von bernhard angerer, werner kurschl
und peter lieber

mehr zum thema:
http://www.angerer.com/SBClinks/
http://www-136.ibm.com/developerworks/grid/

„MAINSTREAM GRID COMPUTING": SOFTWAREENTWICKLUNGEN ZUR GLOBALEN VERNETZUNG

Der Ausdruck „Grid Computing" (GC) wurde Mitte der 90er Jahre geprägt und bezeichnete in erster Linie verteilte Systeme mit dem Hauptaugenmerk auf hoch entwickelte, wissenschaftliche Anwendungen. Der Artikel versucht den Begriff dingfest zu machen und beleuchtet aus Sicht der Softwareentwicklung, welche Technologien den Weg zu breiter Anwendbarkeit bereiten werden.

Dieser Artikel versucht einen Überblick über das undurchsichtige (weil neue und dynamische) Gebiet des GC zu geben und weist darüber hinaus auf eine neue Technologie hin, die gemeinhin recht unbekannt ist, sich jedoch in einem vergleichsweise kleinen aber sehr engagierten Kreis von Softwareexperten zunehmender Beliebtheit erfreut und sehr interessante neue Lösungen zu althergebrachten Problemen bietet. Folgende Aspekte werden in dem Artikel behandelt:

- Anliegen von GC
- architektonische Aspekte
- Verhältnis von GC zu anderen Technologien
- GC von IBM und Oracle
- Service-orientierte Architekturen und Agenten (*Agents*)
- *Space Based Computing (SBC)*
- Standardisierungen

Da das, was gestern noch „High Performance Computing" war, heute bereits Mainstream ist, wird GC immer mehr zum Thema von breiterem Interesse und von wirtschaftlicher Relevanz (vgl. [Pla], [Dav02]). Unter dem Schlagwort GC werden nunmehr Infrastrukturen verstanden, die eine „koordinierte, gemeinsame Nutzung jeglicher Ressourcen in dynamischen, verteilten IT-Systemen" ermöglichen (vgl. [Ber03]). Ein Zusammenschluss von mehreren Institutionen zu solch einer Infrastruktur wird als *virtuelle Organisation (VO)* bezeichnet.

VOs sind mit einer Reihe von Fragen und Anliegen konfrontiert, die neue technologische Antworten erfordern. Im Detail werden folgende Punkte identifiziert: sehr flexible Kommunikationsmodelle von Client/Server bis hin zu *Peer-to-Peer (P2P)*, lokale und globale Mechanismen für die Zugriffskontrolle, transparente Nutzung von unterschiedlichen Ressourcen (z. B. Programme, Dateien, Daten, Rechner, Sensoren) sowie unterschiedliche Modi der Benützung (performanz-sensitiv, kosten-sensitiv, im Einbenutzer- oder Mehrbenutzer-Betrieb). Durch die gemeinhin große Anzahl an Benutzern und den stark verteilten Charakter einer VO sind die angeführten Punkte unter Aspekten wie Fehlertoleranz, Realtime-Messaging, Load-Balancing und Clustering (welche im besten Fall von darunter liegenden Ebenen in transparenter Art übernommen werden) zu sehen.

Grid-Architektur

Wie immer in der Informatik ist die Frage der Abstraktionsebenen von entscheidender Bedeutung. Abbildung 1 zeigt Ebenen einer *Grid*-Architektur.

Der *Fabric-Layer* abstrahiert den Zugriff auf lokale Ressourcen bzw. Komponenten und ermöglicht so den koordinierten Zugriff über das *Grid*. Eine Ressource ist typischerweise eine logische Einheit, wie beispielsweise ein verteiltes Dateisystem oder ein Computer-Cluster. Die Aufgabe des *Grids* ist es, diese lokale Komplexität (z.B. spezielle interne Protokolle) vor dem Gesamtsystem abzuschirmen. Anfragemechanismen müssen dabei ein Minimum an Funktionalität erfüllen. Einerseits sollen Auskünfte über Struktur und Status möglich sein, andererseits sollen Möglichkeiten der Kontrolle auf die gelieferten Ergebnisse gegeben sein.

