Softwarearchitektur für einen Containerterminal

Vom Informationssystem in J2EE bis zur Gerätesteuerung in Java

Hartmut Krasemann[1], Ulrich Spindel[2]

[1] T-Systems GEI GmbH, Lademannbogen 21-23,
22339 Hamburg, Germany
hartmut.krasemann@t-systems.com
http://www.t-systems.com
[2] Hamburger Hafen und Lagerhaus AG, Bei St. Annen 1,
20457 Hamburg, Germany
spindel@hhla.de
http://www.hhla.de

Abstract. We describe the software architecture of the new container terminal Altenwerder operated by Hamburger Hafen- und Lagerhaus-Aktiengesellschaft, which is completely based on Java technologies. While the central information system utilizes J2EE technology, we implemented active Java objects and asynchronous JMS messages for the control of the automated transports. Asynchronous JMS messages also constitute the middleware for integrating the different make and buy components of the system. CTA successfully operates since summer 2002.

Zusammenfassung. Wir beschreiben die Architektur der Software für den neuen Container Terminal Altenwerder der Hamburger Hafen- und Lagerhaus-Aktiengesellschaft, die umfassend auf Java-Technologien basiert. Während das zentrale Informationssystem J2EE Technologie nutzt, haben wir für das Gerätesteuerungssystem der automatischen Transporte aktive Objekte und asynchrone Nachrichten in Java implementiert. Asynchrone JMS-Nachrichten bilden ebenfalls die Middleware für die Integration der verschiedenen, teils zugekauften Teilsysteme. CTA ist seit Sommer 2002 erfolgreich in Betrieb.

Der Container Terminal Altenwerder

Die Hamburger Hafen- und Lagerhaus AG (HHLA) betreibt 3 Container-Terminals im Hamburger Hafen. Im Containerumschlag liegt Hamburg an zweiter Stelle in Europa und an achter Stelle in der Welt. Im letzten Jahr ging der Container-Terminal Altenwerder (CTA) in Betrieb, der modernste der Welt. Für CTA wurde ein neuartiges Automatisierungskonzept entworfen und realisiert, das auf die Produktivitätsanforderungen der Zukunft ausgerichtet ist. Der größte Teil der Umschlagsgeräte und der Software wurde komplett neu entwickelt.

CTA wird in zwei Phasen gebaut: der erste Bauabschnitt ging am 25.Juni 2002 in Betrieb, der zweite Abschnitt ist im Bau. Die Kaimauerlänge beträgt zunächst 800, später 1.400 m. Im Endausbau werden 1,2 Millionen Container pro Jahr umgeschlagen.

Die wesentlichen Ziele, die mit CTA verfolgt werden, lauten

- Steigerung der Produktivität (beim Ent- und Beladen der Schiffe)
- Verbesserung der Qualität (weniger Fehler und damit Korrekturarbeiten)
- kürzere Schiffsliegezeiten
- bessere Planbarkeit der Abläufe
- Reduzierung der Kosten
- Rationalisierung durch Einsatz automatischer Umschlagstechniken
- Verringerung des Flächenbedarfs und der Anzahl an Containerbrücken sowie kürzere Kaimauerlänge gegenüber traditionellen Terminals.

Im Endausbau kommen zum Einsatz:

- 14 Zwei-Katz-ContainerBrücken (halbautomatisch)
- 53 vollautomatische Fahrzeuge (AGV = Automated Guided Vehicle) für den Horizontaltransport zwischen Containerbrücke und Lagerblock
- 44 vollautomatische Lagerkrane (DRMG = Double Rail Mounted Gantry Crane)
- 3 halbautomatische Bahnkrane
- diverse Terminal-eigene Trailer (Chassis) und Zugmaschinen mit Datenfunk

Für die Realisierung des Terminals hatte die HHLA ein großes Projekt aufgelegt, das aus den 4 „Gewerken“

- Bau (Hoch- und Tiefbau)
- Umschlagssysteme (Spezifikation und Beschaffung der Umschlagsgeräte)
- Betriebsvorbereitung (Organisation, Personal, Abläufe, Genehmigungen)
- Informationsssyteme (Hardware, Software, Netzwerk)

bestand.

Geschäftsprozesse und Funktionen

Das Kerngeschäft eines Container-Terminals besteht in der Ent- und Beladung von Containerschiffen, der Zwischenlagerung der Container sowie ihrer landseitigen An- und Auslieferung. Um diese Prozesse effizient zu gestalten, sind ausgeklügelte logistische Verfahren notwendig.

Um diesen operativen Betrieb ranken sich die Geschäftsprozesse der administrativen Vor- und Nachbereitung: Planung der Schiffsbeladung und des Geräteeinsatzes, Konfiguration des Containerlagers, Zollabwicklung, Kommunikation (EDI) mit Reedereien, Fuhrunternehmen, Bahnbetreibern und Behörden, Behandlung von Spezialcontainern (Kühlcontainer, Container mit Übermaßen, Gefahrgut), Personaleinteilung, weitgehend automatische Fakturierung und so weiter.

Eine wesentliche Anforderung an die Software ist „Hafentauglichkeit“. Sie muss robust sowie außerordentlich flexibel sein. Der Terminal-interne Workflow ist eng verzahnt mit Informationen, die im Vorwege von Kunden geliefert werden, wie z.B. Buchungsdaten für einzelne Container. Das System muss deshalb für den Fall des Ausbleibens solcher Daten die Möglichkeit für geeignete Substitutionen bieten.

