Bachelorarbeit

Web Services in der Gebäudeautomation

mit Schwerpunkt auf oBIX

Helene Oberhumer
0626627, E535
e0626627@student.tuwien.ac.at
WS 2009/10

betreut von
ao. Univ. Prof. Dr. Wolfgang Kastner

Inhaltsverzeichnis

1 Einleitung

Der erste Teil der Arbeit bietet einen Überblick über Web Services in der Gebäudeautomation. Dabei werden drei existierende Realisierungen von Web Services aufgezeigt. Weiters wird die Bedeutung der Anwendung der Web Services erläutert und die Vor- und Nachteile des Einsatzes aufgelistet.
Im Hauptteil wird ausführlich auf das Web Service oBIX eingegangen. oBIX steht für Open Building Information eXchange und wird von OASIS [1] entwickelt. Nach einer Beschreibung der Architektur von oBIX wird auszugsweise die Spezifikation v1.0 dargestellt.

2 Web Services in der Gebäudeautomation

Dieses Kapitel gibt eine Einführung in Web Services in der Gebäudeautomation. Es wird ein Überblick über existierende Web Services gegeben. Weiters wird auf die Bedeutung des Einsatzes von Web Services in der Gebäudeautomation eingegangen. Zuletzt werden Vor- und Nachteile aufgelistet.

2.1 Definition

Web Services sind eigenständige, modulare Software Komponenten, die über das Web beschrieben, veröffentlicht und aktiviert werden können [1]. Sie verwenden existierende IT Standards und stellen für die Gebäudeautomation (BA) neue Möglichkeiten für die in Abschnitt 2.3 dargestellten Einsatzbereiche dar. Zentrale Komponenten von Web Services sind XML für die Datendarstellung und standardisierte Protokolle für den Zugriff und Austausch. Für letztere finden typischerweise SOAP oder HTTP Verwendung [13]. Ein Hauptaugenmerk von Web Services ist es, eine (technische) Schicht zu gestalten, die unabhängig von Plattformen oder Programmiersprachen ist. Im Gegensatz zu herkömmlichen Webapplikationen werden Web Services hauptsächlich von anderen Anwendungen und nicht von menschlichen Benutzern verwendet [1]. Da Web Services einen auf XML-Nachrichten basierten Ansatz verfolgen, ermöglichen sie das Erstellen von lose gekoppelten und verteilten Systemen über das Internet [2].

2.2 Realisierungen

In der BA existieren neben oBIX noch andere Implementierungen von Web Services. BACnet/WS (Building Automation and Control Network/Web Services) stellt eine Erweiterung des BACnet Standards dar und wurde 2004 von ASHRAE [2] ergänzt. OPC UA

[1] OASIS steht für Organization for the Advancement of Structured Information Standards
[2] ASHRAE steht für American Society of Heating, Refrigerating and Air Conditioning Engineers

steht für OPC [3] Unified Architecture und wurde im Juni 2006 von der OPC Foundation veröffentlicht.
Alle drei Standards bieten historischen Datenzugriff, Ereignis- und Alarmmanagement und Datenpunktabstraktion. BACnet/WS und oBIX wurden speziell für die BA entwickelt und sind frei verfügbar [13].

BACnet/WS
BACnet/WS definiert kein SOAP (Simple Object Access Protocol) Nachrichtenformat und untersützt keine XML Schemas. Stattdessen wird eine Schnittstelle definiert, die von höheren Programmiersprachen, wie C++ oder Visual Basic, verwendet werden kann [7]. BACnet/WS wurde in der Annahme entwickelt, dass individuelle, maßgeschneiderte Schnittstellen den SOAP/XML Nachrichtendefinitionen vorangehen [7].

OPC UA
OPC UA stellt die XML-Variante von OPC für den Zugriff auf Datenpunkte dar [10]. Der Nachteil, dass OPC auf Windows Plattformen eingeschränkt ist, wird durch die Lösung mit Web Services beseitigt [13].
OPC UA wird durch XML und Web Services zu einer SOA (Service Orientierten Architektur). Es werden zwei Protokolle für den Nachrichtentransport eingesetzt: ein binäres TCP-Protokoll und ein auf Web Services basierendes Protokoll (SOAP).

2.3 Bedeutung der Anwendung

In der Gebäudeautomation sollen heterogene Subsysteme in einer effizienten Art miteinander arbeiten. Da Geräte und Systeme oft von verschiedenen Herstellern kommen, können Unterschiede in den Betriebssystemen, Programmiersprachen und der Hardware bestehen [18]. Die verschiedenen Teilsysteme zu managen, erweist sich als schwierig.
Man möchte eine gemeinsame Ausführung von Aufgaben, die zur Maximierung von Komfort und Effizienz führen sollen, erreichen [18]. Dafür müssen diese Einheiten gemeinsam integriert werden und interagieren können. Wichig ist, dass dies geschieht, ohne dass darunterliegende Systemplattformen oder Protokolle verändert werden müssen. Erschwert wird das Erreichen dieser Interoperabilität [4] oft durch das Faktum, dass Hersteller Systeme in Isolation und ohne Bedenken auf ein gemeinsames Arbeiten mit anderen Systemen, entwickeln. In [18] wird betont, dass Web Services eine Technologie ist, mit welcher hohe Interoperabilität erzeugt werden kann.

[3] OPC heißt OLE for Process Control; OLE steht für Object Linking and Embedding

[4] [18] beschreibt die Interoperabilität eines Systems als die Fähigkeit, dass zwei oder mehrere Systeme Informationen austauschen und diese verarbeiten können. Interoperabilität erlaubt Informations- und Ressourcenaustausch zwischen definierten Subsystemen.

Der Umstand das BA-Systeme mit verschiedenen standardisierten Protokollen wie BACnet, Lontalk und KNX arbeiten, erschwert die Integration. [1] spricht dabei von isolierten Informationsinseln, die aufgrund der Implementierung von verschiedenen (Kommunikations-)Protokollen entstehen. In Abbildung 1 sind die drei Arten der Integration dargestellt. Einerseits ist die Integration von BA- und Enterprise-Systemen abgebildet. Zweitens gibt es das Integrationsproblem von BA-Systemen, welche beispielsweise BACnet, Lontalk und andere Protokolle verwenden. Als Drittes stellt die Grafik die Integration von BA-Systeme mit verschiedenen Funktionalitäten dar, zum Beispiel den Zusammenschluss von HVAC-, Feueralarm- und Beleuchtungssystemen [1]. Web Services sind aber nicht dazu gedacht, die standardisierten Protokolle (z.B. BACnet, Lontalk und KNX) auf Feldbusebene zu verdrängen. Stattdessen adoptieren sie diese um Integration in höhreren Schichten zu generieren [2].

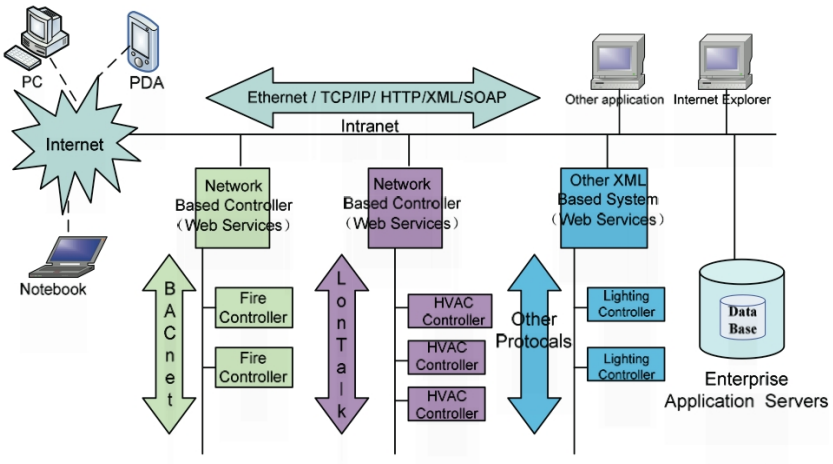

Abb. 1: Integration von BA- und Enterprise-Systemen (Quelle: [1])

oBIX und Enterprise Systeme

[5] beschreibt oBIX als ein sehr Enterprise freundliches Protokoll mit hoher Verlässlichkeit. oBIX wird typischerweise zur Systemintegration von Systemen der Gebäudeautomation mit Enterprisesystemen (wie Customer Relationship Management, Human Resources Systeme, Supply Chain Management Systeme, Facility Management Systeme) verwendet [4], [14]. Ein Beispiel solch einer Integration ist ein Campus Schedular. Ein System, das einen Konferenzraum reserviert, kann auch aufgrund des Wissens über die Reservierungszeiten automatisch die Einrichtung, das Licht und die Sicherheitssysteme anpassen [14].

Gebäude und die Einrichtung sind heute signifikante Bereiche, die in Enterprisesystemen inkludiert werden sollen. Die Effizienz der Anlage wirkt sich auf den Gewinn des Unternehmens aus [4].

Bei der Integration soll auch auf die Bedingungen der IT Abteilung, also deren verwendeten Sprachen, Regeln, Standards, Tools, XML und Web Services Bezug genommen werden [4]. Die Tatsache, dass oBIX diese Technologien verwendet, macht dessen Einsatz noch attraktiver [4].

2.4 Vorteile und Nachteile

Die Verwendung von Web Services ermöglicht es, dass mehrere eigenständige Applikationen, die auf verschiedenen Plattformen und Betriebssystemen laufen, über das Internet/Intranet kooperieren können [13].

Als Vorteile für Web Services ergeben sich:

+ Web Services sind plattformunabhängig: sie können unabhängig von der Programmiersprache implementiert werden und sind auf vielen Hardware Plattformen oder Betriebssystemen lauffähig. Dies erhöht die Interoperabilität und die Wiederverwendbarkeit der angebotenen Services [13].
+ Web Services ermöglichen eine "nahtlose"- und Echtzeit-Integration von Enterprise-Anwendungen und BA-Systemen [1].
+ Web Services ermöglichen eine Integration von BA-Systemen, die verschiedene Kommunikationsprotokolle implementieren, ohne zusätzliche Gateways [1].
+ Web Services können durch kooperatives Arbeiten zwischen den Subsystemen (wie beispielsweise HVAC System, Beleuchtungssystem, Sicherheitssystem, Feueralarmierungssystem und Enterprise-Anwendung) BA-Systeme noch intelligenter machen [1].
+ Web Services werden von den wichtigsten Herstellern von Enterprise-Systemen, wie Microsoft.NET [5], IBM WebSphere [6], Sun Microsystems [7], HP Web Services platform [8], unterstützt [1].

[5].NET ist eine Laufzeitumgebung für Computerprogramme. Sie beinhaltet Klassenbibilotheken, eine API und Dienstprogramme. Für verteilte Programmierung und Web Services bietet es unter anderem eine moderne Implementierung des Standards Webservices [12].

[6]IBM WebSphere ist eine Produktlinie von IBM, die unterschiedliche Software für Anwendungsintegration, Infrastruktur und eine integrierte Entwicklungsumgebung umfasst. Es wurde entwickelt, um Business Anwendungen quer über verschiedene Plattformen erstellen und integrieren zu können. Dabei werden Java basierende Web Technologien verwendet. Beispielsweise verwendet WebSphere Application Server offene Standards wie Java EE (Java Platform, Enterprise Edition), XML und Web Services [9].

