

## Projektgruppe BSCWeasel SS05

Marco Lütticke ([luetticke@freenet.de](mailto:luetticke@freenet.de))  
Steffen Schücke ([steffen@schuecke.com](mailto:steffen@schuecke.com))

### Zielsetzung und Projektentwicklung

Zu Beginn des Projekts bestand die Idee, eine Offline-Funktionalität für das BSCWeasel zu entwickeln. Bei Abbruch der Verbindung oder bewusstem Arbeiten ohne Netzwerkanbindung sollte das Umschalten auf einen „lokalen Server“ in Form einer Datenbank möglich sein, der den letzten Inhalt des BSCW-Servers enthält. Sobald wieder eine Verbindung zum Server besteht, hätte man alle Änderungen auf dem Server in die lokale Datenbank einfügen können, ebenso wäre ein Abgleich neuer Dokumente oder Metadaten auf dem Server sinnvoll gewesen.

Als ersten Ansatz zur Synchronisation haben wir SyncML gewählt, da der BSCW-Server mit XML-Schnittstellen arbeiten kann. Dabei ergaben sich einige Probleme, das Konzept musste geändert werden. Leider haben wir kein Java-Plugin für SyncML gefunden, die verfügbaren Lösungen setzen alle auf C++ auf oder sind kostenpflichtig.

Dazu kommt ein konzeptionelles Problem: SyncML kennt verschiedene Synchronisations-Modi. Da das Protokoll für Geräte mit starker Speicherbeschränkung und langsamer Netzwerkanbindung entwickelt wurde, ist nur eine komplette Synchronisation der lokalen Datenbank mit einem Server möglich. Verwendet wird diese Technik häufig bei Smartphones, PDAs und ähnlichen Geräten. Meist gleicht man eine Adressdatenbank oder einen Kalender ab, dabei werden aber nur geringe Datenmengen übertragen.

Die verfügbaren Modi sind sehr einfach gehalten: Der Client kann die komplette Datenbank des Servers herunterladen bzw. die komplette Datenbank des Servers durch die lokale ersetzen. Es gibt dabei eine Unterscheidung welche Seite den Transfer initiiert, dabei ergeben sich unterschiedliche Ergebnisse, diese sind hier nicht interessant.

Eine Synchronisation einzelner Objekte ist leider nicht möglich – damit ist es für unsere Zwecke unbrauchbar. Die zu übertragende Menge an Daten wäre viel zu groß, dazu würden vorgenommene Änderungen anderer User nicht berücksichtigt.

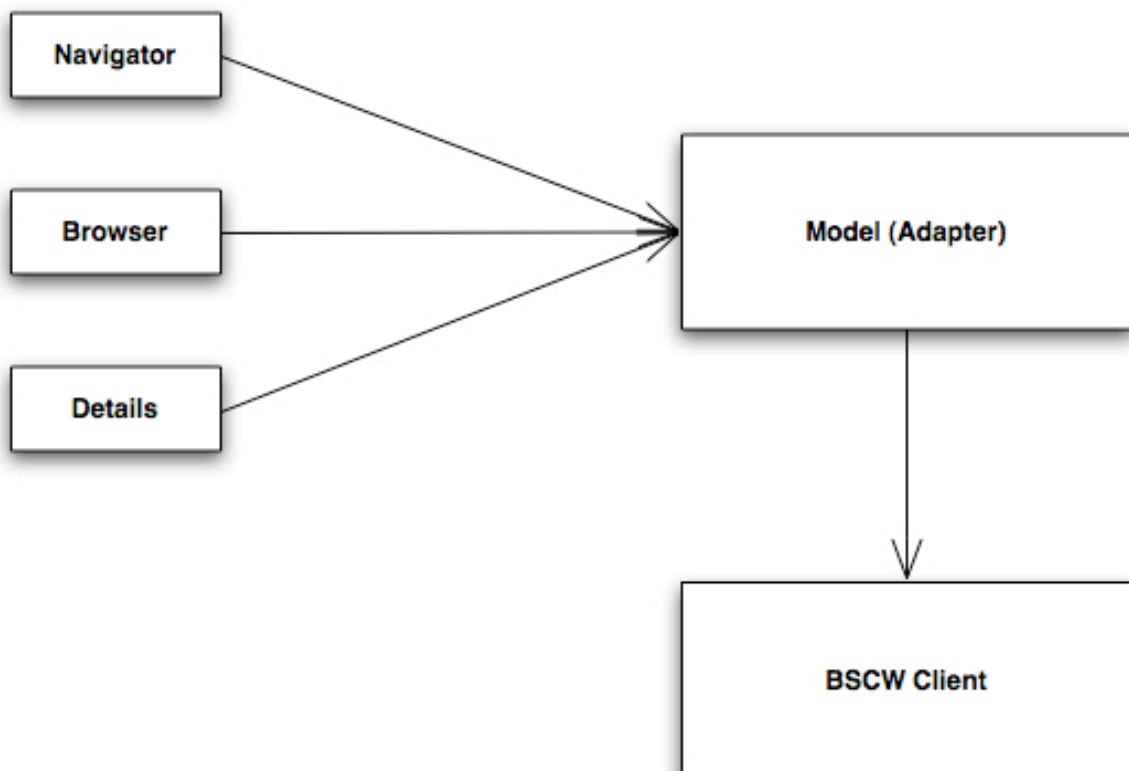
Die Zielsetzung wurde danach leicht modifiziert: Der komplette Offline-Betrieb wurde zurückgestellt, wir begannen eine Caching-Lösung zu entwickeln.

