Java3D statt WebCam

Effiziente Visualisierung in der Remote-Administration

Frank Burchert, Christian Hochberger, Ulrike Lucke, Djamshid Tavangarian

Universität Rostock, Institut für Technische Informatik
Lehrstuhl für Rechnerarchitektur
Albert-Einstein-Straße 21, 18059 Rostock
Tel. +49 (0)381 / 498 - 3386 Fax: -3440
E-Mail: burchert | hochberg | kleinau | tav @informatik.uni-rostock.de

Kurzfassung. Dieser Beitrag gibt einen ersten Einblick in die Möglichkeiten der Java-Remote-Administration und die Visualisierung mit Java 3D, wie sie sich im Rahmen einer Studie anhand eines exemplarischen Hochregallager-Modells ergeben haben. Durch den Einsatz von Objekt-Modellen sowohl im Client als auch im Server ist eine effektive Datenübertragung möglich. Die Visualisierung im Client wird mittels Java3D realisiert, was wiederum die zu übertragende Datenmenge reduziert. Die Kommunikation mit der Steuerungseinheit erfolgt über Java-RMI. Das Objekt-Modell im Client erlaubt semantische Überprüfungen der Steuer- und Administrationsanweisungen, um frühzeitig etwaige Bedienfehler erkennen und verhindern zu können.

1 Einleitung, Motivation

Durch den Anschluss an ein Kommunikationsnetzwerk lässt sich die Funktionalität eingebetteter Systeme erheblich erweitern. Hauptsächlich betrifft dies die Administration, Überwachung und / oder Steuerung solcher Systeme aus der Ferne. Damit lassen sich in der Regel erhebliche Einsparungen bei der Wartung oder dem Service erzielen, die insbesondere bei Industrieanlagensteuerungen entscheidende Wettbewerbsvorteile bedeuten können. Im Zuge der permanent zunehmenden Verbreitung eingebetteter Systeme werden diese zusätzlichen Funktionalitäten von wachsender ökonomischer Relevanz sein.

Um die große Anzahl von Plattformen im Bereich eingebetteter Systeme mit diesen zusätzlichen Funktionen versehen zu können, werden in der Forschung und Entwicklung mehrere Wege untersucht, wie man hierfür zu einem weitestgehend generischen und zukunftsträchtigen Ansatz gelangt. Dabei wird IP-basierte Remote-Administration und -Steuerung von eingebetteten Systemen aufgrund der faktischen Allgegenwart des Internets favorisiert. Die Anbindung kann je nach Einsatzort drahtgebunden (konventionell oder mittels Powerline über die Betriebsspannungszuleitung) oder auch drahtlos (WLAN, Bluetooth, Funk-Modems, ...) erfolgen; i.A. wird hier eine möglichst preiswerte, gegebenenfalls auch nur temporäre Anbindung angestrebt. Für die Remote-Administration werden dabei sowohl CORBA- als auch Java-RMI-Ansätze (unter Verwendung hochportabler Java-Infrastrukturen für eingebettete Systeme) verfolgt [1][2]. Letztere sollen helfen, von der Vielzahl von Plattformen im Embedded-Bereich weitestgehend zu abstrahieren und diese Anwendungen auf einer einheitlichen Basis implementieren zu können. Die Nutzung des Internets als Kommunikationsplattform erfordert dabei zahlreiche Sicherheitsmaßnahmen, um das eingebettete System vor unrechtmäßigen Angriffen zu schützen [3].

Unabhängig davon ist in jedem Fall die Frage nach einer geeigneten Visualisierung des fernadministrierten oder -gesteuerten Systems zu beantworten, mit der der Administrator mit der ihm zur Verfügung stehenden Benutzungsoberfläche einen möglichst einfachen und intuitiven Blick auf das reale System (und dessen aktuellen Zustand) bekommen soll [4]. Prinzipiell sollte der Administrator das reale System jederzeit aus der Ferne beobachten können, mindestens jedoch für die Dauer seines Eingriffs in das System. Dies gilt anschaulich insbesondere für Industrieanlagen mit z.B. beweglichen mechanischen Komponenten, kann darüber hinaus aber ebenso verallgemeinert werden.

