

Order.xml von Gimahhot

- Bestellvorgänge automatisieren -

Version 1.1 vom 13. Mai 2009

Dr. Stefan Kuhlins
kuhlins@gimahhot.de

1. Motivation

Gimahhot (<http://www.gimahhot.de/>) ist eine Produktbörse, auf der Produkte von zahlreichen Online-Shops angeboten werden. Kauft ein Kunde bei Gimahhot ein Produkt, wird ein Bestellvorgang ausgelöst. Dabei sendet Gimahhot ein XML-Dokument (`order.xml`) mit allen Daten, die mit der Bestellung zusammenhängen, an den Shop, der das Produkt an den Kunden liefern soll. Der Shop kann so die Bestellung voll automatisiert entgegennehmen.

Dieses Dokument beschreibt den technischen Ablauf solcher Bestellvorgänge und zeigt PHP-Programmcodebeispiele zur Implementierung.

2. Vorbereitung

2.1. Anmeldung eines Shops bei Gimahhot

Am einfachsten mithilfe einer Shopdatei `shopinfo.xml`, die sämtliche Stammdaten des Shops sowie die Beschreibung der Produktdaten enthält. Shopdateien können unter <http://elektronischer-markt.de/shopinfo> erzeugt werden.

2.2. Bereitstellung von Produktdaten

Üblicherweise stellen Shops eine Produktdaten im CSV-Format zum Download bereit. Die Produktdaten enthält die Daten sämtlicher Produkte, die der Shop über Gimahhot anbieten möchte, und wird vom Shop regelmäßig aktualisiert.

Zusätzlich ist der Einsatz einer Echtzeitschnittstelle zum Zugriff auf aktuelle Produktdaten möglich, z. B. um die Lieferbarkeit zu prüfen.

Für jedes Produkt muss neben dem Preis, bei dem es sich um einen Endkundenpreis inklusive Mehrwertsteuer und Versandkosten handelt, eine im Shop eindeutige und unveränderbare Artikelnummer (Produkt-ID) angegeben werden.

3. Ablauf eines Bestellvorgangs

3.1. Übersicht

Mithilfe von HTML-Eingabefeldern sammelt Gimahhot Daten über Kunden und die von ihnen bestellten Produkte. Für eine Bestellung werden diese Daten in ein XML-Dokument (`order.xml`) verpackt und per

HTTP-POST an ein Programm beim Shop geschickt, welches z. B. unter https://shop.de/gimahhot_order.php läuft. Der Shop wertet die Bestellung aus und antwortet seinerseits mit einem XML-Dokument (`order-status.xml`), das wiederum von Gimahhot bearbeitet wird.

3.2. Gimahhot sendet Bestelldaten

Für die XML-Bestelldokumente `order.xml` ist ein XML-Schema (<http://www.gimahhot.de/schema/1/order-1.xsd>) definiert, das festlegt, welche Elemente (Tags) mit welchen Attributen zulässig sind. Mithilfe des XML-Schemas können `order.xml`-Dokumente validiert werden, d. h., ein XML-Parser kann prüfen, ob das Dokument fehlerfrei aufgebaut ist. Dazu beginnt ein `order.xml`-Dokument mit den Zeilen:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Order id="999" xmlns="http://gimahhot.de/schema/order/1"
      xsi:schemaLocation="http://gimahhot.de/schema/order/1
      http://www.gimahhot.de/schema/1/order-1.xsd"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      schemaVersion="1.1" dateCreated="2008-03-05T08:55:01+01:00">
```

Das XML-Schema ist vollständig dokumentiert mit `<annotation>` und `<documentation>` Elementen. Die HTML-Seiten der Dokumentation stehen unter: <http://www.gimahhot.de/schema/>

Im PHP-Code erzeugt man solche XML-Dokumente bspw. mit der Erweiterung `XMLWriter` oder durch Einsetzen von Variablenwerten in ein XML-Template, z. B. in „heredoc Syntax“:

```
$doc = <<<EOT
<?xml version="1.0" encoding="ISO-8859-1"?>
<Order xmlns="http://gimahhot.de/schema/order/1"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://gimahhot.de/schema/order/1
      http://www.gimahhot.de/schema/1/order-1.xsd"
      id="{ $theOrder->ID }" schemaVersion="1.1"
      dateCreated="2008-03-05T08:55:01+01:00">
    <Sender>Gimahhot</Sender>
    <Receiver>$shop</Receiver>
    ...
</Order>
EOT;
```

Das fertige `order.xml`-Dokument soll bei Gimahhot in der Datenbank unter der eindeutigen Order-ID gespeichert werden, um es bei Fehlern, bspw. wenn die Übertragung zum Shop nicht klappt, nochmals senden zu können.

Das XML-Dokument wird anschließend per HTTP-POST (ähnlich zu einem HTML-Formular) an den Shop geschickt, bei dem ein PHP-Skript die Daten entgegennimmt. Im folgenden Beispiel ist das PHP-Skript unter der URL https://shop.de/gimahhot_order.php erreichbar, sodass die ge-

schickten Daten automatisch verschlüsselt werden. Außerdem wird ein Webserver-übliches Login mit Passwort benutzt. Die jeweilige Schnittstellen-URL für den Shop kommt aus der Gimahhot-Datenbank.

```

$url = 'https://gimahhot:geheim@shop.de/gimahhot_order.php';
$url_parts = parse_url($url);
$host = $url_parts['host'];
$path = $url_parts['path'];
$cred = base64_encode($url_parts['user'].':'.$url_parts['pass']);
$prot = $url_parts['scheme'] == 'https' ? 'ssl://' : 'tcp://';
if (isset($url_parts['port'])) $port = $url_parts['port'];
else $port = $url_parts['scheme'] == 'https' ? 443 : 80;

if ($sock = fsockopen($prot.$host, $port, $errno, $errstr)) {
    $len = strlen($doc);
    $request = <<<EOT
POST $path HTTP/1.0
Host: $host
User-Agent: Gimahhot/1.0 (http://www.gimahhot.de/)
Authorization: Basic $cred
Content-Type: application/xml; charset=ISO-8859-1
Content-Length: $len
Connection: close

$doc
EOT;
    fwrite($sock, $request);
}

```

Darüber hinaus kann Gimahhot das XML-Dokument auch an einen speziellen E-Mail-Account des Shops schicken, sodass der Shop eine Sicherungskopie der Daten hat. Anhand der Order-ID kann das Dokument bei Bedarf jederzeit wieder aus der Gimahhot-Datenbank geholt werden.

3.3. Shop nimmt Bestelldaten entgegen

Der Shop installiert ein Programm, das die von Gimahhot geschickten Bestelldokumente entgegennimmt. Das Programm kann mit den beim Shop verfügbaren Programmiersprachen implementiert werden, zum Beispiel mittels Java Servlet, JSP, ASP, PHP usw.

3.3.1. PHP 5

Im Folgenden wird die Technik mit PHP 5 demonstriert, wobei SimpleXML eingesetzt wird.

```

$request = file_get_contents('php://input');
$order = simplexml_load_string($request);
echo $order->Sender . ' ' . $order['id'];

```

Auf die Daten des XML-Dokuments kann anschließend über das Objekt `$order` bequem zugegriffen werden, bspw. liefert `$order['id']` die ID

der Bestellung und `$Order->Sender` den Absender. Das Skript kann die Bestelldaten dann in der Datenbank des Shops speichern.

Bei Problemen mit Sonderzeichen wie deutschen Umlauten können die Werte mit `utf8_decode` von UTF-8 nach ISO-8859-1 (Latin-1) konvertiert werden, z. B.: `utf8_decode($Order->Customer->LastName)`

3.3.2. PHP 4

Bei PHP 4 muss man ohne SimpleXML auskommen, was die Programmierung erschwert. Eine Möglichkeit ist der Einsatz der Klasse `XMLParser` von Adam A. Flynn (<http://www.criticaldevelopment.net/xml/doc.php>), um ähnlich zu SimpleXML zugreifen zu können.

