Ein weiteres Beispiel ist das erste GUI (graphical user interface) auf dem Xerox Star. Hier wurde der Begriff Desktop eingeführt und somit der Vergleich zu einem realen Schreibtisch gezogen. Es wird sozusagen der real existierende Schreibtisch mit der Kommando- Ebene eines Computers in einem Mashup vereint, sodass eine virtuelle Arbeitsfläche (Desktop) entsteht. Auch hier steht wieder die Funktionalität im Vordergrund. Man nimmt zwei oder mehrere schon Vorhandene Dinge und macht was neues daraus, um eine Funktionalität zu erschaffen, die es so bislang noch nicht gibt. In diesem Fall eben die Funktionalität auf dem Computer bequem und auch ohne spezielle Fachkenntnisse arbeiten zu können bzw., speziell auf den Desktop bezogen Dinge ablegen und verwalten zu können, wie man es auch auf einem realen Schreibtisch mit Akten tun würde.

Eine der wichtigsten Grundvoraussetzung für die Umsetzung von Mashups im Web 2.0 ist wohl die sogenannte API, was nichts anderes bedeutet als application programming interface. Eine API ist ein Programmteil, der von einem Softwaresystem anderen Programmen zur Anbindung an das System zur Verfügung gestellt wird. Somit ist es möglich auch entfernten Webservices eine Anbindung an Daten wie z.B. News zu ermöglichen. Das ist eine unabdingliche Notwendigkeit, um Mashups vernünftig erstellen zu können, und insbesondere, um diese stets auf aktuellem Stand zu halten. Im weitesten Sinne kann jede Schnittstelle einer Bibliothek (wie beispielsweise jquery) als API bezeichnet werden.
Um das anschaulich an einem Beispiel zu zeigen, haben wir ein Mashup ausgesucht, das sich "Open City Agora" nennt. Wie in Abbildung x.y zu sehen bedient sich das Mashup auf jeden Fall der Google Maps API und bindet Kartenmaterial ein. Es bedient sich jedoch noch diversen anderen API's um die Maps- Daten mit weiteren Daten in einer Grafik zu vereinen. So kann man sich zum Beispiel Basketball- Plätze, Bäume oder Stau-Daten in der Karte anzeigen lassen. Welche zusätzlichen Daten man in der Karte haben möchte, kann man aus einer vordefinierten Sammlung in der linken Sidebar oben auswählen, die ausgewählten Elemente werden einem dann zu Kontrolle in der Sidebar unten nochmal angezeigt.
Momentan gibt es auf Open City Agora nur Daten für New York und San Francisco, aber wer weiß, vielleicht folgen bald andere. Wer das ganze gerne mal interaktiv testen möchte, kann das Mashup unter http://opencityagora.appspot.com im Web finden.

Methoden: Mashups im Web lassen sich im Wesentlichen in 2 Kategorien gliedern: Server-Side Mashups und Client-Side Mashup. Im Falle eines Server-Side Mashups hat der Client selbst nur eine Verbindung zum Webserver, der das Mashup bereit stellt. Der eigentliche Mashup wird dann vom Server produziert. Das heißt, der Server holt sich von den verschiedenen Quellen / Content-Providern die Informationen, die er für den Mashup braucht. Das können News, Karten, Zahlen oder andere Daten sein. Die Informationen werden nachher auf dem Server weiterverarbeitet und werden letztendlich als fertiges Endprodukt bzw als fertiger Mashup an den Konsumenten weitergereicht. Folgende Grafik soll diesen Vorgang nochmal veranschaulichen:

Etwas anders verhält sich das ganze bei einem Client-Side Mashup. Hier erhält der Browser, also der Client, vom Webserver, der den Mashup bereit stellt, lediglich das Grundgerüst der Seite sowie etwas Javascript. Im Javascript stehen nun Anweisungen von welchen Content-Providern sich der Browser die für den Mashup benötigten Inhalte holen soll. Diese werden dann mittels Ajax dynamisch nachgeladen und in das vom Webserver bereitgestellte und bereits vom Browser geladene Grundgerüst eingefügt. Was Ajax ist und was es tut wird im nächsten Abschnitt noch genauer erklärt, wichtig ist hier, dass bei einem Client-Side Mashup der Client selbst sich die

Inhalte vom entsprechenden Content-Provider holt und sie nicht zentral vom Webserver des Mashups bekommt. Auch hier wieder eine Grafik, die den Vorgang veranschaulichen soll:

Jetzt wird sich der ein oder andere fragen: "Warum ist das jetzt so wichtig ob jetzt der Webserver die Inhalte holt und an den Client weiterreicht oder ob der Client sich die Inhalte selbst holt? Hauptsache sie sind am Ende für den Konsumenten verfügbar." Ja das mag prinzipiell schon stimmen, aber wie alles im Leben hat auch hier jede Vorgehensweise seine Vor- und Nachteile.
So ist der Client-Side Mashup z.B. auf die Nutzung von Javascript beschränkt wohingegen der bei einem Server-Side Mashup ganz verschiedene Scriptsprachen für die weiterverarbeitung der Daten zum Einsatz kommen können. Denken wir beispielsweise an PHP, ASP oder auch Perl. Denkbar ist auf dem Server prinzipiell jegliche Datenweiterverarbeitung die auf einem Server automatisiert realisierbar ist. So ergeben sich bei einem Serer-Side Mashup deutlich bessere Möglichkeiten der Datenverarbeitung wie z.B. bessere Filtermöglichkeiten oder auch komplexere Datenmanipulationen. Hinzu kommt, dass in der Regel eine gute Fehlerbehandlung in Javascript deutlich schlechter realisierbar ist als bei der Datenverarbeitung auf einem Webserver, allein schon weil die Daten hier theoretisch auch von mehreren verschiedenen Scriptsprachen und / oder anderen Tools in Serie bearbeitet werden können. Ein weiterer Vorteil, der bei einem Server-Side Mashup genutzt werden kann, ist Caching. Auf einem Webserver bietet sich die Möglichkeit, Daten der verschiedenen Content-Provider von früheren Anfragen zu cachen, also zwischenzuspeichern. So ist es möglich, wenn ein Content-Provider aus technischen Gründen einmal nicht erreichbar sein sollte, auf die den Cache zurückzugreifen und die zwischengespeicherten Daten für den Mashup zu nutzen. Ergo bekommt der Endbenutzer seinen fertigen Mashup trotzdem, obwohl ein Content-Provider gerade nicht erreichbar ist. Bei einem Client-Side Mashup würden die Daten vom entsprechenden Content Provider schlichtweg fehlen.
Das hört sich jetzt langsam fast so an, als ob es überhaupt nur vorteile hätte den Mashup auf dem Server direkt zu bilden, dem ist natürlich nicht so. Einen gravierenden Nachteil bringt nämlich allein schon ein Großteil der bisher genannten Vorteile mit sich. Alle Daten müssen auf dem Server verarbeitet werden. Das kann natürlich speziell bei einem stark frequentierten Mashup schnell zu extremen Performance-Einbußen oder auch gravierenderen Problemen bis hin zum Stillstand führen, weil die CPU und der Arbeitsspeicher des Servers eben nur begrenzt vorhanden sind und nur eine gewisse Anzahl gleichzeitiger Anfragen bewältigen können. Auch die Anzahl der Festplattenzugriffe kann hierbei schnell zum Problem werden. Die häufigste Folge dieser Tatsachen sind meist gravierende Geschwindigkeitseinbußen. Würde sich der Client die Quellen selbst holen (Client-Side Mashup) würde die Datenverarbeitung auf dem Client stattfinden und wäre damit sozusagen ausgelagert. Insbesondere muss dann jeder Client nur seine eigenen Daten verarbeiten und eben nicht ein Server die Daten für alle. Ein weiteres Problem beim Server-Side Mashup, das ebenfalls bei entsprechender Frequentierung ins Gewicht fällt, ist die begrenzte Kapazität der Datenleitungen, die den Server ans Internet anbinden. Der Server stellt hier, wie man auch in der Grafik sehr schön sehen kann, eine Art Engpass dar, den alle Daten passieren müssen. Vergleichen wir hier wieder mit dem Client-Side Mashup stellen wir fest, dass hier dieser Engpass nicht besteht, weil sich jeder Client die für ihn benötigten Daten selbst direkt vom Content-Provider holt. Hier stellt sich dann eben je nach Anwendung die Frage, ob das Data Processing wirklich so komplex ist, dass es auf dem Server stattfinden muss oder ob man es nicht doch vielleicht mittels Javascript auf dem Client lösen kann. Weiter stellt sich die Frage, welche Daten wirklich über den Server laufen müssen aus dem Aspekt heraus, dass die individuellen Mashup Daten zumeist nur und zwar ausschließlich nur Relevanz für den Nutzer haben. Man sieht also hier klar, dass auch der Client-Side Mashup seine Vorteile aufzuweisen hat. So lassen sich hier beispielsweise auch Interaktionen mit dem Benutzer besser und vor allem sauberer bewältigen. Einen Nachteil muss ich allerdings noch erwähnen, um nochmal auf die Performance einzugehn. Es ist leider zumindest derzeit noch so, dass, je nach Browser, die Anzahl der gleichzeitigen XMLHttpReqests und somit die Anzahl der

parallel nachladbaren Inhalte, auf zwei bis drei beschränkt ist. Das heißt bei enem Mashup mit besonders vielen verschiedenen Quellen kann es auch hier durchaus zu Geschwindigkeitseinbußen kommen (genaueres zu XMLHttpRequests kann dem Abschnitt Ajax entnommen werden). Letztendlich sind heut die meisten Mashups ein gesunder Mix aus beidem. Wen wundert das bei den gerade beschriebenen Sachverhältnissen.