Wir werden in diesem Beitrag hierzu als Beispiel ein Hochregallager betrachten, in das Produkte nach definierbaren Strategien ein- und wieder ausgelagert werden können. Die Bestandteile des Modells sind zwei Rampen zum Ein- und Auslagern, ein Regal mit 5 mal 10 Fächern zur Lagerung verschiedenfarbiger Bausteine (die die unterschiedlichen Produkte repräsentieren) und ein seitlich verschiebbarer Turm mit einem Fahrstuhl und einem speziellen Ausleger zur Platzierung der Bausteine (siehe Abb. 1).

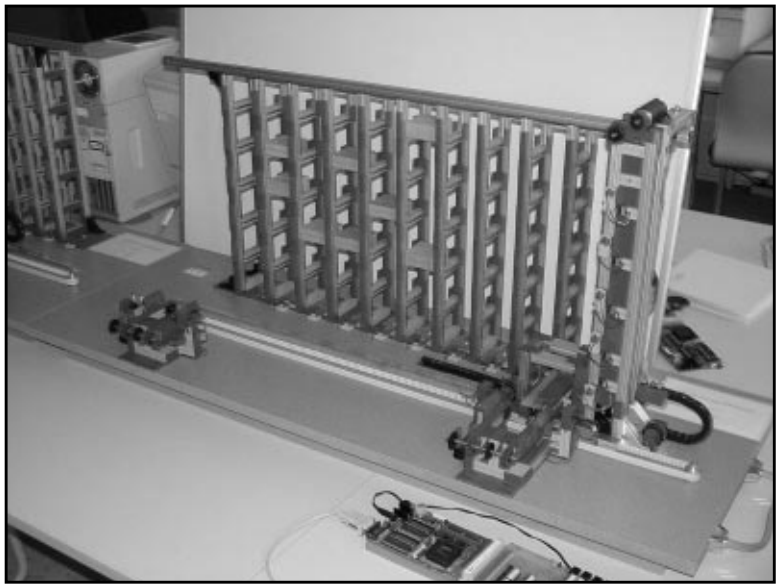


Abb. 1. Foto des realen Hochregallagers

Eine einfache Lösung zur Beobachtung des Systems ist bei vorhandenem Netzzugang sicherlich die Übertragung eines Bildes des zu steuernden Objekts (oder Teilen davon) mittels einer (Web-) Kamera. Der Vorteil liegt in dem verhältnismäßig niedrigen Aufwand, der hierzu unter Einsatz moderner Produkte getrieben werden muss. Diese Lösung beinhaltet aber eine Reihe von Nachteilen:

- Das Datenaufkommen bei einer Video- oder allgemeiner bei einer Bildübertragung ist auch unter Verwendung moderner Komprimierungs- und Reduktionsverfahren nach wie vor verhältnismäßig hoch. Dies bedingt einen vergleichsweise breitbandigen Internetzugang vor Ort, der wiederum i. A. höhere Kosten verursacht.

- Optische Übertragungen setzen bezüglich der Kameramontage entsprechende Licht- und Raumverhältnisse am Standort der Anlage voraus, die nicht immer gegeben sein müssen.
- Verschiedene (Teil-) Ansichten der zu steuernden Anlage sind nur von mehreren zoom-fähigen oder beweglichen Kameras zu bewerkstelligen. In jedem Fall erhöhen sich hierdurch Aufwand und Kosten.
- Sehr schnelle Vorgänge werden bei der Übertragung periodisch aktualisierter Bilder unter Umständen nicht erfaßt.