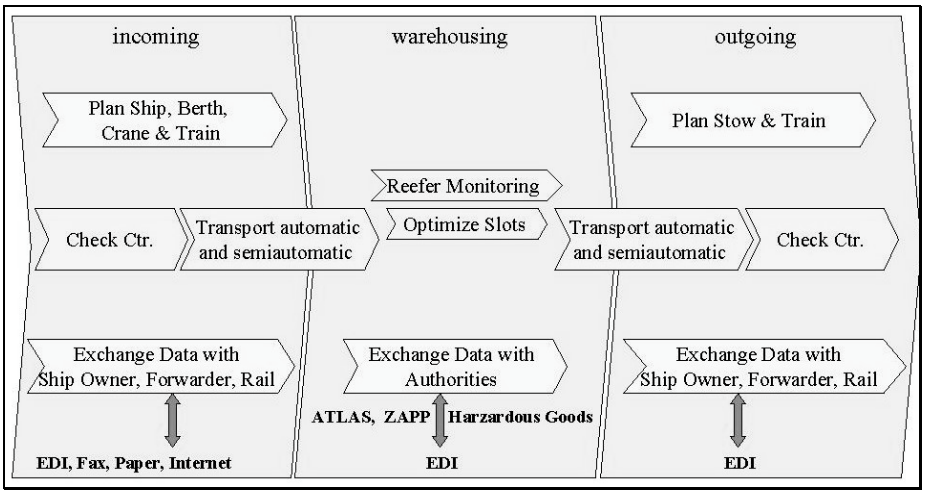

Fig. 1. Die Geschäftsprozesse eines Container-Terminals

Wir haben es also mit einer Ausgangslage zu tun, in der gleichzeitig fließband-artige, nahezu in Echtzeit abzuwickelnde Geschäftsprozesse und solche, die ein hohes Maß an unsicheren Daten verarbeiten und somit stark nicht-deterministisch sind, unterstützt werden müssen.

Herausforderungen

Mit der Entwicklung der neuen CTA-Software verfolgte die HHLA mehrere herausfordernde Ziele gleichzeitig:

- Die Programmiersprache M (früher: MUMPS) hatte bereits keine Zukunft mehr und musste abgelöst werden. Die Wahl fiel auf Java, weil Java die am schnellsten wachsende objektorientierte Programmiersprache war, die auch die größte Zukunftssicherheit versprach. HHLA-Programmierer hatten die ersten Erfahrungen mit Java gesammelt, aber fundiertes Java-Knowhow musste im wesentlichen noch aufgebaut werden.
- Die Host-Architektur sollte durch eine 3-schichtige Client/Server-Architektur abgelöst werden. Die Server dieser Architektur sollten – wie zuvor der Host – im Rechenzentrum in der Hamburger Speicherstadt stehen. CTA ist 15 km vom Rechenzentrum entfernt.

- CTA sollte das erste HHLA-Terminal werden, in dem ein komplett neues Steuerungssystem für den vollautomatischen Transport von Containern sorgt.
- Für die operationale Abwicklung der vollautomatischen Transporte mussten vier große und einige kleine Typen von Umschlagsgeräten unterschiedlicher Hersteller mit der automatischen Steuerung integriert werden.
- Die komplette Software-Entwicklung musste auf jeden Fall rechtzeitig zur Inbetriebnahme des Terminals im Jahre 2002 abgeschlossen sein, damit gab es eine unverrückbare Zeitbox. Entsprechend rechtzeitig mussten die Tests beginnen.
- Die vorgesehene Zeit (und die geplanten Kosten) würden nicht ausreichen, um die komplette Software neu zu implementieren. Deshalb sollte ein Rest mit Kernfunktionalität aus dem alten M-System weiter benutzt werden. Wie groß dieser Rest sein würde, hing vom Verlauf der geplanten Entwicklung ab. Auf jeden Fall würde eine Programmverbindung zwischen dem Java- und dem M-System benötigt werden.
- Einige Komponenten sollten auch von Anfang an zugekauft werden, so z.B. das Planungssystem und das Leerlagersystem. Im Verlauf des Projektes wurde es auch immer offensichtlicher, dass auch die Hinterland-Steuerung (Zulauf zum Lager und zum Container-Bahnhof) zugekauft werden konnte.
- Die Entwicklung der Software begann, bevor alle Anforderungen bekannt waren, es musste also eine agile, d.h. inkrementelle und ggf. iterative Methodik in der Entwicklung der Software angewandt werden, in der die erforderlichen Präzisierungen der Anforderungen jeweils rechtzeitig vor der spätest möglichen Implementierung erfolgte.
- Die neue CTA-Software sollte als Grundlage und Vorbild für die in Zukunft erforderliche Migration der M-Programme der anderen HHLA-Terminals dienen, sie sollte deshalb so weit wie möglich bereits „produktreif“ sein.

Vorsichtige Zeitgenossen empfehlen, höchstens eines der großen Risiken:
- die Einführung einer neuen, objektorientierten Programmiersprache,
- den Umstieg von einer Host-Architektur auf eine Client/Server-Architektur oder
- die Erstimplementierung einer umfangreichen Gerätesteuerung für vollautomatischen Transport

zur Zeit auf sich zu nehmen. Um so größer erscheint der Erfolg, der im Gelingen dieses Projektes liegt. Entscheidend für diesen Erfolg waren neben der herausragenden Leistung der Projektmannschaft drei Dinge:
- Die Migrationsstrategie, die es erlaubte, den in M verbliebenen Funktionsumfang des CBS flexibel zu steuern,
- die Kaufstrategie, die es erlaubte, noch relativ spät im Projekt über Kauf statt Eigenentwicklung zu entscheiden, und
- die agile Entwicklungsmethodik, die es erlaubte, sowohl die Anforderungen sehr spät und schritthaltend mit der Entwicklung zu konkretisieren, als auch die erforderlichen Kompromisse bei der Implementierung transparent zu machen.

Auch die Details der logistischen Algorithmen wurden erst “just in time” während der Entwicklung durch Simulationen untersucht und festgelegt.