AJAX
Ajax hat weder etwas mit einem Putzmittel noch mit einem Fußballverein zu tun, sondern steht schlicht und einfach für Asynchrounus Javascript and XML. Ajax ist für Mashups aus zweierlei Gründen relevant. Zum einen allein schon aus dem Grund, weil Ajax selbst ein Mashup ist zum anderen deshalb, weil Ajax im Prinzip unerlässlich ist, wenn es um das Thema Client-Side Mashup geht. Ein Mashup ist Ajax, weil es sich verschiedener bereits vorhandener Features bedient, um einen neuen Zweck, einen funktionalen Zweck zu erfüllen. So bedient sich Ajax zwingend an Javascript und dem sogenannten XMLHttpRequest. Letzteres ist eine spezielle Form eines HttpReuests, über den es möglich ist, Inhalte dynamisch nachzuladen. Näheres hierzu gleich. Diese nachgeladenen Inhalte können nun in verschiedenen Formaten vorliegen, wie zum Beispiel XML. In diesem Fall könnte man Ajax dann als einen Mashup von Javascript XMLhttpRequest und XML bezeichnen, was auch allein schon der Name beinhaltet. An dieser Stelle sei allerdings erwähnt, dass es längst möglich ist, auch Inhalte in ganz anderen Formaten nachzuladen. Speziell zu erwähnen ist hier vielleicht JSON (Java Script Object Notation), da es neben XML mittlerweile sehr häufig im Zusammenhang mit XMLHttpRequests verwendet wird. Aber auch jeder andere Datentyp ist möglich, denkbar sind z.B. auch Daten aus einer MySQL-Datenbank, die dann allerdings serverseitig mit PHP, ASP oder anderen Boardmitteln ausgelesen und aufbereitet werden müssen. Kommen wir nun zum zweiten Punkt. Warum ist Ajax im Prinzip unverzichtbar, wenn es um Client-Side Mashups geht? Weil Ajax genau die Technik ist, der man sich normalerweise bedient, um Inhalte auf einer Webseite clientseitig dynamisch nachzuladen. Wie das ganze funktioniert bzw. was man sich darunter vorzustellen hat, zeigt folgende Grafik ganz gut.

<img src="./ajax-funktionsweise.jpg" alt="Grafik zur Veranschaulichung von Ajax">

Der blaue Zweig zeigt einen herkömmlichen HttpReqest. Das heißt, der Browser sendet eine Anfrage an den Server und bekommt entsprechen eine Antwort. Die Antwort beinhaltet die komplette HTML-Seite sowie zusätzlich benötigte Informationen wie CSS Informationen (Cascading Stlye Sheet, zumeist ausgelagert in seperater Datei), Scripte oder ähnliches. Beim ersten Seitenaufruf muss das natürlich zwingend so sein, denn zu Beginn ist auf der Client-Seite noch nichts vorhanden, ergo auch nichts, in das man etwas einfügen könnte. Ist das Grundgerüst allerdings erst einmal da, so lassen sich nun über einen sogenannten XMLHttpRequest Teilinhalte nachladen, es werden also erstens tatsächlich nur die Inhalte der Seite aktualisiert, die sich auch ändern, und zweitens auch nur die Daten übertragen die sich auch verändert haben, wohingegen bei einem herkömmlichen HttpRequest die komplette Seite neu angefordert werden oder sage wir korrekterweise zumindest auf Änderungen überprüft werden muss. Fakt ist, dass es bei herkömmlichen Anfrage in jedem Fall einen Full Page Refresh gibt und mehr Daten zwischen Server und Client übertragen werden müssen. Die folgenden zwei Grafiken veranschaulichen nochmal den Unterschied mit dem Schwerpunkt auf dem Datenverkehr:

<img src="./ajax-vs-normal-request.png" alt"Unterschied zwischen herrkömmlichen und Ajax Request">