Der *Connectivity-Layer* definiert Kommunikations- und Authentifizierungsprotokolle, die einer *grid*-spezifischen Infrastruktur gerecht werden. Die Komplexität dieser Ebene wird durch deren Sicherheitsaspekte bestimmt und umfasst Charakteristika wie *Single Sign On, Delegation* oder *Trusted Relationships*. Dabei soll die flexible Handhabung von den verschiedenen Sicherheitskontexten, die benutzerbasiert oder durch spezifische Sicherheitssysteme (beispielsweise „Unix-Security") bestimmt sein können, ermöglicht werden.

Der *Resource-Layer* ist für die Überwachung, Kontrolle und Protokollierung individueller Ressourcen verantwortlich. Diese

die autoren

Dr. Bernhard Angerer (E-Mail: bernhard@angerer.com) ist unabhängiger Berater und widmet sich ganz den Space-Technologien, deren Anwendungen und Kommerzialisierungen.

Dr. Werner Kurschl (E-Mail: Werner.Kurschl@fh-hagenberg.at) ist Professor an der Fachhochschule Hagenberg und beschäftigt sich mit verteilten Systemen und „Space Based Computing".

Peter Lieber (E-Mail: peter.lieber@lieberlieber.com) beschäftigt sich mit „Enabling Business". Seine Schwerpunkte sind objektorientierte Softwarearchitektur mit UML und strategische Firmenbeteiligungen.

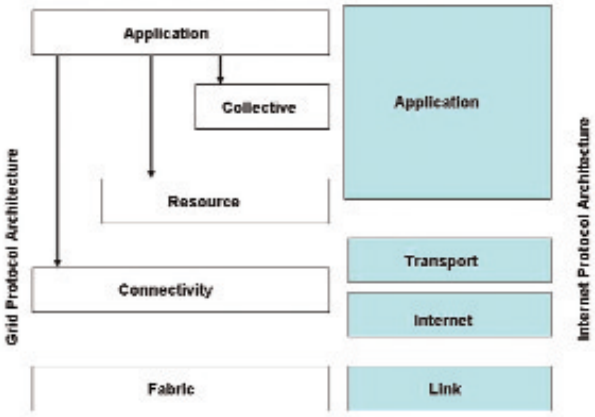

Abb. 1: Ebenen der Grid-Architektur (aus [Fos01])

Ebene bedient sich des *Fabric-Layers* und ignoriert Fragestellungen globaler Statusinformation oder verteilten transaktionalen Verhaltens. Informations- und Management-Protokolle sind die Kernbereiche dieser Schicht, wobei unter dem ersten Punkt die Informationsabfrage über Struktur und Status (z. B. aktuelle Auslastung oder Konfigurationsdaten) zu verstehen ist, wohingegen die zweite Protokollart auf die Kontrolle und Instanziierung der jeweiligen Operationen abzielt. Verfahrensweisen bezüglich der Berechtigungen, Anforderungen und Reservierungen von Ressourcen sind hierbei wichtige Bestandteile.

Der *Collective-Layer* befasst sich mit „globalen Fragen" und umfasst Interaktionen quer über unterschiedliche Bereiche. Diese Schicht führt eine Reihe von Funktionen ein, ohne mit der tatsächlichen Ressource in direktem Kontakt zu stehen. Die folgenden Dienste charakterisieren tatsächlich die Eigenschaften und Komplexität eines *Grid*-Systems:

- Verzeichnisdienste
- Vermittlungs- und Einteilungsdienste
- Überwachungs- und Diagnosefunktionen
- Datenreplikation
- Programmiersysteme (z. B. *grid*-fähige Nachrichtenbus-Systeme oder *Master/Worker*-Frameworks, vgl. [Gou00])
- Auslastungsmanagement
- *Community*-Authentifizierung
- *Community*-Buchführungs- und Zahlungssysteme
- *Collaboration Services*

Zu beachten ist, dass der *Collective-Layer* bereits nahe an der Anwendungsschicht liegt und daher ein weites Spektrum an Eigenschaften implementiert, von sehr allgemeinen bis hin zu speziellen Funktionen, die nur in bestimmten VOs zum Einsatz kommen mögen. Manifest wird diese Schicht entweder in persistenten Diensten mit assoziierten Protokollen oder in Form von SDKs (mit assoziierten Programmierschnittstellen), um die Anbindungen an Anwendungen bewerkstelligen zu können.

