VSDB

Space Based Computing

Ausarbeitung

Daniel Dimitrijevic Thomas Traxler

19. Dezember 2013

5AHITT

Inhaltsverzeichnis

1 Erklärung

1.1 Wofür steht Space-Based Computing?

Space-Based Computing (fortan SBC) hat seine Ursprünge im parallel programming, und stellt dabei hauptsächlich ein Datenorientiertes Modell zur Koordination dar. Wie aus dem Namen eindeutig hervorgeht handelt es sich hierbei um ein Modell welches auf 'Spaces' basiert. Ein Space ist hierbei gleichbedeutend mit einem (logischen) Ort auf welchem Daten von mehreren Komponenten geteilt verwendet werden. Diese Komponenten können im einfachsten Fall 'write', 'read' und 'take' Aktionen ausführen. Ein Write steht hierfür für das zur Verfügung Stellen eines neuen Datenteil an alle anderen Komponenten in diesem Space, ein Read für das Lesen von Daten ohne diese zu entfernen und ein Take für das Lesen und entfernen einer Datei aus dem Space, auch destructive read genannt. Im einfachsten Fall hat ein Space nun das Datenmodell eines Tuples.
Eine Hauptanforderung die an einen Space hier nun gestellt wird ist das persistente Aufbewahren aller Daten die sich in ihm befinden, auch bei Systemausfällen. Wie das SBC modelliert und implementiert wird kann sehr stark variieren, je denn gegebenen Anforderungen und Wünschen entsprechend. Was hierbei vor allem variiert ist die Zahl der Geräte auf dem der Space implementiert ist und die Zahl der Geräte die auf diesen Space zugreifen, diese Zahlen sind prinzipiell, wenn nicht durch den Anwendungsfall anders umgesetzt, voneinander unabhängig und befinden sich jeweils im Bereich von 1 bis n.
Das SBC-System stellt hierbei eine logische zentrale Einheit dar welche nicht spezifiziert wo genau sie Physikalisch vorhanden ist. Der Ort wo eine Datei schlussendlich wirklich abgespeichert wird kann dabei nach verschiedensten Methoden ausgewählt werden, in der simpelsten Form wird das selbe Prinzip wie bei Tuples angewendet oder es werden andere Prinzipien verwendet wie FIFO, LIFO, keys, geo-coordinates oder noch kompliziertere, auch das ist vom gegebenen Anwendungsfall abhängig.

1.1.1 EAI

Enterprise Application Integration (EAI) Systeme sind der heutige Ansatz um der oft allgemein üblichen 'chaotischen' Kommunikation entgegen zu wirken. Gemeint sind damit beispielsweise Sockets, MOM, RPC oder RMI die meistens nur so wie es gerade notwendig ist direkt mit allen Möglichen anderen Komponenten kommunizieren, zwar genau so wie es für ihre Aufgabe notwendig ist im Gesamtbild aller Komponenten jedoch eben Chaotisch.
EAI Systeme sind Beispielsweise Object Request Broker (CORBA DCOM...), Message

Queues, Message Broker und Enterbrise service Buses.
All diese Systeme sind jedoch hauptsächlich Nachrichten-orientiert und ziemlich komplex, daher für Kommunikation zwar brauchbar, jedoch für tatsächliches Arbeiten mit verteilt abgespeicherten Daten nur bedingt ratsam, hier kommt nun SBC ins Spiel.

1.1.2 Unterschied zwischen SBC und Cloudcomputing

Um nun der Verwechslung von SBC und Cloudcomputing vor zu Beugen sie hier nun gesagt, zu aller erst behandelt SBC lediglich die Daten und Cloudcomputing stellt noch viele weitere Ressourcen (wie zB. Rechenleistung) zur Verfügung. Auch zu Cloud-Storrage kann man abgrenzen, Cloud-Storrage behandelt nicht nur die interne Abspeicherung der Daten sondern auch vor allem wie diese dem Benutzer zur Verfügung gestellt und abstrahiert werden, bei SBC geht es hauptsächlich um die interene Verteilung und Kommunikation der Komponenten untereinander und stellt dabei einen Datenkanal zwischen den Komponenten des Spaces dar beziehungsweise zur Verfügung.