Alternativ kann die Visualisierung der Anlage durch ein auf der Seite des Administrators (nachfolgend Client-Seite genannt) befindlichen grafischen (3D-) Modells erfolgen, das das reale System visuell nachbildet. Über spezielle Informationen vom eingebetteten System und seinen Sensoren und Aktoren kann es manipuliert werden und so die Vorgänge des realen Systems nachbilden. Hierzu bieten sich bereits heute verschiedene Techniken an (VRML, OpenGL, Java3D ...), von denen wir in diesem Beitrag ausschließlich die Variante Java 3D zur Lösung des Problems einsetzen und vorstellen wollen. Dies begründet sich damit, dass Java mehr und mehr zu einer Sprache avanciert, deren Einsatzbereich sich zunehmend auch in das Umfeld eingebetteter Systeme, deren Spezifikation (z.B. in Form eines Objektmodells des eigentlichen Systems) oder auch Programmierung ausdehnt.

Der wesentliche Vorteil dieses Verfahrens zur Beobachtung einer fernbedienten Anlage ist, dass die Anzahl der Informationen, die das eingebettete System zur Manipulation des 3D-Modells an den entfernt stehenden Client senden muss, dabei erheblich geringer ist als die einer Bildübertragung, was geringere Anforderungen an die Übertragungsbandbreite zwischen dem eingebetteten System und dem Administrationssystem stellt. Darüber hinaus

- sind keine Bedingungen mehr an die Lichtverhältnisse vor Ort zu stellen,
- können Anlagenteile auch einzeln modelliert und dargestellt werden,
- können ggf. auch schnelle Vorgänge visualisiert werden (wenn auch nicht in Echtzeit),
- kann das Modell auf Client-Seite beliebig gedreht oder vergrößert und damit in verschiedensten Ansichten betrachtet werden.

Aufgrund der knappen Ressourcen in eingebetteten Systemen werden im Allgemeinen Thin-Server- und Fat-Client-Lösungen bei der Remote-Administration bevorzugt. Die Visualisierung des zu steuernden Systems als 3D-Modell fügt sich nahtlos in dieses Modell ein, indem sich hier die hohen Anforderungen lediglich auf Client-Seite ergeben.

Der weitere Aufbau dieses Beitrags gliedert sich in drei Abschnitte. Der folgende Abschnitt behandelt die Grundkonzepte des Java3D-APIs und die sich dadurch bietenden Möglichkeiten zur Implementierung eines 3D-Modells des exemplarisch betrachteten Hochregallagers. Im Kapitel 3 wird anschließend aufgezeigt, wie das geschaffene Modell für die Remote-Administration an das System angebunden werden kann und wie durch ein vorgelagertes Objektmodell Managementbefehle vor der Übermittlung an das zu steuernde System auf Plausibilität und Durchführbarkeit geprüft werden können. Der Beitrag endet mit einer Zusammenfassung der bisher geleisteten Arbeit und einem Ausblick, welche Potentiale in dieser Art der Visualisierungstechnik für die Zukunft stecken.

2 Visualisierung des zu steuernden Systems

Für die dreidimensionale Modellierung eines fernzusteuernden Systems stehen vielfältige Werkzeuge zur Verfügung (z.B. 3D Studio MAX, VRML). Für die Remote-Administration über das Internet ist jedoch auch der Aspekt der Kommunikation von Interesse. Java hat sich als Standard für verteilte Anwendungen im Internet etabliert und bietet mit seinem 3D-API zugleich die Möglichkeit zur Erzeugung und Visualisierung dreidimensionaler Modelle [5].

2.1 Grundkonzepte des Java3D-APIs

Das Java3D-API stellt basierend auf plattformunabhängigen nativen Bibliotheken (wie OpenGL oder Direct3D) verschiedene Klassen und Methoden zur Erstellung von dreidimensionalen Szenen in Applikationen und Applets bereit. Im Folgenden werden die Definition von visuellen Objekten, der Umgang mit Licht sowie das Animations- und Interaktionskonzept kurz vorgestellt [6][7]. Die Anwendung dieser Konzepte für die Implementation des Hochregallagers wird im nächsten Abschnitt beschrieben. Dies erfolgt unter Verwendung von Java3D-spezifischen Fachbegriffen, um Fehlinterpretationen durch die Übersetzung vorzubeugen.