[7]Unter Verwendung von Java Technologie APIs und Metro, einem Web Service Stack, können Web Services entwickelt werden. Es kann alles vom einfachen Hello World Web Service bis zu verlässlichen, sicheren Web Services die .NET Services involvieren, erstellt werden [19].

[8]Die HP Web Services Plattform bietet modulare, standardbasierte Architektur für die Zusammenstellung von XML Komponenten zum Entwickeln von Web Services [8].

+ Web Services sind eigenständig gebaut und lose gekoppelt: sie können flexibel zu komplexen Applikationen arrangiert werden [13].
+ Web Services verfolgen ein modulares Konzept: das erlaubt die Verwendung von Standards für Übertragung, Ereignissteuerung, Datenermittlung und Schutz [13].
+ Web Services sind geeignet für Fernzugriff auf einen Server [13].

Als Nachteile von Web Services erwähnt [13]:

- Es entsteht ein zusätzlicher Overhead aufgrund XML kodierter Nachrichten, speziell wenn kleine Mengen von Daten sehr oft übertragen werden.
- Das Empfangen von Ereignissen vom Server erfordert für den Client periodisches Polling oder eine Implementierung des Clients als Web Service Server. Dies stellt einen Nachteil auf Feldebene dar, da schnelle Antwortzeiten und Ressourceneffizienz entscheidend sind.

3 oBIX - Open Building Information eXchange

Dieses Kapitel beschreibt die Entstehung von oBIX und gibt eine kurze Definition der Bezeichnung an. In Abschnitt 3.3 wird die Architektur erläutert.

3.1 Einführung

oBIX steht für Open Building Information eXchange und ist ein XML- und Web Service basierter Mechanismus zur Darstellung und Übertragung von Daten in der Gebäudeautomation. oBIX wird von OASIS (Organization for the Advancement of Structured Information Standards) implementiert.
oBIX war ursprünglich eine Sammlung von Richtlinien, die von CABA (Continental Automated Buildings Association) und Vertretern der Industrie initiiert waren. Das CABA Komitee hat sich 2003 in oBIX umbenannt. Mit Hilfe von OASIS haben sich diese Richtlinien dann als Standard etablieren können. CABA unterstützt die Arbeit der oBIX Group, bietet Services und fördert die oBIX Aktivitäten in der Industrie [4].

Mit oBIX möchte man eine öffentlich zugängige Web Service Schnittstellen Spezifikation schaffen. Diese soll für das Einholen von Daten von BA-Systemen auf einem einfachen und sicheren Weg und zum Austausch von Daten zwischen diesen Systemen und Enterprise Anwendungen Verwendung finden [1]. Die Daten werden über TCP/IP Netzwerke wie das Internet ausgetauscht [4].
oBIX ist kein Netzwerkprotokoll, sondern eine Menge von Standards, wie Systeme in der Gebäudeautomation untereinander und mit Enterprise Systemen kommunizieren können. Deshalb stellt oBIX keinen Konflikt zu LonTalk oder BACnet dar [14]. oBIX deckt die

Bereiche Heating, Ventilation und Air conditioning (HVAC), Fahrstühle, Labor Einrichtungen, Life- und Safety Systeme, Zutrittskontrollen, Einbruchserkennung (intruder detection) und CCTV Monitoring ab [14]. Weiters werden die Bereiche Security, Lighting und Energy miteinbezogen [17]. oBIX bietet Echtzeit-Datenzugriff auf Sensordaten [14]. oBIX stellt auch eine Art Meta-Modell, um andere Modelle zu beschreiben, bereit [6].

3.2 Definition

In [11] ist die Definition von oBIX wie folgt zusammengefasst:

- **o**pen: alle technischen Details sind frei verfügbar
- **b**uilding: alle Gebäude(automations)systeme
- **i**nformation: sachbezogene Systemdaten
- e**x**change: Interoperabilität

3.3 Architektur

Abbildung 2 bietet einen Überblick über die oBIX Architektur (Stand 2006). In der Grafik werden das Object Model von oBIX und die Darstellungsmöglichkeiten mit Contracts, sowie die zwei protocol bindings SOAP und HTTP zusammengefasst veranschaulicht. Die "Future Bindings" und "Future Standards" deuten auf zukünftige Entwicklungen hin. Auf diese Punkte wird in den nächsten Abschnitten näher eingegangen.

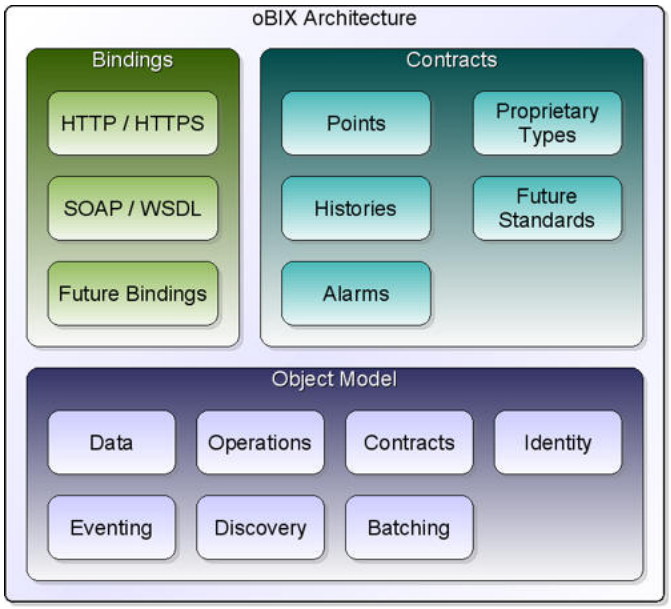

Abb. 2: oBIX Architektur, Mai 2006 (Quelle: [6])

3.3.1 Object Model

In oBIX basiert alles auf Objekten. Objekte werden zur Beschreibung von Datentypen (Klassen) und Operationen (Signaturen von Methoden) verwendet. Die Flexibilität von oBIX ist durch die Möglichkeit der Definition jeglicher Art von Objekten hergestellt [13]. Dabei werden die Beziehungen "is-a" und "has-a" (Komposition) unterstützt [13]. Jedem Objekt kann eine URI zugewiesen werden und jedes Objekt kann mehrere andere Objekte beinhalten.

3.3.2 Erweiterbarkeit

Um in oBIX eine Erweiterbarkeit zu gewährleisten, werden sogenannte Contracts verwendet [6]. Ein Contract stellt eine Zusammenfassung von Elementen dar, die zu einer komplexen Datengröße gehören. Eine Analogie in der objektorientierten Programmierung stellen die Klassen dar. Diese Contracts werden dann zur Darstellung von Daten im System, wie den Datenpunkten, den historischen Verläufen und den Alarmen aber auch zur Beschreibung von proprietären Herstellerdaten verwendet [6]. Es können beliebige Contracts definiert werden, ohne dass das oBIX Schema verändert werden muss. In typischen Web Services für Business-Anwendungen bedeutet das Hinzufügen von neuen Datenstrukturen auch die Definition von neuen Schemata. Anders als bei oBIX, können die (web-basierten) Enterprise-Anwendungen nicht mit diesen unbekannten Schemata umgehen und ignorieren diese unerwarteten Daten. Bei oBIX hingegen werden einfach neue Contracts definiert. Der Client kann dann aufgrund des oBIX Schemas (XML Schemas) auf die Daten zugreifen. Alle Daten werden als erstklassig behandelt, d.h. es werden keine Daten ignoriert nur weil sie nicht erwartet wurden [6].

3.3.3 XML Darstellung

Objekte werden in oBIX durch XML-Code dargestellt. Objekte, die nicht von Basistypen wie beispielsweise `<int>` abgeleitet werden, werden als Standardelement `<obj>` übersetzt.
Attribute von Objekten, genannt facets, können über XML Attribute dargestellt werden. Facets beinhalten Metadaten über das Objekt, wie beispielsweise der Name oder der Status. Subobjekte können in deren vollen XML Darstellung oder über eine Referenz inkludiert werden [13]. Ein Alarmobjekt könnte beispielsweise so aussehen [13]:

```
<obj name=''einalarm'' is=''obix:Alarm''>
   <ref name=''source'' href=''/meinHaus/irgendwo'' />
   <abstime name=timestamp'' val=''2010-02-12_12:22:22'' />
</obj>
```

Der Contract `obix:Alarm` beim "is" Attribut dient als eine Vorlage (Klasse in der objektorientierten Programmierung) für das Objekt. Das Objekt "einalarm" erbt somit

alle Subobjekte von `obix:Alarm` . Natürlich besteht die Möglichkeit diese Defaultwerte zu überschreiben. oBIX unterstützt auch die Vererbung von mehreren Contracts.

3.3.4 RESTful Ansatz

REST bedeutet Representational State Transfer und beschreibt das Architekturmerkmal des World Wide Web. Dieser Ansatz besagt, dass sich die Ressourcen eine einheitliche Schnittstelle teilen und nur eine limitierte Anzahl von Operationen auf diesen Ressourcen möglich sind. Die Standards HTTP, URL, XML und HTML, die das Web so erfolgreich gemacht haben, verfolgen das REST Modell [6]. Mit den drei Operationen GET, PUT und POST auf unzählige Ressourcen zugegriffen werden [13].
Auch oBIX Services implementieren diesen Ansatz. Es sind die drei Netzwerkanfragen `read` , `write` und `invoke` definiert. `Read` und `write` sind vergleichbar mit "get" und "set". Weiters ist es möglich zusätzliche Operationen wie Objekte zu definieren. Diese können über die Netzwerkanfrage `invoke` aktiviert werden [13].

3.3.5 Protocol Binding

oBIX ist in Bezug auf das Protokoll Binding agnostisch [6]. Einerseits gibt es das SOAP Binding, sodass oBIX mit dem WS-* (Web Service stack, dem Schichtenmodell für Web Services) interagieren kann, andererseits das HTTP Binding, welches oBIX zu einem "RESTful" Standard macht.
oBIX verwendet die erfolgreichen Standards URL, XML und HTTP. Es identifiziert Objekte mit URIs (Uniform Resource Identifier), repräsentiert das Objekt mittels XML und transferiert die Objekte mit Hilfe von HTTP (Hyper Text Transfer Protocol) [6]. Ein Vorteil ist, dass so über einen Webbrowser auf den oBIX Server zugegriffen werden kann. Die Server können von Suchmaschinen indiziert werden, auf andere Webseiten verlinkt werden und grundsätzlich mit den anderen verbreiteten Webtechnologien interagieren [6].

Anmerkung: SOAP (Simple Object Access Protocol) ist ein Kommunikationsmechanismus, der den Austausch von strukturierter Information in verteilten System erlaubt. SOAP tauscht Information in Form von Nachrichten aus [18]. Eine SOAP Nachricht besteht aus einem SOAP Envelope, das den Namespace enthält, welcher die SOAP Version definiert. Das SOAP Envelope beinhaltet einen SOAP Header (optional) und einen SOAP Body. Der Header schließt Informationen, die vom SOAP Prozessor interpretiert werden, ein. Im Body stehen die Daten zum Senden und Empfangen.

