Regenerative Modellregion Harz

Schnittstellen und Kommunikationsprotokolle
Abschlussbericht AP1.4 und AP3.1

Dokumententyp	Spezifikation
Sichtbarkeit	Internes Dokument
Arbeitspaket	AP3.1: IKT Infrastruktur
Editor	Martin Winter, SAG
Beitragende	Martin Winter
Version	02
Letzte Änderung	08.11.2011
Seitenzahl	26
Dateiname	RMH_AP3.1_D0224_V02.doc

Gefördert durch das

Bundesministerium
für Umwelt, Naturschutz
und Reaktorsicherheit

im Rahmen von

E-ENERGY

Inhaltsverzeichnis

Abbildungsverzeichnis

1. Einführung

Die Einbindung dezentraler Energieanlagen in das Energiesystem, die Marktliberalisierung und die damit einhergehenden neuen Rollenmodelle am Energiemarkt sowie der Wandel der Energieerzeugung hin zu einem System mit einem hohen Anteil fluktuierender Erzeuger (Wind, PV, usw.) erfordern eine völlig neue Struktur der Kommunikation zwischen den beteiligten Einheiten.

Statt einer Leittechnik, die eine feste Anzahl abgesetzter Einheiten über „Remote Terminal Units“ mit bekannten Eigenschaften einbindet, ist nun eine flexible, serviceorientierte Sicht notwendig: Dezentrale Energieanlagen bieten ihre Dienste an: sowohl am Markt, als auch auf technischer Ebene. Betreiber von virtuellen Kraftwerken, aber auch Energieversorger und Netzbetreiber nehmen diese Dienste in Anspruch.

Neue Dienste kommen dabei dynamisch hinzu, andere fallen wieder weg. Auch die Erreichbarkeit und Zuverlässigkeit der dezentralen Anlagen erreicht oftmals nicht das Niveau, das man bisher im Energiesystem gewohnt war. Trotzdem gilt es, daraus ein Gesamtsystem zu schaffen, das nicht nur umweltfreundlich ist, sondern auch den Anforderungen an Zuverlässigkeit und Wirtschaftlichkeit entspricht.

Diese Spezifikation beschreibt die IKT Architektur, die in RegModHarz als Antwort auf die genannten Anforderungen eingesetzt wird. Die Architektur setzt in vielen Fällen auf standardisierten und bewährten Technologien auf und schafft so die Voraussetzungen für eine rasche Einführung und eine hohe Akzeptanz.

Das vorliegende Dokument beschreibt deshalb außerdem die eingesetzten Technologiestandards, Schnittstellen und die konkrete Umsetzung und dient damit auch als Spezifikation für die Realisierung der Komponenten für den Feldtest.

Das vorliegende Dokument schließt damit die Arbeiten des Arbeitspaketes 1.4 ab, indem es die Systemarchitektur bezogen auf die Einbindung dezentraler Energieanlagen über die PowerBridge beschreibt. Es stellt weiterhin die in Arbeitspaket 3.1 vorgesehene Spezifikation der IKT-Infrastruktur dar und beinhaltet alle relevanten Informationen, die zur Kommunikation mit den dezentralen Energieanlagen erforderlich sind.

1.1. Architektur

Die genannten Anforderungen spiegeln sich in einer Architektur mit drei wesentlichen Komponenten wider:

- Die PowerBridge ist die Repräsentation einer dezentralen Energieanlage nach außen hin. Es stellt damit eine einheitliche Service-Schnittstelle zur Verfügung, über die die Anlage ihre Eigenschaften, Dienste, aber auch ihren Zustand und weitere Messwerte zur Verfügung stellen kann.
- Diese Dienste werden über eine **Service Registry** publiziert. Dort können die angebotenen Dienste registriert werden. Eine Service Registry bietet aber darüber hinaus auch die Möglichkeit, weitere zum Service gehörende Daten und Dokumente abzulegen und so für Menschen oder Maschinen zugänglich zu machen. Außerdem enthält die Service Registry Daten, die zum Zugriff auf die angebotenen Dienste notwendig sind (z.B. eine URL).
- Die im nachfolgenden Bild dargestellt **Leitwarte** repräsentiert einen Nutzer der angebotenen Dienste. Die Bezeichnung ist dabei generisch zu verstehen und unabhängig davon, ob es sich um die Leitwarte eines Verteilnetzbetreibers oder eines Virtuellen Kraftwerkes handelt. Die Leitwarte subskribiert sich zunächst bei der Service Registry und wird dann von dieser über neu hinzukommende, geänderte oder wegfallende Dienste informiert. Mit den von der Service Registry erhaltenen Daten kann er auch die von den dezentralen Anlagen über die PowerBridge angebotenen Dienste nutzen.