SceneGraph. Die strukturierte Darstellung einer 3D-Szene in Java3D erfolgt in einem Baum. Ein solcher *SceneGraph* ist eine Ansammlung von Objekten, um geometrische Eigenschaften, Licht, Sound, Positionen und das Erscheinungsbild einer Szene zu definieren. Jeder *SceneGraph* besitzt ein so genanntes *VirtualUniverse.* Dieses beinhaltet eine Liste von *Locale*-Objekten, die Referenzpunkte im virtuellen Universum darstellen. Ein *Locale* ist Wurzel von so genannten *BranchGraphen*: der *ContentBranchGraph* bestimmt Eigenschaften und Verhaltensweisen des virtuellen Universums, während der *ViewBranchGraph* den Standort des Betrachters und die Blickrichtung definiert. Mit der Verwendung des bereits vordefinierten *SimpleUniverse* ist es nicht notwendig, eigene *VirtualUniverse-*, *Locale-* und *ViewBranchGraph*-Objekte zu erstellen.

Geometrie. Die Erzeugung visueller Objekte in der Szene ist über vordefinierte Klassen (z.B. für Quader, Zylinder, Kugel und Kegel), über die Angabe von Eckkoordinaten zum Aufspannen von Linien oder Polygonflächen sowie über extern erstellte und in Java3D eingeladene Geometrien möglich. Einzelne Basiskomponenten lassen sich gruppieren und über Translationen und Rotationen, die in *TransformGroup*-Objekten definiert werden, zu einer komplexen Szene zusammensetzen. Außerdem sind Farben, Normalen und Texturen im *Geometry*-Objekt festzulegen. Weitere Attribute werden durch das *Appearance*-Objekt bestimmt (z.B. Linienstärke, Materialeigenschaften). Gleiche Eigenschaften können an unterschiedliche Elemente vergeben werden, indem dasselbe *Appearance*-Objekt referenziert wird.

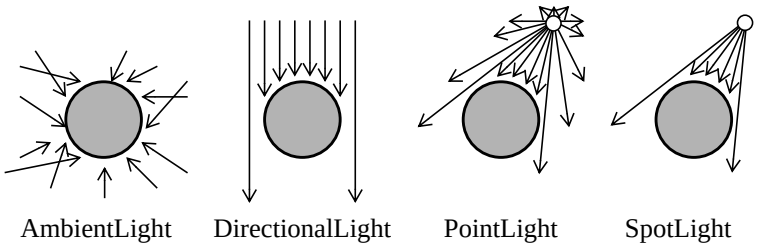


Abb. 2. Verschiedene Beleuchtungsmodi

Licht. Reflexionen sind abhängig von Materialeigenschaften und Lichtquellen. In Java3D sind die Lichtarten *AmbientLight*, *DirectionalLight*, *PointLight* und *SpotLight* definiert. Eine Berechnung von objektübergreifenden Schatten ist nicht vorgesehen, kann aber durch eigene Berechnungen nachgebildet werden. Die Geschwindigkeit beim Rendering hängt neben der Komplexität der Geometrie auch stark von den verwendeten Beleuchtungsmodellen ab. Mit *Bound*-Objekten kann deshalb ein begrenzter Bereich der Szene definiert werden, auf den sich die Lichtquellen beschränken.

Interaktion / Animation. Für Interaktionen (Veränderungen aufgrund von Eingaben des Nutzers) und Animationen (nach zeitlichen Vorgaben) kann die Klasse *Behavior* die Attribute von Objekten in Abhängigkeit von bestimmten Bedingungen wie Nutzeraktionen oder Zeitkriterien zur Laufzeit ändern. In Java3D sind als *Behavior* die so genannten Interpolatoren für Rotationen, Translationen, Größen-, Farb- oder Erscheinungsbildänderungen vordefiniert. Eigene *Behavior*-Objekte können zusätzlich implementiert werden, z.B. um das Modell zu drehen bzw. den Blickpunkt zu verlagern oder die Beleuchtung zu verändern.