Die Herausforderungen und großen Risiken des CTA-Projektes legten eine risikomindernde Architekturstudie nahe, die von Mai bis September 1999 durchgeführt wurde. Diese Studie entschied die folgenden Alternativen:

- Java RMI als Alternative zu einem zusätzlichen Einsatz von CORBA;
- den Einsatz von J2EE, insbesondere Enterprise Java Beans, als Alternative zu möglicherweise funktionsreicherem selbstgeschriebenen Code;
- den Einsatz von JMS als Alternative zu möglicherweise performanterem selbstgeschriebenen Code.

Darüber hinaus wurde hier die Anwendungsarchitektur zum ersten Mal skizziert, insbesondere mit der neuen Umschlagskomponente für zentrale Dienste, die von den ausführenden (Execution) Komponenten Fuhre, Bahn und Schiff genutzt werden sollten.

Die explizite Modellierung des abstrakten Containerumschlags von der ersten Vormeldung bis zur Auslieferung erlaubt es der CTA-Software, anders als auf allen anderen HHLA-Terminals mit mehr als einem Umschlagsvorgang pro Container umzugehen.

In der entscheidenden Frage der M-Anbindung gelang der Nachweis einer zuverlässigen und performanten Implementierung ebenfalls noch in 1999.

Integrations-Architektur: Teilsysteme und Middleware

Die CTA-Software ist aus mehreren Teilsystemen aufgebaut. Dies ist einerseits das Ergebnis der flexiblen “Make AND Buy” Strategie, trägt aber andererseits gezielt zur Modularität und damit Skalierbarkeit und Pflegbarkeit bei.

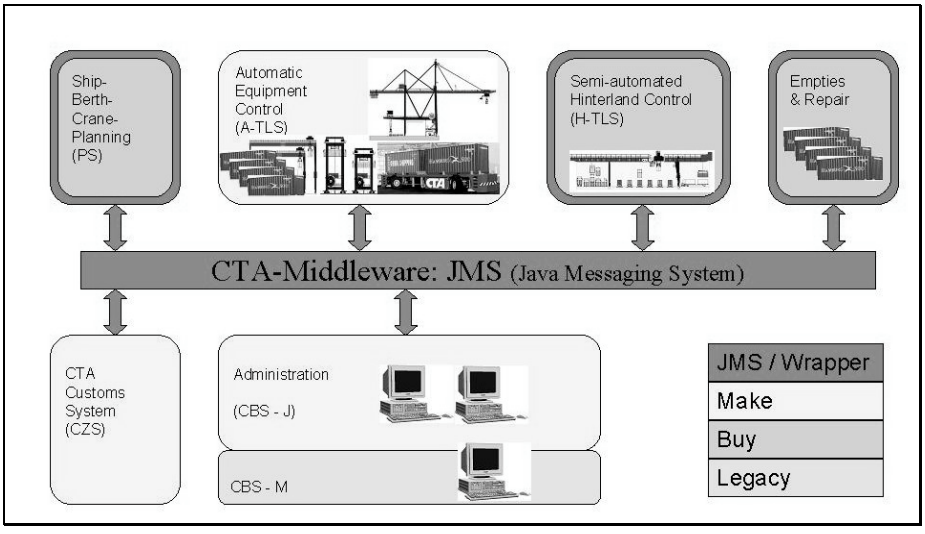

Fig. 2. Die Integrationsarchitektur der CTA-Software.

Neben dem Container-Basis-System CBS, das in seiner M-Ausprägung auf einer VAX im Cluster läuft, und in seiner Java-Ausprägung in einem BEA-Weblogic Applikationsserver auf dem zentralen CTA-Server unter Solaris, gibt es auf der Solaris-Maschine noch jeweils einen oder mehrere Prozesse für

- die Terminal-Logistik und -Steuerung (Die Automatik-TLS steuert den vollautomatischen Containerverkehr mit Containerbrücken, AGVs und Lagerkranen),
- die Hinterland-TLS (bedient die halbautomatischen Bahnkrane und disponiert Zugmaschinen und LKW) und
- das Container-Zoll-System CZS (kommuniziert mit dem ATLAS-System der Zollverwaltung).

Das Planungssystem SPARCS der Fa. Navis für die Schiffs- und Brückeneinsatzplanung läuft auf einem eigenen Windows-Server unter der Kontrolle ausgewählter Nutzer.

Alle Teilsysteme kommunizieren untereinander mit Hilfe des Java-Messaging-Systems JMS, dessen Server ebenfalls auf der Solaris-Maschine läuft.

Alle eingekauften Teilsysteme haben jeweils einen JMS-Wrapper erhalten, der die Übersetzung ihrer Kommunikationsschnittstellen nach JMS und zurück übernimmt und sie dadurch über die Middleware genauso einfach integrierbar und testbar macht wie die selbstgeschriebenen Java-Anwendungen.

JMS ist die Java-Spezifikation einer sogenannten Message Oriented Middleware. JMS bietet neben Punkt-zu-Punkt-Verbindungen auch abonnierbare Broadcast-Verbindungen und wahlfrei eine Zustellgarantie (der Absender kann die Nachricht getrost "vergessen", er darf sich sicher sein, dass sie ungeachtet aller Software- oder Rechnerausfälle vom Empfänger verarbeitet werden wird).

Die Zustellgarantie vereinfacht die Programmierung erheblich, da kein Handshake mehr erforderlich ist. Sie entkoppelt aber auch das Lastverhalten der beteiligten Systeme, da der Nachrichtenverkehr ganz ohne Synchronisation und Warten auf den Empfänger auskommt. Kurze Überlastspitzen eines Teilsystems werden vom Gesamtsystem so einfach in den JMS-Postfächern "weggesteckt".