Der *Application-Layer* definiert sich im Wesentlichen durch das Konstruieren von Anwendungen mit Hilfe der darunter liegenden Ebenen und kann dabei mit anderen anspruchsvollen Architekturen, wie beispielsweise der *Common Object Request Broker Architecture (CORBA)* (vgl. [Gan99] [Lop00]) oder der *Common Component Architecture* (vgl. [Arm99]), kommunizieren.

Das Globus-Toolkit

Das Globus-Toolkit (siehe www.globus.org) ist ein open-source *Grid*-Projekt, das für die unterschiedlichen Komponenten eines *Grid*-Systems die Protokolle und Programmierschnittstellen definiert und dazugehörige Referenzimplementierungen in C (für Client-seitige Programmierschnittstellen in Java) produziert und anbietet (Trademarks sind in Besitz der Universität von Chicago und Southern California):

- *Fabric-Layer:* Das Toolkit verfolgt die Strategie, bereits existierende *Fabric*-Komponenten verwenden zu können, vervollständigt andererseits jedoch fehlende Funktionalität im Sinne der übergeordneten *Grid*-Ziele.
- *Connectivity-Layer:* Public-key basierte *GSI*-Protokolle (*Grid Security Infrastructure*) werden definiert.
- *Resource-Layer:* Auf LDAP basierendes *Grid Resource Information Protocol (GRIP)* oder eine erweiterte Version von *FTP (GridFTP)* sind definiert.
- *Collective-Layer:* Hier werden Meta-Verzeichnisdienste bereitgestellt. Der *GIIS (Grid Information Index Server)* bietet frei wählbare Sichten auf unterschiedlichste Ressourcen.

Kasten 1: Das Globus-Toolkit

Verhältnis zu anderen Technologien

Halten wir fest: Die in Produktion befindlichen Technologien bieten keine Frameworks an, die generelle Mechanismen und Funktionalität zur verteilten Benutzung von Ressourcen in VOs adressieren.

Systeme wie CORBA oder J2EE ermöglichen die Realisierung von verteilten Systemen, zielen aber eher auf lokales Ressourcemanagement ab. Operationen quer durch Firewalls sowie Echtzeit- und Multimedia Unterstützung waren nicht Teil des ursprünglichen Systemdesigns.

Die grundsätzliche Natur von P2P-Computing (es existiert kein Server, stattdessen arbeitet eine „Föderation von Engines") entspricht ganz der Philosophie des GC. In der tatsächlichen Anwendung ist P2P zur Zeit noch auf sehr „vertikale" Bereiche beschränkt (z. B. Datei-Tauschbörsen) und wird nicht mit Middleware, generellen Protokollen und Funktionalität zur Interoperabilität identifiziert. Das Projekt „JXTA" von Sun (vgl. [JXTA]) bietet eine Menge genereller P2P-Schnittstellen und -Protokolle, die sich breiter Akzeptanz erfreuen und daher zu einem Industriestandard werden könnten.

Web-Services (WS) bieten *Dynamic-Discovery* von Diensten quer durch heterogene Systemlandschaften sowie die automatische Generierung der dazu notwendigen Proxys zur Kommunikation, basierend auf definierten Schnittstellen. Da WS ein weithin akzeptierter Industriestandard sind und so Interoperabilität garantieren, stellen sie eine zentrale Komponente auf dem Weg zum globalen GC dar. Was WS nicht bieten, sind neue Kommunikationsmuster, die das Abstraktionsniveau erhöhen und damit die