Bei einem herkömmlichen HttpRequest wird bei jeder Interaktion des Benutzers (Klick auf einen Link) eine Anfrage an den Webserver gesendet, der dann entsprechend reagieren bzw. antworten muss. Der Client muss hier in jedem Fall nach jeder Anfrage warten bis er eine Antwort vom Server bekommen hat, bevor er weitere Aktionen ausführen kann. Hierbei könnte es sich zum Beispiel um

eine Weiterverarbeitung der Daten handeln oder auch um eine andere Interaktion mit dem Benutzer. Schaut man sich nun im Vergleich an, wie das ganze bei einem Ajax Request abläuft, so stellt man fest, dass der Browser hier selbst gar nicht direkt mit dem Server kommuniziert, sondern mit der Ajax Engine, in der Grafik auch als Ajax Communication Layer bezeichnet. Die Kommunikation mit dem Server übernimmt die Ajax Eninge asynchron, also im Hintergrund. Währenddessen können problemlos andere Aufgaben erledigt werden oder andere Interaktionen stattfinden. Trifft die Antwort vom Server bei der Ajax Engine ein, so wird sie über eine Callback Funktion an den Client weitergereicht und die Daten können dort nach Wunsch weiterbearbeitet und / oder an ensprechender Stelle eingefügt werden. So lassen sich wie, bereits erwähnt, anstandslos Teile einer Website aktualisieren ohne, dass dabei ein Full Page Refresh von Nöten wäre. Zu den bereits erläuterten Vorteilen kommt eine deutliche Reduzierung des Datenverkehrs, was an sich schon eine einiges an Zeit einspart, sowie Zusätzliche Zeitersparnis durch die asynchrone Arbeitsweise von Ajax.

Was hat das ganze mit Mashups zu tun? Nun ja Teilinhalte und Mashups gehören ja wohl zusammen wie Pech und Schwefel, ich möchte also eine Seite haben, die aus verschiedenen Teilinhalten von verschiedenen Quellen besteht. Da bietet sich Ajax gerade zu an, zumindest wenn der Mashup clientsetig erstellt werden soll, wäre da nicht das wörtchen "verschieden" vor den Quellen. Denn genau hier gibt es bei Ajax ein kleines Problem. Aus Sicherheitsgründen kann man mittels XMLHttpRequest also letztendlich auch mit einer Ajax Anfrage nur Teilinhalte nachladen, die der selben Domain zugeordnet sind, wie die, von der das Javascript geladen wurde. Zu deutsch: Es lassen sich nur Inhalte nachladen, die auf dem selben Webserver liegen wie die Mashup-Page. Das ganze fällt unter den Namen "same-origin-policy" bzw "same-origin-security-policy" und schützt uns zu einem gewissen Grad davor, dass im Hintergrund umbemerkt Schadsoftware auf unseren Rechner geladen wird bzw Code oder Daten von einem ganz anderen Webserver als dem, den wir bewusst aufgerufen haben. Für unser vorhaben, einen Mashup zu erstellen, ist das allerdings denkbar schlecht, weil wir so beispielsweise auch keine News von verschiedenen Quellen clientseitig in unseren Mashup einbinden können. Aber das Web wäre nicht das Web, wenn es hierfür nicht diverse Lösungsansätze gäbe, mit denen man das Problem lösen kann und einen gesunden Kompromiss zwischen Sicherheit und Funktionalität bildet. Da wäre zum einen das sogenannte JSONP (JSON with Padding) zu nennen, dass sich allerdings nicht wirklich des XMLHttprequests bedient sondern in die Seite ein "<script></script>"- Element injiziert, über das wiederum JSON-Daten nachgeladen werden können. Näher möchte ich das hier nicht erläutern, weil das dann doch den Rahmen sprengen würde. Der Nachteil hier ist, dass man für jede externe Quelle einen Neuen Script-Tag injizieren muss. Außerdem kann man hier nur mit Http-GET Anfragen arbeiten. Als zweite Möglichkeit gibt es da noch CORS, was in meinen Augen die deutlich bessere und auch sauberere Lösung ist. CORS steht für cross-origin resource sharing und ermöglicht somit, wie der Name schon sagt, ebenfalls das Nachladen von Inhalten aus Fremdquellen. Bei CORS funktioniert das ganze so, dass der Server dem Client über einen HTTP-Header Eintrag mitteilt, von welchen Quellen das Script Inhalte nachladen darf. Das Nachladen kann hier dann ganz normal über einen XMLHttpRequest erfolgen, sodass hier clientseitig keine weiteren Nachteile auftreten wie bei JSONP. Auch die Anfragemethoden sind hier nicht auf Http-GET beschränkt. Die Verantwortung der Sicherheit liegt hier auf Seiten des Servers, ergo des Providers. Denkbar ist auch eine Wildcard- Header- Angabe, die das nachladen von allen Quellen ermöglicht, was sicherheitstechnisch jedoch absolut unverantwortlich ist. Weitere Infos zu den hier kurz angeschnittenen Verfahren JSONP und CORS finden sich wie gewöhnlich im Internet.