Das bei CTA eingesetzte Produkt SwiftMQ ist ein ausgereifter, stabiler Nachrichtenserver in nativem Java, der zusätzlich den Einsatz einzelner JMS-Router auf dedizierten Rechner-Knoten erlaubt.

Die Java-Technologie sorgt für die Portabilität der Teilsysteme, so wurden z.B. die Server-Komponenten auf Windows-NT entwickelt, auf NT sowie Solaris getestet und auf Solaris deployed. Für das Projekt bedeutete dies, dass die tatsächliche Entscheidung für das Betriebssystem nicht mehr von der Programmiersprache abhängt und in der Tat recht spät, nämlich erst im Frühjahr 2001 getroffen wurde.

Auch sorgt die Java-Technologie für Skalierbarkeit der Anwendung im TCP/IP-Netz geeigneter Bandbreite. Ob die Enterprise Beans in einem oder mehreren Applikationsservern bzw. einem Cluster deployed werden, wird durch das Programm nicht mehr festgeschrieben. In der A-TLS kann dank der Kommunikation mittels JMS ebenfalls jede Instanziierung einer Komponente in einem eigenen Prozess und damit prinzipiell auf einem eigenen Rechner erfolgen.

Integrations-Architektur: Verteilung

Die neue CTA-Software sollte - wie alle operative Software der HHLA - auf zentralen Servern in der Hamburger Speicherstadt laufen, und zwar selbst die Gerätesteuerung der A- und H-TLS. Die Gründe hierfür liegen vor allem in den - verglichen mit einer dezentralen Installation - geringeren Installations- und Betreuungskosten.

Damit ist klar, dass alle Serverprogramme (mit Ausnahme des Planungssystem-Servers, der von speziell geschulten Nutzern betreut wird) im Rechnerverbund des Rechenzentrums laufen, alle Clients hingegen im LAN des CTA selbst. Der Preis für die zentrale Installation ist eine notwendige Bandbreite von 34 MBit/s für die WAN-Strecke zwischen CTA und dem Rechenzentrum. Über diese Leitung gehen

- (in Richtung Rechenzentrum) Meldungen der Geräte, insbesondere die Statusmeldungen der Containerbrücken und der Lagerkrane sowie Daten aller Dialoge;
- (in Richtung CTA) Kommandos an die Geräte und Bildschirminhalte der Clients, Daten- und Handfunkgeräte;
- (in Richtung CTA) Statusinformation über alle Transportvorgänge und Aufträge der A-TLS sowie über alle Geräte, und zwar gleichlautend für bis zu etwa 10 Leitstandsclients.

Kritisch für die WAN-Bandbreite von 34 MBit/s sind einerseits die Gerätemeldungen und andererseits die Statusinformationen für die Leitstände. Der Bandbreitenbedarf für die Gerätemeldungen wird dadurch minimiert, dass die Ortsmeldungen beim Verfahren jedes einzelnen Fahrzeugs und Krans auf die jeweils minimal erforderliche Frequenz eingestellt wird.

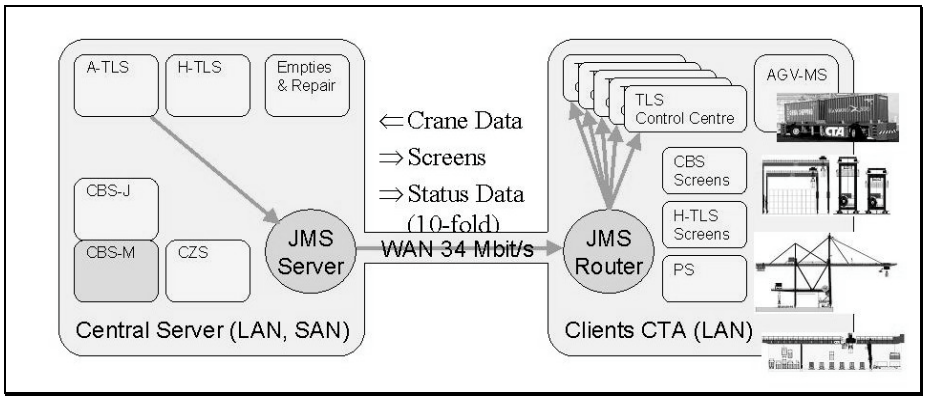

Fig. 3. Physische Verteilung der CTA-Komponenten und Daten über das WAN. Die Pfeile heben die Bandbreiten-kritischen Statusinformationen der A-TLS hervor, die im CTA-LAN zu den Leitstandsclients vom JMS-Router vervielfacht werden.

Die Statusinformationen für die Leitstände werden mehrfach übertragen, nämlich an jeden Leitstand. Um mit der gegebenen Bandbreite auszukommen, empfängt ein JMS-Router im CTA-LAN jedes Update nur einmal und repliziert es n-fach für die

Leitstandsclients. Diese Replikation musste nicht programmiert werden, der geeignet konfigurierte JMS-Router erledigt sie automatisch.

Für alle anderen Komponenten der CTA-SW ist die Bandbreite des 34 MBit/s WAN ausreichend, dort spielt die Netztopologie keine Rolle.

Querschnittliche Architektur-Entscheidungen

Zu den querschnittlichen Architekturentscheidungen gehören:

- Die XML-Spezifikation der Schnittstellen zwischen den Teilsystemen und zu den Geräten (OSI Schicht 6). Mit ihrer Hilfe wurden syntaktische Fehler weitgehend ausgeschlossen.
- Die Entscheidung, alle zugekauften Teilsysteme so in Java zu wrappen, dass sie für die Middleware genauso aussehen wie die selbst geschriebenen.
- Die Spezifikation des Verbindungsauf- und -abbaus zu den Geräten. Dies ist die OSI Schicht 5 für die Datagramme der Geräte. Dadurch wird für die Steuerung von vielen Details der Geräte abstrahiert.
- Ein sehr flexibles Logging mit dem Open Source Framework Log4J. Dieses Logging wurde sogar produktionstauglich.
- Ein Systemparameter-Server, dessen Parameter nicht nur nutzerdefinierbar sind, sondern bei Bedarf auch dynamisch, d.h. sofort im Betrieb wirken.