1.2 Einsatzbereiche

SBC wird vor allem immer dort verwendet wo Daten verteilt abgespeichert werden sollen. Dies kann auf verschiedenste Arte durchgeführt werden um so 'Bottlenecks' zu schließen. SBC bietet durch die Verteilung des Spaces auf mehrere Komponenten grundlegend immer mehr zur Verfügung stehenden Speicherplatz, wodurch man je nach Implementierung verschiedene Vorteile gewinnen kann. Man kann diesen Speicherplatz beispielsweise direkt logisch zur Verfügung stellen und so den logisch zur Verfügung stehenden Speicherplatz erhöhen, man kann die notwendige Kommunikation auf einer Komponente verringern (durch Aufteilung) und man kann für Redundanz sorgen. Die Vorteile lassen sich hierbei auch annähernd beliebig Kombinieren wobei zu beachten ist, das 2 Kombinierte Vorteile auch bei bester Umsetzung so gut wie immer einen dritten Vorteil verringern oder noch andere Nachteile nach sich ziehen. Redundanz schlägt sich Beispielsweise immer auf den zur Verfügung stehenden Speicherplatz nieder, was durch erhöhte Kommunikation zwischen den Komponenten jedoch wieder verringert werden kann.
Ein Beispiel für einen weiteren Vorteil wäre zum Beispiel die Möglichkeit an jeder Komponente Zugang zum ganzen Space zu haben, was in einem Master-Client System erhöhte Kommunikation zum Master mit sich bringt, außerdem müssen die Clients in diesem Fall auch durchgehend die Möglichkeit haben zumindest einen Master ansprechen zu können. Dieser Vorteil stellt sich automatisch ein wenn man auf ein Peer-to-Peer System wechselt, wobei auch oftmals stark belastete Master-Komponenten nicht mehr notwendig sind aber in der Regel wieder mehr Kommunikation zwischen den Komponenten notwendig ist.

2 Grundlegende Prinzipien

2.1 Space-Based Computing Paradigma

Interaktionen sind bei SBC in 3 Teile geteilt

- Zeit

  Applikationen können jederzeit Daten Lesen und schreiben.

- Space

  Applikationen müssen nur auf den selben Space zugreifen um miteinander zu kommunizieren

- Referenzierung

  Applikation die miteinander Kommunizieren müssen nicht tatsächlich voneinander wissen.

Durch diese Aufteilung sind die Applikation tatsächlich voneinander unabhängig, ändert sich nun an einer Applikation etwas (Ort, Ausführungszeit etc.) so ist dies für die andere gänzlich irrelevant (abgesehen von sich ändernden Inhalt) solange sich der Space auf den beide zugreifen und über welchen sie Kommunizieren der gleiche bleibt. Auch wenn zu einem späteren Zeitpunkt weitere Applikation hinzugefügt werden, so ist das (erneut, ausgenommen Nachrichteninhalt) für die ersten beiden Applikation insofern irrelevant als das man an ihnen nichts ändern muss deswegen. Zu beachten ist jedoch dass so keine direkte Kommunikation mehr stattfindet und daher auch keine direkte Reaktion zu erwarten ist in der Regel. (Ausgenommen es werden notifies aus geschickt)

2.2 Mapping

3 Im genaueren betrachtet

3.1 Tuple Spaces

Tuple Spaces

Ein Tupel Raum ist eine Ausführung des assoziativen Gedächtnisses Modell für verteiltes / paralleles Verarbeiten. Es bietet eine Bibliothek von Tupeln, auf die gleichzeitig zugegriffen werden kann. Tupel sind Mengen, mit null oder mehr Argumente und einem Schlüssel.

Die Sammlung von Tupeln unterstützt einige Grundfunktionen, wie zB das Hinzufügen eines Tupels in den Raum (Schreiben) und Entfernen eines Tupels aus dem Raum (nehmen). Das Tupel Sammlung wird von einem Netzwerk über mehrere Server aufbewahrt und gemanaged. Mehrere Threads können zur selben Zeit auf den selben Raum zugreifen.

Nun gehen wir ein wenig auf Linda ein.