Da hier die Behandlung der Nutzdaten erstmal keine Rolle spielt, haben wir uns auf die Verarbeitung der Metadaten beschränkt. Eine Datenbank zur Speicherung bereits vom Server geholter Metadaten haben wir beibehalten und um einen im Speicher befindlichen Cache erweitert.

## Aufbau des Systems ohne/mit Caching

Bisher fand im System kein explizites Caching statt. Bei Bedarf wird eine Verbindung zum Server aufgebaut, die Daten werden in einer Hashtable gespeichert. Alle Zugriffe auf den Server finden über die Client-Klasse statt. Die Grafik zeigt den Aufbau, bevor Änderungen vorgenommen wurden.

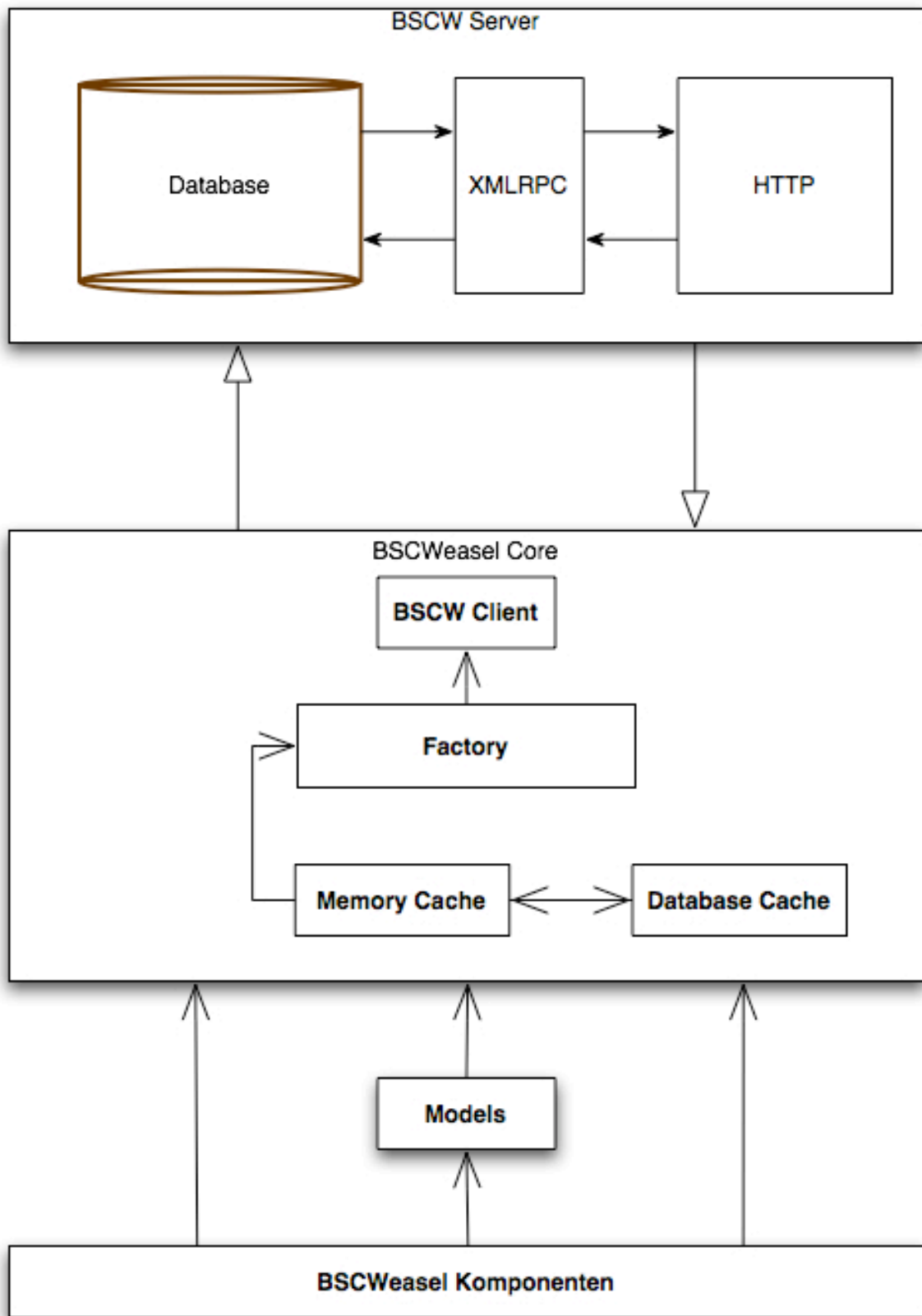
### Aktuell: System ohne Caching



Um einen optimalen Punkt zu haben, an dem man mit einer Caching-Strategie ansetzen kann, haben wir eine „Factory“ eingeführt, die alle Anfragen verarbeitet, die momentan an den Client gehen. Die Factory soll als zentrale Schnittstelle BSCW-Objekte erzeugen und notwendige Server-Zugriffe weiterleiten. Hinter die Factory haben wir zwei neue Klassen geschaltet, die das Caching realisieren.

Jetzt bekommt man die Hashtable entweder aus dem Cache (sofern vorhanden), oder durch eine Serveranfrage.

Die Grafik auf der nächsten Seite zeigt den veränderten Aufbau des Systems und die angebundenen Caches.



## Funktionsweise der Caches

Bisher sah der Zugriff z.B. auf einen Ordner aus wie folgt:

Der User klickt einen Ordner an (d.h. ein Objekt wird angefragt), sofort ist ein Server-Zugriff (über den Client) nötig. Die benötigten Daten werden geholt und verarbeitet. Erfolgt später ein erneuter Zugriff auf den Ordner, wiederholt sich der Ablauf.

Nach dem Einbau der Factory und den notwendigen Änderungen ist die Verarbeitung wesentlich komplexer geworden:

### 1. Stufe: Memory Cache

Wird jetzt ein Objekt benötigt, prüft die Factory ob das Objekt bereits im Speicher (d.h. im Memory Cache) liegt. Dazu verwaltet der Memory Cache zwei Hashtables:

In einer Hashtable stehen die Attribute zu einem Objekt (genauer: Die Table enthält die Hashtable, die vom Server zurückgegeben wird), in der zweiten werden die Kindknoten (z.B. Unterordner eines Folders) in Form eines Vectors gespeichert.

Die Identifizierung (d.h. der Schlüssel) erfolgt dabei über die BSCW-ID des Objekts.

Werden die benötigten Daten nicht im Speicher gefunden, können sie auch nicht in der Datenbank stehen (dazu später mehr). Hier ist wie bisher ein Zugriff auf den Server notwendig, um die benötigten Daten abzurufen. Die Zugriffe auf den Server laufen ab wie bisher, die Factory leitet hier nur weiter.

Die abgerufenen Daten werden nun in die beiden Hashtables geschrieben. Findet später ein erneuter Zugriff auf das Objekt statt, befinden sich die Daten bereits im Speicher. Damit erspart man sich einen erneuten Zugriff auf den Server.

Zusätzlich werden alle vom Server geholten Daten auch in den Database Cache geschrieben.

Beim Testen hat sich herausgestellt, dass das Arbeiten eindeutig flüssiger abläuft.