XML-Spezifikation der Schnittstellen

Spätestens mit dem Advent der WebServices ist die Nutzung von XML-Dokumenten auf Schnittstellen populär geworden. Da die XML-Tags für CTA-Schnittstellen etwa doppelt soviel Datenvolumen darstellen würden wie die Inhalte selbst, wurde im CTA-Projekt die XSLT-Technologie (eXtended Stylesheet Language for Transformation) ausprobiert und eingesetzt.

Alle Schnittstellen wurden in XML spezifiziert. Aus diesen Spezifikationen wurden mit der Hilfe jeweils zweier XSL Stylesheets

- die Dokumentation der Schnittstelle als HTML-Text und
- sowohl die generierenden als auch die parsenden Java-Klassen

generiert.

Diese Technik garantierte die Abwesenheit von syntaktischen Fehlern auf den Schnittstellen, gegen die semantischen hilft auch das natürlich nicht. Bemerkenswert war in der Tat die geringe Zahl von ausschließlich semantischen Fehlern in den Integrationstests.

Die Schnittstellen zu den Umschlagsgeräten waren alle auf Papier spezifiziert - mit all den üblichen Fehlerquellen. Hier half zunächst die nachträgliche Spezifikation in XML, viele Zweifelsfragen zu entdecken und zu klären. Der A-TLS-seitige Code wurde mit der gleichen, eben beschriebenen Technik generiert. In der Folge traten syntaktische Fehler auf den Geräteschnittstellen ausschließlich auf der Geräteseite auf.

Wrapper für die Kaufkomponenten

Die Entscheidung für JMS-Nachrichten als Kommunikationsmittel zwischen den Teilsystemen führte zu der Notwendigkeit, alle anderen Kommunikationsschnittstellen zu wrappen. Dies geschah entweder durch den Lieferanten oder durch das Projektteam selbst. Jeder so entstandene JMS-Wrapper sorgte

- für die Übersetzung der Socket-Datagramme in JMS-Nachrichten und umgekehrt und
- für das Handshake mit dem eingekauften Teilsystem.

Dadurch wurde nicht nur der Test des Teilsystems vereinfacht, da zur Ansteuerung ja immer nur ein vorhandener Generator von JMS-Messages benötigt wurde, sondern es wurden auch Konfigurationsprobleme gelöst. Der JMS-Wrapper wurde konfiguriert wie jedes andere Teilsystem auch. Grundsätzlich läuft er auf derselben Maschine wie das gewrappte Programm. Da das gewrappte Programm aber nur den Wrapper kennen muss, ist seine Installation immer lokal - mit entsprechend konstanten Installationsparametern. Die Konfigurationsparameter des Wrappers erlauben nun eine Fernbedienung und -administration.

Gerätemanager

Eine große Herausforderung bestand in der Integration von Umschlagsgeräten verschiedener Hersteller. Deshalb wurde den Geräteherstellern der Auf- und Abbau der Kommunikation mit den Geräten mit einer speziellen Schnittstellen-Spezifikation vorgeschrieben. Diese Spezifikation spezieller Socket-Datagramme und speziellen Verhaltens deckt praktisch die OSI-Schicht 5 (Session) ab.

Im CTA-System übernimmt der Gerätemanager das Session-Management (Auf- und Abbau der Sessions, Reset) völlig transparent für das Steuerungsprogramm, das nur noch wissen muss, welche Geräte verfügbar sind. Wie der Nachrichtenserver übernimmt auch der Gerätemanager die Zustellgarantie für ein empfangenes Telegramm.

Die zweite Aufgabe des Gerätemanagers ist die Übersetzung von Socket-Datagrammen in JMS-Nachrichten und zurück. Die Nachspezifikation der Datagramme der Umschlagsgeräte in XML (OSI-Schicht 6) erlaubte, auch die Java-Klassen des Gerätemanagers für die Übersetzung zwischen den Datagrammen des jeweiligen Gerätes und den entsprechenden JMS-Messages automatisch zu generieren. Probleme auf der OSI-Schicht 6 (Syntax der Datagramme) traten dem zufolge während der Integration auch nur noch auf der Geräteseite auf.

Die dritte Aufgabe des Gerätemanagers ist die Überwachung aller Geräteverbindungen. Dies ist wichtig, weil sich die logischen Verbindungen zwischen Gerät und Steuerprogramm nicht mit Netzwerkmitteln allein überprüfen lassen.

Logging

Eine wichtige Architekturentscheidung fiel mit dem Einsatz des Logging-Frameworks Log4J. Dieses Framework erlaubt nicht nur eine flexible Strukturierung des Loggings unabhängig von der Java-Klassenhierarchie, sondern auch die dynamische Beeinflussung des Loggings im Betrieb, z.B. wenn Performanzengpässe auftreten.

Es werden prinzipiell viele Log-Dateien erzeugt, die jedoch nur selten wieder gelesen werden müssen. Deshalb ist das Schreiben der Log-Dateien der performanzkritische Aspekt. Hier ist es vorteilhaft, das Logverzeichnis in das Temp-Device der Maschine zu legen. Faktisch ist dies schneller Hauptspeicher, der von der Maschine recht bald ausgelagert wird und die Hauptspeicherressourcen dadurch nicht wesentlich belastet. Durch diesen "Trick" ist es möglich, das Logging dauerhaft ohne Performanznachteile im System zu betreiben.