3.3.6 Naming

Das Naming wird in oBIX über zwei verschiedene, komplementäre Konzepte realisiert [13]. Einerseits hat jedes Objekt einen "Namen" der verwendet wird, um das Subobjekt innerhalb eines "Composite-Objects" zu identifizieren. Andererseits kann die Namensgebung über die Zuweisung einer URI (einem "href") geschehen [13]. Dieser Vorgang ist nötig, wenn auf das Objekt von außen referenziert werden muss, nämlich bei einem "read", "write" oder "invoke" Netzwerkzugriff [13].

3.3.7 Vordefinierte Objekte

In oBIX werden die zwei praktischen Konzepte "list" und "feed" angeboten. List stellt einen Container mit statischem Inhalt dar, Feed hingegen hat den Charakter einer Queue. Weiters gibt es Objekte für Fehler, Referenzen zu anderen Objekten und Operationen [13]. Beliebige Klassen können von (auch mehreren) Typen abgeleitet werden, d.h. es wird Mehrfachvererbung unterstützt [13]. oBIX bietet auch einfache Wege um physikalische Werte und deren Einheiten (in SI-Einheiten) darzustellen [13].
Das `Lobby` Objekt hat eine festgelegte Position am Server und dient als Einstiegspunkt [13]. Es bietet nicht nur Informationen über den Server (über das `About` Objekt), sondern auch Batch Operationen, um unnötigen Protokoll-Overhead zu reduzieren [13]. Es ist auch möglich, dass Clients einen Watch Service für bestimmte Objekte/Daten registrieren. Jedesmal, wenn der Client diese Watch pollt, erhält er vom Server ein Feed von Ereignissen (beispielsweise eine Wertänderung), die seit dem letzten Pollen passiert sind [13].
In oBIX sind Points spezifiziert, mit denen primitive Werte dargestellt werden können. Schreibgeschützte Punkte dienen als semantische Kennzeichnung, beschreibbare Punkte hingegen definieren eine Operation, um den Wert ändern zu können.
Um historische Trends darstellen zu können, kann in oBIX ein History Record, welches einen Punktwert und dessen Zeitstempel zusammenfasst, verwendet werden. Ein `History` Objekt besteht aus mehreren History Records und Methoden, um diese abzufragen. Es können auch Filter für eine Berechnung (beispielsweise Mittelwertbildung) spezifiziert werden [13].
oBIX definiert ein Modell zum Abfragen, Beobachten und Bestätigen von Alarmen [13]. Jedesmal, wenn der Server einen Alarm entdeckt, d.h. ein Wert liegt in der vordefinierten Alarmbedingung, wird ein `Alarm` -Objekt zu einem `Alarm` -Feed hinzugefügt. Dieses Objekt beinhaltet einen Zeitstempel und die Alarmquelle. Alarme können stateful sein, d.h. der Zeitpunkt, wann die Quelle wieder in den normalen Zustand übergeht, wird aufgezeichnet. Weiters kann aufgenommen werden, ob und von wem ein Alarm bestätigt wurde.

4 oBIX Spezifikation

Folgender Abschnitt beschreibt einige interessante Teile der "Spezifikation oBIX 1.0", [16]. Das Object Model wird verwendet um alle oBIX Informationen zu definieren. Dieses wird mittels XML Syntax dargestellt. URIs werden dazu verwendet, um Daten im Object Model zu identifizieren. REST steht für das Architekturmerkmal, wie auf die oBIX Objekte (über deren URIs) zugegriffen wird und diese transportiert werden. Um neue oBIX Klassen zu definieren, wird das Konzept Contracts verwendet, welches auch die Erweiterbarkeit gewährleistet.

4.1 Object Model

Im Object Model sind acht primitive Datentypen definiert, mit welchen einfache Daten gespeichert werden können: booleans, integers, floating point numbers, strings, enumerations, absolute time values (timestamp), relative time values (duration or time span) und URIs. Es gibt auch spezielle Objekttypen wie list, op, feed, ref und err.
Das oBIX Object Model ist in Abbildung 3 dargestellt. In dieser Grafik stellt jede Box einen Objekttyp inklusive seiner Attribute dar. Das Wurzelobjekt "object" wird in XML mit dem "obj" Element modelliert. Jedes XML Element in oBIX ist vom "obj" Element abgeleitet. Jedes "obj" Element kann andere "obj" Elemente enthalten.

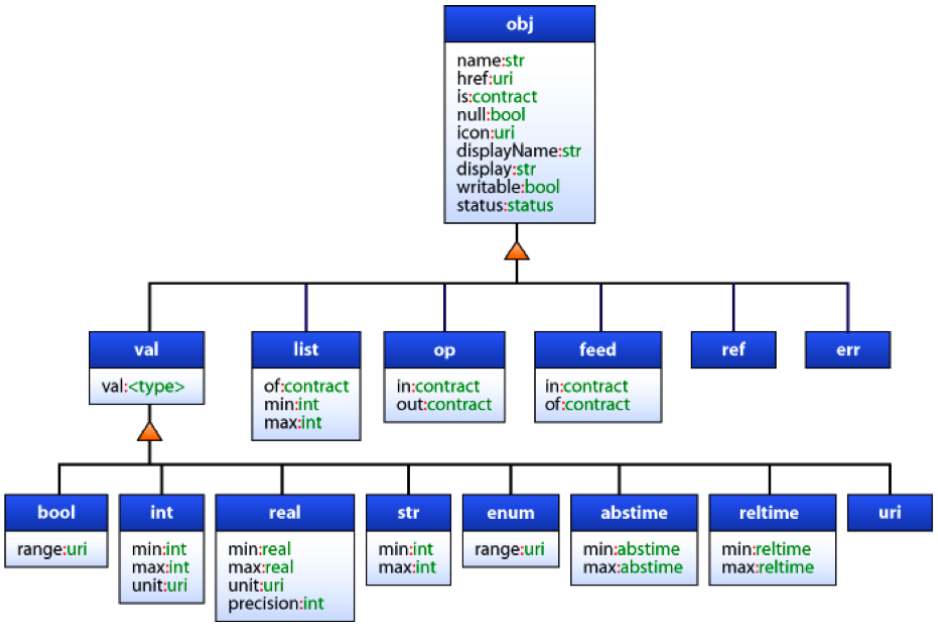

Abb. 3: oBIX object model (Quelle: [16])

Die Attribute des obj Elements sind:

name : definiert den "Zweck" des Objektes im Elternobjekt, also eine semantische Benennung [9]

href : enthält die URI, um das Objekt zu identifizieren

is : definiert die Contracts, die dieses Objekt implementiert

null : unterstützt null - Objekte

facets : sind eine Menge von Attributen, um Metadaten über das Objekt herstellen zu können

val : dieses Attribut wird nur bei Werteobjekten (bool, int, real, str, enum, abstime, reltime und uri) verwendet, um den aktuellen Wert zu speichern

Auszugsweise folgen Contract Definitionen (in XML) einiger Objekte.

- obj

```
<obj href=''obix:obj'' null=''false'' writeable=''false'' status=''ok'' />
```

- bool

```
<bool href=''obix:bool'' is=''obix:obj'' val=''false'' null=''false'' />
```

- str
  Mit dem str Typ werden Strings aus Unicode Zeichen dargestellt. Das val Attribut wird als xs:string abgebildet und enthält als Default den leeren String.

```
<str href=''obix:str'' is=''obix:obj'' val='''' null=''false'' />
```

  Ein Beispiel eines str Objektes:

```
<str val=''hello world'' />
```

- enum

```
<enum href=''obix:enum'' is=''obix:obj'' val='''' null=''true'' />
```

  Ein Beispiel eines enum Objektes:

```
<enum range=''/enums/OffSlowFast'' val=''slow''/>
```

- list

```
<list href=''obix:list'' is=''obix:obj'' of=''obix:obj'' />
```

[9] Die Benennung eines Objektes sollte semantisch geschehen. Beispielsweise sollte ein Alarmobjekt als Alarm bezeichnet werden und nicht als XYZ oder "Schalter".

Ein Beispiel eines list Objektes:

```
<list of=''obix:str''/>
   <str val=''one'' />
   <str val=''two'' />
</list>
```

- op
  Das op Objekt wird zur Definition einer Operation verwendet. Es hat Objekte als Eingangs- und Ausgangsparameter. Diese Eingangs- und Ausgangscontracts werden über die `in` und `out` Attribute definiert:

```
<list href=''obix:op'' is=''obix:obj'' in=''obix:Nil'' out=''obix:Nil'' />
```

- feed
  Feed Objekte werden verwendet, um einen Zulauf von Events definieren zu können. Sie werden in Watches verwendet, um die kommenden Events, wie zum Beispiel Alarme, handhaben zu können. Das `in` Attribute kann beispielsweise für Filter verwendet werden, durch welchen die Eingabewerte gereicht werden.

```
<feed href=''obix:feed'' is=''obix:obj'' in=''obix:Nil'' of=''obix:obj'' />
```

- facet
  Facets bieten zusätzliche Metadaten über Objekte. Die Menge der facets ist:
  - displayName: für Menschen lesbarer Name des Objektes, z.B. name="spaceTemp" displayName="Space Temperatur"
  - display: für Menschen lesbare Beschreibung des Objektes, vergleichbar mit Object.toString() in Java oder C#
  - icon: gibt eine URI zu einer Grafik an, welche zur Darstellung des Objektes im Benutzerbereich verwendet werden kann
  - min: spezifiziert einen eingeschlossenen minimalen Wert
  - max: spezifiziert einen eingeschlossenen maximalen Wert
  - precision: gibt die Anzahl der Dezimalstellen vor dem Komma einer Gleitkommazahl an
  - range: gibt den Wertebereich einer Enumeration an, wird in Zusammenhang mit bool und enum Typen verwendet, z.B. range="/enums/OffSlowFast
  - unit: definiert die physikalische Einheit des Objektes
  - writeable: gibt an, ob ein Objekt vom Client überschreibbar ist
  - status

Der Facet `status` wird verwendet, um die Aussagen über die Qualität und den Zustand der Information zu treffen. Beispiel:

```
<real val=''35.2'' status=''alarm'' />
```

Für `status` kann aus folgenden string Werten gewählt werden:

- disabled: zeigt auf, dass dieses Objekt "out of service" ist, also für die normale Operation "abgeschaltet" ist. Sind die Objekte Operationen oder Feeds, so wird mit Hilfe dieses States die Unterstützung für Operation oder Feed abgeschaltet.
- fault: gibt an, dass die Daten (in Bezug auf die Fehlerbedingung) ungültig oder nicht verfügbar sind. Zum Beispiel Daten, die abgelaufen sind, Konfigurationsprobleme, Software Fehler oder Hardware Fehler.
- down: bezieht sich auf einen Fehler in der Kommunikation.
- unackedAlarm: es existiert ein Alarm, der noch nicht vom vom User quittiert (ack) worden ist. Dieser Zustand stellt eine Kombination aus `alarm` und `unacked` states dar. Im Unterschied dazu impliziert Alarm, dass der Benutzer den Alarm bereits quittiert hat oder dass kein menschliches Acknowledgement benötigt wird. Der Unterschied zwischen `unackedAlarm` und `unacked` ist, dass das Objekt bei `unacked` zum normalen Zustand zurückgesprungen ist.
- alarm: gibt an, dass das Objekt derzeit im Alarmzustand ist. D.h., das Objekt arbeitet außerhalb der normalen Bedingungen, also in unerwünschten Zuständen.
- unacked: zeigt auf, dass eine vergangene Alarmbedingung noch nicht quittiert wurde.
- overridden: bedeutet, dass die Daten in Ordnung sind, aber aktuell eine lokale Überschreibung passiert. Zum Beispiel wird ein Sollwert temporär von seinem geplanten Sollwert überschrieben.
- ok: gibt den normalen Zustand an. Dieser ist angenommer Default Wert für alle Objekte.