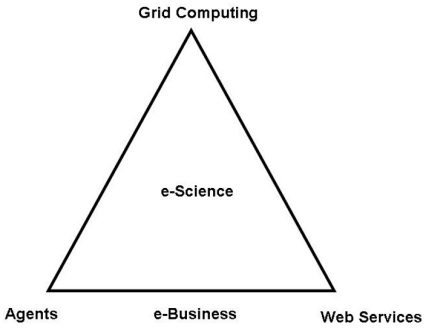

Abb. 2: der Weg zum E-Science (nach [Ber03])

Entwicklung von verteilten Systemen erleichtern. Aus diesem Grund sind Technologien erforderlich, die das Bild vervollständigen und zusätzliche notwendige Bausteine darstellen. Hier ist als zentrale Komponente *Agent-Based Computing* (vgl. [Jen01] und [Jen99]) zu nennen (siehe Abb. 2).

GC von IBM und Oracle

Betrachtet man die verschiedenen Hersteller *grid*-basierter Systeme oberflächlich, so scheint es, dass jeder dieser Hersteller seinen eigenen Ansatz verfolgt.

Das Hauptaugenmerk der Hersteller liegt offensichtlich auf der Vermarktbarkeit von *Grid* bei ihren Zielgruppen. Hinter den Kulissen wird aber sehr wohl zusammen gearbeitet, wie das Projekt „Globus" (vgl. [GGF]) zeigt, wobei sich dieses auf die Ebene der Entwicklungswerkzeuge und Applikationsserver beschränkt.

Nach Bekanntgabe der neuesten Version der Oracle-Datenbank „Database 10g" wurden meistens zwei Fragen gestellt: Die erste Frage richtete sich auf die Bedeutung des „g", das zusammen mit der Versionsnummer das altbekannte „i" für „Internet" abgelöst hat. Bei der zweiten Frage (nach entsprechender Aufklärung, dass das „g" für „Grid" steht) ging es darum, ob *Real Application Cluster (RAC)* mit *Grid Computing* gleich zu setzen sind. Diese Frage kann sicher verneint werden, obgleich aus Datenbanksicht RAC die Basis für ein ausfallsicheres und skalierbares GC ist. Während der Service-Gedanke innerhalb eines *Enterprise-Grids* sicherlich auch die Applikationen und damit rein technologisch Applikationsserver oder ähnliche Architekturen mit berücksichtigen muss, kann er aus Sicht von RAC auf die Datenbank und datenbanknahe Applikationen beschränkt werden. Deshalb ist RAC auch nicht gleich *Grid*.

RAC ist eine Weiterentwicklung des lange bewährten „Oracle Parallel Servers" (OPS) und bietet neben hervorragender Ausfallsicherheit eine über Rechnergrenzen hinausgehende Skalierbarkeit praktisch aller Anwendungen. Oracle hat die Zeichen der Zeit offensichtlich erkannt und nutzt die gerade im Großkundenbereich einsetzende Euphorie für die von IBM eingeläutete Renaissance des Mainframes, die durch *Grid* nun wieder sinnvoll und möglich erscheint, vor allem in Bezug auf die Integration, das Clustering usw., was im Großrechnerbereich schon seit mehreren Jahrzehnten ganz normaler Alltag war.

IBM hat bisher weltweit über 100 verschiedene *Grid*-Projekte bei Kunden in Produktion genommen und in den letzten Monaten 19 industriespezifische *Grid*-Lösungen mit Partnern auf den Markt gebracht. Der Fokus von IBM bei der Vermarktung im Bereich *Grid* liegt auf Hardware und Infrastruktur – ausgehend vom Großrechner hin zu virtuellem Speichern und virtuellen Prozessoren. Das wichtigste Ziel von IBM ist die Nutzung von unnötig brach liegenden Systemressourcen; dabei werden folgenden Zahlen angeführt: Großrechner sind 40% ihrer Zeit „unbeschäftigt". Server im universitären Umfeld arbeiten weniger als 10% ihrer Rechenzeit. Und die meisten PCs machen 95% ihrer Zeit nichts. (Zum Vergleich stelle man sich nur einmal eine Fluglinie vor, bei der 90% der Flotte am Boden steht, oder einen Autohersteller, der 40% der Zeit nicht produziert, oder ein Hotel, das zu 95% leer steht.)