```
require('parser_php4.php');
$request = file_get_contents('php://input');
$parser = new XMLParser($request);
$parser->Parse();
$order =& $parser->document;
echo $order->sender[0]->tagData . $order->tagAttrs['id'];
```

Im Vergleich zu SimpleXML ist der Zugriff durch zusätzliche Feldindizes sowie `tagData` und `tagAttrs` umständlicher. Außerdem werden alle Elementnamen klein geschrieben. Es gibt den `XMLParser` auch für PHP 5, sodass notfalls einheitlich für beide Versionen programmiert werden kann.

3.4. Shop antwortet

Abschließend muss das Shop-Skript, das die Bestellung entgegen genommen hat, dem Gimahhot-Skript antworten, ob alles fehlerfrei verlaufen ist. Dazu wird ein XML-Dokument `order-status.xml` passend zum XML-Schema `order-status-1.xsd` erstellt und als Antwort ausgegeben. Das XML-Schema ist vollständig dokumentiert mit `<annotation>` und `<documentation>` Elementen. Die HTML-Dokumentation steht unter: <http://www.gimahhot.de/schema/>

Der folgende Programmcode läuft sowohl in PHP 4 als auch PHP 5 und nutzt „heredoc Syntax“, um ein XML-Template mit Werten zu füllen.

```
header('Content-Type: application/xml; charset=ISO-8859-1');
$doc = <<<EOT
<?xml version="1.0" encoding="ISO-8859-1"?>
<OrderStatus id="$order_id" schemaVersion="1.0"
    dateCreated="2008-03-05T08:55:03+01:00"
    xmlns="http://gimahhot.de/schema/order/1"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation=
        "http://gimahhot.de/schema/order/1
        http://www.gimahhot.de/schema/1/order-status-1.xsd">
<OK>
<OrderID>$my_order_id</OrderID>
```

```

        <CustomerID>$my_customer_id</CustomerID>
        <StatusCode>$my_order_status_code</StatusCode>
        <StatusText lang="de">$my_order_status_text</StatusText>
    </OK>
</OrderStatus>
EOT;
echo $doc;

```

Falls Fehler auftreten, sieht das Dokument bspw. wie folgt aus:

```

$doc = <<<EOT
<?xml version="1.0" encoding="ISO-8859-1"?>
<OrderStatus schemaVersion="1.0"
    dateCreated="2008-03-05T08:55:02+01:00"
    xmlns="http://gimahhot.de/schema/order/1"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation=
        "http://gimahhot.de/schema/order/1
        http://www.gimahhot.de/schema/1/order-status-1.xsd">
    <Error>
        <Code>102</Code>
        <Message lang="en">Invalid request</Message>
    </Error>
</OrderStatus>
EOT;

```

Die einsetzbaren Fehlercodes sind im XML-Schema deklariert.

Wichtig ist, dass immer geantwortet wird. Denn sonst ist ein Serverfehler zu vermuten und das ursprüngliche XML-Dokument muss zu einem späteren Zeitpunkt nochmals geschickt werden.

Auftretende Fehler sollten möglichst schnell behoben werden. Es bietet sich daher an, im Fehlerfall eine E-Mail an den Shop-Administrator zu schicken.

3.5. Gimahhot wertet die Antwort aus

Zum abschließenden Auswerten der Antwort benutzt Gimahhot den folgenden (hier vereinfacht ohne Fehlerbehandlung dargestellten) PHP 5-Code:

```

$content = stream_get_contents($sock);
fclose($sock);
$p = strpos($content, "\r\n\r\n") + 4;
$headers = substr($content, 0, $p);
$body = substr($content, $p);
$regex = '#^HTTP/\d.\d\s+(\d+)\s+(.+)#i';
if (preg_match($regex, $headers, $matches)) {
    $http_status_code = $matches[1];
    if ($http_status_code == 200) {
        $orderStatus = simplexml_load_string($body);
    }
}

```