4.1.1 Naming

In oBIX gibt es zwei Bezeichner: name und href. Ein Objekt kann entweder nur einen name oder href, keines oder beide haben. Es ist gebräulich für Objekte innerhalb einer Liste keine names zu verwenden, da die meisten Listen unbenannte (unnamend) Sequenzen von Objekten sind. Man soll von keiner Beziehung zwischen names und hrefs ausgehen, in der oBIX Spezifikation wird eine klare Trennung eingeführt [13]. Aus praktischer Sicht möchten viele Hersteller eine href Struktur aufbauen, welche die name Struktur abbildet, jedoch sollte bei Client Software von keinem Zusammenhang zwischen name und href ausgegangen werden.

name
Name wird verwendet, um die Rolle eines Objektes innerhalb seines Elternobjektes zu definieren und wird durch das `name` Attribut dargestellt. Der primäre Zweck eines names ist es dem Subobjekt semantische Bedeutung zuzuweisen. Sie werden auch dazu verwendet, um das Überschreiben von Contracts anzuzeigen. Eine Analogie zu names sind die field/method names einer Klasse in Java oder C# [16].

href
Mit dem `href` Attribut werden Objekten URIs zugewiesen. Mit Ausnahme vom Wurzelobjekt, für dieses muss die URI absolut sein, sind auch relative URIs zum Referenzieren möglich. Die absolute URI des Wurzelobjekts kann als Basis URI zum Auflösen einer relativen URI innerhalb eines Dokumentes herangezogen werden.
Im Allgemeinen muss jedes Objekt, auf das mittels Lesezugriff read zugegriffen werden kann, eine URI spezifizieren. Jedes oBIX Dokument, das von einer read Anfrage zurückgeben wird, muss eine Wurzel-URI spezifizieren. Ist das Objekt transient, wie beispielsweise ein Objekt das sich nach einem invoke einer Operation ergibt, gibt es keine Wurzel-URI. D.h. es gibt keine Möglichkeit dieses Objekt später nochmals zu erhalten. Wird keine Wurzel-URI angegeben, so wird die URI des Servers (server's authority URI) herangezogen, die als Basis für die URI Auflösung der relativen URI gilt.
Das URI Schema kann frei gewählt werden, es wird jedoch empfohlen, das HTTP URI Schema anzuwenden, weil es eine klare Definition der Normalisierungssemantik (normalization semantics) bietet. Die Regeln sind:

- Startet die URI mit "scheme:", so ist es eine global absolute URI
- Startet sie mit einem Slash, dann ist sie eine Server absolute URI
- Startet sie mit einem "#", ist es ein sogenannter "fragment identifier"
- Startet sie mit "../", so muss der Pfad mit der Basis zusammen gestellt werden (http://server/a/b/ + ../c = http://server/a/c)

Der beste Ansatz ist es, wenn die URI mit einem Slash endet. In der Praxis sollten Systeme, die in hierarchischen URIs organisiert sind, die Basis URI mit einem "trailing slash" enden.
Um Objekte intern in einem oBIX Dokument zu referenzieren, werden Fragment URIs (die mit "#" starten) verwendet. Beispiel:

```
<obj href=''http://server/whatever/''>
   <enum name=''switch1'' range=''#onOff'' val=''on'' />
   <enum name=''switch2'' range=''#onOff'' val=''off'' />
   <list is=''obix:Range'' href=''onOff''>
      <obj name=''on'' />
      <obj name=''off' />
   </list>
</obj>
```

In diesem Beispiel referenzieren zwei Objekte mit einer `range` Facet auf eine Fragment URI. Das "#" deutet an, dass sich diese Referenz auf ein Objekt innerhalb des selben oBIX Dokumentes bezieht. Das Objekt mit der href "onOff" is das "Ziel" der Fragment URI. Es hat die absolute URI "http://server/whatever/onOff".

4.1.2 Contracts

Ein Contract ist grundsätzlich ein normales oBIX Objekt, das aber als Art "Template Objekt" für andere Objekte verwendet werden kann. Die Umsetzung geschieht mittels `is` Attribut. Mit Contracts kann durch die Definition eines "types" konsistente Semantik innerhalb von Implementationen eines Herstellers geschaffen werden. Beispielsweise wird für alle Alarme derselbe `Alarm` Contract verwendet. Somit kann die Client Software auf normalisierte Alarminformationen eines Systems zugreifen. Weiters bieten Contracts einen einfachen Weg, Objekten Defaultwerte zuzuweisen. Die dritte Problemlösung bieten Contracts für "Type Export". Contracts können einfach in Sprachen wie Java oder C#, welche häufig von Hersteller für Client-Software verwendet werden, exportiert werden. Konzeptuell bieten Contracts ein elegantes Model, um verschiedene Probleme mit einer Abstraktion zu lösen. Aus Sicht der Spezifikation werden neue Abstraktionen mit der oBIX XML Syntax definiert. Aus Sicht der Implementierung können die Ideen der objektorientierten Programmierung mit den Klassen und Instanzen angewendet werden.

Terminologie

- Contract: eine wieder verwendbare Objektdefinition, welche durch ein oBIX XML Dokument ausgedrückt wird
- Contract list: stellt eine Liste von einem oder mehreren URIs zu Contract Objekten dar. Diese Liste wird für die Attribute `is` , `of` , `in` und `out` verwendet. Die Liste sind durch Leerzeichen getrennte URIs.
- Implements: ein Objekt implementiert ein Contract, wenn der Contract vom Objekt in seiner Contract Liste spezifiziert ist
- Implementation: ein Objekt, das ein Contract implementiert, ist eine Implementierung des Contracts

4.1.3 Contract Inheritance

Aufgrund der Contracts kann die klassische "is a" Beziehung erstellt werden. Abstrakt gesehen kann durch einen Contract ein Typ vererbt werden. Es wird unterschieden zwischen einer expliziten und impliziten Contract Implementierung.

- explizit: definiert eine Objektstruktur, mit der alle Implementationen übereinstimmen müssen

- implizit: definiert Semantiken, die mit dem Contract assoziiert sind

Implizierte Implementierung heißt, dass das Objekt dieselbe Semantik hat als der Contract, den es implementiert. Implementiert bspw. ein Objekt den Contract `Alarm` , so hat man unmittelbar das Kindobjekt `timestamp` . Diese Struktur ist im expliziten Contract von `Alarm` und ist definiert in XML. Zusätzlich verbinden wir dieses Objekt mit einer semantischen Bedeutung, nämlich dass das Objekt Informationen über ein Alarmereignis enthält, also eine implizite Implementierung. Wird ein Contract implementiert, so soll es immer implizit und explizit geschehen.
Die Kindobjekte eines Contracts werden automatisch an die Implementierung weitergeben. Die Implementierung kann das Kindobjekt entweder überschreiben (*override*) oder als *default* verwenden.
Beispiel für einen Contract:

```
<obj href=''/def/television/''>
   <bool name=''power'' val=''false'' />
   <int name=''channel'' val=''2'' min=''2'' max=''200''/>
</obj>
```

Die Implementierung dieses Contracts:

```
<obj href=''/livingRoom/tv/'' is=''/def/television''>
   <int name=''channel'' val=''8'' />
   <int name=''volume''  val=''22''/>
</obj>
```

Das Livingroom-TV Objekt wird mittels `is=''/def/television''` instanziert. In diesem Objekt wird Defaultwert 2 von channel mit 8 überschrieben, power wird mit Defaultwert false übernommen. Die Facets `min` und `max` von channel werden auch vererbt. Das Objekt volume wird neu hinzugefügt, da es noch nicht im Contract vorkommt. Möchte man also ein Kindobjekt in der Implementierung überschreiben, so geschieht das über das `name` Attribut.
Die Attribute `val` , `of` , `in` und `out` inklusive ihrer Facets werden vom Contract vererbt. Das `href` wird nie vererbt.

Contract Override Rules
Werden die Werte, Facets oder Contract lists überschrieben, so spricht man von "overriding the explicit contract". Beim Überschreiben können keine inkompatiblen Werttypen verwendet werden. Spezifiziert der Contract das Kind mit `real` , so müssen auch alle Implementierungen `real` sein. Ein Spezialfall stellt `obj` dar, dass zu irgendeinem Typen beschränkt werden kann. Wenn man Attribute eines Contracts überschreibt, muss man darauf achten, dass man keine Beschränkungen verletzt. Das bedeutet, man kann den Wertebereich eines Contracts nur spezialisieren (*to spezialize*) oder schmälern (*to narrow*), aber niemals verallgemeinern (*to generalize*) oder erweitern (*to widen*). Dieses Konzept wird als *covariance* bezeichnet.

Beispiel:

```
<int name=''channel'' val=''8'' min=''2'' max=''200'' />
```

Der Contract definiert einen Wertebereich von 2 bis 200. Würde man `min` mit -100 überschreiben, so würde das den Wertebereich erweitern, also die Begrenzung verletzen. Man kann den Wertebereich jedoch schmälern, indem man für `min` einen Wert größer 2 oder für `max` einen Wert kleiner 200 angibt.

Multiple Inheritance
Von einem Objekt können auch mehrere Contracts durch Angabe ihrer URIs in der Contract list implementiert werden. Mit der Mehrfachvererbung hängen die zwei Punkte *flattening* und *mixins* zusammen.

1. Flattening
   Bei einer Verkettung der Vererbungshierachie kann es zu schlechter Performance bei Netzwerkanfragen kommen, wenn die Objekte über ihre Contracts "lernen" müssen. Aus diesem Grund soll der Server seine Contract Vererbungshierarchie bei Spezifizierung von `is` , `of` , `in` und `out` Attribute zu einer Liste abflachen (*to flatten*).

   Bei folgender Implementierung müssen drei weitere Anfragen (eine für C, B und A) gestellt werden, um völlig über das Objekt D zu lernen.

   ```
   <obj href=''A'' />
   <obj href=''B'' is=''/A'' />
   <obj href=''C'' is=''/B'' />
   <obj href=''D'' is=''/C'' />
   ```

   Abgeflacht sieht die Hierachie folgendermaßen aus:

   ```
   <obj href=''A'' />
   <obj href=''B'' is=''/A'' />
   <obj href=''C'' is=''/B /A'' />
   <obj href=''D'' is=''/C /B /A'' />
   ```

   Bei dieser Implementierung erkennt der Client mit einer einzigen Anfrage sofort, dass C, B und A von D implementiert werden.