3.1.1 Linda

Ziel von Linda ist es, Prozessen einer Anwendung zu erlauben, miteinander zu kommunizieren, ohne identifikations Informationen des Datensatzes zu wissen. Linda geht von einem Tupelraum als Umgebung aus. Verschiedene nebenläufige Prozesse eines verteilten Programms kommunizieren über einen gemeinsamen Tupelraum dadurch, dass jeder dieser Prozesse diesem Tupelraum beliebig Tupel hinzufügen und Tupel daraus entfernen kann.

Die daraus resultierende zeitliche und räumliche Entkoppelung ermöglicht einen Entwurf verteilter Protokolle, die flexibel und robust auf die Herausforderungen verteilter Programmierung reagieren können wie z.B auf Latenz, erhöhten Synchronisationsaufwand und mögliche Teilausfälle des Systems.

3.2 Triple Spaces

Bis jetzt werden die meisten "Messages"dirket zwischen den Machinen ausgetauscht. Triple Space Computing möchte das die Daten in einen Space geschrieben wird und das jeder auf die Elemente des Spaces zugreifen kann und diese lesen/schreiben/entnehmen kann.
Zitat:İn der Zukunft könnte Triple space Computing das Web für die Machinene werden wie HTML das Web für Menschen geworden ist".

Triple Space Computing bietet einen Kommunikations Paradigmas des anonymen und asynchronen Informations Austausch und ebenfalls die Persistenz und Einzigartige Identifikation von Daten.

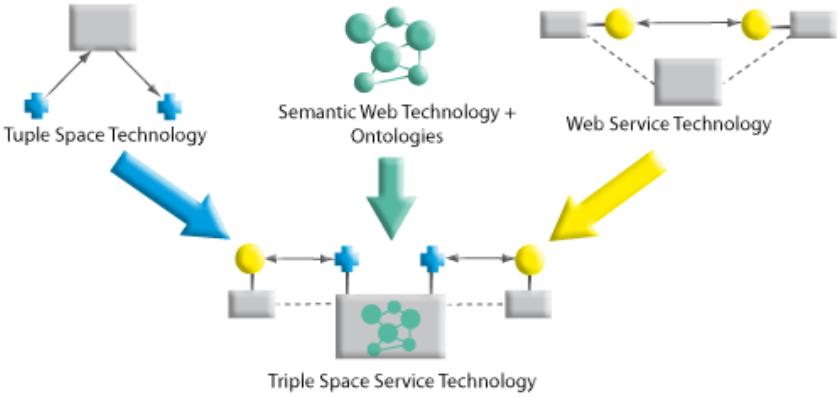

Abbildung 3.1:

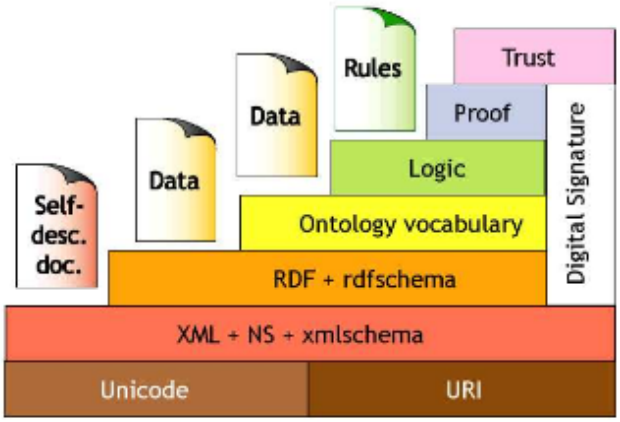

Abbildung 3.2:

Die Semnatische Web Technologie ist eine erweiterung des heutigen Webs in dem die Informationen eine gut definierte, Maschinel verarbeitbare Bedeutung und ein ermöglicht es Maschinen und Menschen leicht zu Co-Operieren.
Es ist gedacht das um das Web mit einem Netzwerk von adressierbare URI Informationen, diese sind so verlinkt und vorgestellt das es Maschinen leicht gemacht wird sowohl syntactisch und semantisch darauf zuzugreifen. Für diesen Zweck sollten die Ressourcen mit maschinen verständlciehn metadaten versehen das standatisiert ist durch verwendung von Allgemeinem Vocabular und einer vordefinierten Semantic auch bekannt als Öntologie"

Web-Service Technologie bietet eine "Virtuelle Komponenten-Modellßur vereinheinfachung der heterogenenen Welt von Komponenten. Es erlaubt es, bestehende Funktionalität zu nutzen ohne die Last der Mittelware spezifischen Eigenheiten, wie die invokation-Mechanismus, Transport-Protokoll usw. Web Services sind eine gut verstandenes Kommunikationsstruktur und-Architektur für Unternehmensanwendungen.
Mehr zu diesem Thema in der Ausarbeitung über SOA.