Systemparameter

Einem Benutzer kann und will man im Allgemeinen nicht zumuten, in einem Java-System Property-Dateien zu editieren. Deshalb wurde ein flexibler Systemparameterdienst eingerichtet, aus dem alle Programme solche Programmkonstanten beziehen, die entweder der Benutzer ändern können soll, oder die dynamisch während des Betriebes änderbar sein sollen.

Bei einfachen Systemparametern wirkt die Änderung beim nächsten Start der betroffenen Komponente. Im Fall "dynamischer" Systemparameter wird das verwendende Programm per JMS-Nachricht von jeder Änderung unterrichtet. Dies ermöglicht es, die geänderten Systemparameter im laufenden Betrieb neu einzulesen und somit die Konfiguration des Gesamtsystems dynamisch zu ändern.

Ein Standard-GUI erlaubt dem entsprechend Berechtigten, einen Systemparameter in der Datenbank zu ändern.

Dieser Dienst hat sich nicht nur als nützlich für die Programmkonstanten und für die dynamischen Parameter erwiesen, sondern er konnte vor allem wegen seines Standard-GUIs unmittelbar für die schnelle und sehr kostengünstige Implementierung einiger notwendiger Funktionalitäten verwendet werden. Hierzu gehörten

- die erste Implementierung von Berechtigungen, bevor der endgültige Berechtigungsdienst verfügbar war, und
- die erste Implementierung des sogenannten Workloadmanagers der A-TLS, mit dessen Hilfe die optimierungsrelevanten Parameter eingestellt werden konnten.

Architektur des Informationssystems CBS

Das Informationssystem CBS ist dreischichtig nach den Regeln der J2EE aufgebaut.

Die oberste oder Dialog-Schicht enthält stateful Session Beans für die Bedienung der Dialoge mit langen Transaktionen. Trotz des größeren Ressourcenverbrauchs der stateful Session Beans ergeben sich Vorteile gegenüber stateless Session Beans dadurch, dass in den Clients auf jegliche Datenspeicherung für die Transaktionssiche-

rung verzichten werden kann. Der Ressourcenverbrauch auf dem Server ist bei einer festen Obergrenze von Clients kalkulierbar.

Die mittlere oder Dienste-Schicht von stateless Session Beans enthält alle Dienste der Domäne, also die „Businesslogik“ mit kurzen Transaktionen.

Die unterste oder Daten-Schicht schließlich besteht aus Entity Beans für den Zugriff auf die Daten. Mit Hilfe des TopLink-Frameworks greifen die Entity Beans auf eine Oracle Datenbank zu.

Als Folge der Migrationsstrategie ist ein Teil des CBS aus der ‚alten‘ M-Software adaptiert, sozusagen als Legacy-Teil. Solange wie das M-System noch benutzt wird, ist auch der Zugriff auf seine Daten erforderlich. Für diesen M-Zugriff wurde ein spezielles M-Bean gebaut [1].

Da es mit der „neuen“ Oracle-Datenbank und dem „alten“ M-System zwei Datenspeicher gibt, ist prinzipiell ein Two Phase Commit Protokoll erforderlich. Von Oracle wird das Two Phase Commit Protokoll unterstützt, im M-System reichte es deshalb aus, mit einem Read-Modify-Write Zyklus zu arbeiten.

Fast alle Komponenten des CBS finden sich mit ähnlicher Funktionalität in anderen Terminalsoftwaresystemen wieder, seien es die ausführenden oder Execution-Komponenten wie ‚Fuhre‘ für den LKW-Verkehr, ‚Bahn‘ für die Zugabfertigung oder ‚Schiff‘ für die Schiffsabfertigung, oder seien es unterstützende Administrations-Komponenten wie ‚Partner‘ für Kundendaten.

Neu ist die Komponente ‚Umschlag‘. Hier wurden alle Dienste und Daten zusammengefasst, die während eines Containerumschlags von wenigen Tagen eine Rolle spielen. Durch die Trennung der Umschlagsdaten von den eigentlichen Containerdaten kann das CBS der CTA-Software - anders als in anderen Terminalsoftwaresystemen - denselben Container in mehreren Umschlagsvorgängen verwalten.

Schließlich vereinheitlicht ein eigenes GUI-Framework „JADIS“ für die CBS-Dialoge Look und Feel der vielen hundert Dialoge.

Architektur des Automatischen Steuerungssystems

Die automatische Steuerung der A-TLS organisiert alle Transporte zwischen den Containerbrücken am Schiff und den Lagerblöcken sowie zwischen den Lagerblöcken und den LKW bzw. Chassis in den landseitigen Übergabespuren. Für das Aufnehmen und Absetzen der Container auf LKW oder Chassis sind aus Sicherheitsgründen Fernsteuerer erforderlich, die mit der Hilfe von Joysticks und Monitoren die Krane vom Betriebsgebäude aus steuern.

Die A-TLS plant, steuert und überwacht die Transporte mit Hilfe zweier abstrakter Objekte:

- Ein Transportvorgang steht für den kompletten Transport z.B. von der Containerbrücke bis ins Blocklager und
- ein Transportauftrag steht für den Teil des Transports, den eines der Geräte erledigen muss.

Korrespondierend ist es Aufgabe der Vorgangskomponente, Transportvorgänge anzulegen, sie zu starten, zu optimieren, zu überwachen und auch wieder zu beenden. Mit Hilfe der Vorgangskomponente kann auch die Leitstandsbesatzung Transportvorgänge anlegen, ändern oder stornieren.