```
        if ($OrderStatus->OK)
            echo $OrderStatus->OK->StatusCode;
        else if ($OrderStatus->Error)
            echo $OrderStatus->Error->Code;
    }
}
```

Über das Objekt `$OrderStatus` kann komfortabel mit den Werten aus der XML-Antwort gearbeitet werden, bspw. um sie in der Datenbank zu speichern oder Fehlermeldungen anzuzeigen.

Die beim Shop verwendeten IDs für die Bestellung und den Kunden werden bei Gimahhot gespeichert. Kunden, die mehrere Bestellungen tätigen, können so leichter identifiziert werden.

4. Sicherheit

Wie beschrieben sollte eine SSL-Verbindung (`https`) verwendet werden, damit insbesondere die in Bestellungen enthaltenen Kundendaten nur verschlüsselt über das Internet verschickt werden. Gleiches gilt für das im Folgenden eingerichtete Login. (Durch die Verwendung von Client-Zertifikaten lässt sich die Sicherheit bei Bedarf erhöhen.)

Störer könnten versuchen, Bestellungen im Namen von Gimahhot an Shops zu senden. Wie gezeigt kann dies durch Vereinbaren eines Logins für den Zugriff auf die Schnittstelle beim Shop erschwert werden. Dies kann beim Webserver Apache in der Regel mithilfe einer `.htaccess`-Datei bewerkstelligt werden, z. B.:

```
<Files "gimahhot_order.php">
    AuthName "Gimahhot Order Interface"
    AuthUserFile ".htpasswd"
    AuthType Basic
    require valid-user
</Files>
```

Da alle gültigen Schnittstellenanfragen vom Gimahhot-Server (IP `188.65.73.118` oder `188.65.73.119` und Domain `gimahhot.de`) kommen müssen, können Zugriffe von anderen Rechnern per `.htaccess` blockiert werden, z. B.:

```
Order deny,allow
Deny from all
Allow from 188.65.73.118
Allow from 188.65.73.119
Allow from gimahhot.de
```

Die Prüfung kann auch innerhalb des PHP-Skripts erfolgen, z. B.:

```
if ($_SERVER['REMOTE_ADDR'] != '188.65.73.118 '
    && $_SERVER['REMOTE_ADDR'] != '188.65.73.119')
{ header('HTTP/1.0 403 Forbidden');
  exit;
}
```

Darüber hinaus ist der Einsatz asymmetrischer Verschlüsselungsverfahren möglich, um die kompletten XML-Dokumente zu kodieren und mit einer elektronischen Signatur zu versehen. Mit OpenSSL bringt PHP alle notwendigen Voraussetzungen dafür mit.

5. Technik

Das von Gimahhot eingesetzte und hier beschriebene technische Verfahren des Austauschs von XML-Dokumenten mittels HTTP-Befehlen nennt sich REST (Representational State Transfer). Es ist technisch verhältnismäßig simpel und somit auf allen Webservern einsetzbar, sodass alle Shops angeschlossen werden können.

Darüber hinaus erlaubt das Verfahren auch den Austausch der XML-Dokumente über andere Wege wie FTP, E-Mail usw.

6. FTP

Alternativ zum synchronen Austausch von XML-Dokumenten mittels HTTP-Befehlen kann auch FTP benutzt werden. Dazu richtet der Shop auf seinem FTP-Server einen Account für Gimahhot ein, sodass Gimahhot XML-Dokumente dort ablegen kann. Gimahhot benötigt dann lediglich die zugehörige FTP-URL wie z. B.: `ftp://gimahhot:geheim@ftp.shop.de/` (Gegebenenfalls ist noch der Pfad für ein Unterverzeichnis anzuhängen.)

Der Shop schaut regelmäßig nach (bspw. per Cron-Job), ob Dokumente eingetroffen und zu verarbeiten sind.

XML-Dokumente, die der Shop an Gimahhot übermitteln möchte, kann er entweder auf seinem eigenen FTP-Server für Gimahhot zur Abholung bereitstellen oder auf Gimahhots FTP-Server hochladen. Dazu erhält er ggf. einen eigenen FTP-Account bei Gimahhot.

Folgende Konventionen benutzt Gimahhot für die Dateinamen, die Gimahhot für den Shop ablegt, z. B.:

```
gimahhot_order_20080313_090406_4321.xml
```

Die erste Zahl ist der Tag (`yyyymmdd`), die zweite die Uhrzeit (`hhmmss`) und die dritte eine interne Nummer von Gimahhot.

Dateien, die der Shop für Gimahhot erzeugt, sollen z. B. so aussehen:

```
amazon_order_status_20080313_150519_1234.xml
```

Auch hier ist wieder die erste Zahl der Tag (`yyyymmdd`), die zweite die Uhrzeit (`hhmmss`) und die dritte eine interne Nummer des Shops. Der Shopname wird vorangestellt.

In beiden Fällen müssen die internen Nummern sicherstellen, dass keine zwei Dateien denselben haben. Am einfachsten ist also eine fortlaufende Nummer oder eine Datenbank-ID, unter der das Dokument zu Protokollierungszwecken gesichert wird.

Auf dem FTP-Server des Shops bietet es sich an, für eingehende und ausgehende XML-Dokumente zwei verschiedene Verzeichnisse wie bspw. **'incoming'** und **'outgoing'** zu verwenden.

Da FTP sämtliche Daten unverschlüsselt überträgt, ist es sicherer, eines der verschlüsselten FTP-Verfahren wie FTPS, *Secure* FTP usw. einzusetzen.

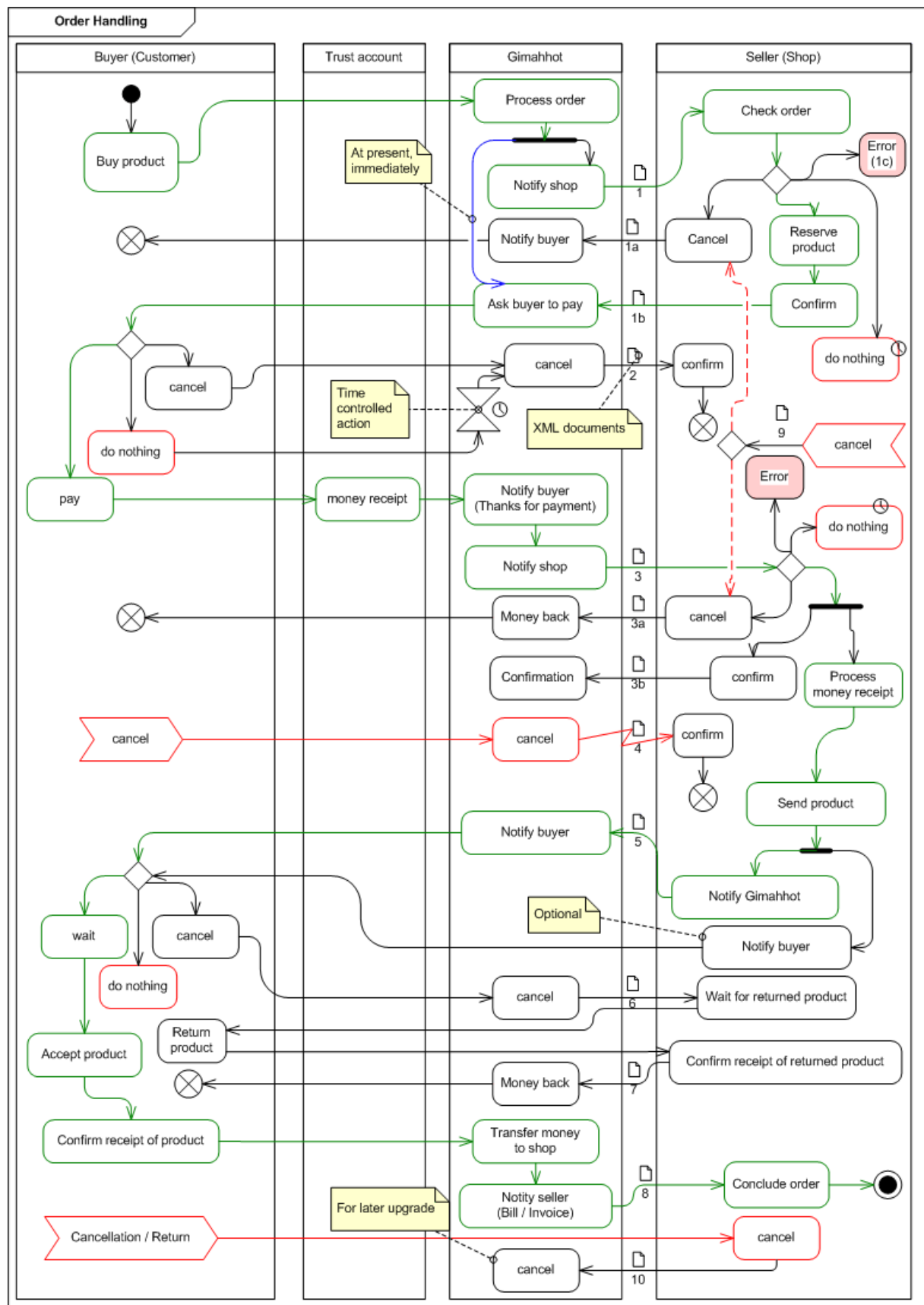
Der asynchrone Austausch von XML-Dokumenten mittels FTP hat den Nachteil, dass zwischen Bereitstellen eines Dokuments und dessen Verarbeitung eine größere Zeitspanne liegen kann, sodass ein Dokument bei der Verarbeitung u. U. schon nicht mehr aktuell ist. Vorteilhaft ist, dass Dokumente bei auftretenden Übertragungsproblemen leicht mehrmals übertragen werden können.

7. Links

- Blog:
<http://blog.gimahhot.de/>
- Testmöglichkeiten für API-Entwickler:
<http://www.gimahhot.de/api/test/>
- Dokumentation:
<http://www.gimahhot.de/schema/>