Daran erkennt man, dass IBM weiterhin auf ihre E-Business-Strategie setzt und *Grid* als wichtige strategische und vor allem infrastrukturelle Maßnahme fürs E-Business sieht (vgl. [Shu02]).

GC im Unternehmensumfeld optimal einzusetzen heißt, alles mit allem auf den unterschiedlichsten Ebenen zu integrieren, was zur service-orientierten Sichtweise führt. Oracle wäre damit kein Datenbankserver mehr, sondern ein Service des Dienstes „Datenbank". *Service-orientierte Architektur (SOA)* ist der wesentliche konzeptuelle Ansatz und GC ermöglicht die Durchführung. Viele Hersteller haben dies erkannt und so ist die Frage nicht, *ob* SOA und GC sich durchsetzen, sondern nur *wann*.

Agenten/SOA

Agenten sind gekapselte Computersysteme, die – eingebettet in einer bestimmten Umgebung – in flexibler und autonomer Art und Weise Aktionen ausführen und damit zu einer übergeordnete Gesamtaufgabe beitragen. Sie stellen *Producer, Consumer* und *Broker* von Diensten dar und führen somit zu SOAs, wobei das Client/Server-Paradigma aufgebrochen wird und jeder Teilnehmer in gleichberechtigter Art und Weise eine Kommunikation starten und führen kann (P2P-Modell).

Nachdem Agenten mit Hilfe von asynchronen und entkoppelten (*loosely coupled*) Mechanismen arbeiten und kommunizieren, stellen sie eine signifikante Erweiterung der Möglichkeiten im Design und der Implementierung dar. Die Entkoppelung führt bei verteilten Systemen zu mehr Stabilität und *aktivem* Verhalten in dem Sinne, dass Aktionen nicht immer in synchroner Abfolge

Evolution – nicht Revolution

Grid Computing (GC) kann als Evolution und Komplettierung von bekannten Entwicklungen und Technologien (wie etwa verteilte Systementwicklung, das Web, Peer-to-Peer und Virtualisierungstechnologien) gesehen werden:

- *Gleich dem Web* verbirgt GC Komplexität, d. h. viele Benutzer erfahren ein vereinheitliche System.
- *Ungleich dem Web,* das hauptsächlich Kommunikation ermöglicht, erlaubt GC volle „Collaboration" zur Erreichung von gemeinsamen Geschäftszielen.
- *Gleich Peer-to-Peer* ermöglicht GC es den Benutzern Daten zu verteilen.
- *Ungleich Peer-to-Peer* erlaubt GC das vielfältige Verteilen und Teilen von jeglichen Ressourcen.
- *Gleich Clustern* bringt GC Computer-Ressourcen zusammen.
- *Ungleich Clustern,* die homogen sein müssen, kann GC mit geografischer Verteilung und Heterogenität umgehen.
- *Gleich Virtualisierungstechnologien* erlaubt GC das Virtualisieren von Computer-Ressourcen.
- *Ungleich Virtualisierungstechnologien,* die einzelne Systeme virtualisieren, erlaubt GC das Virtualisieren von sehr großen, unterschiedlichen Systemen.

Kasten 2: ein IBM Statement (siehe: www-1.ibm.com/grid/about_grid/what_is.shtml*)*

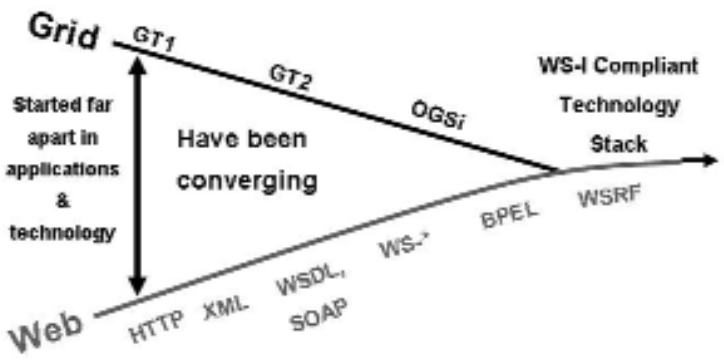

Abb. 3: Grids und Web-Services-Standards (nach [Bro04])