Besonders bei langsamer Verbindung zum Server (d.h. alle Verbindungen, die sich nicht im gleichen Netzwerk wie der Server befinden) ist die „gefühlte Geschwindigkeit“ des Systems merklich verbessert. Allerdings scheint der Start des Systems durch die neuen Komponenten (und das Einlesen der Datenbank) noch langsamer zu sein wie bisher.

### 2. Stufe: Database Cache

Hinter den Memory Cache ist ein weiterer Cache geschaltet, der Datenbank Cache.

Die Datenbank hat 2 Funktionen:

1. Alle Daten die man im Memory Cache speichert, werden auch in den Database Cache geschrieben. Wird das Programm beendet, sind die abgerufenen Daten persistent vorhanden.  
Beim erneuten Start des Systems (und Herstellen der Verbindung zum Server) wird die zugehörige Datenbank ausgelesen. Der Memory Cache wird mit den gespeicherten Daten gefüllt, so dass beim Zugriff auf ein Objekt keine Serveranfrage notwendig ist, falls sich das benötigte Objekt bereits im Speicher befindet.  
Neben dem Einlesen der Datenbank beim Start des Programms finden keine weiteren Lese-Zugriffe auf die Datenbank statt. Schreibzugriffe finden immer dann statt, wenn auch in den Memory Cache geschrieben wird. Damit kann die Datenbank nie einen neueren Stand wie der Memory Cache enthalten.

2. Arbeitet man mit mehreren Usern/Servern, werden alle abgerufenen Daten in die Datenbank geschrieben, bevor ein Wechsel der Verbindung erfolgt. Der Name der erzeugten Datenbanken ergibt sich aus der BSCW Server-Location (zusammengesetzt aus Username und Server). Bei erneutem Wechsel der Verbindung kann der vorherige Zustand des Memory Cache aus dem Database Cache wiederhergestellt werden.

### Datenbankschema

Jede erstellte Datenbank hat das gleiche Schema und ist sehr simpel aufgebaut. Es existieren zwei Tabellen, „attributeCache“ und „children“. Es gibt keinen Primärschlüssel, ebenso existieren keinerlei Relationen zwischen den Tabellen.



### Aktualisierungen/Synchronisation mit dem Server

Eine automatische Aktualisierung bzw. Synchronisation der Caches mit dem Server ist noch nicht implementiert (Siehe „Offene Baustellen“). Wir haben allerdings eine Möglichkeit geschaffen, bereits gespeicherte Objekte aus dem Cache zu entfernen, um eine Aktualisierung zu erreichen.

Klickt man im Navigator ein Objekt an (z.B. einen Folder) und wählt aus dem Kontextmenü (rechte Maustaste) den Punkt „Aktualisiere“, werden aus beiden Caches sämtliche Attribute sowie alle Kindknoten (Children) gelöscht. Die Attribute und Kindknoten zu diesem Objekt werden neu vom Server geholt (anhand der ID), damit erhält man den aktuellsten Stand.

Eine Einschränkung gibt es: Es werden nur die Attribute/Kindknoten für das aktuelle Objekt neu geladen – nicht aber die Attribute/Kindknoten zu allen untergeordneten Elementen (z.B. Unterordner). Diese werden erst in dem Moment neu vom Server geladen, wenn der User sie anklickt. Ein vollständiges Löschen des Caches kann durch Aktualisieren des Arbeitsbereichs erreicht werden.

Den Menüpunkt „Aktualisiere“ haben wir verwendet, weil es sich anbietet die vorhandene Lösung zu erweitern.

Die Lösung war simpel zu realisieren, wir mussten keine Änderungen/Erweiterungen an der GUI vornehmen. Allerdings ist man auf das Kontextmenü angewiesen, was z.B. für Mac-User (Eintasten-Maus bzw. Touchpad) kein optimales Arbeiten erlaubt.

Man könnte die Menüleiste um einen eigenen Button zur Aktualisierung erweitern, um dieses Problem eleganter zu lösen.

In der Detailansicht ist ein neues Info-Feld hinzugekommen: „in den Cache geschrieben“ zeigt an, wann ein Objekt in die beiden Caches geschrieben wurde. Da die Aktualisierung momentan manuell erfolgt, hat man einen guten Anhaltspunkt, ob eine Aktualisierung sinnvoll sein könnte.

