SPACE BASED COMPUTING 2

Implementationen und Anwendungsfälle

Adrian Bergler 5BHIT

Inhalt

Abstract

This elaboration extends the **"Space-Based Computing 2015"** elaboration created by **Dominik Scholz** and **Samuel Schmidt**, which should be read first. This continuation will go into further detail about some Space-Based Computing implementations and demonstrate their value by explaining some real life projects in which SBC is used.

This elaboration is for educational purposes only.

LIME

LIME steht für „Linda in a Mobile Environment“ und ist eine Java-basierte Middleware, die für die Entwicklung von Applikationen die entweder physische oder logische Mobilität anfordern ausgelegt.

Grundsätzlich unterscheidet man zwischen 3 verschiedenen Versionen (LIME, TinyLime und TeenyLime) die auf unterschiedliche Anforderungen und Umgebungen ausgelegt sind. Dabei verwendet man TinyLIME sowie TeenyLIME für mobile Umgebungen die mit **Wireless Sensor Networks** arbeiten und LIME für andere mobile Anwendungsgebiete die nicht direkt auf Sensorik angewiesen sind. [3]

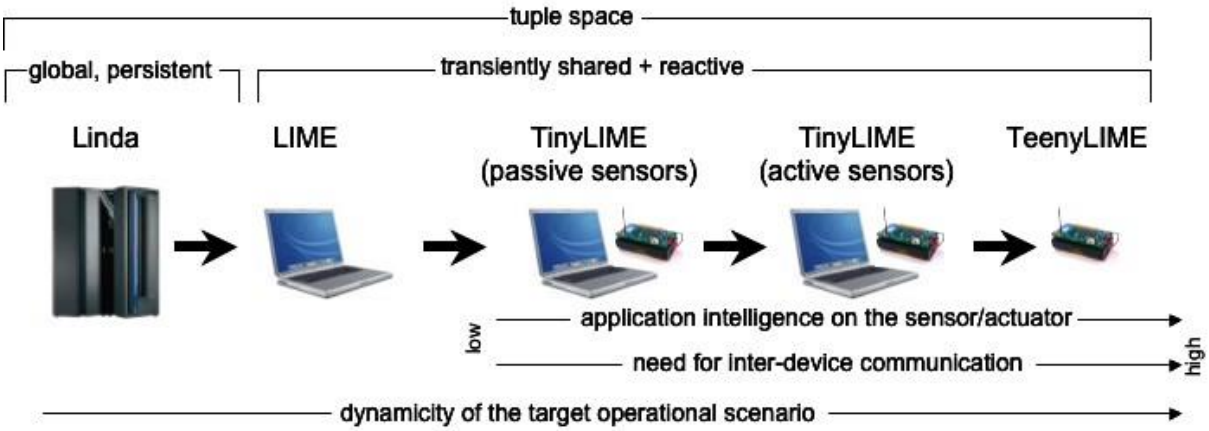


Figure 1: http://lime.sourceforge.net/img/limeProgression.jpg

Wireless Sensor Networks

Wireless Sensor Networks (WSNs) sind Netzwerke bestehend aus mehreren verteilten Sensormodulen, die zur flächenabdeckenden Aufzeichnung von Daten verwendet werden. Sie wurden anfangs primär entwickelt um für einen zentralen Knotenpunkt bzw. einer zentralen Überwachungsstation Sensorwerte zu sammeln, wo diese dann erst später analysiert werden. Ein Beispiel hierfür wäre das Aufzeichnen von Wetterdaten und darauffolgendes Erstellen von Statistiken.

Gegenwärtige WSNs haben allerdings oft auch die Anforderung auf externe Stimuli zu reagieren. Zusätzlich bringen Lösungen die mit einem zentralen Knotenpunkt arbeiten ähnliche Probleme mit sich wie Client-Server Architekturen (Skalierbarkeit usw.). TinyLIME und TeenyLIME bieten die Möglichkeit solche zentralen Knotenpunkte zu dezentralisieren oder gar komplett darauf zu verzichten. [7]

TinyLIME

Mit TinyLIME ist es möglich anstelle eines zentralen Knotenpunktes mehrere mobile Überwachungsstationen für die Auslesung der Sensordaten zu verwenden. Diese Überwachungsstationen lesen die Werte der Sensoren in ihrer Nähe aus und teilen die gesammelten Daten mithilfe drahtloser Verbindungen. Dies ist vor allem dann sinnvoll wenn die Sensormodule sehr verteilt platziert sind, sowie Anwendungsfälle wo Daten vor Ort bzw. ortsbezogen gespeichert werden sollen.