2. Mixins

   Ein Objekt kann auch von mehreren Contracts erben.
   Ein Beispiel:

   ```
   <obj href=''acme:Device''>
      <str name=''serialNo'' />
   </obj>

   <obj href=''acme:Clock'' is=''acme:Device''>
      <op name=''snooze'' >
   ```

```
   <int name=''volume'' val=''0'' >
</obj>

<obj href=''acme:Radio'' is=''acme:Device''>
   <real name=''station'' min=''87.0'' max=''107.5'' >
   <int name=''volume'' val=''5'' >
</obj>

<obj href=''acme:ClockRadio'' is=''acme:Radio acme:Clock acme:Device'' />
```

Via *flattening* von `Clock` und `Radio`, implementiert `ClockRadio` auch `Device`. In oBIX wird das als *mixin* bezeichnet. `ClockRadio` erbt die vier Kindobjekte: `serialNo`, `snooze`, `volume` und `station`.

Contract Lists und Feeds
Werden Implementationen von `list` oder `feed` Contracts abgeleitet, so erbt das Objekt das `of` Attribut. Man kann das `of` Attribut überschreiben. Jedoch nur Contract kompatibel, d.h. alle URIs des `of` Attributes müssen übernommen werden. Wenn benötigt können zusätzliche hinzugefügt werden.

```
<list href=''/def/list'' of=''obix:str''/>
   <str val=''one'' />
   <str val=''two'' />
</list>

<list href=''/mylist'' is=''/def/list'' of=''obix:str obix:int''>
   <int val=''3'' />
</list>
```

4.1.4 Object Composition

Im übertragenen Sinne kann oBIX mit dem World Wide Web verglichen werden. Einfach gesehen basiert das WWW auf einem Web (Gespinst) aus HTML Dokumenten, die mit URIs miteinander verlinkt sind. Was das WWW für die HTML Dokumente "erledigt", macht oBIX für die Objekte. Das logische Model von oBIX ist ein globales Netz von oBIX Objekten, die über URIs miteinander verlinkt sind. Objekte können statische Dokumente wie Contracts oder Gerätebeschreibungen sein, andere Objekte zeigen Echtzeitdaten oder Services. Egal welche Objekte, alle sind miteinander verlinkt, um zum "oBIX Web" zu werden. Individuelle Objekte können via "containment" oder via "reference" aus anderen Objekten zusammengesetzt werden.

Containment
Containment wird in XML-Syntax durch Verschachtelung der XML-Elemente repräsentiert. Jedes Objekt kann kein oder mehrere Kindobjekte enthalten. Diese Kindobjekte können auch primitive Objekte wie `bool` oder `int` sein. Kindobjekte können benannt oder unbenannt sein, je nachdem ist das `name` Attribut spezifiziert.

Im nachfolgenden Beispiel bindet das Objekt "/a" das Objekt "/a/b" ein, das wiederum "/a/b/c" beinhaltet. Im Beispiel ist "/a/b" Objekt benannt und "/a/b/c" unbenannt. Gebräuchlicherweise werden benannte Kindobjekte oft für Felder in Records, Strukturen oder Klassentypen verwendet und Unbenannte in Listen.

```
<obj href=''/a/''>
   <list name=''b'' href=''b''/>
      <obj href=''b/c''/>
   </list>
</obj>
```

References
Eine oBIX Referenz ist so ähnlich wie ein HTML Anker. Anker werden innerhalb einer HTML Datei verwendet, um Passagen zu verlinken. Eine oBIX Referenz dient als Platzhalter, um zu einem anderem oBIX Objekt via URI zu verlinken.

Containment wird am besten für kleinere Datenbäume verwendet, Referenzen hingegen sollten eher für größere Bäume oder Graphen von Objekten herangezogen werden.

4.1.5 Operations

Operationen kann man auf oBIX Objekte anwenden. Sie sind vergleichbar mit den Methoden in der objektorientierten Programmierung. Im Gegensatz zu einem Werteobjekt, welches das Objekt und den aktuellen Zustand darstellt, definiert das `op` Element eine Operation, die aufgerufen werden kann.

Ein Beispiel einer Operation:

```
<op href=''/addTwoReals'' in=''/def/AddIn'' out=''obix:real''/>

<obj href=''/def/AddIn''>
   <real name=''a''/>
   <real name=''b''/>
</obj>
```

Die `in` und `out` Attribute definieren die Contracts für die Eingabe- und Ausgabeobjekte. Benötigt man mehrere Eingabe- oder Ausgabeparameter, so kann man diese einfach in einem Objekt zusammenfassen. Für Operationen ohne Parameter kann man einfach `in` als `obix:Nil` definieren. Hat die Operation keinen Rückgabeparameter, so definiert man `out` als `obix:Nil` . Operationen können über ihr `href` Attribut aktiviert werden.

4.2 XML

Dieses Kapitel beschreibt, wie das Object Model von oBIX in XML dargestellt wird. XML stellt das kleine, fixe Schema für alle oBIX Dokumente dar. Wenn ein Client die XML Syntax versteht, kann er garantiert auf den Objektbaum des Servers zugreifen unabhängig davon, ob es vordefinierte oder keine vordefinierten Contracts sind. Die höherliegende semantische Bedeutung ist über die Contracts festgelegt, Contracts beeinflussen die XML Syntax nicht.

Contract XML Syntax
Das abstrakte Object Model wird sehr genau auf die oBIX XML Syntax abgebildet. Zusammengefasst ergibt sich:

- Jedes oBIX Objekt wird auf genau ein XML Element gemapped (abgebildet).
- Kindobjekte werden auf XML Kindelemente abgebildet.
- Der XML Elementname stellt den built-in Objekttyp dar.
- Alles andere eines Objektes wird als XML Attribut dargestellt.

Contract XML Encoding
Das Encoding von oBIX Dokumenten basiert auf folgenden Regeln:

- oBIX Dokumente müssen wohlgeformtes XML sein.
- oBIX Dokumente sollen mit einer XML Deklaration beginnen, welche das Encoding spezifiziert.
- Die Verwendung von UTF-8 encoding ohne Byte Order Mark wird empfohlen.
- oBIX Dokumente dürfen kein Document Type Declaration enthalten - sie können kein internes oder externes Subset enthalten.
- oBIX Dokumente sollen einen XML Namespace enthalten.
  Grundsatz ist "http://obix.org/ns/schema/1.0" als Default-Namespace zu deklarieren. Falls oBIX in einem anderen Typ von XML Dokument eingebettet ist, dann soll "o" als Prefix für den Namespace verwendet werden. Der Prefix "obix" sollte nicht verwendet werden.

Contract XML Decoding
Folgende Regeln gelten für das Decoding von oBIX Dokumenten:

- Eine Konformität zu XML Processing (Verarbeitung), wie in XML1.1 definiert, muss vorhanden sein.
- Dokumente, die nicht wohlgeformt in XML sind, müssen abgelehnt (rejected) werden.

- Es werden keine Parser benötigt, um Document Type Declaration zu verstehen.
- Unbekannte Elemente müssen ignoriert werden ohne Rücksicht auf deren XML Namespace.
- Unbekannte Attribute müssen ignoriert werden ohne Bezug auf deren XML Namespace.

Als Daumenregel gilt: "strict in what you generate, and liberal in what you accept". oBIX Parser sollen Elemente und Attribute, die sie nicht verstehen, ignorieren. Sie sollen nie XML Dokumente akzeptieren, die nicht wohlgeformt sind, wie zum Beispiel Dokumente mit mismatched tags.

Contract XML Namespace
XML namespaces innerhalb von oBIX sollten zu folgendem Muster konform sein:

```
http://obix.org/ns/{spec}{version}
```

Für oBIX Version 1.0 gilt die Namespace Definition:

```
http://obix.org/ns/schema/1.0
```

Contract Namespaces Prefixes in Contracts
Die XML Namespace Präfixe innerhalb eines oBIX Dokuments können als Präfix von URIs eines Contracts verwendet werden. Beginnt eine URI (innerhalb einer Contract List) mit einem definiertem Präfix gefolgt von einem Doppelpunkt ":", so wird der Präfix der URI mit dem Namespace Wert ersetzt. Der XML Namespace Präfix "obix" ist vordefiniert. Dieser wird für alle in oBIX definierten Contracts verwendet. Der Präfix "obix" wird zu "http://obix.org/def/".
Ein Beispiel eines oBIX Dokuments, wo die XML Namespace Präfixe aufgelöst/normalisiert sind:

```
<obj xmlns:acme=''http://acme.com/def/''  is=''acme:CustomPoint  obix:Point''/>
<obj is=''http://acme.com/def/CustomPoint''  http://obix.org/def/Point''/>
```

4.3 Protocol Binding

Das "Herz" von oBIX ist das Object Model und die damit verbundene XML Syntax. Die primäre Verwendung von oBIX ist der Zugriff auf Daten und Services über ein Netzwerk. Grundsätzlich basiert oBIX auf einem Client/Server-Modell. Die Server sind Software, die oBIX freigegebene Daten und Services zur Verfügung stellen. Sie antworten auf Anfragen von Clienten über das Netzwerk. Ein Client ist eine Software, welche Anfragen an Server über das Netzwerk stellt, um Zugriff auf oBIX Daten und Services zu erhalten.
Es gibt die drei Netzwerkanfragen:

- **read**: Rückgabe des aktuellen Objektzustandes einer gegebenen URI als oBIX XML Dokument. Ist der Lesevorgang nicht erfolgreich, so kann der Server ein `err` Objekt zurückgeben.
- **write**: Aktualisierung des Objektzustandes einer gegebenen URI. Der zu schreibende Zustand wird als oBIX XML Dokument über das Netzwerk übergeben. Das aktualisierte Objekt wird wieder über das Netzwerk zurückgegeben. Bei fehlerhaftem Schreibvorgang soll der Server ein `err` Objekt zurückgeben.
- **invoke**: Aktivieren einer Operation die über eine URI (eines `op` Objekts) identifiziert ist. Eingabe- und Ausgabeparameter werden als oBIX XML Dokumente über das Netzwerk geschickt.

Wie diese drei Anfragen/Antworten (Requests/Responses) zwischen Client und Server über das Netzwerk ablaufen wird als **protocol binding** bezeichnet. In oBIX gibt es zwei protocol bindings: das HTTP Bindung und das SOAP Binding.

HTTP Binding
Das HTTP binding bildet das einfache REST-Modell von oBIX Anfragen auf HTTP ab. Zusammengefasst ergibt sich Tabelle 1.

oBIX Request	**HTTP Methode**	**Target (Ziel)**
Read	GET	Objekt mit href
Write	PUT	Objekt mit href und schreibbar=true
Invoke	POST	operation Objekt

Tabelle 1: Abbildung REST-Modell auf oBIX Anfragen

Als MIME- (Multipurpose Internet Mail Extensions) oder auch Content-Typ soll "text/xml" für die Übertragung von oBIX Dokumenten zwischen Client und Server in den HTTP Headers festgelegt werden. Um Authentifizierungs- und Kodierungsservices anzubieten, können die existierenden Standards wie "HTTP Authentication: Basic and Digest Access Authentication", HTTPS und das TLS Protokoll inkludiert werden.

SOAP Binding
Das SOAP binding bildet eine SOAP Operation auf jede der drei oBIX Anfragen ab: read, write und invoke. Im Gegensatz zum HTTP binding wird auf die Anfragen nicht über die URI vom Zielobjekt zugegriffen, sondern über die URI vom SOAP Server, welcher die URI des Objekts im Body des SOAP envelope kodiert hat.