## Die verwendete Datenbank: HSQLDB

Um eine persistente Speicherung der Daten zu erreichen, standen verschiedene Konzepte zur Wahl. Da wir den Sicherheitsaspekt (Verschlüsselung usw.) völlig außer Acht gelassen haben, wäre auch eine Speicherung der Daten in einer normalen Text-Datei oder einem anderen Format möglich gewesen. Wenn neben der Speicherung der Metadaten auch eine Sicherung von Nutzdaten eines Servers (sozusagen ein „Content Cache“) erfolgen würde, wäre eine normale Datenbank wohl ungeeignet. Hier müsste wohl eine Möglichkeit zur Speicherung der Daten im normalen Dateisystem gefunden werden.

Da wir uns hier auf die Metadaten beschränkt haben, war eine SQL-Datenbank erste Wahl. Gesucht war ein frei verfügbares und plattformunabhängiges Java-Plugin.

Allerdings wollten wir den Betrieb eines ausgewachsenen SQL-Servers auf dem System vermeiden – hier wäre der Aufwand zu groß.

Zudem wären Sicherheitsbedenken (Absicherung des Servers und drohende Korrumpierung des Systems), Probleme mit Zugriffsrechten (z.B. Arbeit unter Windows ohne Administrator-Rechte) und erhöhte Anforderungen (Speicherverbrauch des Servers) inakzeptable Folgen gewesen, die es zu vermeiden galt.

Momentan verwenden wir HSQLDB in der Version 1.8.0 ([www.hsqldb.org](http://www.hsqldb.org)).

HSQLDB ist eine relationale Datenbank und unterstützt nahezu alle SQL-Befehle. Es sind (laut Entwickler) Datenbanken bis zu mehreren hundert MB Größe möglich.

Die Datenbank kann in verschiedenen Modi betrieben werden („only in Memory“, Servlet-Mode usw.), hierzu sei auf die Dokumentation auf der HSQLDB-Website verwiesen.

Die von uns verwendete Variante hält alle Daten zur Laufzeit im Speicher und schreibt alle 20 Sekunden mögliche Änderungen in die Datenbank. Dies ist ein Standardfeature der Datenbank und kann beliebig geändert werden.

HSQLDB bietet in der neuen Version einen recht guten Schutz bei Abstürzen und brauchbare Recovery-Funktionen. Da die Datenbanken ja keine wichtigen Daten enthalten, haben wir keine Stabilitäts-Tests durchgeführt. Ebenso haben wir die keinerlei Optimierungen der Servereinstellungen vorgenommen. Durch Anpassungen auf bestimmte Bedingungen wäre evtl. ein schnellerer Start bzw. eine bessere Geschwindigkeit zu erreichen.

Getestet haben wir die neuen Funktionen unter Windows 2000, Windows XP, Mac OS X 10.3.x, Mac OS 10.4.x, sowie Linux. Damit haben wir die wichtigsten Systeme abgedeckt, auf anderen Systemen sollte es aber ebenfalls keine Probleme geben.

Um einen fehlerfreien Betrieb der Datenbank zu gewährleisten, empfehlen wir vom localhost (127.0.0.1) alle Verbindungen zum lokalen Port 1701 zu erlauben.

## Offene Baustellen und Ansatzpunkte

Erste Baustelle ist sicherlich die Synchronisation des Caches mit dem Server. Mit der aktuellen Lösung kann man gut arbeiten, trotzdem wäre eine automatische Synchronisation ein sehr komfortables Feature. Vielleicht kann man an der Arbeit der Awareness-Gruppe ansetzen und die neuen Funktionen zusammenführen. Dieser Schritt war uns aus Zeitgründen (und aufgrund der Komplexität) nicht mehr möglich.

Als nächstes Projekt wäre eine Erweiterung des Cachings denkbar, um auch Nutzdaten (geschriebene Beiträge, Dokumente) zu erfassen. Dazu ist die aktuelle Lösung nicht mehr ausreichend, da man sich um eine (plattformübergreifend funktionierende) Speicherung der Nutzdaten kümmern müsste.

Eine echte Offline-Funktionalität ist immer noch ein lohnendes Ziel, welches man nicht aus den Augen verlieren sollte. Hier kann man sicher an der aktuellen Caching-Lösung ansetzen, allerdings werden viele Änderungen in anderen Bereichen notwendig sein, um völlig ohne Serverzugriffe auszukommen. Auch hier dürfte eine spätere Synchronisation mit dem Server eines der Hauptprobleme darstellen.