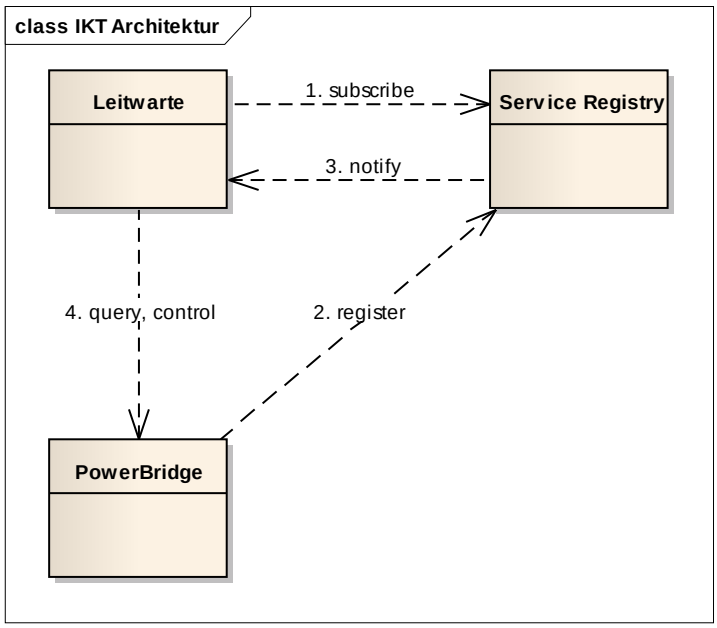

Abbildung 1: Übersicht der IKT Architektur

Dies entspricht dem Grundaufbau einer SOA (Service Oriented Architecture), wie sie in der folgenden Grafik[1] dargestellt ist:

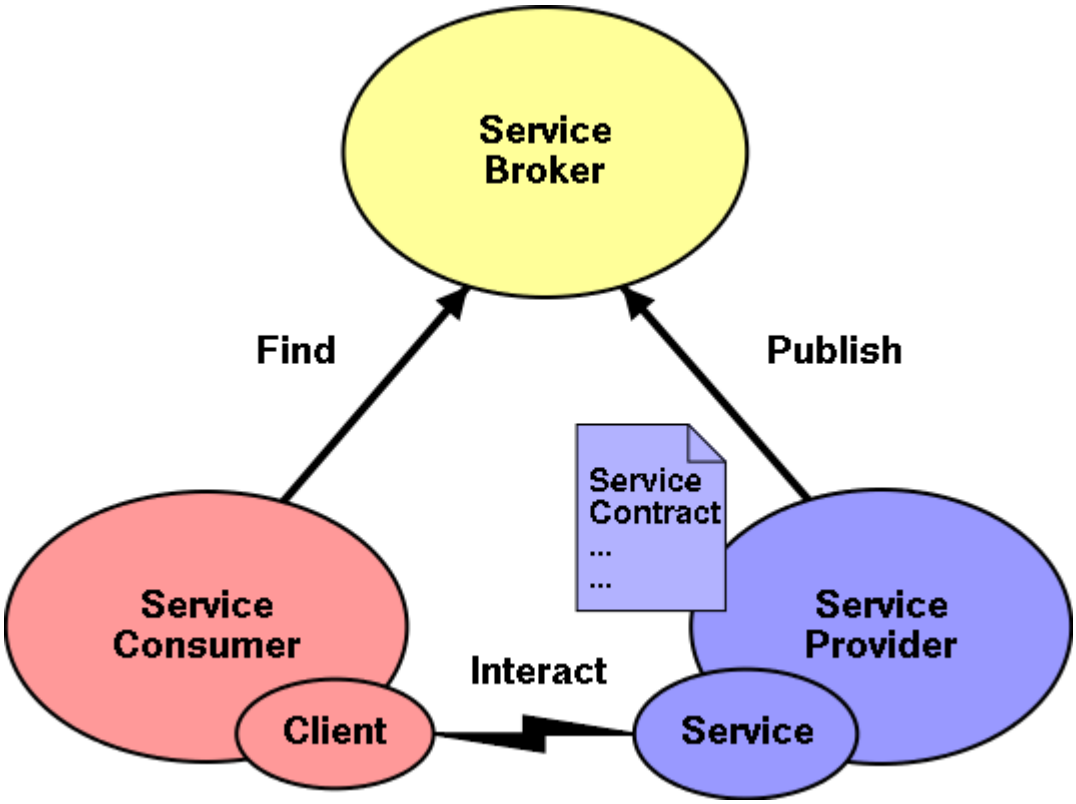

Abbildung 2: Generelle Struktur einer SOA

Die PowerBridge ist dabei der Service Provider. Die Registry agiert als Service Broker und die Leitwarte ist der Service Consumer.

[1] [1] W3C 22.05.2003

1.2. PowerBridge

Die PowerBridge spielt eine zentrale Rolle im Gesamtsystem von RegModHarz. Als Service Provider stellt es die wesentlichen Dienste der dezentralen Energieanlagen zur Verfügung. Durch die weitgehend einheitliche Abbildung der Eigenschaften von ganz unterschiedlichen dezentralen Energieanlagen nach außen hin wird es erst möglich, die heterogene Landschaft von Energieanlagen in einheitlicher Weise zu verwalten und zu beeinflussen.

Das Gateway trennt aber auch die Verantwortungsbereiche des Anlagenbetreibers und des VPP Betreibers (oder Netzbetreibers, …) voneinander und ermöglicht eine klare Zuordnung von Zuständigkeiten.

