch als kostengünstiges EIB-Gateway (z.B. Twisted Pair to Powernet) agieren.

Ein EIB/Jini-Manager besteht nun aus einer Reihe von Group-Managern (Abbildung 2). Da letztere modular aufgebaut sind und voneinander unabhängig agieren, ist es nicht weiter schwierig, neue EIB-Geräte (oder aber, wenn sich neue Anwendungsgruppen in einem bestehenden EIB-System herauskristallisieren, diese Bereiche) in einem eigenen Group-Manager zusammenzufassen, der dynamisch in das System eingebunden werden kann.

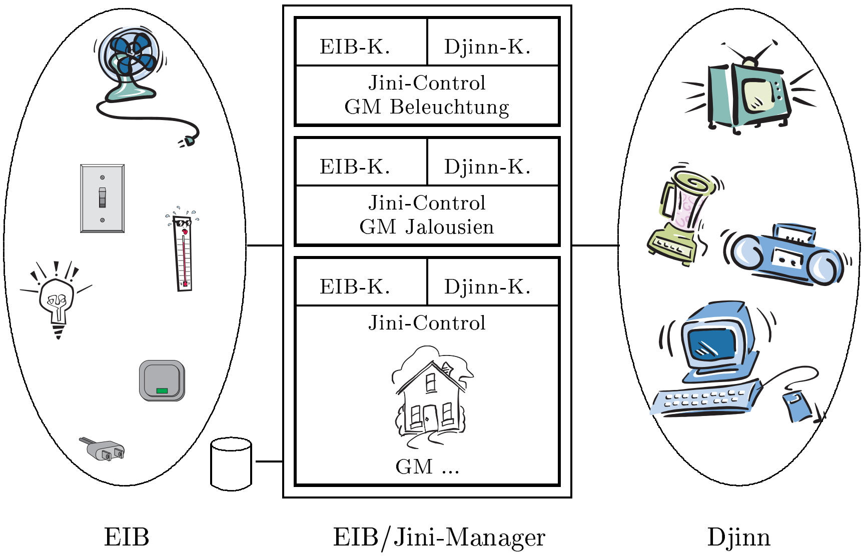

Abbildung2. EIB im Jini-Verbund

Das Hinzufügen einer neuen EIB Komponente in den Djinn läuft somit in zwei Schritten ab.

1. Physikalische Geräte-Installation im EIB-System
2. Integration des Group-Managers
   (a) EIB-Kommunikationsmodul übernimmt weitere Konfiguration im EIB
   (b) Djinn-Kommunikationsmodul meldet sich am Lookup Service an

Ein wichtiger Punkt ist die Modellierung der Group-Manager als Java-Klasse und die Definition einer geeigneten Klassenhierarchie. Da die unterschiedlichsten Hersteller und Geräte die Services eines Group-Managers nutzen wollen und zugleich Jini Services am effizientesten durch Angabe des gewünschten Typs gefunden werden, muß für den EIB (und auch andere Feldbussysteme der Gebäudeleittechnik) eine einheitliche Schnittstelle definiert werden.

Das aktuelle Konzept sieht vor, jede Gruppe von EIB-Services durch eine Basisklasse zu repräsentieren, von der wiederum Subklassen abgeleitet werden, die Sonderfunktionen bieten. Alle Klassen stammen von einer gemeinsamen, abstrakten Superklasse ab, die Basisfunktionen vorsieht. Bei diesen Methoden könnte es sich um eine textuelle Beschreibung der Komponente handeln (Hersteller, Aufstellungsort, Funktionalität), um ein Applet, das Statusinformationen anzeigt und um ein weiteres Applet, das eine manuelle Konfiguration des

EIB-Gerätes vorsieht. Dadurch ist auch eine Fernwartung gewährleistet. Diese abstrakte Superklasse könnte folgendes Aussehen haben:

```
public interface eib_device {
  public String get_info();
  public Applet monitor_status();
  public Applet do_maintenance();
}
```

In einer abgeleiteten Klasse (z.B. einer Klasse, welche die Beleuchtung eines Gebäudes zugänglich macht), müssen diese Services implementiert sein, zugleich kann diese neue Klasse um spezifische Dienste erweitert werden:

```
public class lighting implements eib_device {
  ...                     // Implementierung Methoden der Superkl.
  public light_on(ID id);  // neues Service: Licht an
  public light_off(ID id); // neues Service: Licht aus
  public set_lux(ID id);   // neues Service: Einstellen Helligkeit
}
```

5 Zusammenfassung

Jini ist – ohne Zweifel – ein vielversprechender und zukunftsträchtiger Mechanismus, der es gestattet, unterschiedliche Geräte einfach und elegant zu flexiblen Föderationen zusammenzuschließen. Dieser Verbund findet automatisch und für den Benutzer transparent statt. Voraussetzung dafür ist allerdings, daß alle beteiligten Komponenten Java-basierend ausgeführt sind.

Im speziellen Anwendungsfall der Gebäudesystemtechnik ist es unserer Ansicht nach nicht notwendig (und auch gar nicht sinnvoll) jedes einzelne Gerät Jini-fähig zu gestalten. Vielmehr bedarf es steuernder Instanzen, die auf das zugrundeliegende Feldbussystem zugreifen und ganze Gerätegruppen dem Djinn zugänglich machen. Somit ist es möglich, vorhandene Netzstrukturen beizubehalten, deren Vorteile auszunutzen und gleichzeitig für größere Anwendungsbereiche zu öffnen.

Das vorgestellte (allgemeingültige) aktuelle Konzept soll nun von reiner Theorie in die Praxis umgesetzt werden. Wir haben dazu den (ebenso wie Jini) zukunftsträchtigen Europäischen Installationsbus (EIB) ausgesucht. Wir hoffen demnächst, EIB-Geräte in einen bestehenden Djinn integrieren zu können, wodurch der Gebäudesystemtechnik und -automation faszinierende neue Anwendungsgebiete offen stehen sollten. Die Präsentation dieser Implementierung bleibt zukünftigen Arbeiten vorbehalten.

Literatur

1. Sun Microsystems: *Jini^{TM} Connection Technology.* http://www.sun.com/jini/
2. European Installation Bus Assocation: *EIB Introduction.* http://www.eiba.com/