Beispiel einer SOAP Anfrage nach einem `About` Objekt eines oBIX Servers:

```
<env: Envelope xmlns:env=''http://schema.xmlsoap.org/soap/envelope/''>
 <env:Body>
   <read xmlns=''http://obix.org/ns/wsdl/1.0'' href=''http://localhost/obix/about />
 </env:Body>
</env:Envelope>
```

Beispiel Antwort zur obigen SOAP Anfrage:

```
<env: Envelope xmlns:env=''http://schema.xmlsoap.org/soap/envelope/''>
  <env:Body>
    <obj name=''about''
            href=''http//localhost/obix/about/''
            xmlns=''http://obix.org/ns/schema/1.0''>
      <str name=''obixVersion'' val=''1.0'' />
      <str name=''serverName'' val=''obix'' />
      <abstime name=''serverTime'' val=''2010-02-08T09:40:55.000+05:00'' />
      ...
      <uri name=''productUrl'' val=''http://www.acme.com/obix'' />
    </obj>
  </env:Body>
</env:Envelope>
```

Die oBIX Spezifikation definiert keine SOAP Fehler. Ein Fehler soll über das `err` Objekt angezeigt werden.

4.4 Core Contract Library

Dieser Abschnitt beschreibt einige fundamentale Contracts die als "building blocks" der oBIX Spezifikation dienen.

- Nil: definiert ein null Objekt. Nil wird meistens dafür verwendet, um bei fehlenden Ein- und Ausgaben die `in` und `out` Attribute einer Operation zu setzen.
- Range: wird verwendet um einen `bool` - oder `enum` Bereich zu definieren. `Range` ist ein Listenobjekt mit null oder mehreren Objekten die als Range Items bezeichnet werden. Definition von `Range` :

  ```
  <list href=''obix:Range'' of=''obix:obj'' />
  ```

  Beispiel: definieren eines Strings für die `bool` Werte:

  ```
  <list href=''/enums/OnOff'' is=''obix:Range'' >
    <obj name=''true'' displayName=''On'' />
    <obj name=''false'' displayName=''Off'' />
  </list>
  ```

- Weekday: ist eine standardisierte Enumeration für die Wochentage. Der Contract sieht folgendermaßen aus:

```
<enum href=''obix:Weekday'' is=''obix:Range'' >
   <list href=''#Range'' is=''obix:Range''>
      <obj name=''sunday'' />
      <obj name=''monday'' />
      <obj name=''tuesday'' />
      <obj name=''wednesday'' />
      <obj name=''thursday'' />
      <obj name=''friday'' />
      <obj name=''saturday'' />
   </list>
</enum>
```

- Month: ist eine standardisierte Enumeration für die Monate
- Units: oBIX bietet einen Framework, um Einheiten mathematisch definieren zu können. Es sind auch vordefinierte Einheiten vorhanden. Die meisten Dimensionen können aus den sieben SI-Einheiten (meter, kilogram, second, Kelvin, ampere, mole und candela) hergeleitet werden.

  Der Contract `obix:Dimension` definiert den Anteil einer SI Einheit mit Hilfe eines Exponenten:

```
<obj href=''Dimension'' >
   <int name=''kg'' val=''0'' />
   <int name=''m'' val=''0'' />
   <int name=''sec'' val=''0'' />
   <int name=''K'' val=''0'' />
   <int name=''A'' val=''0'' />
   <int name=''mol'' val=''0'' />
   <int name=''cd'' val=''0'' />
</obj>
```

  Möchte man beispielsweise die Beschleunigung in $\frac{m}{s^2}$ darstellen, so sieht das Objekt so aus:

```
<obj is=''Dimension'' >
   <int name=''m'' val=''1'' />
   <int name=''sec'' val=''-2'' />
</obj>
```

  Der `obix:Unit` Contract ist wie folgt festgelegt:

```
<obj href=''obix:Unit'' >
   <str name=''symbol'' />
   <obj name=''dimension'' is=''obix:Dimension'' />
   <real name=''scale'' val=''1'' />
   <real name=''offset'' val=''0'' />
</obj>
```

  `Symbol` gibt eine Abkürzung für die Einheit an, z.B. "$^\circ F$" für Grade in Fahrenheit. `dimension` gibt die Dimension des Wertes (der Messung) als Verhältnis der SI Einheiten an. Das `scale` Element gibt die Skalierungsvariable für die "toNormal" Gleichung an. Defaultwert von `scale` ist 1. `Offset` definiert die offset Variable der "toNormal" Gleichung, Defaultwert ist 0.

Die "toNormal" Gleichung `scalar * scale + offset` wird verwendet um den Wert in eine Einheit basierend auf SI Einheiten umzuwandeln. Um zum Beispiel die Energie in Wattstunden = 3600 Joule nach Joule = $m^2 * kg * s^{-2}$ zu wandeln.

4.5 Standard Contracts

4.5.1 oBIX Server Contracts (Lobby, About, Batch, Error)

Lobby
Alle oBIX Server müssen ein Objekt anbieten das `obix:Lobby` implementiert. Das `Lobby` Objekt stellt einen zentralen Einstiegspunkt "in" den oBIX Server dar und listet URIs von bekannten Objekten der oBIX Spezifikation auf. Nach Konvention ist die URI der `Lobby` Instanz "http://server/obix", aber Hersteller können auch eine andere wählen.
Der Contract der `Lobby` ist folgendermaßen definiert:

```
<obj href=''obix:Lobby'' >
   <ref name=''about'' is=''obix:About'' />
   <op name=''batch'' in=''obix:BatchIn'' out=''obix:BatchOut'' />
   <ref name=''watchService'' is=''obix:WatchService'' />
</obj>
```

Mit der `Lobby` Instanz sollten die Anbieter Hersteller-spezifische Objekte für Daten und Service Discovery bereitstellen.

About
Im `obix:About` Objekt werden Informationen über einen oBIX Server zusammengefasst. Clients können die URI direkt vom `Lobby` Objekt ableiten.
Der Contract sieht folgendermaßen aus:

```
<obj href=''obix:About'' >
   <str name=''obixVersion'' />

   <str name=''obixName'' />
   <abstime name=''serverTime'' />
   <abstime name=''serverBootTime'' />

   <str name=''vendorName'' />
   <uri name=''vendorUrl'' />

   <str name=''productName'' />
   <str name=''productVersion'' />
   <str name=''productUrl'' />
</obj>
```

obixVersion gibt an, welche Version der oBIX Spezifikation der Server implementiert. serverName, serverTime und serverBootTime bieten Informationen über den Server. vendorName und vendorUrl stellen die Daten über den Softwarehersteller des Servers dar.

productName, productUrl und productVersion bieten Informationen über die Software des Servers.

Batch
Das `Lobby` Objekt definiert eine `batch` Operation, mit der man mehrere Netzwerkanfragen zu einer einfachen Operation stapeln kann. Dadurch wird die Performance erhöht. Eine Batch-Anfrage ist eine Aggregation von Read-, Write- und Invoke-Anfragen implementiert als Standard oBIX Operation. Auf der Schicht des procotol binding ist es dann nur als einzige Invoke-Anfrage unter Verwendung der `Lobby.batch` URI. Die Abarbeitung eines Batch Objekts ist semantisch äquivalent zur Abarbeitung der einzelnen Requests nacheinander.
Das Eingabeobjekt `BatchIn` und Ausgabeobjekt `BatchOut` der Operation sind:

```
<list href=''obix:BatchIn'' of=''obix:uri'' />
<list href=''obix:BatchOut'' of=''obix:obj'' />
```

Der `BatchIn` Contract spezifiziert eine Liste von Anfragen zur Verarbeitung identifiziert durch den `read` , `write` und `invoke` Contract:

```
<uri href=''obix:Read'' />

<uri href=''obix:Write''>
   <obj name=''in'' />
</uri>

<uri href=''obix:Invoke''>
   <obj name=''in'' />
</uri>
```

Der `BatchOut` Contract spezifiziert eine geordnete Liste von Antwort Objekten passend zu jeder Anfrage. Das erste Objekt im `BatchOut` muss das Ergebnis der ersten Anfrage sein. Jede `uri` die über das `BatchIn` für einen Lese- oder Schreibvorgang übergeben wird, muss ein korrespondierendes Objekt im `BatchOut` mit einem `href` Attribut mit derselben String Darstellung wie im `BatchIn` haben.
Es folgt ein einfaches Beispiel in dem eine Leseanfrage für "/someStr" und "/invalidUri" und eine Schreibanfrage für "/someStr" spezifiziert wurde. Der zu schreibende Wert ist im Kindobjekt "in" angegeben.

```
<list is=''obix:BatchIn />
   <uri is=''obix:Read'' val=''/someStr'' />
   <uri is=''obix:Read'' val=''/invalidUri'' />
   <uri is=''obix:Write'' val=''/someStr'' />
      <str name=''in'' val=''new string value'' />
   </uri>
</list>
```

Der Server antwortet dann mit einem Objekt pro Anfrage.

```
<list is=''obix:BatchOut />
   <str href=''/someStr'' val=''old string value'' />
```

```
   <str href=''/invalidUri'' is=''obix:BadUriErr'' display=''href not found'' />
   <str href=''/someStr val=''new string value'' />
</list>
```

Die erste Leseanfrage bekommt ein `str` Objekt zurück, das den aktuellen Wert von "/someStr" enthält. Die zweite Leseanfrage enthält eine falsche URI, somit gibt der Server ein `err` Objekt zurück. Der Server macht mit den Anfragen weiter. Bei der Schreibanfrage überschreibt der Server den Wert von "someStr" und gibt den neuen Wert zurück. Da die Anfragen der Reihe nach abgearbeitet werden, ergibt sich: Die erste Anfrage gibt den Originalwert von "someStr" zurück und die dritte Anfrage beinhaltet den neuen Wert. Das selbe erhält man wenn die einzelnen Anfragen nacheinander durchgeführt werden.

Error Contracts
Der Server sollte immer ein `err` Objekt zurückliefern, falls die Anfrage des Clients nicht bearbeitet werden kann. Es gibt bereits vordefinierte Contracts für häufige Fehler:

- BadUriErr: zeigt eine falsche oder unbekannte URI an.
- UnsupportedErr: indiziert eine Anfrage welche nicht vom Server unterstützt wird. Beispielsweise das Aktivieren einer Operation die nicht vom Server unterstützt wird.
- PermissionErr: zeigt an, dass der Client wegen fehlender Sicherheitserlaubnis nicht auf die Daten zugreifen kann.

Die Contracts dieser Fehler sind folgendermaßen definiert:

```
<err href=''obix:BadUriErr'' />
<err href=''obix:UnsupportedErr'' />
<err href=''obix:PermissionErr'' />
```

Ist so ein Contract für einen Fehlerfall sinngemäß, so kann er im `is` Attribut des `err` Elements inkludiert werden. Empfohlen wird auch eine brauchbare Beschreibung des Problems im `display` Attribut einzutragen.