Voraussetzung hierfür sind jedoch klare Spezifikationen der Schnittstellen und der verwendeten Datenmodelle, da nur so die genannten Ziele erreicht werden können.

RegModHarz setzt dabei auf die Empfehlungen der Normungsroadmap Smart Grid des DKE[2] auf und definiert eine Kommunikationsarchitektur basierend auf der Standardreihe IEC 61850. In diesem Rahmen spezifiziert das Projekt ein einfaches Datenmodell[3], das eine generische Abbildung unterschiedlicher dezentraler Energiesysteme ermöglich und dabei Erzeuger, Verbraucher und Speicher sowie deren Variabilität berücksichtigt. Für die Kommunikation setzt das Projekt das Web Service Mapping gemäß IEC 61400-25-4 ein und setzt somit mit Web Services auf einen Standard, der durch seine hohe Verbreitung, offene Standards und hervorragende Tool-Unterstützung ideal für diesen Zweck geeignet ist.

1.3. Service Registry

Die Service Registry ist der zentrale Punkt, an dem die von den dezentralen Energieanlagen über die PowerBridge angebotenen Dienste und die an diesen Diensten interessierten Akteure (Energieversorger, Netzbetreiber, Betreiber eines virtuellen Kraftwerkes, usw.) zusammenfinden. Die Service Registry macht die angebotenen Dienste erst im System bekannt und ermöglicht die gezielte Suche nach geeigneten Diensten basierend auf den unterschiedlichsten Kriterien.

Auch hier setzt RegModHarz einerseits auf eine standardisierte Lösung. Wichtig bei der Auswahl war aber auch die flexible Erweiterbarkeit und gute Skalierbarkeit der eingesetzten Technologie. Beide Anforderungen werden von der ebXML Registry/Repository[4] erfüllt. Auch diese Technologie basiert auf Web Services und passt somit gut in die gewählte Gesamtarchitektur.

1.4. Leitwarte

Der Begriff „Leitwarte“ steht hier stellvertretend für alle Service Consumer, die an Diensten der dezentralen Energieanlagen interessiert sind. Die beschriebene Architektur lässt dabei unterschiedlichste Service Consumer zu. Die Sicherheitsmechanismen ermöglichen es dabei, die jeweiligen Zugriffsrechte detailliert und spezifisch festzulegen. Die Kontrolle über die Daten verbleibt dabei stets beim Service Provider, also bei der PowerBridge.

[2] [2] Ratz 13.04.2010

[3] [3] RegModHarz

[4] [4] OASIS ebXML Registry TC

2. PowerBridge mit IEC 61850

Die zentrale Schnittstelle im operativen Betrieb ist die zwischen einer Leitwarte (unabhängig davon, welche Rolle der Leitwartenbetreiber einnimmt) und der PowerBridge. Für diese Schnittstelle setzt das Projekt auf die Spezifikationen von IEC 61850 auf, wobei das Mapping auf Web Services gemäß IEC 61400-25-4[5] eingesetzt wird.

Weiterhin besitzt die PowerBridge ein User Interface, das über einen Browser angesprochen werden kann. Es gibt Zugriff auf die Daten im Datenmodell und ermöglicht es, die PowerBridge bei der Registry zu registrieren oder zu deregistrieren.

Zur Anbindung an die Anlagen können unterschiedliche Device Adapter eingesetzt werden. Diese setzen die anlagenspezifischen Protokolle und Datenstrukturen auf das IEC 61850 Datenmodell um.

Weiterhin können interne Algorithmen an das Datenmodell angebunden werden, die z.B. Fahrpläne bearbeiten oder sonstige Operationen auf den Daten des Datenmodells vornehmen. Diese Schnittstelle ist in einer getrennten Spezifikation[6] ausführlich beschrieben.