Zum Abschluss des Themas sei vielleicht noch erwähnt, dass Ajax bereits jetzt schon äußerst relevantes Werkzeug im Web ist und künftig noch stark an Bedeutung gewinnen wird.

Yahoo Pipes

Ein weiteres Beispiel, dass die Mashups im Web 2.0 sehr gut verdeutlicht, ist das Mashup Tool "Yahoo Pipes", mit dessen Hilfe prinzipiell jeder selbst einen Mashup auf einfache Weiße erstellen kann. Voraussetzung ist allerdings die Anmeldung bei Yahoo. Einmal angemeldet gelangt man über http://pipes.yahoo.com/pipes/ mit einem Klick auf "Create Pipe" in das grafische Interface mit dessen Hilfe man sich seinen ganz eigenen Mashup erstellen kann. Hauptsächlich gedacht ist das Tool zum erstellen von Mashups aus RSS-Feeds. Das fertige Mashup kann ebenfalls bei bedarf wieder per RSS abonniert werden. Die Funktionsweise erklärt sich am Besten anhand des folgenden Screenshots.

<img src="Yahoo.png" alt="Inferface von Yahoo Pipes">

Zu sehen ist die Oberfläche mit der wir unsere Pipe erstellen die uns letztendlich unseren fertig Mashup ausgibt. Das Fenster ist in drei wesentliche Bereiche eingeteilt. Links befinden sich die Werkzeugleiste, die verschiedene Bausteine bereithält, gleich rechts daneben ist das Hauptfenster. Hier werden nachher die ausgewählten Bausteine eingefügt und miteinander verknüpft. Zum Abschluss gibt es unten ein Output- Fenster, das uns den entsprechenden Output des momentan im Hauptfenster ausgewählten Bausteins anzeigt. Das kann zum Beispiel auch der "PipeOutput"-Baustein sein, dann sind im Output-Fenster alle Daten zu sehen die am "PipeOutput" ankommen, d.h. alle Daten die letztendlich der Mashup beinhaltet.

Will man sich nun eine neue Pipe bauen und somit einen neuen Mashup erstellen, sucht man sich links aus der Werkzeugleiste die entsprechenden Bausteine aus. Zu finden Sind hier unter anderem "Sources"- Bausteine, die z.B. ein RSS- Feed oder andere Daten integrieren. Des weiteren gibt es "Operator"-Bausteine, mit denen man den Ouput von Bausteinen manipulieren kann, wie z.B. Filter, Sortierungswerkzeuge und andere. Wichtig ist hier vielleicht noch der "Merge"- Baustein, mit dessen Hilfe sich z.B. mehrere "Sources"- Bausteine auf einen Output führen lassen. Es gibt noch weitere Bausteingruppen, um die Erklärung jedoch nicht unnötig zu verkomplizieren, sollen die genannten hier mal genügen. Hat man die Bausteine nun ins Hauptfenster gezogen, kann sie dort miteinander verbinden und in den Bausteinen selbst die entsprechenden Optionen einstellen. Optionen können in einem "Source"-Baustein z.B. die Feed-Adresse sein oder in einem "Operator"-Baustein, wie einem Filter, die Filterkriterien. Schlussendlich leitet man alle gewünschten Outputs z.B. mit dem "Merge"-Baustein, falls man mehrere Inputs hat, auf den Baustein PipeOutput. Jetzt muss man die Pipe noch speichern mit "Save-Button" und anschließend auf "run pipe" klicken, um das fertige Ergebnis betrachten zu können. Im Screenshot- Beispiel wurden zur Veranschaulichung als Quellen zwei RSS-Feeds eingebunden. Einmal dass Feed von Tageschau.de links und als zweites das Feed von derStandard.at rechts. Das Feed von der Tagesschau wird einfach absteigend sortiert und das Feed von derStandard wird nach allen Feeds, deren Titel mit "A" anfangen, gefiltert. Abschließend werden noch beide Feeds mit dem "Merge"-Baustein auf dein PipeOutup geleitet.

Wie bereits erwähnt kann das Ergebnis selbst auch wieder als RSS-Feed aboniert oder über diesen Weg auch in andere Webseiten eingebunden werden. Wer noch mehr Details zu diesem Mashup Tool wissen möchte, am besten selbst ausprobieren unter pipes.yahoo.com/pipes/.