Der Zugriff auf die Sensordaten erfolgt mithilfe eines Tuple Space Interfaces und ermöglicht somit einen quasi geteilten Speicher zwischen Applikation und Sensorik. [3]

TeenyLIME

Ein anderer Ansatz ist es die Steuerlogik im Netzwerk auf die Komponenten zu verteilen. Solang die entsprechend komplexere Koordination mehrerer Prozesse noch möglich ist können Ressourcen und teilweise systemkritische Latenz eingespart werden.

Die **TeenyLIME** Middleware adressiert diese Herausforderungen mit einer quasi Peer-to-Peer Version des Space-Based Computing Paradigmas. Spaces befinden sich bereits auf den einzelnen Komponenten und können nur von benachbarten (one-hop) Nachbarn verwendet werden. [7]

TRITon

TRITon ist ein Forschungsprojekt mit dem Ziel, mithilfe eines WSNs eine effiziente und sichere Tunnelbeleuchtung zu realisieren.

„TRITon is a research and innovation project funded by the project members and the Autonomous Province of Trento (Provincia Autonoma di Trento, PAT) aimed at advancing the state of the art in the management of road tunnels, specifically to improve safety and reduce energy costs." [5]

Herkömmliche „intelligente" Tunnelbeleuchtungen versuchen meist mithilfe bestehender, gesammelter Daten die Lichtintensität je nach Uhrzeit und Datum anzupassen, ohne dabei die tatsächlichen Wetterbedingungen und andere Einflüsse zu beachten. Da sich bestimmte Umwelteinflüsse oft nicht vorhersehen lassen, führt dies oft dazu, dass der Tunnel entweder zu hell oder zu dunkel beleuchtet ist. Dies führt nicht nur zu ineffizientem Energieverbrauch, sondern weiters auch zu einem Sicherheitsrisiko für Verkehrsteilnehmer.

TRITon hingegen arbeitet mit einem auf **TeenyLIME** basierendem WSN, mithilfe dessen die Beleuchtung des Tunnels in Echtzeit an die Umwelteinflüsse angepasst wird. [5]

GigaSpaces

GigaSpaces ist ein Unternehmen, das Lösungen für verteilte Systeme und XTP (Extreme Transaction Processing) entwickelt. Zu ihren Kunden gehört mitunter **HP, IBM, Bloomberg, Goldman Sachs** und zig andere wichtige Unternehmen. [6]

XAP

GigaSpaces **XAP** ist dabei ihr wichtigstes und meist verbreitetes Produkt. XAP steht für „eXtreme Application Platform“ und ist eine Space-based Architecture in der Programmiersprache Java. Ursprünglich basierte XAP auf Konzepten von **Jini** bzw. **JavaSpaces**, heutzutage unterscheidet sich XAP allerdings enorm und bringt etliche Vorteile mit sich. [2]

Skalierbarkeit

Stufenbasierte Architekturen (tier-based)

GigaSpaces vertritt nach außen die Meinung, dass sich stufenbasierte Architekturen nicht vernünftig skalieren lassen. Tatsache ist, dass bei stufenbasierten Architekturen die Architektur selbst der Bottleneck ist und nicht die einzelnen Komponenten. Das liegt daran, dass in einer stufenbasierten Architektur, sobald Module zur Verbesserung der Skalierbarkeit verwendet werden, die Kommunikation zwischen den einzelnen Stufen deutlich komplizierter und damit teurer wird. Dies verlängert weiter die Responsetime und kann zu Problemen bzgl. Datenkonsistenz führen (fig. 2). Weitere typische Probleme bei stufenbasierten Architekturen sind Bottlenecks beim Zugriff auf die Datenbank, Message-Overhead, Latenz des Netzwerks, ineffizientes clustern und unzuverlässige Fehlerbehebung. [9]