4.5.2 Watches

Damit ein Client auf sich schnell ändernde Daten Echtzeitzugriff hat, wird in oBIX das Konzept der Watches eingeführt.
Ein watch Zyklus ist wie folgt aufgebaut:

- Der Client erstellt mit der Operation `make` der `WatchService` URI des Servers ein neues `Watch` Objekt. Der Server erstellt ein neues `Watch` Objekt und bietet dem Client eine URI zum Zugriff.
- Der Client registriert (und meldet ab) Objekte, die beobachtet werden sollen, über die Operationen des `Watch` Objektes.

- Der Client kann nun die `Watch` URI mit der Operation `pollChanges` pollen. Er erhält somit die Events, die seit dem letzten Pollen aufgetreten sind.
- Der Server gibt das `Watch` Objekt aufgrund zweier Bedingungen frei. Entweder gibt der Client `Watch` mit der Opertaion `delete` frei, oder der Server löscht diese automatisch, weil der Client nicht innerhalb einer vordefinierten Zeit (lease time) gepollt hat.

Das `WatchService` Objekt ist direkt vom `Lobby` Objekt verfügbar. Der Contract sieht folgendermaßen aus:

```
<obj href=''obix:WatchService'' >
  <op name=''make'' in''obix:Nil'' out=''obix:Watch'' />
</obj>
```

Die `make` Operation liefert ein leeres `Watch` Objekt als Ausgabe. Der href dieses Objektes kann dann vom Client verwendet werden, um Operationen zu aktivieren. Das `Watch` Objekt wird verwendet, um eine Menge von Objekten die periodisch von Clients gepollt werden, um die letzten Events zu erhalten, zu managen.
Der Contract ist so aufgebaut:

```
<obj href=''obix:Watch'' >
  <realtime name=''lease'' min=''PTOS'' writeable=''true'' />
  <op name=''add'' in=''obix:WatchIn'' out=''obix:WatchOut'' />
  <op name=''remove'' in=''obix:WatchIn'' />
  <op name=''pollChanges'' out=''obix:WatchOut'' />
  <op name=''pollRefresh'' out=''obix:WatchOut'' />
  <op name=''delete'' />
</obj>

<obj href=''obix:WatchIn''>
   <list name=''hrefs'' of=''obix:WatchInItem'' />
</obj>

<uri href=''obix:WatchInItem''>
   <obj name=''in'' />
</uri>

<obj href=''obix:WatchOut''>
   <list name=''values'' of=''obix:obj'' />
</obj>
```

Das Objekt `WatchIn` besteht aus einer Liste von (meistens zum Server relativen) URIs, welche der Client zum Identifizieren von Objekten zum `add` und `remove` von der Poll-Liste verwendet.
Watch.add wird zum Hinzufügen von Objekten zur Watch verwendet. Diese Operation hat als Eingabe eine Liste von URIs (via `WatchIn`) und als Ausgabe die aktuellen Werte der Objekte referenziert. Wenn ein Objekt nicht verarbeitet werden kann, so wird das mit der Rückgabe eines `err` Objektes und der respektiven href verdeutlicht. Die Operation Watch.remove wird verwendet, um Objekte von der Watch List zu entfernen. Das Nil Objekt wird zurückgegeben. Mit der Operation Watch.pollChanges kann der Client den Server periodisch pollen. Diese Operation gibt eine Liste mit Objekten zurück, die sich

geändert haben. Server sollen nur die Objekte zurückgeben, die sich seit dem letzten Pollen für die Watch geändert haben. Watch.pollRefresh initiiert eine Aktualisierung aller Objekte in der Watch List. Der Server muss jedes Objekt zurückgeben. Dazu verwendet er den href mit exakt derselben String Darstellung, die der Client beim `add` mitgegeben hat.
Mit Watch.delete kann eine existierende Watch gelöscht werden.

4.5.3 Points

Es gibt mehrere Definitionen für Point. Generell gehören Points direkt zu einem Sensor oder Aktor, auch bezeichnet als hard points. Manchmal ist ein Point auch bezogen auf eine Konfigurationsvariable wie einem Software setpoint, genannt soft points. In manchen Systemen ist ein Point eine atomare Variable, in anderen wiederum eine Zusammenfassung von Status und Konfigurationsinformationen.
oBIX hat das Ziel eine normalisierte Darstellung für Points einzuführen. oBIX definiert für den Point eine Abstraktion, nämlich einen primitiven Datentyp und dazugehörige Statusinformation. Ein Point stellt grundsätzlich einen Marker Contract für ein Objekt dar, das eine "point" Semantik hat:

```
<obj href=''obix:Point'' />
```

Der Contract darf nur für die primitiven Werttypen `bool` , `real` , `enum` , `str` , `abstime` und `reltime` verwendet werden. Points verwenden das `status` Attribut (Abschnitt 4.1) um qualitative Information darzulegen. Tabelle 2 gibt an, wie Semantik zu Werttypen gemapped wird.

`bool`	digital point	< bool is="obix:Point" val="true" />
`real`	analog point	< real is="obix:Point" val="22" units="obix:units/celsius" />
`enum`	multi-state point	< enum is="obix:Point" val="slow" />

Tabelle 2: Mapping: Semantik zu Werttypen

Verschiedene Regelsysteme behandeln das Schreiben von Points auf unterschiedliche semantische Weise. Öfters wird ein Point ab einem spezifiziertem Prioritätslevel geschrieben. Manchmal wird ein Point für eine gewisse Zeitspanne überschrieben, dann fällt er wieder zurück auf den Defaultwert. oBIX bietet einen Standard `WriteablePoint` Contract, welcher mit verschiedenen Mixins erweitert werden kann um spezielle Fälle abzudecken. `WriteablePoint` definiert eine Operation, welche eine `WritePointIn` Struktur übernimmt die den neuen Wert beinhaltet.

Die Contracts sind:

```
<obj href=''obix:WriteablePoint'' is=''obix:Point'' >
   <op name=''writePoint'' in=''obix:WritePointIn'' out''obix:Point'' />
</obj>

<obj href=''obix:WritePointIn''>
   <obj name=''value'' />
</obj>
```

Der Wert der zu `writePoint` übergeben wird, muss zum Typ des Points passen.

4.5.4 History

Viele Automatisierungssysteme besitzen die Möglichkeit, periodisch Punktdaten zu messen, um eine Trendlinie (Geschichte) über die Werte zu erhalten. In oBIX kann dies mit Hilfe der History realisiert werden. Eine History stellt eine Liste von zeitgestempelten point values dar. Es gibt folgende Features:

- History Object: die normalisierte Darstellung der History selbst
- History Record: ein Record eines Point samplings zu einem bestimmten Zeitpunkt
- History Query: ein Standardweg, um historische Daten als Points abzufragen
- History Rollup: ein Mechanismus, um Rollups von historischen Daten durchzuführen

Das History Object ist wie folgt aufgebaut:

```
<obj href=''obix:History'' >
   <int          name=''count'' min=''0'' val=''0'' />
   <abstime name=''start'' null=''true'' />
   <abstime name=''end'' null=''true'' />
   <op         name=''query'' in=''obix:HistoryFilter'' out=''obix:HistoryQueryOut'' />
   <feed      name=''feed'  in=''obix:HistoryFilter'' of=''obix:HistoryRecord'' />
   <op         name=''rollup'' in=''obix:HistoryRollupIn'' out=''obix:HistoryRollupOut'' />
</obj>
```

Die Elemente bedeuten folgendes:

- count: gibt die Anzahl der history records in der History an
- start: gibt den Zeitstempel des ältesten Records an
- end: gibt den Zeitstempel des neuesten Records an
- query: wird verwendet, um History Records aus der History abzufragen
- feed: wird verwendet, um Echtzeit Feeds von History Records zu abonnieren
- roll: wird für History rollups verwendet

4.5.5 Alarming

Das oBIX Alarming spezifiziert ein Model, um Alarme abzufragen, zu beobachten oder zu bestätigen. Alarme sind Zustände, die einem Benutzer oder anderem Programm angezeigt werden sollten. Die Bestätigung eines Alarms besagt, dass jemand Aktionen durchführt / durchgeführt hat, um den Alarm zu beseitigen.
Typischerweise läuft ein Alarm so ab:

- **source monitoring**: Algorithmen auf einem Server, die eine Alarmquelle beobachten. Alarmquellen sind Objekte, die einen Alarm auslösen können. Beispielsweise Sensorpunkte (im Raum ist es zu heiß), Hardware Probleme (Hard Disk ist voll) oder Anwendungsprobleme (Gebäude verbraucht zuviel Energie).
- **alarm generation**: wenn der Algorithmus erkennt, dass ein Alarm ausgelöst wurde, wird ein *alarm* Record generiert. Jeder Alarm ist eindeutig durch eine href identifiziert und wird durch den Contract `obix:Alarm` dargestellt. Manchmal wird für den Übergang zum Alarm von *off-normal* gesprochen.
- **to normal**: die meisten Alarmquellen besitzen einen Zustand, sie sind *stateful*. Das Auftreten des Alarmzustandes wird als zurückgehen zu *to-normal* bezeichnet. Alarme, die stateful sind, implementieren den Contract `obix:StatefulAlarm` . Wenn die Alarmquelle in *to-normal* übergeht, wird die Zeit `normalTimestamp` aktualisiert.
- **acknowledgement**: manchmal ist es notwendig, dass Benutzer oder die Applikation die Behandlung eines Alarms bestätigt. Solche Alarme werden durch den Contract `obix:AckAlarm` implementiert. Wenn der Alarm bestätigt wird, wird der Zeitstempel `ackTimestamp` und das Element `ackUser` aktualisiert.

Alarm Zustände
Der Alarmzustand kann in zwei Variablen zusammengefasst werden:

- **in alarm**: gibt an, ob die Alarmquelle gerade in Alarmzustand oder im Normalzustand ist. Diese Variablen bilden sich auf den `alarm` Status Zustand (s. Abschnitt 4.1, Facet `status`) ab.
- **acknowledgement**: gibt an, ob der Alarm bestätigt oder unbestätigt ist. Diese Variable bildet sich auf `unacked` Status Zustand ab.

Diese Zustände sind unabhängig voneinander. Beispielsweise kann eine Alarmquelle in den Normalzustand übergehen, bevor oder nachdem der Alarm bestätigt wurde.
Anmerkung: nicht alle Alarme haben einen Zustand. Ein Alarm, der weder `StatefulAlarm` noch `AckAlarm` implementiert ist zustandslos. Diese Alarme stellen ein Ereignis dar.
Der aktuelle Alarmzustand einer Alarmquelle wird durch das `status` Attribut repräsentiert (siehe Abschnitt 4.1). Ein `Alarm` Record wird zur Zusammenfassung des Lebenszyklus eines Alarmes verwendet. Wenn der Alarm `StatefulAlarm` implementiert, kann der

Übergang von off-normal zu normal getrackt werden. Implementiert der Alarm `AckAlarm`, so beschreibt dies den Bestätigungsvorgang.
Tabelle 3 listet die vier diskreten Alarmzustände auf.

alarm	acked	normalTimestamp	ackTimestamp
true	false	null	null
true	true	null	non-null
false	false	non-null	null
false	true	non-null	non-null

Tabelle 3: Vier Alarmzustände

Aus dieser Tabelle kann man die Zustände der `normalTimestamp` und `ackTimestamp` ablesen.