3.3 Reliable Message

Reliable Message verhält sich wie Tuple Space computing. Nur das man hier nicht von Tuples spricht sondern von Messages. Die Reliable Message Technologie geht von mehreren Channels aus die die geteilten Daten beinhalten und jeder dieser Channel kann von einem oder mehreren Server „unterstützt“ werde d.H. das die geteilten Daten auf einem oder mehreren Servern liegen und damit abgesichert werden können.

3.4 Peer-to-Peer Architekturen

Peer-to-Peer (fortan p2p) Architekturen bieten nun ganz eigene Möglichkeiten aber auch Anforderungen an und für ein SBC System. P2p bedeutet Gleichheit aller Komponenten, mindestens insofern, als dass es keine bestimmenden und steuernden (Master) Komponenten gibt, alle Komponenten gleichgestellt sind. Wobei zu beachten ist, dass es nicht nur erlaubt, sonder erwünscht ist, wenn Unterschiede bezüglich zB. den Komponenten zur Verfügung stehenden Ressourcen berücksichtigt werden.

3.4.1 Weshalb p2p bei SBC?

Ein (gutes) p2p System bietet vor allem die Möglichkeit einfach Komponenten hinzu zu fügen, heraus zu nehmen oder aus zu tauschen, was gerade für einen Space der nicht vollständig definiert sein muss einen guten Vorteil bieten kann. Durch eine p2p Lösung kann sehr oft hohe Flexibilität des Systems erreicht werden, hier sei nun gleich angemerkt, sollten die Anforderungen möglichst effiziente Nutzung von begrenzten oder klar definierten Ressourcen sein, so ist dies zu berücksichtigen und ein p2p Ansatz sehr wahrscheinlich nicht die beste Lösung.
Ein p2p System liefert einem außerdem den Vorteil, dass man bei sehr dynamischen Systemen nicht oder nur bedingt darauf achten muss, dass alle Komponenten Arten (zB. Master-Servant) in einem effizienten Gleichgewicht zueinander stehen da Prinzipiell alle Komponenten der selben Art entsprechen sollten.

3.4.2 P2p Arten/Routing

Die Arten von p2p Systemen kann man in der Regel dadurch unterscheiden wie das Routing innerhalb des Systems funktioniert, d.h. wie schafft es Komponente A mit Komponente B-Z Kontakt auf zu nehmen. Das Routing in (größeren) p2p Systemen

ist entscheidend dafür wie skalierbar, effizient, sicher und ressourcenschonend es ist.

Prinzipiell kann man p2p Systeme in Unstrukturierte und Strukturierte Systeme unterteilen. Unstrukturierte Systeme stellen dabei den weit bekannteren Teil der Systeme und umfassen unter anderem Zentralisierte Systeme (zB. Napster), Pure p2p Systeme (zB. Gnutella 0.4) oder Hybride p2p Systeme (zB. Gnutela 0.6) wärend Strukturierte p2p Systeme beispielsweise mit Distributed-Hash-Tables realisiert werden können.

3.4.3 Unstrukturierte p2p Systeme

Zentralisiertes p2p

Zentralisiertes p2p ist nur bedingt ein tatsächliches p2p System. Es besteht zwar durchaus aus vielen Gleichberechtigten Komponenten (zb. Datenspeichereinheiten) welche die Kommunikation und Organisation selbstständig regeln, jedoch existiert hierbei ein zentraler Server bei welchem sich die Peers registrieren müssen und welcher den Peers die Information mitteilt wie diese den von ihnen gesuchten Peer erreichen können. Dies führt zwar einerseits zu geringerem Traffic zwischen den Peers da diese lediglich mit der zentralen Einheit kommunizieren müssen um einen anderen Peer zu erreichen, außerdem müssen die Peers lediglich abspeichern wie sie den zentralen Server erreichen, jedoch entspricht es nur bedingt tatsächlich dem p2p Konzept.
Die Sicherheit des Systems hängt hierbei direkt von der der Sicherheit der Zentralen Einheit ab.