2.1. Struktur

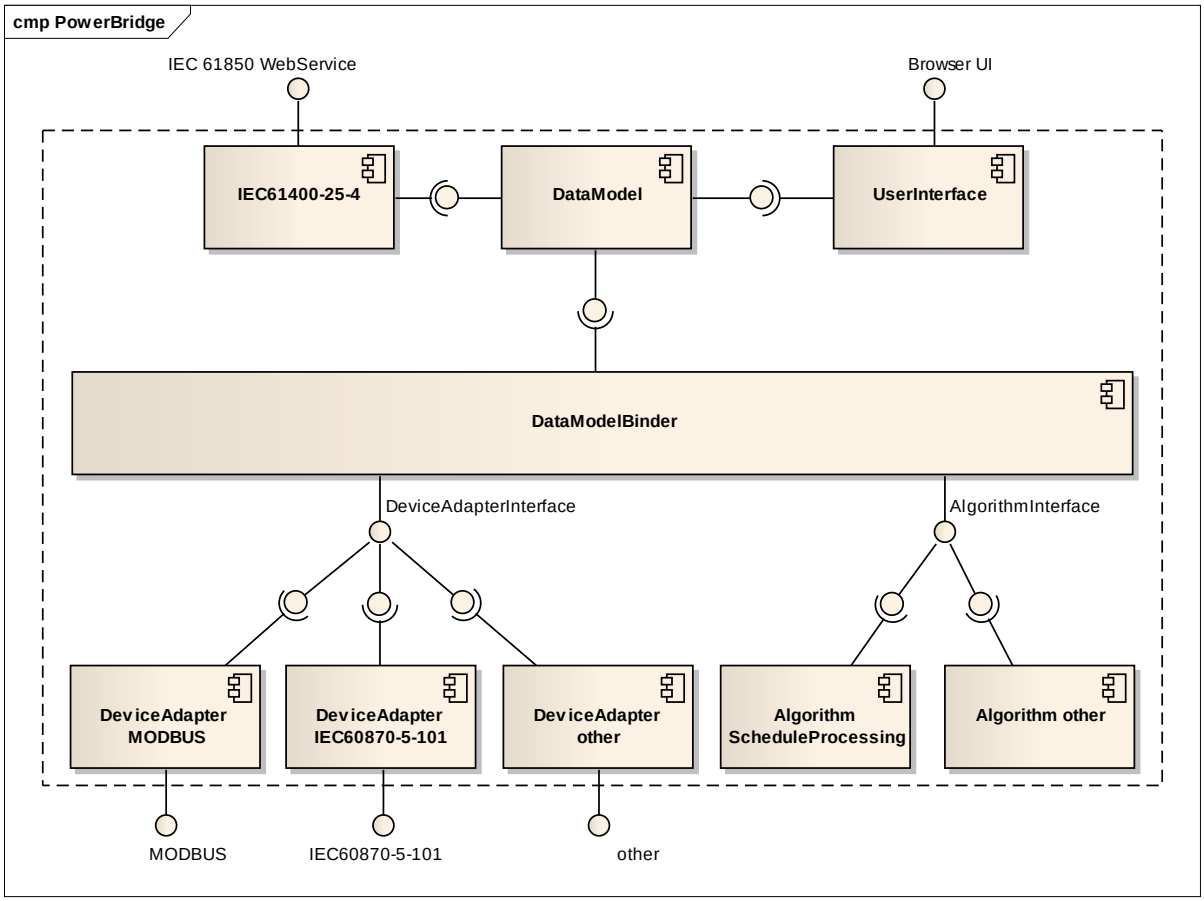

Abbildung 3: Struktur der PowerBridge

Die zentrale Komponente der PowerBridge ist das Datenmodell, das über einen SCL Konfigurationsfile gemäß IEC 61850-6[7] konfiguriert wird. In diesem Konfigurationsfile sind auch die Vorbelegungen

[5] [5], IEC 61400-25-4

[6] [6] RegModHarz

[7] [7], IEC 61850-6

der Datenwerte festgehalten. Außerdem werden dort Benutzerkennungen und die Zugriffsberechtigungen der Benutzer konfiguriert.

Eine weitere Komponente ermöglicht den Zugriff über IEC 61400-25-4 Web Services auf das Datenmodell. Eine detaillierte Beschreibung der hierüber abrufbaren Informationen liefert das nachfolgende Kapitel.

Für den direkten lokalen Zugriff auf die Daten des Datenmodells steht außerdem ein User Interface zur Verfügung, das in strukturierter Weise Zugriff auf die Daten erlaubt.

Zur Anbindung der PowerBridge an vorhandene Systeme der Energieanlagen steht eine Reihe von sogenannten Device Adaptern zur Verfügung. Diese stellen die Implementierung bestimmter Protokolle zur Verfügung, z.B. MODBUS oder IEC 60870-5-101. Der DataModelBinder stellt die Verbindung zwischen den Protokollen und den Datenwerten des Datenmodells her. Weitere Adapter können als Bausteine nach Bedarf hinzugefügt werden.

Der DataModelBinder kann außerdem in der PowerBridge realisierte Algorithmen an das Datenmodell anbinden. Er stellt dazu die Verbindung zwischen einem vom Algorithmus spezifizierten Daten-Interface und den Datenpunkten des Datenmodells her. Algorithmen können zyklisch aufgerufen oder durch die Änderung von Datenwerten getriggert werden. Eine detaillierte Beschreibung dieser Schnittstelle findet sich in einem getrennten Dokument[8].

2.2. ACSI Services mit Web Service Mapping

Für die Kommunikation zwischen PowerBridge und Leitwarte ist es auf Applikationsebene unerheblich, welche Protokolle, Datenformate oder Kommunikationstechnologien eingesetzt werden. IEC 61850-7-2 definiert deshalb einen abstrakten Satz von Diensten, das „Abstract Communication Service Interface“ ACSI, das unabhängig von der eingesetzten Übertragungstechnologie, aber auch unabhängig von der verwendeten Programmiersprache auf abstrakte Weise Datenzugriffsmethoden spezifiziert.

Eine konforme Implementierung muss dabei nicht alle Dienste implementieren. Auch für RegModHarz wird lediglich ein Subset der spezifizierten Dienste eingesetzt. Diese werden im Folgenden kurz beschrieben.

Trotz der prinzipiell abstrakten Beschreibung der Services werden zusätzlich Programmierbeispiele in Java angeboten, um die Benutzung der Services zu erleichtern. Die Beispiele basieren auf den Klassen, die durch das in jedem JDK ab Version 6 enthaltenen Tool `wsimport` aus der WSDL generiert werden.