(meist ausgehend von einem initialen Anstoß) vonstatten gehen müssen, sondern beliebig, den logischen Prozessketten folgend abgearbeitet und beantwortet werden können. Batch-Prozesse werden eliminiert und durch Echtzeit-Verhalten ersetzt. Damit stellen Agenten eine adäquate Architektur für sich dynamisch ändernde Systeme dar.

Loosely Coupling / Space Based Computing

Loosely Coupled Computing – also das Entwickeln von entkoppelten Komponenten – wird in der Industrie immer intensiver diskutiert (vgl. [Loo]). Das Thema wird jedoch fast ausschließlich anhand von Web-Services aufgerollt und besprochen. Um jedoch tatsächlich flexible – weil entkoppelte – Systeme bauen zu können, ist die Realisierung von SOAs auch in feineren Granularitäten, als es mit Web-Services sinnvoll bzw. möglich ist, unabdingbar.

Hier kommt *Space Based Computing* ins Spiel, welches das RPC-Modell ablöst, eine vollkommen entkoppelte, transaktionale, P2P-Kommunikation bereitstellt (vielfältige Kommunikationsmuster werden unterstützt) und daher das ideale Medium darstellt, um agentenbasierte bzw. service-orientierte Architekturen zu bauen. Ein Aspekt der den *Space*, wie er gemeinhin genannt wird, auszeichnet, ist die Eigenschaft, Kommunikations- und Koordinationsmedium gleichzeitig zu sein (wobei hier durch die Eigenschaften des virtuell verteilten Hauptspeichers jeglicher Kontext in persistenter Art und Weise mittransportiert wird) (vgl. [Fre99], [Fle02], [Küh01]). Damit werden sowohl der Designer als auch der Entwickler mit einem Schlag von einer Reihe von Problemen befreit (vgl. [Kur04]), die bei der Entwicklung von verteilten Systemen auftauchen und mit Infrastrukturen wie CORBA nicht in diesem Maße zu lösen sind. Industrieprojekte zeigen sehr gute Ergebnisse und signifikante Quellecode-Einsparungen (vgl. [Ang02]). Das *Space*-Modell unterscheidet sich auch insofern grundlegend von klassischen *MOM*-Servern (*Message Oriented Middleware*), als dass die *Space-Engine* selber verteilt ist und es damit zu keinem Server-Flaschenhals kommen kann. Die Firma Tecco aus Wien ist in diesem Feld ein Vorreiter (vgl. [Tec]).

SBC kann nicht als *Grid Computing* im Sinne einer „VO-Infrastruktur" bezeichnet werden, weil es die höheren Ebenen eines GC-Frameworks (wie z. B. in dem Globus-Projekt definiert) nicht besetzt. SBC wird trotzdem mit GC in engen Zusammenhang gestellt, da es die Abstraktionsebene bei der Entwicklung von verteilten Systemen erhöht (ein transaktional sicherer Datenraum vereint heterogene Systemlandschaften), ohne dabei die Granularität im Sinne der Kontrolle und der Eingriffsmöglichkeiten zu verlieren (vgl. [Li03]). SBC virtualisiert alle Ressourcen mit einer einheitlichen, flexiblen und sehr schlanken Kommunikationsschnittelle und bietet dabei Eigenschaften, wie Ausfallssicherheit durch Replikation, verteiltes Rechnen (*Master/Worker*-Muster) oder *Real-Time-Messaging* durch inhärentes Caching.

Standardisierungen

Im Rahmen des Globus-Projekts arbeitet IBM an der *Open Grid Services Architecture (OGSA)* (vgl. [GGF]), deren Ziel es ist, Web-Services und GC zu vereinen (siehe Abb. 2). Die Hauptanstrengungen der Standardisierungen, was sowohl Web Services als auch die Globus *Grid*-Ebenen betrifft, richten sich auf die Schnittstellen. Ein Zusammenwachsen dieser beiden Richtungen liegt auf der Hand (siehe Abb. 3).