Pures p2p

Ein pures p2p System besteht aus x gleichberechtigten Komponenten die eine Verbindung mit y (y<x) Komponenten besitzen. Hier existiert keine zentrale Einheit und jede Komponente versucht möglichst direkt Kontakt mit jeder anderen Einheit auf zu nehmen. Wie das Routing innerhalb des Systems durchgeführt wird kann sehr Unterschiedlich sein, jedoch ist es in der Regel bis zu einem gewissen Grad 'chaotisch' da es auf sehr kurzfristige Änderungen reagieren können muss. Pures p2p ist in der Regel sehr dynamisch da jederzeit neue Komponenten an jedem Ort hinzu kommen können und diese Verbindungen zu allen Möglichen anderen Komponenten ziehen können, bzw. permanent abspeichern wie diese erreichbar sind. Da dies jedoch meist (pseudo-)zufällig vonstatten geht ist Effizienz sehr oft nicht gegeben da zB. geographische Verteilung oder Belastbarkeit der Knoten nicht berücksichtigt wird. Das hingegen kann dazu führen dass man eine Komponente erreichen will die im Nebenraum platziert ist aber aufgrund des chaotischen Routings und der Chaotischen Verbindungsfindung dafür über 3 Kontinente gerouted wird.

Hybrid p2p

Hybrid p2p ist eine Kombination aus zentralisiertem und purem p2p. Bei hybridem p2p bilden die Komponenten des Systems 'Gruppen' nach bestimmten Faktoren (meis-

tens geographische Verteilung) und bestimmen eine oder mehrere Hauptkomponenten welche den Großteil des Routings zu den anderen Gruppen übernimmt. Die bestimmte Hauptkomponente ist in der Regel die welche am meisten Netzwerklast verarbeiten kann und am meisten Kontaktdaten zu anderen Komponenten abspeichern muss. Diese lokale Hauptkomponente ist dadurch in der Ausübung ihrer eigentlichen Tätigkeit eingeschränkt, jedoch erhält das gesamte Netzwerk so eine Struktur welche den Nachteilen der Chaotischen Verbindungsführung entgegenwirkt und es kann dennoch leicht dynamisch gehalten werden da bei Ausfall einer Hauptkomponente einfach eine neue bestimmt werden kann. Auch kann man hier Gruppen wieder zu Gruppen zusammenfassen. (zB. Gruppenebene 1 fasst alle Komponenten eines Landes zusammen, Gruppenebene fasst alle Gruppen der ersten Ebene eines Kontinents zusammen)
Hybrid p2p Systeme werden auch als 2. Generation unstrukturierter p2p Systeme bezeichnet.

3.4.4 Distributed Hash Tables

Distributed Hast Tables (fortan DHT) stellen den hier angeführten Vertreter von strukturierten p2p Systemen dar. Bei einem strukturierten System ist zu jederzeit eine ganz bestimmte Komponente für eine (bzw. mehrere) ganz bestimmte Anfrage zuständig. und zwar insofern als dass für jede theoretisch mögliche Anfrage innerhalb des Systems eine Komponente zuständig ist, nicht mehr und nicht weniger.
Um diesen recht abstrakten Satz nun ein wenig zu veranschaulichen, nehmen wir an ziel des Systems ist es eine Datenbank auf zu bauen. So hat jeder Datensatz einen eindeutigen ihm zugehörenden Hashwert, für diesen Hashwert ist nun eine spezielle Komponente zuständig und diese muss wissen wo dieser Datensatz genau zu finden ist.
Ein konkreteres Beispiel: Die Datensätze bekommen einen Hashwert im Bereich 1 bis 9 und wir haben 3 Komponenten. Jede Komponente ist nun für einen Bereich der Hashwerte zuständig, sprich Komponente 1 muss wissen wo sich datensätze 1 bis 3 befinden und welche Komponente sie benachrichtigen kann um Datensätze 4-9 zu finden. Wie vorallem das finden der Datensätze 4-9 Implementiert ist und wie aufgeteilt wird welche Komponente für welchen Hashwert zuständig ist hängt vom verwendeten DHT-Algorithmus ab.
Um das vorherige Beispiel ein wenig praxisnäher zu bringen könnte man den Hashwert 1 ersetzen durch alle MD5 Haswerte die mit 1 beginnen oder ähnliches.