2.2.1. An- und Abmelden: Associate und Release

Bevor ein Client über Web Service auf weitere Funktionen zugreifen kann, muss er sich zuerst durch einen Associate Request mit Benutzernamen und Passwort anmelden. Sind die Anmeldedaten korrekt, so erhält er eine AssocID zurück, die er bei jedem weiteren Request mit angeben muss.

Die Benutzerdaten werden in der PowerBridge über den SCL-Konfigurationsfile eingestellt. Dort wird auch verwaltet, welcher Benutzer welche Rechte hat. Dabei kann für jedes Datenobjekt des Datenmodells, aber auch für ganze Unterbäume festgelegt werden, ob diese gelesen und/oder geschrieben werden dürfen. Ist weder Lese- noch Schreibzugriff erteilt, so ist das entsprechende Element für diesen Benutzer komplett verborgen, d.h. erscheint auch nicht bei den im nachfolgenden Abschnitt beschriebenen Verzeichnisdiensten.

Eine Anmeldung wird ungültig, wenn sie der Benutzer über einen expliziten Release Request freigibt oder wenn sie für eine bestimmte Zeit (30 min) nicht benutzt wird.

Bemerkung: Die in IEC 61400-25-4 standardisierte WSDL enthält einen Fehler beim Release Request. Dieser liefert dort eine AbortResponse zurück. Richtig wäre natürlich eine ReleaseResponse. Unabhängig von diesem Fehler funktioniert der Aufruf jedoch wie gewünscht.

[8] [6] RegModHarz

2.2.2. Verzeichnisdienste: GetServer / LogicalDevice / LogicalNodeDirectory

Diese drei Aufrufe ermöglichen es einem Client herauszufinden, welche Logical Nodes auf einer PowerBridge für ihn sichtbar sind. Wie die Namen schon andeuten, liefern die Aufrufe folgende Informationen:

- GetServerDirectory: Liefert eine Liste aller Logical Devices des Servers.
- GetLogicalDeviceDirectory: Liefert eine Liste aller Logical Nodes in einem Logical Device
- GetLogicalNodeDirectory: Liefert eine Liste aller Datenobjekte in einem Logical Node

Dabei werden jeweils nur solche Objekte zurückgegeben, zu denen der jeweilige Nutzer (siehe vorhergehendes Kapitel) auch Zugriff hat. Damit können die Ergebnisse dieser Aufrufe für unterschiedliche Nutzer durchaus unterschiedlich sein.

2.2.3. Daten lesen und schreiben: Get / SetDataValues

Diese beiden Dienste stellen die zentralen und wesentlichsten Methoden zum Zugriff auf das Datenmodell dar. Sie ermöglichen es, Datenobjekte auszulesen oder Werte zu setzen.

Der Aufruf GetDataValues liefert für einen gegebenen Pfad im Datenmodell und ein Functional Constraint (siehe Spezifikation des Datenmodells) alle dazugehörigen Datenwerte zurück. Mit SetDataValues können umgekehrt ein oder mehrere Datenwerte im Datenmodell geschrieben werden. Die Beispiele in Kapitel 7.1 zeigen das Lesen oder Schreiben eines einzelnen Attributes. Für mehrere Attribute können sie entsprechend erweitert werden.

2.2.4. Datengruppe lesen: GetDataSetValues

Oft ist es erforderlich, eine Gruppe von Daten aus unterschiedlichen Zweigen des Datenmodells gemeinsam auszulesen. Hierzu können im SCL Konfigurationsfile der PowerBridge sogenannte DataSets definiert werden. Diese können dann mit GetDataSetValues angesprochen und mit einem Aufruf als Gruppe ausgelesen werden.

2.2.5. Ereignisse beobachten: Add / RemoveSubscription

Mit diesen Aufrufen kann sich ein Client dafür registrieren, bei Updates oder Änderungen von Datenwerten informiert zu werden. Für den Prototyp werden nicht alle der relativ komplexen Parameter einer Subscription unterstützt.

Der Standard unterscheidet zwischen Buffered Reports und Unbuffered Reports. Bei letzteren können Benachrichtigungen verloren gehen, wenn sie vom Client nicht rechtzeitig bearbeitet werden. Die PowerBridge implementiert jedoch ausschließlich Buffered Reports.

Um Daten zu beobachten, muss man zunächst ein im SCL Konfigurationsfile der PowerBridge in DataSet definieren. Der AddSubscription Request bezieht sich auf das DataSet und beobachtet Änderungen an den Werten des DataSets.

Die weiteren Parameter des AddSubscription Requests werden wie folgt benutzt (siehe auch WSDL File in IEC 61400-25-4[9]):

- RCBRef: Muss belegt werden (mandatory), wird in der PowerBridge aber lediglich für RemoveSubscription benutzt
- RCBType: Stets BRCB
- RptID: Muss belegt werden. Wird zur Abfrage der Ereignisse benutzt
- RptEna: Nicht benutzt, stets „enabled“
- DatSet: Verweis auf das DataSet, das auf Veränderungen beobachtet werden soll

[9] [5], IEC 61400-25-4

- OptFlds: Nicht benutzt
- BufTm: Nicht benutzt
- TrgOp: Sollte stets „dchg“ = Data Changed sein
- IntgPd: Nicht benutzt
- DSMbrRef: Nicht benutzt

2.2.6. Report

Die Web Services in IEC 61400-25-4 arbeiten streng nach dem Client-Server Prinzip: Die Power-Bridge ist Server. Anfragen werden stets vom Client zum Server geschickt und von diesem beantwortet.