2.2 Implementierung eines Java3D-Modells

Zur Java3D-Realisierung des exemplarisch betrachteten Hochregallagers sind sämtliche Komponenten sowie die funktionale Abfolge beim Ein- und Auslagern nachzubilden. Die Hauptaufgabe liegt dabei im Erstellen des *ContentBranchGraphen*, in dem visuelle Objekte, Lichtquellen, Verhaltensweisen etc. angeordnet werden (siehe Abb. 3). Die *Geometry* besteht aus mehreren einzelnen, zu den Komponenten des Hochregallagers zusammengesetzten Quadern. Durch einfarbige glatte Oberflächen wird der Aufwand für das Rendering der Szene minimal gehalten, ohne an Anschaulichkeit zu verlieren. Als Beleuchtungsmodell wurde *DirectionalLight* verwendet, da dies den natürlichen Beleuchtungsverhältnissen (Sonnenlicht) entspricht.

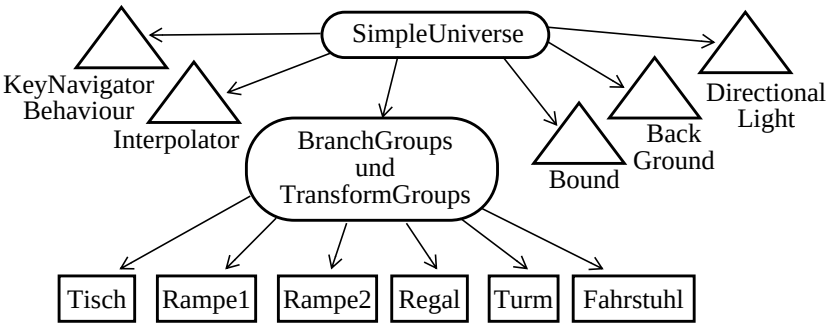


Abb. 3. Vereinfachter *SceneGraph* des Java3D-Modells des Hochregallagers

Für Bewegungen innerhalb des Modells ist eine eigene *Interpolator*-Klasse entwickelt worden, die Positionsänderungen der Steine durch Ein- oder Auslagerungsvorgänge bzw. die dafür nötigen Verschiebungen des Fahrstuhls und des Turms nachbildet. Auf Steuerkommandos des Nutzers wird folglich mit Animationen im Modell reagiert.

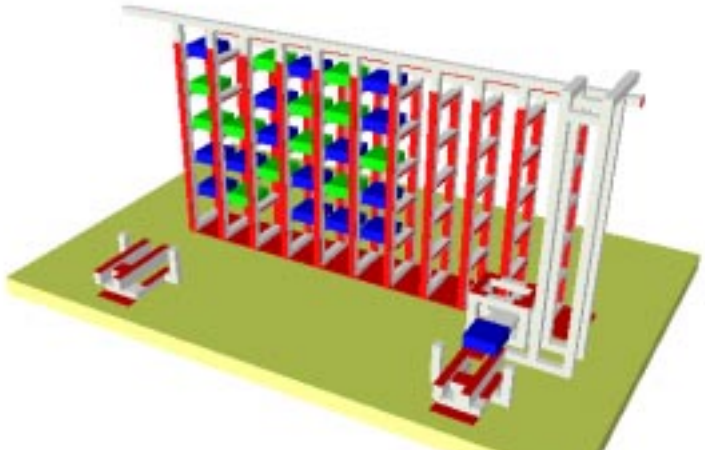


Abb. 4. Mit Java3D nachgebildetes Hochregallager-Modell

Das Ergebnis der Implementierung mittels Java3D ist ein einfaches und auf die wesentlichen Merkmale des Hochregallagers beschränktes Modell. Die Detailtreue ist nicht mit der Wiedergabe realistischer Bilder wie von einer Kamera vergleichbar, allerdings wird die Menge der über das Netz zu übertragenden Daten erheblich minimiert, ohne auf eine anschauliche, dreidimensionale Darstellung des zu überwachenden Systems verzichten zu müssen.