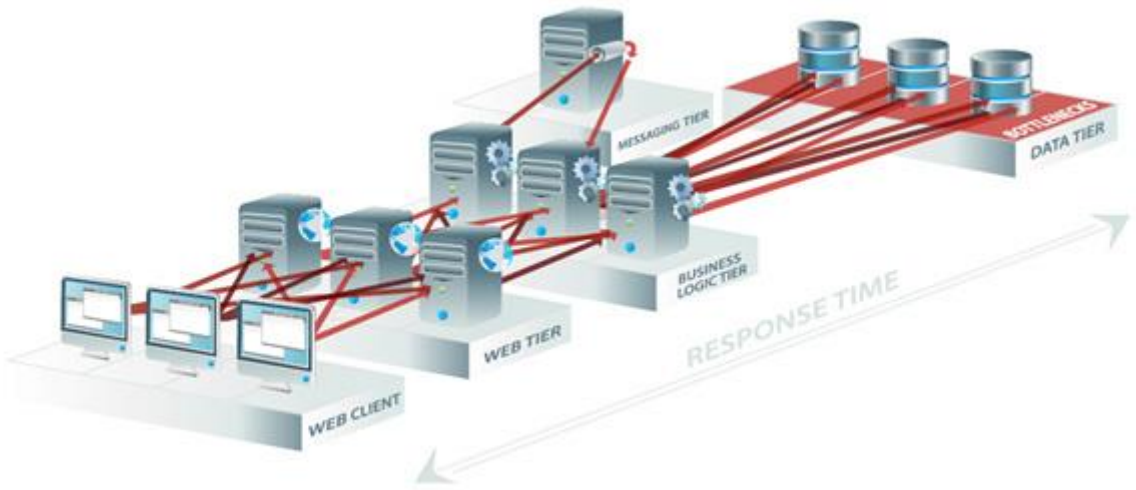


Figure 2: [2]

GigaSpaces behandelt jedes einzelne dieser Probleme mithilfe einer Plattform, die das Arbeiten mit einer Space-Based Architecture ermöglicht. [9]

Space-Based Architecture

GigaSpaces XAP bearbeitet Daten komplett im Arbeitsspeicher (In-Memory Computing) und verwendet die Datenbank ausschließlich zur Archivierung. Anstelle einer tier-based Architecture wird eine Space-based Architecture verwendet. Hiermit lässt sich (laut GigaSpaces) eine fast lineare Skalierbarkeit erreichen. Man kann sich also vorstellen, dass für doppelt soviel Zugriffe auch lediglich doppelt soviel Rechenleistung bzw. Hardwarekomponenten und somit Kosten anfallen. Desweiteren verlängert sich die Responsetime des Systems nur minimal im Vergleich zu tier-based Architectures (fig. 3). [9]

Figure 3: [2]

Razorfish iPhone Activation

Obwohl der iPhone-launch in der USA in 2007 an sich erfolgreich verlief, gab es anfangs, aufgrund der unerwartet großen Menge an Kunden, einige Probleme mit der Aktivierung der iPhones. Innerhalb der ersten 30 Stunden nach Markteinführung des Smartphones war die iPhone Activation Applikation ganze 3 Stunden nicht erreichbar. Da sich iPhones erst nach der Aktivierung verwenden lassen, hat dies etliche Kunden verärgert und es weiters auch in die Medien geschafft.

Da Apple nach diesem Missgeschick keine Risiken für den UK-Launch eingehen wollte, beauftragen sie Razorfish eine iPhone Activation Applikation zu entwickeln. Diese Applikation hatte folgende Anforderungen:

12000 iPhone Aktivierungen pro Stunde

Es müssen bis zu 12000 Aktivierungen pro Stunde möglich sein, damit die Applikation auch mit etwaigen Spikes klarkommt, die zu bestimmten Einkaufszeiten, allerdings auch bei zeitlich begrenzten Sonderangeboten anfallen können.