Chord Algorithmus

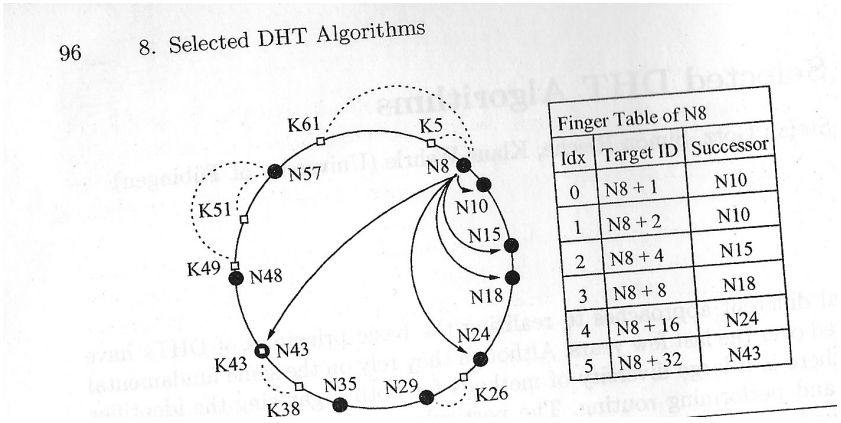

Abbildung 3.3: 6-Bit Chord identifier Space. Gepunktete Linien geben an welche Komponenten (Nodes) welchen Key zu den Datensätzen betreuen.

Der Chord Algorithmus legt alle möglichen Hashwerte in einem Kreis ab. Jede Komponente muss nun wissen wo sie die Datensätze findet für die sie selbst zuständig ist (Für N8 Beispielsweiße K58-K8) und eine gewisse Anzahl anderer Komponenten nach folgendem Schema:
'Each node maintains a routing table, the finger table, poiting to other nodes on the identifier circle. Given a circle with l-bit identfiers, a finger table has a maximum of l entries. On node n, the table entry at row i identifies the first node that succeeds n by at least 2^{i-1}, i.e., successor $n + 2^{i-1}$, where $1 <= i <= l$. For example, the second finder of node N8 ($8 + 2^1 = 10$) is node 10 and the third finder ($8 + 2^2 = 12$) is node 15' Wobei zu beachten ist,dass N15 für den Bereich der Keys 11 bis 15 zuständig ist.
Dadurch ergeben sich selbst bei großen Speicher-Spaces (zB. 256 Bit) relativ kleine Finger Tables nur und somit auch nur kleiner Speicheraufwand zum Routing und erreichen aller Datensätze. Das Routing selbst wird immer genauer je näher man dem gewünschten Key kommt, die Anzahl der Routing-schritte befindet nicht daher im Bereich von O(log(N)).
Würde N8 in unserem Beispiel K38 erreichen wollen so würde er den näherstehen vorhergehenden Node im Figure Table fragen, d.h. N24. N24 würde das selbe nun wieder machen, daher die Anfrage zu N35 weiterschicken. Bei N35 ist der näherste Node zu K38 ident mit dem ersten Node im Figur Table (N43), N35 würde N8 daher nun eine Nachricht schicken dass N43 für K38 zuständig ist. Würde N8 dagegen K51 wissen wollen wären die Schritte N43-N48 und dieser würde dann N57 als Antwort schicken.
In einem Chord netzwerk mit 1000 Komponenenten werden ungefähr O(10) Schritte gebraucht um die zuständige Komponente für den gesuchten Key zu finden. Der Chord Algorithmus hat außerdem die Möglichkeit zur Selbstorganisation bei neu hinzukommenenden Komponenten und Komponentenausfall

CAN Algorithmus

4 Namhafte Implementierungen

4.1 JavaSpaces

JavaSpaces ist eine Spezifikation des Konzepts Object Spaces in der Programmiersprache Java. Ein Object Space ist hierbei ein assoziativer Speicher von verteilten, über das Netz erreichbaren Objekten. Kommunikationspartner (peers) kommunizieren ausschließlich indirekt über diese Objekte. Dadurch etabliert der JavaSpace einen äktiven, verteilten Datenraum", wie er in keiner anderen Technologie geschaffen wird (traditionelles Grid-Computing). Einige Ansätze der Jini-Technologie kommen hierbei zur Anwendung. Bei der Idee, die sich hinter den JavaSpaces verbirgt, handelt es sich nicht um eine revolutionäre Neuerung, sondern sie basiert im Wesentlichen auf den Linda TupelSpaces.