3 Anbindung des Modells an das System

Um eine dreidimensionale Darstellung des aktuellen Zustands des Hochregallagers zu erzeugen, kann man zwei verschiedene Wege einschlagen:

1. Der vollständige Status des Hochregals wird in regelmäßigen Abständen oder bei Änderungen zum Client übertragen und dort visualisiert. In diesem Fall werden eine Reihe redundanter Daten übertragen.
2. Es werden nur die Änderungen des Zustandes des Hochregallagers an den Client übertragen, der diese in einem lokalen Objektmodell vorhält.

Da die Verwendung eines lokalen - also vorgelagerten - Objektmodells erhebliche Vorteile mit sich bringt, soll im Weiteren nur dieser Fall betrachtet werden.

3.1 Vorgelagertes Objektmodell

Das vorgelagerte Modell sollte eine möglichst detaillierte und exakte Implementierung des tatsächlichen Hochregallagers sein. Dieses Modell sollte alle sensorisch erfassbaren Parameter des tatsächlichen Hochregallagers enthalten und Methoden zur Verfügung stellen, um alle Operationen des Hochregallagers zu simulieren. Die Struktur des Gesamtsystems wird in Abb. 5 dargestellt.

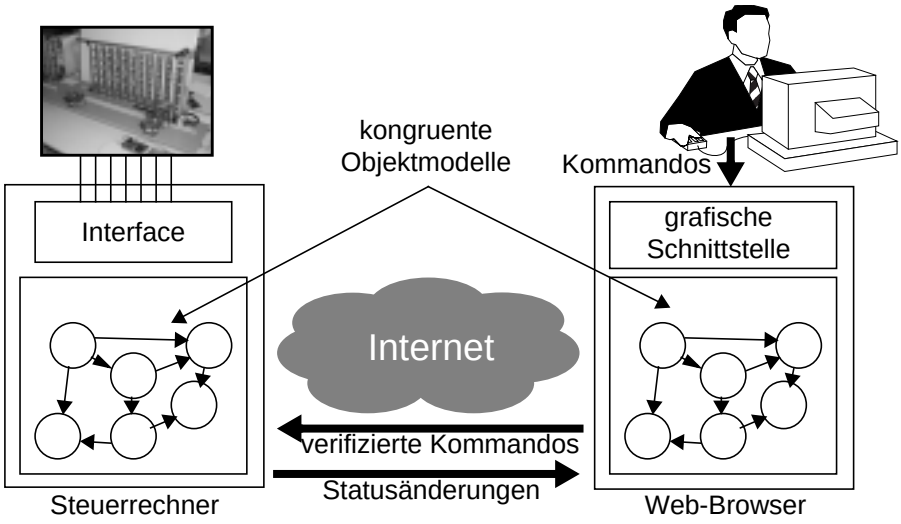


Abb. 5. Kommunikation der Objektmodelle

Ein wesentlicher Vorteil bei der Verwendung eines solchen Objektmodells im Client besteht darin, alle Managementbefehle vor der Übermittlung an das Hochregallager auf Plausibilität und Durchführbarkeit prüfen zu können. Dadurch wird einerseits ein erheblicher Kommunikationsaufwand vermieden und andererseits die Möglichkeit gegeben, eine bessere Argumentation zu geben und ggf. dem Anwender direkt zu zeigen, warum seine Anweisung nicht durchführbar ist.

Ein weiterer Vorteil eines lokalen Objektmodells besteht in der leichteren Abbildbarkeit auf einen *SceneGraph*. Bei geeigneter Modellierung lässt sich dieser durch einfaches Traversieren des Objektmodells gewinnen.