Jede Aktivierung besteht aus mehreren Transaktionen. Dabei wird auf Daten aus mehreren Quellen und auf Systeme von zwei verschiedenen Unternehmen zugegriffen.

Ständige Verfügbarkeit

Die Aktivierung muss **jederzeit, ohne Unterbrechungen**, möglich sein.

„If you can't activate iPhones, you can't sell them." [12]

Unterbrechungen wirken sich erheblich auf Apple's Image auswirken muss die Lösung über eine sehr hohe Ausfallsicherheit verfügen. Das heißt, dass die Applikation sowohl Software-, Hardware- als auch Netzwerkbezogen absolut fehlertolerant sein muss.

Datenschutz

Kundendaten, wie Kreditkartennummern, Adressen, Accountinformationen und andere sensible Daten, müssen mit Vorsicht behandelt werden.

Integration und Einbindung in andere Systeme

Razorfish musste weiters die Kompatibilität bzw. Interaktion mit anderen Systemen wie zum Beispiel der iTunes Plattform und dem Apple CRM (Customer Relationship Managment) ermöglichen. Dazu

gehören weiters auch separate Systeme zur Authentifizierung, Autorisierung, Zahlung und viele mehr. Da es sich hier um teils heterogene Datenquellen und Systeme handelt, stellt dies das zu entwickelnde System deutlich auf die Probe.

Zeit

Man würde meinen, die oben angeführten Anforderungen sind bereits Herausforderung genug. Ein weiterer Punkt ist allerdings, dass Razorfish lediglich 12 Wochen Zeit hatte, um das gesamte System zu entwickeln und zu deployn.

Lösung

Razorfish überlegte anfangs diese Problemstellung mithilfe eines Java Application Servers (JEE) zu lösen, allerdings entschieden sie sich letztendlich dagegen. Der Hauptgrund dafür war der, dass sich so eine Ausfallsicherheit in dem hier geforderten Ausmaße nur sehr schwer realisieren lässt.

Ein wesentlicher Vorteil elastischer Applikationsplattformen (z.b. Space-based) ist, dass sich sowohl Daten als auch Arbeit bzw. Last auf zwei oder mehrere Nodes verteilen lassen. Fällt eine der beiden Nodes aus, wird die Arbeit schlichtweg von einer der anderen Nodes durchgeführt. Eine ähnliche Ausfallsicherheit lässt sich natürlich auch mithilfe von Clustern typischer Java Application Servers realisieren, allerdings steigt dadurch die Komplexität des Source-Codes und die Serverkosten enorm an.

XAP macht dies komplett transparent. Der Source-Code muss nicht extra verändert werden um eine höhere Skalierbarkeit zu gewährleisten. Für den Programmierer ist es also quasi irrelevant ob die Applikation nun auf einem einzigen Server oder auf mehreren Servern betrieben wird.

Razorfish entschied sich daher aus folgenden Gründen für **GigaSpaces XAP**:

- **Tight integration with Java**: Razorfish ist bekannt für Java und Spring Development. Aufgrund der Zeitknappheit wurden Systeme, die in Java/Spring entwickelt werden können, bevorzugt.
- **Performance and Scalability**: GigaSpaces XAP in-memory Architektur ermöglicht eine fast lineare Skalierbarkeit
- **High availability**: Razorfish deployte GigaSpaces XAP auf mehreren redundanten Servern. Wenn bei einem der Server ein Fehler auftritt, verteilt XAP die Aufgaben automatisch und ohne Unterbrechung an einen der anderen (Backup-)Server.
- **Security baked into the application**: GigaSpaces XAP lässt sich sehr einfach mit bestehenden Authentifizierungs- und Autorisierungsmodellen bzw. Systemen wie zum Beispiel LDAP und ähnlichem kombinieren.

[12]

Vergleich

In diesem Vergleich werden wichtige SBC-Implementationen anhand einiger Features verglichen. Dieser Vergleich erweitert lediglich die Vergleichs-Tabelle der Prequel (siehe Abstract). Transaktionen werden von allen relevanten Implementierungen unterstützt und werden deshalb hier nicht erwähnt.