Alarm Contracts

- `Alarm` contract:

```
<obj href=''obix:Alarm''>
   <ref name=''source'' />
   <abstime name=''timestamp'' />
</obj>
```

  Source ist die URI, die die Alarmquelle identifiziert. Als Quelle soll auf ein oBIX Objekt, das den Alarm generiert, verwiesen werden. Timestamp gibt den Zeitpunkt an, wann die Alarmquelle vom normalen Zustand zum Zustand off-normal übergeht und der Alarm Record generiert wurde.

- `StatefulAlarm`

```
<obj href=''obix:StatefulAlarm'' is=''obix:Alarm''>
   <abstime name=''normalTimestamp'' null=''true'' />
</obj>
```

  Das (zusätzliche, neben den veerbten) Kindobjekt ist normalTimestamp. Dieses Feld ist null, wenn die Alarmquelle noch im Alarmzustand ist. Ansonsten beinhaltet es die Zeit, wenn das Objekt wieder in den Normalzustand übergeht.

- `AckAlarm`
  Alarme, die eine Bestätigung unterstützen implementieren den `AckAlarm` Contract:

```
<obj href=''obix:AckAlarm'' is=''obix:Alarm''>
   <abstime name=''ackTimestamp'' null=''true'' />
   <str name=''ackUser'' null=''true'' />
   <op name=''ack'' in=''obix:AlarmAckIn'' out=''obix:AlarmAckOut'' />
</obj>
```

```
<obj href=''obix:AckAlarmIn''>
   <str name=''ackUser'' null=''true'' />
</obj>

<obj href=''obix:AckAlarmOut''>
   <obj name=''alarm'' is=''obix:AckAlam obix:Alarm'' />
</obj>
```

Bei einem unbestätigten Alarm ist AckTimeStamp Null. Es indiziert den Zeitpunkt der Bestätigung. AckUser gibt einen String an, welcher kennzeichnet, wer für das Acknowledgement verantwortlich war.

- `PointAlarms`
  Es ist weit verbreitet, dass eine Alarmquelle ein `obix:Point` ist. Respektive stellt ein `PointAlarm` den Wert dar, der den Alarm ausgelöst hat. Der Contract sieht folgendermaßen aus:

```
<obj href=''obix:PointAlarm'' is=''obix:Alarm''>
   <obj name=''alarmValue'' />
</obj>
```

AlarmSubject

Server, die das oBIX Alarming anbieten, müssen ein oder mehrere Objekte von `AlarmSubject` Contract implementieren. Mit dem `AlarmSubject` können die Alarme, welche der Client erhält oder beobachten kann, gruppiert und kategorisiert werden. Zum Beispiel kann der Server ein `AlarmSubject` für alle Alarme und andere `AlarmSubjects` basierend auf der Priorität oder der Tageszeit anbieten.
Der Contract für ein `AlarmSubject` sieht folgendermaßen aus:

```
<obj href=''obix:AlarmSubject''>
   <int     name=''count'' min=''0'' val=''0'' />
   <op     name=''query'' in=''obix:AlarmFilter'' out=''obix:AlarmQueryOut'' />
   <feed  name=''feed'' in=''obix:AlarmFilter'' of=''obix:Alarm'' />
</obj>

<obj href=''obix:AlarmFilter''>
   <int     name=''limit'' null=''true'' />
   <abstime name=''start'' null=''true'' />
   <abstime name=''end'' null=''true'' />
</obj>

<obj href=''obix:AlarmQueryOut''>
   <int     name=''count'' min=''0'' val=''0'' />
   <abstime name=''start'' null=''true'' />
   <abstime name=''end'' null=''true'' />
   <list    name=''data'' of=''obix:Alarm'' />
</obj>
```

Das `AlarmSubject` verfolgt dieselben Designprinzipien wie `History` . Das `AlarmSubject` spezifiziert die aktive Anzahl `count` von Alarmen. Es beinhaltet eine `query` Operation, um die aktuelle Liste der Alarme mit Anwendung des Filters `AlarmFilter` auszulesen. Mit `AlarmFilter` können die Zeitgrenzen (time bounds) eingeschränkt werden.

`AlarmSubject` enthält auch ein Feed Objekt, was zum Abonnieren der Alarm Events verwendet werden kann.

4.6 Security

Das Thema Sicherheit überdeckt die Bereiche:

- **Authentifizierung**: überprüfen, ob ein User (Client) der ist, für den er sich ausgibt
- **Kodierung, Verschlüsselung**: oBIX Dokumente schützen
- **Zugriffskontrolle**: kontrollieren, ob ein User Erlaubnis zum Zugriff auf Schreib- und Leseobjekte oder das Aktivieren von Operationen hat
- **User Management**: managen von User Konten und Erlaubnissen (permission levels)

oBIX baut diese Bereiche nicht in die Spezifikation ein. Authentifizierung und Verschlüsselung wird dem protocol binding überlassen. Privilegien und User Management werden den Herstellern überlassen. Es ist zwar möglich, ein Management Model mittels oBIX zu definieren, jedoch sind in der Spezifikation keine Standard Contracts vorhanden.

5 Fazit

XML ist heutzutage einer der wichtigsten Bestandteile im Web. Bereits 2006 wird in [6] darauf hingewiesen, dass mit oBIX und der maßgebenden XML Technologie die Integration von Systemen der Gebäudeautomation und Enterprisesystemen einfacher wird. Mit oBIX ist es möglich, größere, reichhaltigere Cross-Systeme zu erstellen um den Bedürfnissen der Anwender gerecht zu werden. oBIX hat das Potential das Enterprise Gateway für den gesamten Maschine-zu-Maschine (m2m) Bereich zu sein [6]. Mit der Entstehung von mehreren standardisierten Protokollen taucht das Problem der Integration von BA-Systemen untereinander auf. Da Web Services eine "nahtlose" Integration dieser Systeme ermöglichen, stellen sie eine interessante Problemlösung dar.

Trotzdem geht die Weiterentwicklung seit der Herausgabe der Spezifikation "oBIX 1.0" im Jahr 2006 nur schleppend voran. Nach [15] kann folgender Fortschritt seit 2006 festgestellt werden.
Ab Mai 2007 wird zur Gründung eines neuen Komitees, das sich mit der Ausarbeitung von neuen Enterprise Contracts beschäftigen soll, aufgerufen. Im November 2007 wird bekannt gegeben, dass die Updates der Spezifikation bis Ende des Jahres fertiggestellt seien. Neben Korrekturen und Klarstellungen wurden Datums- und Zeitobjekte, die das nachfolgende Scheduling Model unterstützen sollen, eingeführt. Im Oktober 2008 wird ein neues Sub-Komitee gegründet, das eine neue Referenzspezifikation erstellen soll. Diese Spezifikation

soll die Charakteristika einer Schnittstelle beschreiben, welche von einem Building System Integrator bereitgestellt werden sollen. Im April 2009 diskutiert das oBIX Komitee über die Ausarbeitung einer Version 1.1 der Spezifikation. Dabei sollen Unklarheiten der Version 1.0 beseitigt werden, um weiters Schedule Elemente des WS-Calender (Web Services Kalender) angliedern zu können. Im Februar 2010 startet ein neues technisches Komitee das Vorhaben WS-Calender. oBIX plant die Verwendung der WS-Calender Spezifikation, um die Performanceerwartungen der Regelungssysteme mit Enterprise- und Smart Grid [10]-Tätigkeiten koordinieren zu können.
Seit Spezifikation 1.0 wurde noch keine neuere Version publiziert.

Laut [3] ist oBIX durchaus weit verbreitet und gedeihend. Die "Stille" wird dadurch begründet, dass oBIX mit vielen anderen Protokollen der Enterprisesysteme kombiniert wird. Daraus resultieren Projekte, deren Fertigstellung sich auf mehrere Jahre ausdehnt. Der Einsatz von oBIX wird in [3] durch laufende Projekte unterstrichen. Beispielsweise wurde oBIX in Gebäudesystemen des Olympic Stadiums und allen Außenbeleuchtungssystemen des Olympic Districts in Peking eingesetzt.

[10] Intelligente Stromnetze

Abbildungsverzeichnis

Tabellenverzeichnis

Literatur

[1] J. Bai, H. Xiao, X. Yang, and G. Zhang. Study on integration technologies of building automation systems based on web services. *ISECS International Colloquium on Computing, Communication, Control, and Management*, 4:262–266, 2009.

[2] J. Bai, H. Xiao, T. Zhu, W. Liu, and A. Sun. Design of a web-based building management system using ajax and web services. *Business and Information Management, International Seminar on*, 2:63–66, 2008.

[3] T. Considine. obix is alive and kicking... `http://www.automatedbuildings.com/news/jun08/articles/considine/080539121012obix.htm`, Juni 2008. [Online-Zugriff 27-Maerz 2010].

[4] P. Ehrlich. update on obix. `http://www.automatedbuildings.com/news/dec03/interviews/obix.htm`. [Online-Zugriff 05-Feburar 2010].

[5] B. Frank. Version .11 presentation. `http://www.oasis-open.org/committees/document.php?document_id=17936`, Mai 2006. [Online-Zugriff 02-Feburar 2010].

[6] A. Hanson. obix unbound. `http://www.automatedbuildings.com/news/may06/articles/obix/060425112552obix.htm`, Mai 2006. [Online-Zugriff 06-Feburar 2010].

[7] H. Himmelbauer. Soap interface for an internet/fieldbus gateway, September 2006. Diplomarbeit.

[8] HP. hp web services platform... `http://archive.devx.com/javasr/whitepapers/hp/default.asp`. [Online-Zugriff 27-Maerz 2010].

[9] IBM. Ibm websphere. `http://en.wikipedia.org/wiki/IBM_WebSphere`. [Online-Zugriff 27-Maerz 2010].

[10] W. Kastner and G. Neugschwandtner, editors. *Datenkommunikation in der verteilten Gebäudeautomation*. Bulletin SEV/AES 17/06, 2006.

[11] P. Manolescue. What is obix. `http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=obix`, August 2004. [Online-Zugriff 02-Feburar 2010].

[12] Microsoft. .net. `http://de.wikipedia.org/wiki/.NET`. [Online-Zugriff 27-Maerz 2010].

[13] M. Neugschwandtner, G. Neugschwandtner, and W. Kastner, editors. *Web Services in Building Automation: Mapping KNX to oBIX*. IEEE, 2007.

[14] oasis. oasis faq. http://www.oasis-open.org/committees/obix/faq.php. [Online-Zugriff 05-Feburar 2010].

[15] OASIS. obix homepage. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=obix. [Online-Zugriff 27-Maerz 2010].

[16] OASIS. obix 1.0. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=obix, Dezember 2006. [Online-Zugriff 05-Feburar 2010].

[17] obix. obix faq. http://www.automatedbuildings.com/news/feb04/articles/faq/faq.htm, Februar 2004. [Online-Zugriff 05-Feburar 2010].

[18] T. Perumal, A. Rahman Ramli, C. Yew Leong, S. Mansor, and K. Samsudin, editors. *Interoperability for Smart Home Environment Using Web Services*. International Journal of Smart Home, Vol. 2, No. 4, Oktober 2008.

[19] Sun. Metro web services overview. http://java.sun.com/webservices/. [Online-Zugriff 27-Maerz 2010].