Schließlich ist auch die Menge der zu übertragenden Daten erheblich kleiner, als sie ohne ein solches Objektmodell wäre, da nur die Änderungen der Objektzustände übertragen werden müssen. Dies gilt natürlich nur für den Dauerbetrieb. Da aber davon auszugehen ist, dass der Client von Zeit zu Zeit beendet und dann in einem anderen Browser neu aufgerufen wird, muss natürlich auch eine Möglichkeit vorgesehen werden, das lokale Objektmodell aus dem Objektmodell der Hochregal-Steuerung zu initialisieren.

3.2 Implementierung der Steuerung des Hochregallagers

An die Implementierung der Steuerung des Hochregallagers sind für eine Internetanbindung einige weitergehende Forderungen zu stellen. Hier ist in erster Linie die IP-Infrastruktur zu nennen, die man bei einer Internetanbindung benötigt. Zusammen mit der erforderlichen Implementierung eines Web-Servers resultiert daraus auch ein Bedarf an mehr Verarbeitungsleistung, um die Kommunikationsprotokolle in ausreichender Geschwindigkeit implementieren zu können.

Um eine möglichst gute Kongruenz zwischen dem lokalen Objektmodell und dem in der Steuerung des Hochregallagers zu erreichen, sollten die beiden Objektmodelle in

der gleichen Programmiersprache implementiert werden. Somit kommt Java nicht nur im Client zum Einsatz, sondern auch in der Steuerung.

Im Bereich der Realisierung von Steuerungen auf der Basis von Java liegen schon einige positive Erfahrungen vor [8]. Für den Zugriff auf das eigentliche Interface zum Hochregallager werden typischerweise native Routinen verwendet. Alternative Mechanismen sind Gegenstand aktueller Forschung [9].

3.3 Kommunikation zwischen Hochregallager und Client

Zur Übermittlung von Kommandos und zum Abgleich der beiden Objektmodelle müssen Client und Steuerung miteinander kommunizieren. In den meisten publizierten Anwendungen dieser Art wird dazu ein proprietäres Protokoll über TCP/IP verwendet. Diese Vorgehensweise wäre im Prinzip auch zwischen den beiden Objektmodellen möglich, hat jedoch einige schwerwiegende Nachteile. Alternativ bietet sich in diesem Umfeld der Einsatz einer Middleware wie CORBA oder Java-RMI an. Die besonderen Vorteile der letzteren Variante sollen hier noch einmal kurz aufgezählt werden:

1. Abstraktion.
   Java-RMI ist nahtlos in das Konzept der Sprache Java integriert, so dass der Programmierer sich nicht mit den Details der Kommunikation belasten muss, sondern in sehr abstrakter Weise den Abgleich der beiden Objektmodelle formulieren kann.
2. Sicherheit.
   Java-RMI stellt Mechanismen zur Verfügung, um den Datenverkehr zwischen zwei Kommunikationspartnern zu verschlüsseln. Der besondere Vorteil gegenüber CORBA besteht in der Möglichkeit, diese Kanalverschlüsselung selbst zu implementieren. Dadurch hat der Entwickler eine gute Kontrolle über den zur Verschlüsselung nötigen Aufwand.

Eine detailliertere Gegenüberstellung der Vor- und Nachteile von Java-RMI für den Einsatz zur Steuerung von eingebetteten Systemen ist in [10] zu finden.

4 Zusammenfassung und Ausblick

In diesem Beitrag wurde eine Visualisierung eines fernbedienbaren Hochregallagers auf der Basis eines Java-3D-Modells vorgestellt, wie es im Umfeld der Remote-Administration zur Beobachtung eines Systems während des Fernsteuerungsvorgangs eingesetzt werden kann. Ausgehend von einer Beschreibung der grundlegenden Konzepte und Möglichkeiten des Java3D-APIs wurde das Modell anhand der funktionalen und konstruktiven Vorgaben des realen Hochregallagers entworfen und demonstriert.