Die Gründe, warum JavaSpaces eingesetzt werden, sind vielfältig. Meist wird Skalierbarkeit und Verfügbarkeit bei gleichzeitiger Reduzierung der Gesamtkomplexität angestrebt.

4.2 Corso

Corso(Co-ORdinated Shared Objects) basiert auf Forschungsarbeiten der Technischen Uni Wien. Software Entwicklung in heterogenen verteilten Systemen bringt immer zusätzlich Komplexität.
• Die Addressierung und Lokalisierung von Ressourcen
• Replikation: Verwalten von Daten Kopien an verschiedenen Orten
• Transaktion
• Skalierbarkeit: Die Möglichkeit der Erweiterung des Systems im Zuge der wachsenden Anforderung und Belastung des Systems.
• Lastverteilung
• Ausfahlsicherheit
Corso ist eine Middleware die auf dem MOM(Message Oriented Middleware) Prinzip aufbaut. Corso nimmt sich als Middleware der oben genanten Problemen an und erleichtert es dem Entwickler da er sich nicht um diese kümmern muss. Corso erleichtert die Addressierung und vereinfacht das arbeiten mit Spaces.

4.2.1 Koordinationsmodell

- Zuverlässige Kommunikation durch gemeinsame Datenobjekte

- Nebenläufigkeit
- Wiederherstellbarkeit und Flexible Koordinationspatterns.

Im Corso Modell kommunizieren autonome Services über gemeinsam genutzte verteilte Objekte. Jedes dieser Objekte hat eine OID(Object IDentifikation) und ist dadurch eindeutig im Netz identifizierbar.

Die Objekte werden von Agenten überwacht. Diese Agenten sollen sicherstellen das alle Daten konsisten sind. Das heißt Transaktionen werden atomar gemacht und entweder ganz oder garnicht ausgeführt.

Die Nebenläufigkeit wird mit Corso Prozessen sichergestellt. Die Prozesse habe eine bestimmte Aufgabe und diese werden entweder durch Systemprozesse oder Threads realisiert. Corso Prozesse unterstützen den Austasch von Objekten da man nur zugriff auf Objekte hat die eine Gehören, ein subObjekt eines erstellten Objektes ist oder man die Referent auf dieses Objekts besitzt. Das heißt Objekte benötigen eine Autorisierung.

4.2.2 Virtual shared Memory

Ein Virtual Shared Memory stellt einen freigegeben Datenraum für die Kommunikation und Zusammenarbeit von mehreren Autonomen System Komponenten zur Verfügung, auch genant Peers. Peers können ihre Anfragen in diesen Datenraum schreiben, als auch User und Nutzdaten, die für die Arbeit erfoderlich sind. Das Konzept des Virtual Shared Memory ist aus dem Parallel Computing bereits bekannt und anerkannt.
Das Virtual Shared Memory Ansatz ermöglicht es autonomen Arbeitern miteinander zu im Virtuellen Datenraum zu interagieren.
Das heißt das alle arbeitenden Peers haben Zugriff auf die geteilten(verteilten) Daten in einem gut strukturiertem, sicheren, virtuellen Speicher.

4.3 XVSM

XVSM (eXtensible Virtual Shared Memory). XVSM ist eine Middleware Technology die Daten in “Container” speichert und für andere Peers teilt. Dieser Weg bringt einige Vorteile wie die Daten werden auf mehreren verschiedenen Computern verteilt, damit wird die Ausfallwahrscheinlichkeit der Daten reduziert.