Features	**Replication**	**Persistency**	**PLMobility**	**WSN**	**Language**
MozartSpaces	X	X*			Java
XcoSpaces	X	X			C#/.NET
TinySpaces	X	X	+*	*	.NET (emb. Sys.)
LIME	X	*	+	*	Java
TinyLIME	X	*	+	Multiple stations	Java
TeenyLIME	X	*	+	Completely decentralized	C
Corso	X	X*			Java
TSpaces		X			Java
XAP	X	X			Java (Spring)

PLMobility: *Diese Spalte beschreibt ob die jeweilige SBC-Implementierung für logische oder physikalische Mobilität der einzelnen Nodes empfehlenswert ist.*
WSN: *Diese Spalte beschreibt ob die jeweilige SBC-Implementierung für die Entwicklung von Wireless Sensor Networks geeignet ist und beschreibt das Feature.*

Legend:
(X): *Supported*
(+): *Recommended*
(?): *Keine Quelle*

Details/Sternchen **(*)**:
XVSM *(MozartSpaces, XcoSpaces, TinySpaces): Persistierung und Replikation auch via Plugable Extensions möglich.*

***MozartSpaces:** Die Persistierung von Containern und deren Einträgen wurde in Version 2.2 hinzugefügt. MozartSpaces verwendet dabei die Berkeley DB (BDB) Java Edition library um die Daten als key-value-Paare abzuspeichern. Es ist nicht erforderlich die BDB zu installieren. [18]

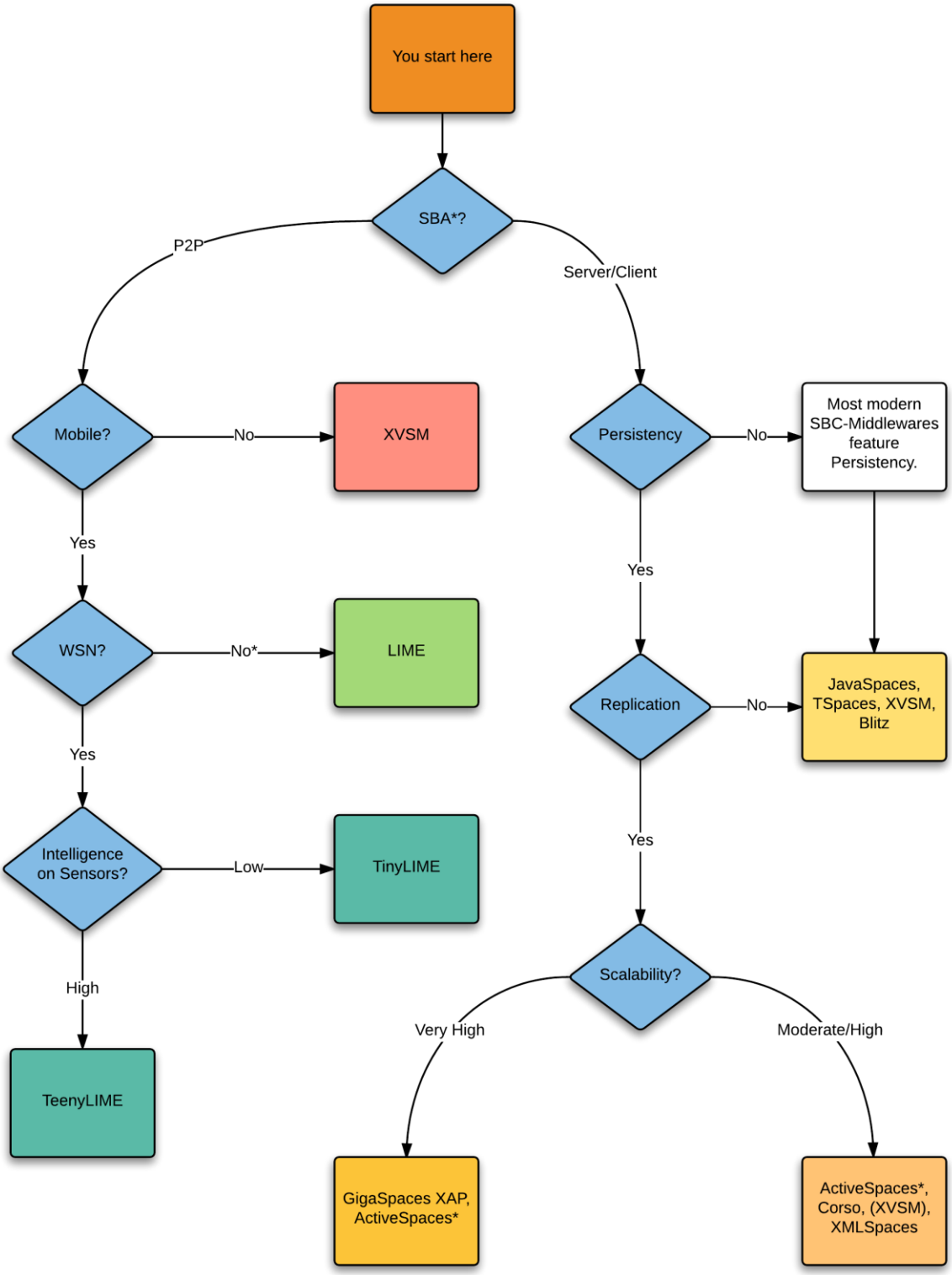
***TinySpaces:** Kann ähnlich wie TeenyLIME verwendet werden, ist allerdings mit mehr Aufwand verbunden.