Bei der Anbindung des Modells an das reale System wurde zum Zwecke der Fernadministration das Konzept eines vorgelagerten Objektmodells propagiert, das Plausibilitätskontrollen für Management-Befehle zulässt, mit denen etwaige Bedienfehler frühzeitig erkannt und verhindert werden können. Aufgrund der sprachlichen Eigenschaften von Java bietet es sich an, das vorgelagerte Objektmodell ebenfalls in Java zu implementieren.

Als Ergebnis kann festgehalten werden, dass die Visualisierung von fernsteuerbaren Anlagen im Client unter Einsatz von Java3D eine interessante Alternative zur Überwachung und Beobachtung dieser Systeme mit (Web-) Kameras darstellt. Sie besitzt bezüglich des zu übertragenden Datenvolumens eindeutige Vorteile gegenüber kamerabasierten Beobachtungstechniken und erlaubt die freie dreidimensionale Be-

trachtung des ferngesteuerten Objekts. Der in unserem Fall durch die manuelle Modellierung höhere Entwicklungsaufwand wird im industriellen Einsatz weniger bedeutend sein, da es hier in der Regel schon 3D-Modelle der Anlagen auf CAD-Systemen gibt, die hierzu lediglich umgewandelt werden müssen. Als einzig verbleibender prinzipieller Nachteil wirkt sich allein der Umstand aus, dass mit dieser Methode ausschließlich die sensorisch erfassbaren Zustände visualisiert werden können. Es wird angestrebt, die Praxistauglichkeit dieses Konzepts im industriellen Umfeld nachzuweisen.

Die intuitive Erfahrbarkeit eines 3D-Modells wird von der in Zukunft zu erwartenden Leistungssteigerung der Grafiksysteme weiter profitieren, die sich insbesondere bei einer größeren Detailtreue der Modelle positiv auswirken wird. Die Kopplung dieser Visualisierungstechnik mit modernen 3D-Scan-Techniken wird aller Voraussicht nach den Entwicklungsaufwand bei der Modellierung weiter herabsetzen.

5 Literatur

1. Hochberger, C.: "DCOM, CORBA und JAVA-RMI; Ein Vergleich verschiedener Middleware-Architekturen im Umfeld eingebetteter Systeme", Technischer Bericht, Universität Rostock 2000
2. Hegering, H.-G.; Abeck, S.; Numair, B.: "Integriertes Management vernetzter Systeme", dpunkt-Verlag, ISBN 3-932588-16-9
3. Ahuja, V.: "Network & Internet Security", AP Professional, Boston 1996
4. Schumann, H.; Mueller, W.: „Visualisierung: Grundlagen und allgemeine Methoden“, Springer, Berlin 2000.
5. Sun Microsystems: "Java3D Documentation", 1999 (http://java.sun.com/products/java-media/3D/forDevelopers/J3D_1_2_API/j3dapi)
6. Bouvier, D.J.; Dankwardt, K.P.: "Getting Started with the Java3D-API - A Tutorial for Beginners", Sun Microsystems, 1999 (http://sun.com/desktop/java3d/collateral/)
7. Sun Microsystems: "The Java3D API Specification", 1999 (http://java.sun.com/products/java-media/3D/j3dguide.pdf)
8. Hergenhan, A.; Weiler, Chr. ; Weiß, K.; Rosenstiel, W.: "Internet-basierte eingebettete Systeme in der industriellen Automation", GI/ITG-Workshop Java und eingebettete Systeme, Karlsruhe 1998, S. 19-31
9. Gloede, G.; Hartwig, V.; Hochberger, C.: "Hardware-Zugriffe aus Java heraus", NetObjectDays 2000
10. Burchert, F.; Hochberger, Ch.; Kleinau, U.; Tavangarian, D.: "Remote-Administration von eingebetteten Systemen mit einem Java-basierten Add-On-Modell", Java-Informations-Tage (JIT) ’99, S. 110-120