8. Anhang: Flussdiagramm

Die Abbildung zeigt den Austausch von XML-Dokumenten zwischen Gimahhot und einem Shop (<http://gimahhot.de/schema/flowchart.wmf>).



Die folgende Tabelle gibt ergänzend zum Flussdiagramm an, welche Dokumente von Gimahhot oder einem Shop mit welchem Statuscode gesendet werden. Die einzelnen Codes sind in den jeweiligen XML-Schemas dokumentiert.

Nr.	Anlass	Gimahhot	Shop	StatusCode
1)	Bestellung	order.xml		1
1a)	Storno		order-status.xml	301
1b)	Bestätigung		order-status.xml	200
1c)	Fehler		order-status.xml	Error Code
2)	Käufer storniert	order.xml		4 oder 5
2a)	Bestätigung		order-status.xml	302
3)	Geldeingang	order.xml		2
3a)	Storno		order-status.xml	301
3b)	Bestätigung		order-status.xml	201
4)	Käufer storniert	order.xml		5
4a)	Bestätigung		order-status.xml	302
5)	Ware verschickt		order-status.xml	202
5a)	Bestätigung	order.xml		10
6)	Käufer storniert	order.xml		11 = Geld bei Gimahhot 6 = Geld beim Shop
6a)	Bestätigung		order-status.xml	302
7)	Käufer retourniert		order-status.xml	303
7a)	Bestätigung	order.xml		12
8)	Käufer hat Ware erhalten	order.xml		3
8a)	Bestätigung		order-status.xml	203
9)	Storno		order-status.xml	301
9a)	Bestätigung	order.xml		7
10)	Käufer storniert nachträglich beim Shop		order-status.xml	304
10a)	Bestätigung	order.xml		13

Gimahhot verschickt von sich aus also nur **order.xml**-Dokumente, und Shops versenden ausschließlich **order-status.xml**-Dateien. Lediglich als Antwort auf per HTTP von Shops geschickte XML-Dokumente setzt Gimahhot für Fehlermeldungen **order-status.xml**-Dokumente ein, die in der Tabelle nicht explizit aufgeführt sind.