In dem skizzierten Szenario stellt sich natürlich auch die Frage nach der Positionierung und den Absichten von Microsoft. Hier werden die Dinge naturgemäß anders benannt und so gibt es keine explizit ausgezeichnete *Grid-Strategie*. Freilich arbeitet man auch in Redmond an Technologien, die das Entwickeln von verteilten Systemen stark vereinfachen (vgl. [Box04]) und die Abstraktionsniveaus in den einzelnen Schichten neu definieren werden.

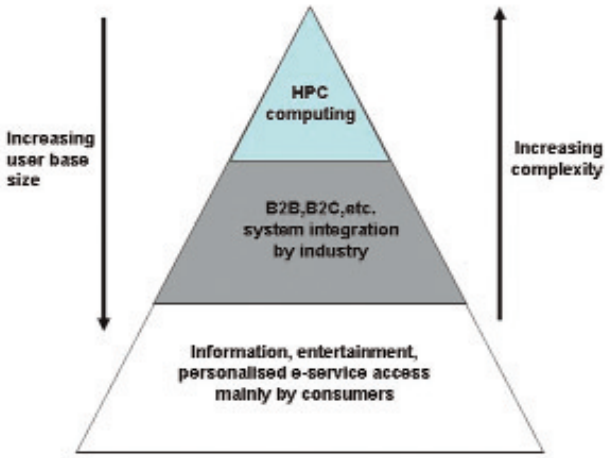

Abb. 4: Ebenen von Grid-Diensten (nach [Juh02])

Fazit

Die Zukunft gehört sicherlich P2P-basierten Virtualisierungstechnologien im Sinne von Agenten und service-orientierter Sichtweise, um damit umfassende Dienste, die auf höherer Ebene einer holistischeren Sicht der IT