XVSM ist ein Virtual Shared Memory dessen Funktionalitäten sehr leicht durch Software Entwicklern, der durch die Verwendung von Aspekten, erweitert werden kann. Man kann nicht davon ausgehen das eine Middleware alle Funktionalitäten bieten kann und man wird nicht ein Produkt nutzen das nicht die benötigten Funktionalitäten hat und man wird nicht gerne eine Middleware nutzen die überladen ist.
Mit XVSM kann man es so Konfigurieren, dass die gewünschten Funktionen hinzufügt werden(Pluggable Komponents) oder, wegnehmen wenn ungewünscht sind.(In der Graphik dargestellt durch die Gelben Elemente)

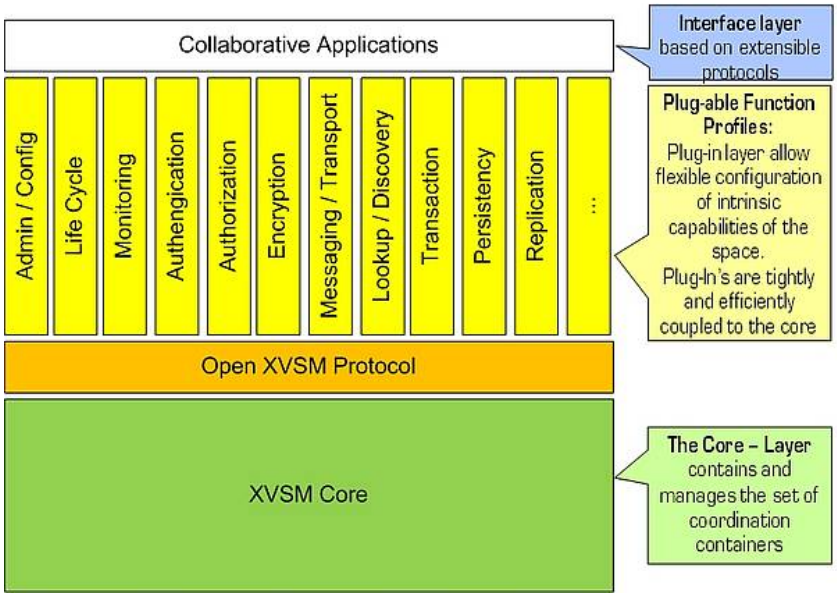

Abbildung 4.1:

Unterschied zum Linda(JavaSpaces)Ansatz ist, dass die Daten in Container gespeichert werden und nicht direkt in den Space und die Daten mit einträgen, coordinaten und containern leichter gefunden werden können. XVSM integriert das Konzept von Peer-2-Peer Netzwerken um ein Verteiltes Geteilten Space zu erzeugen.

4.3.1 Implementierungen

- MozardSpaces
- XcoSpaces
- TinySpaces

4.4 TinySpaces

Tinyspaces setzt auf das XVSM prinzip auf und benutzt das .Net Framework. Da es bei dem XVSM Prinzip noch nicht vollkommen mit allen Implementierungen kompatibel war erschuf man Tinyspaces. TinySpaces soll es erleichtern mit space based computing zu arbeiten. Da es unnötige schichten entfernt. TinySpaces nutzt die unterstützung von Contracts die sagen wie man ein bestimmtes Interface ansprechen soll. Durch die Nutzung von Contracts ist es möglich die angesprochenen Componenten zu ändern. Da man bei der Programmierung der Aplication darauf zielen sollte das Alle Componenten auf denn Contract aufbauen sollten. TinySpaces setzt auf CAPI-1 auf damit sie denn Wechsel von Componenten während der Laufzeit zu ändern.

4.5 GSpaces

Geht von dem selben Ansatz aus wie JavaSpaces(Tuple Spaces) hat aber einen großen Unterschied. GSpaces hat zwar Tupel, benutzt diese aber mit Replikationsrichtlinien. Bei einem Aufruf wird ein lokaler Aufrufshändler aufgerufen und es wird nachgeschaut welche Operation ausgeführt werden muss. Bei einem Aufruf wird erst mal nachgeschaut welche Richtlinien angewendet werden müssen. Nachdem man die Auswahl getroffen hat wird die Anfrage weiter an einen Verteilungsmanager weitergeleitet. Wenn dann zB gelesen werden soll und es auf Master-Slave-Richtlinie herrscht wird dann einfach die Read Operation auf dem lokalem Datensatz(Slice) gelesen. Falls es aber eine write Operation ist muss der Verteilungsmanager beim Master Server anfragen ob er schreiben darf und dann kann er erst weiter schreiben.

5 Conclusio

6 Quellen