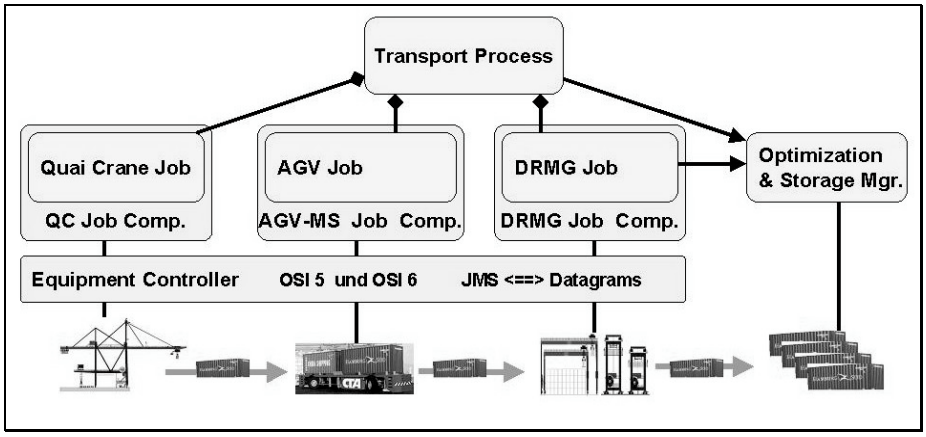

Fig. 4. Die Organisation der automatischen Transporte in der A-TLS.

Eine Auftragskomponente ist verantwortlich für ‚ihr' Gerät. Die AGV-MS-Auftragskomponente verwaltet alle AGVs und ist für die optimierte Belegung aller AGVs mit den anstehenden Transportaufträgen verantwortlich. Eine DRMG-Auftragskomponente verwaltet jeweils 2 Krane in einem Lagerblock. Sie ist für die optimale Zuordnung der anstehenden Transportaufträge zu ihren beiden Kranen verantwortlich.

Die globale Optimierung der Vorgangskomponente wägt z.B. mehrere alternative Einlagerorte bzgl. der Stellplatzqualität und der implizierten Belastung der betroffenen Geräte ab. Die lokale Optimierung der Auftragskomponenten findet i.w. eine optimale Reihenfolge der Transportaufträge für jedes einzelne Umschlagsgerät (Sequenzierung).

Der Gerätemanager übernimmt für die Auftragskomponenten das An- und Abmelden der Geräte, er überwacht und restauriert notfalls die Verbindung, und er garantiert der Auftragskomponente, dass eine für das Gerät empfangene Nachricht dort auch ankommt. Umgekehrt sorgt er dafür, dass eine dem Gerät quittierte Nachricht bei der Auftragskomponente ankommt. Für die Auftragskomponenten ist auch dieses Handshake völlig unsichtbar. Dank der frühen Spezifikation dieses Verhaltens der Geräte können alle Geräte vom Gerätemanager mit der gleichen Logik behandelt werden.

Daneben hat der Gerätemanager die Aufgabe, zwischen den Datagrammen der Geräte und den JMS-Nachrichten der Auftragskomponenten zu übersetzen. Die hier verwendeten Mechanismen entsprechen denen der Teilsystem-Wrapper, d.h. die Übersetzung wird von Java-Klassen vorgenommen, die aus den XML-Spezifikationen der Telegramme generiert wurden.

Wie implementiert man die Vorgangs- und Auftragskomponenten in Java? Bieten sich hier Session Beans an? Wie werden die Zustände der Transportvorgänge und -aufträge gesichert? Bieten sich hier Entity Beans an?

In den Begriffen klassischer Steuerungsarchitekturen geht es um die Implementierung aktiver nebenläufiger Objekte, die Zustände tragen und ein eigenes ‚Leben' in Form von Threads oder Prozessen führen. Diese aktiven Objekte tauschen Nachrichten aus, die ebenfalls einen jeweils relevanten Zustand tragen. Sie erledigen ihre Aufgaben jeweils mit Hilfe anderer aktiver Objekte, um deren Ressourcen sie konkurrieren. Die transaktionale Einheit in dieser Architektur ist eine Nachricht und ihre ungeteilte Verarbeitung.

Wenn die Zustände der Nachrichten und der aktiven Objekte im Hauptspeicher der Maschinen leben, müssen sie zwar für den Notfall gesichert werden, brauchen in der Regel aber nie wieder eingelesen werden. Entity Beans sind hier also das falsche Modell. Stattdessen wurde für die Zustandssicherung der aktiven Objekte ein eigenes kleines Framework geschrieben.

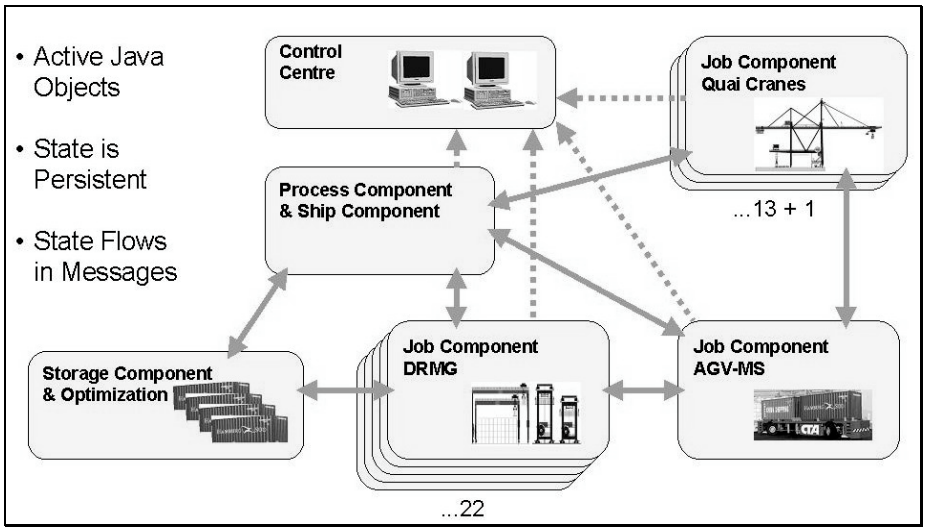

Fig. 5. Die aktiven Objekte der A-TLS und ihre Zusammenarbeit.