Literatur & Links

[Ang02] B. Angerer, „Space Based Programming": J2EE bekommt Konkurrenz und das aus dem eigenen Lager, in: OBJEKTspektrum 04/02
[Arm99] R. Armstrong et al, Toward a Common Component Architecture for High Performance Scientific Computing, in: Proc. 8th IEEE Symp. on High Performance Distributed Computing, 1999
[Ber03] F. Berman, G. Fox, T. Hey, Grid Computing – Making the Global Infrastructure a Reality, Wiley 2003
[Box04] D. Box, A Guide to Developing and Running Connected Systems with Indigo, in: MSDN Magazine 2004 (siehe: msdn.microsoft.com/events/pdc/default.aspx?pull=/msdnmag/issues/04/01/indigo/default.aspx)
[Bro04] M. Brooks, Service Oriented Architecture & Grid Computing, The MITRE Corp., 3rd Emerging Components Technology Conference, Washington DC 2004 (siehe: www.web-services.gov/Brooks32404.ppt)
[Dav02] C. Davidson, JP Morgan unveils Project Compute Backbone, in: Waters Technology, Vol.9, No. 18, 10/02
[Fle02] R. Flenner, Jini and JavaSpaces Application Development, Sams Publishing, 2002
[Fos01] I. Foster, C. Kesselman, S. Tuecke, The Anatomy of the Grid – Enabling Scalable Virtual Organizations, The Globus Alliance, 2001 (siehe: www.globus.org/research/papers/anatomy.pdf)
[Fre99] E. Freeman, S. Hupfer, K. Arnold, JavaSpaces Principles, Patterns, and Practice, Addison-Wesley, 1999
[Gan99] D. Gannon, A. Grimshaw, Object-Based Approaches, in: I. Foster, C. Kesselman (Hrsg.), The Grid: Blueprint for a New Computing Infrastructure, Morgan Kaufmann, 1999
[GGF] Open Grid Services Architecture, siehe www.globus.org/ogsa/ und www.ggf.org/ Global Grid Forum
[Gou00] J.P. Goux et al, An Enabling Framework for Master-Worker Applications on the Computational Grid, in: Proc. 9th IEEE Symp. On High Performance Distributed Computing, IEEE Press 2000
[Jen99] N.R. Jennings, Agent-Based Computing: Promise and Perils, Computers and Thought Award Lecture: IJCAI-1999 (siehe: www.ecs.soton.ac.uk/~nrj/download-files/computers-thought.pdf)
[Jen01] N.R. Jennings, An agent-based approach for building complex software systems, in: Comm. of the ACM, 2001
[Juh02] Z. Juhasz, Making Grid Computing Mainstream. Dept. of Information Systems, University of Veszprem, Ungarn 2002
[JXTA] Project JXTA – open, generalized peer-to-peer protocols, siehe: www.jxta.org
[Küh01] E. Kühn, Virtual Shared Memory for Distributed Architecture, Nova Science Publishers, 2001
[Kur04] W. Kurschl, Space-Based versus Message-Passing Communication – A Comparison, Technical Report, 2004 (siehe: webster.fh-hagenberg.at/staff/kurschl/pubs/TR.2004.01.html)
[Li03] S. Li, High-impact Web tier clustering, Part 2: Building adaptive, scalable solutions with JavaSpaces, Wrox Press, 2003 (siehe: www-106.ibm.com/developerworks/java/library/j-cluster2/)
[Loo] Portal für „Loosely coupled business process automation", siehe: www.looselycoupled.com
[Lop00] I. Lopez, G. Follen, R. Gutierrez, I. Foster, B. Ginsburg, O. Larsson, S. Martin, S. Tuecke, NPSS on NASA's IPG: Using CORBA and Globus to Coordinate Multidisciplinary Aeroscience Applications, in: Proc. NASA HPCC/CAS Workshop, NASA Ames Research Center 2000
[Pla] Platform Computing, Grid solutions for the electronics industry, siehe: www.platform.com
[Shu02] L. Shu, S. Bell, Real-Time Strategies Process for a Changing World, Gartner Group 2002
[Tec] Tecco Software Entwicklungs AG, Coordinated Shared Objects / Corso Virtual Server Technology, siehe: www.tecco.at

gerecht werden und *Grids* bilden, adäquat implementieren zu können. In der Praxis ist freilich noch ein weiter Weg zu gehen. Die Schwierigkeit sowie die Herausforderung gleichermaßen besteht dabei in dem Paradigmenwandel, der zu durchlaufen ist. So fordert beispielsweise SBC ein „Vergessen" gewohnter Herangehensweisen und Lösungsmethoden (verteilte Datenstrukturen bzw. Objekte im *Space* als zentrale Kommunikationsdrehscheiben im Gegensatz zum Publizieren von Interfaces), zu vergleichen mit der Objektorientierung, die nur zur Entfaltung kommt, wenn prozedurale Lösungswege über Bord geworfen werden.

Ein wichtiger Erfolgsfaktor für die *Grid*-Initiativen auf dem Weg zum Mainstream wird sicher die Flexibilität und Modularität im Sinne einer breiten Anwendbarkeit werden. Die alleinige Nutzbarkeit von *Grid*-Systemen im Bereich des *High Performance Computing (HPC)* (siehe Abb. 4) ist zu wenig. Eine intensive Kommunikation zwischen den einzelnen „Gemeinden", auch auf menschlicher Ebene, wird hier sicherlich Vorraussetzung sein (wie durch das Web ja auch schon bedingt). Ebenso muss die leichte Anwendbarkeit mit all den Aspekten der Entwicklungswerkzeuge und des *Deployments* sowie der Wartung mehr als in der Vergangenheit im Blickfeld stehen. Microsoft macht der Industrie immer wieder aufs Neue vor, was durch leicht zu bedienende Benutzungsoberflächen und somit für den Mainstream zugängliche Software (und zwar in allen Bereichen) zu erreichen ist. Die Stringenz von „VisualStudio" des „.NET Frameworks" ist hier nur ein weiteres Beispiel.