***LIME:** LIME teilt alle Daten auf alle Teilnehmer auf (Replikation). Da es sich hierbei um generativen Space handelt, kann dies theoretisch zur Persistierung ausreichen. Eine eigentliche Persistierung findet allerdings nicht statt, ist aber über Extensions möglich. TinyLIME und TeenyLIME sind Extensions von LIME, die beide auf die Verwendung mit Wireless Sensor Networks ausgelegt sind.

***Corso:** Die Persistierung erfolgt intern mithilfe einer Datenbank.

Cheat-Sheet

Dieser Flowchart soll die Entscheidung, welche Implementierung verwendet werden soll, erleichtern.



***SBA** = Space-based Architecture: „Wird SBC lediglich für die Serverarchitektur verwendet (anstelle einer tier-based Architecture) oder für die gesamte Kommunikation (vergleichbar mit P2P)?“*
***LIME:** TinyLIME und TeenyLIME sind Extensions von LIME.*
***ActiveSpaces:** kommt in verschiedenen Versionen. Sollte nach Anforderungen gewählt werden (Enterprise Version wird kommerziell vertrieben)*

References

[2] *XAP Solution*
http://wiki.gigaspaces.com/xap/solution

[3] *„LIME: Linda in a Mobile Environment“- Homepage*
http://lime.sourceforge.net/

[4]: *XVSM/SBC*
http://www.complang.tuwien.ac.at/eva/SBC-Group/sbcGroupIndex.html

[5] *„TRITon: Trentino Research & Innovation for Tunnel Monitoring”- Homepage*
http://triton.disi.unitn.it/

[6] *GigaSpaces Customers*
http://www.gigaspaces.com/customers

[7] *“TeenyLIME: Transiently Shared Tuple Space Middleware for Wireless Sensor Networks”*
https://www.sics.se/~luca/papers/costa06teenylime.pdf
Autor: Paolo Costa, Luca Mottola, Amy L. Murphy, Gian Pietro Picco
Version: 2006A

[9] *„Space-Based Architecture by GigaSpaces“*
https://web.archive.org/web/20100703053432/http://javapulse.net/2008/04/22/spaced-based-architecture-by-gigaspaces/

[12] *„Razorfish iPhone Launch”*
http://d3a0pn6rx5g9yg.cloudfront.net/sites/default/files/private/free/resource/iphone_launch_razorfish_Forrester.pdf
Autor: Noel Yuhanna and Mike Gualtieri
Version: March 19, 2010

[18] *„MozartSpaces Tutorial“*
http://www.mozartspaces.org/2.3-SNAPSHOT/docs/MozartSpaces_Tutorial.pdf
Autor: Michael Wittmann, Bernhard Efler, Tobias Dönz & Martin Planer
Version: 21.12.2012