Wenn die aktiven Objekte Session Beans wären, dann müssten die Ressourcenkonflikte beim Aufruf anderer aktiver Objekte mit Hilfe von Transaktionen gelöst werden, denn Session Beans kommunizieren synchron. In der Folge wäre das Lastverhalten der Steuerung sehr hart, bei lokalen Ressourcenengpässen würde das System global ausgebremst. Hier bietet sich das JMS mit asynchronen Nachrichten und Warteschlangen (Briefkästen) vor den aktiven Objekten an. Der JMS-Server erledigt auf Wunsch die Zustandssicherung der JMS-Nachrichten, und das Lastverhalten der Steuerung wird sehr elastisch. Das System steckt lokale, zeitlich begrenzte Überlast sehr gut weg. Selbst die neuen, zu Beginn der Entwicklung noch nicht vorhandenen Message Driven Beans sind überflüssig, denn der JMS-Server übernimmt das Ressourcenmanagement der Threads vollständig.

Die lose Kopplung der aktiven Objekte über Nachrichten und Warteschlangen führt auch zu einer hohen Robustheit des Gesamtsystems, in dem notfalls ein einzelnes aktives Objekt beendet und neu gestartet werden kann, ohne die Gesamtfunktionalität zu gefährden.

Auch der Leitstand ist lose an das System angekoppelt. Er erhält einerseits seine Zustandsinformation über abonnierte JMS-Nachrichten und schickt andererseits JMS-Nachrichten mit Queries oder Kommandos an die aktiven Objekte der Steuerung.

Besonderheiten des Entwicklungsprojektes

Die großen Risiken und Herausforderungen dieses Projektes wären ohne besondere Aufmerksamkeit und Vorkehrungen nicht zu bewältigen gewesen.

Zum einen ist da die schiere Größe. Die selbstgeschriebenen Komponenten des Steuerungssystems und des Informationssystems umfassen ca. 1,5 Millionen Codezeilen. Hinzu kommen Kaufkomponenten, für die nochmals ca. 1 Million Codezeilen geschätzt werden müssen. Das ganze System wurde innerhalb von 2,5 Jahren entwickelt bzw. beschafft, integriert und getestet.

Zu den besonderen Maßnahmen des Projektes zählt zunächst die oben bereits erwähnte vorgeschaltete Architekturstudie zur Risikominderung in 1999.

Die eigentliche Entwicklung begann im Jahr 2000. Angesichts der entwicklungsbegleitenden fachlichen Spezifikation und logistischen Simulation war auch eine sehr adaptive Software-Entwicklung gefragt. Dies leistete ein iteratives und inkrementelles Vorgehensmodell mit parallelen Workshop-Strängen für die wichtigsten Komponenten und Teilsysteme, in die Entwickler und fachliche Spezifizierer eingebunden waren. Dort wurden auch letztlich die Notwendigkeiten und Termine für die fachlichen Spezifikationen und logistischen Simulationen just in time festgelegt.

Dank der portablen Programmiersprache Java konnte die Entwicklung auf Windows NT begonnen und durchgeführt werden, obwohl die Hardware- und Betriebssystemauswahl noch nicht abgeschlossen war. Etwa zur Halbzeit des Projektes wurden Betriebssystem und Hardware mit Unix/Solaris festgeschrieben. Die Programme waren problemlos Bytecode-portabel, d.h. es wurde die Produktion aufgenommen mit einem Programm, das unter NT übersetzt und unter Solaris getestet worden war.

Der Test des Steuerungssystems wurde früh automatisiert, um möglichst keine unnötigen Fehler mehr in der Produktion aufschlagen zu lassen, die im Sommer 2002 begann. Auch wurde die Testabdeckung systematisch anhand der Fehler aus Integrationstest und Produktion verbessert. Dies hat sich als sehr segensreich erwiesen, hatte der Betrieb doch noch genug zu tun mit den Restfehlern aus der Geräteintegration.

Zusammenfassung

Die Programmiersprache Java eignet sich für sehr große Systeme mit weit mehr als einer Million Codezeilen. Die Sprache ist erwachsen, sie ist nicht nur geeignet für Informationssysteme nach der J2EE Referenzarchitektur, sondern dank JMS auch für

automatische Steuerungen mit einer Reaktionszeitanforderung im 10-Sekundenbereich.

Gleichzeitig ist JMS mit dem Produkt SwiftMQ für CTA eine ausgezeichnete und stabile Middleware. Aus Performanzgründen wurde in der Middleware zwar weitgehend auf den Austausch von XML-Dokumenten verzichtet, jedoch reichte bereits die XML-Spezifikation der Nachrichten aus, um die Integrationsschicht syntaktisch fehlerfrei zu machen, dies zahlte sich in den Integrationstests aus.

Auf den Geräteschnittstellen, die zunächst mit Datagrammen spezifiziert waren, hat sich sogar die Nachspezifikation mit XML als segensreich erwiesen, einerseits ließ sich dadurch ein Teil des Codes generieren, andererseits waren zumindest auf Seiten der Steuerung syntaktische Fehler ausgeschlossen.

Weiter hat es sich bei der Integration der Geräte verschiedener Hersteller ausgezahlt, die Session-Schicht der Socket-Kommunikation durch einen eigenen HHLA-Standard festzuschreiben.

Mitte 2002 ist CTA mit der neuen Java-Software in Betrieb gegangen, mittlerweile ist die Vollauslastung erreicht.

Referenzen

1. Gerken, A., Krasemann, H., Spindel, U.: J2EE meets MUMPS, Legacy Data Access through Enterprise JavaBeans. NetObjectDays 2000