Damit können jedoch keine spontanen Nachrichten vom Server zum Client gesendet werden. Stattdessen muss der Client zuvor explizit nach diesen Nachrichten fragen. Genau das ist der Report Request.

Möchte der Client also Informationen zu seinen Subscriptions erhalten, so muss er einen Report Request absetzen. Der Server wird diesen Request nun nicht unbedingt sofort beantworten, sondern nur

- Wenn eine Mindestantwortzeit abgelaufen ist und Nachrichten für den Client anstehen
- Oder nach dem Ablauf einer maximalen Antwortzeit

Der Client muss dann einen erneuten Report Request anstoßen, um weitere Nachrichten zu erhalten. Das Verfahren benutzt also einen „hängenden“ Request, der solange unbeantwortet bleibt, bis Informationen für den Client verfügbar sind. Der Timeout dient dazu, dass der Client überprüfen kann, ob der Server überhaupt noch antwortet.

3. Sicherheitsarchitektur

Die Kommunikation zwischen Leitwarte und PowerBridge ist stets Client-Server basiert. Dabei ist die PowerBridge der Server und die Leitwarte der Client.

Zum Aufbau einer verschlüsselten und vertrauenswürdigen Verbindung benutzt dabei der Server (also die PowerBridge) ein Server-Zertifikat, mit dem es sich dem Client (also der Leitwarte) gegenüber identifiziert. Der Client überprüft dieses Zertifikat auf Vertrauenswürdigkeit und baut dann eine verschlüsselte Verbindung zum Server auf.

Damit der Client dem Server vertrauen kann, gibt es grundsätzlich zwei Möglichkeiten:

- Der Client bekommt vorab eine Liste der Server Zertifikate aller PowerBridges und legt diese in seinem Trust Store ab. Dies erfordert jedoch eine Erweiterung des Trust Store, sobald eine neue PowerBridge hinzukommt.
- Eine Certificate Authority (CA) wird damit beauftragt, die Zertifikate für die PowerBridges zu erstellen und zu signieren. Die Leitwarte nimmt lediglich das Zertifikat der CA in ihren Trust Store auf. Sie vertraut damit automatisch allen Zertifikaten, die diese CA signiert hat. Beim Hinzufügen einer neuen PowerBridge ist damit auf Client-Seite kein Eingriff erforderlich. Die neue PowerBridge muss lediglich sein Server-Zertifikat von der CA signieren lassen.

Die Rolle der CA könnte ein PowerBridge Betreiber übernehmen oder eine andere Instanz, die für die Installation und den Betrieb der Gateways zuständig ist.

RegModHarz benutzt dieses CA-Modell, das nachfolgend noch einmal grafisch dargestellt ist:

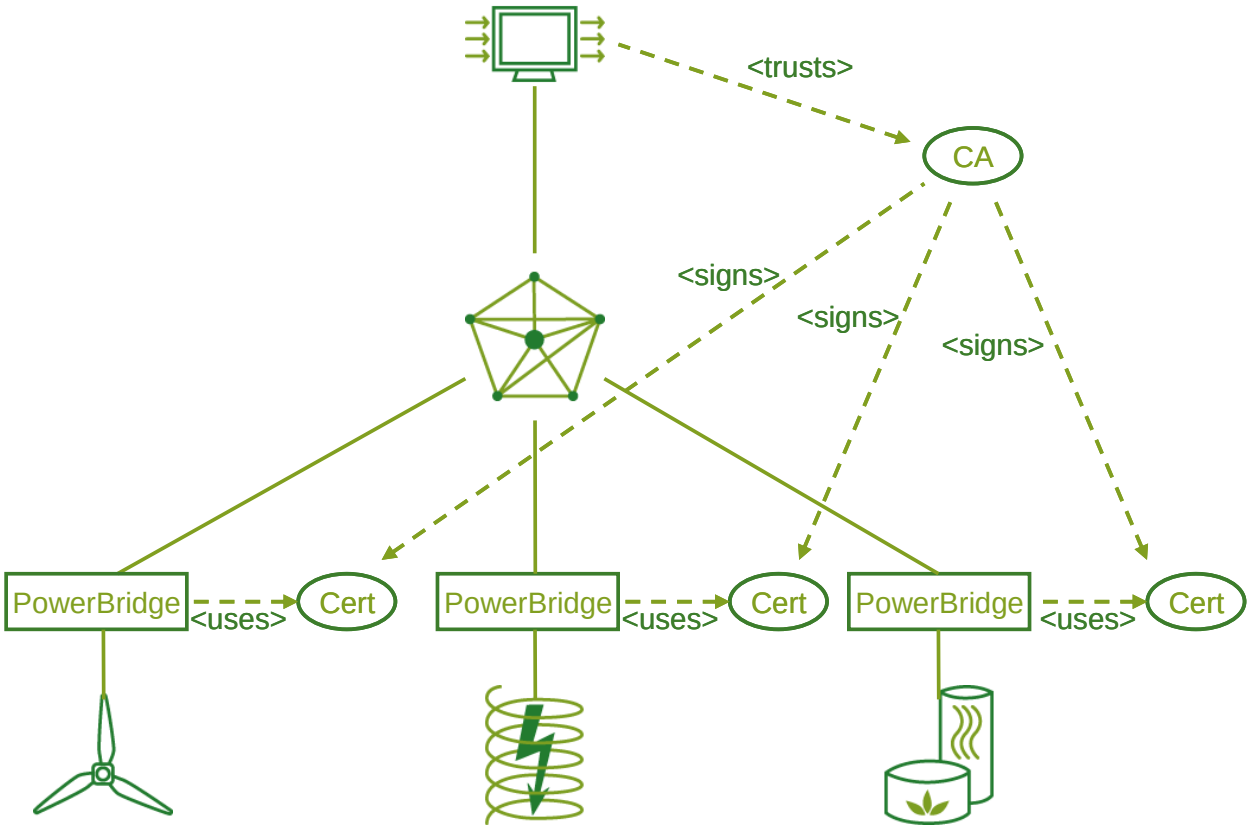

Abbildung 4: Sicherheits-Architektur

4. Schnittstelle zur Registrierung

4.1. Ablauf der Registrierung

Das nachfolgende Diagramm zeigt grob die Abläufe bei der Installation und Registrierung einer neuen PowerBridge:

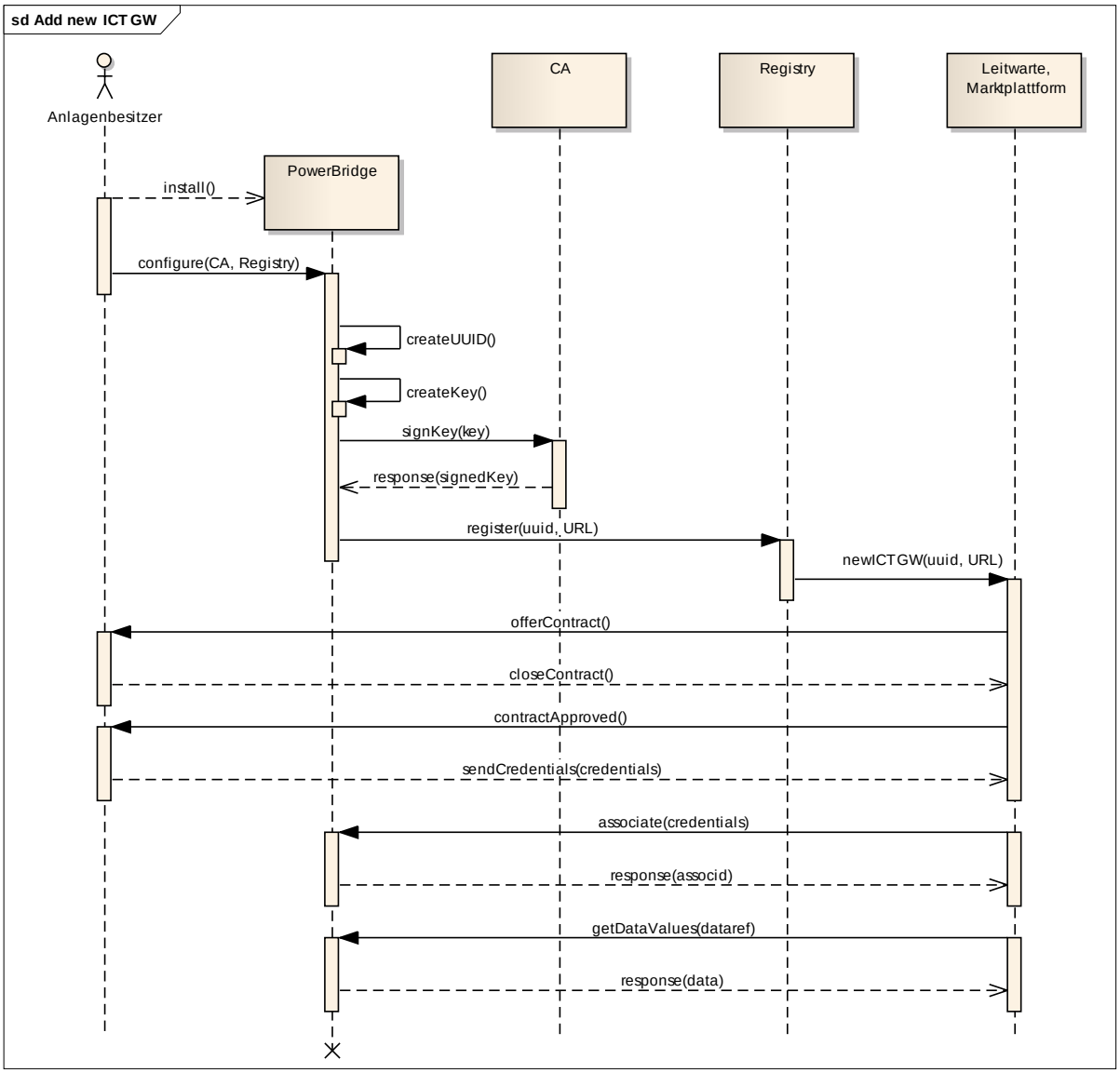

Abbildung 5: Registrierung einer neuen PowerBridge

Der Ablauf ist wie folgt:

- Der Anlagenbesitzer (oder eine andere von ihm beauftragte oder bevollmächtigte Person) installiert die PowerBridge und sorgt für die nötigen Anschlüsse an seine Anlage.
- Er konfiguriert nun über die Bedienoberfläche der PowerBridge die URLs einer CA, die für das Ausstellen eines Zertifikates verantwortlich ist, sowie einer Registry, bei der er sich schließlich registrieren möchte.
- Die PowerBridge erzeugt nun eine eindeutige UUID, mit der es sich selbst in Zukunft identifizieren möchte. Diese UUID wird im Datenmodell unter LPHD1.PhyNam.mrID eingetragen. Sofern die PowerBridge mehrere Logical Devices beinhaltet, wird die gleiche Information in allen Logical Devices wiederholt (siehe auch IEC 61850-7-1, Abschnitt 8.2).

- Die PowerBridge erzeugt außerdem ein Schlüsselpaar.
- Nun beantragt die PowerBridge bei der CA ein Server Zertifikat. Die CA kann für diesen Vorgang weitere Identifizierungsmerkmale verlangen. So könnte sie z.B. auch offline den Anlagenbesitzer über PostIdent oder ein anderes Verfahren identifizieren, bevor sie bereit ist, das Zertifikat auszustellen. Dies ist jedoch nicht Betrachtung dieses Ablaufes und wird in RegModHarz auch nicht weiter untersucht.
- Die CA stellt nun ein Server-Zertifikat für die PowerBridge aus und sendet es zurück. Die PowerBridge installiert dieses Zertifikat auf seinem Server. Die PowerBridge ist nun betriebsbereit.
- Im nächsten Schritt registriert sich die PowerBridge bei der Registry und übergibt dorthin die Daten des Betreibers, die UUID der Anlage, die eigene Server-URL sowie weitere Eigenschaften der dezentralen Energieanlage, die für einen potentiellen Nutzer der Dienste von Interesse sind.
- Der Leitwartenbetreiber hat bereits im Vorfeld eine Abfrage in der Registry eingerichtet, über die er bei für ihn relevanten Änderungen informiert wird. Meldet sich nun eine neue PowerBridge an, das zu dieser Anfrage passt, so wird die Leitwarte von der Registry aktiv darüber informiert. Sie kann daraufhin dem Anlagenbetreiber aktiv einen Vertragsabschluss anbieten.
- Ist dieser Vertrag zustande gekommen, dann übermittelt der Anlagenbetreiber die für den Zugriff auf das Gateway benötigten Credentials (z.B. Benutzername, Passwort).
- Die Leitwarte meldet sich daraufhin mit diesen Daten bei der PowerBridge an (ACSI Service „Associate“) und erhält eine Association ID zurück.
- Nun kann die Leitwarte mit dieser Association ID auf die freigegebenen Daten der PowerBridge zugreifen. Es könnte zunächst einige Stammdaten abfragen (z.B. UUID, Anlagentyp, Standort) und dann die neue Anlage auf der Benutzeroberfläche anzeigen.

5. ebXML Registry

ebXML steht für Electronic Business using XML, d. h. XML für elektronische Geschäftsprozesse.

ebXML ist eine 1999 gestartete, gemeinsame Initiative von UN/CEFACT und OASIS. Ziel der Initiative ist die Entwicklung eines technischen Rahmens zur Nutzung von XML für elektronische Geschäftsprozesse sowie eine Senkung der Eintrittsbarrieren für klein- und mittelständische Unternehmen (KMU) und Entwicklungsländer.

ebXML ist kein Standard an sich, sondern eine Familie verschiedener Standards von UN/CEFACT und OASIS. Zu den ebXML-Standards gehören u. a. die grundlegende technische ebXML-Architektur (ebXML Technical Architecture Specification), ein XML-Schema für Geschäftsprozesse (Business Process Specification Schema), ein Registrierdienst (Registry Services Specification) mit einem Registry Information Model ebRIM und ein Nachrichtendienst (Message Service Specification) [10].

2005 wurde die Version 3 der Spezifikation freigegeben[11]. In RegModHarz wird aus diesem Strauß an Spezifikationen lediglich den Registrierdienst und das dazugehörige Informationsmodell eingesetzt.

Neben der Möglichkeit Dienste zu registrieren, bietet der Registrierdienst auch umfangreiche Abfragemöglichkeiten sowie einen Notification Mechanismus, mit dem ein Anwender bei Neueinträgen, Änderungen oder Löschungen von Objekten benachrichtigt werden kann.

5.1. Das ebXML Informationsmodell (ebRIM)

Zur Ablage von Informationen stellt ebRIM ein erweiterbares Basis-Informationsmodell zur Verfügung, das in der folgenden Abbildung gezeigt ist:

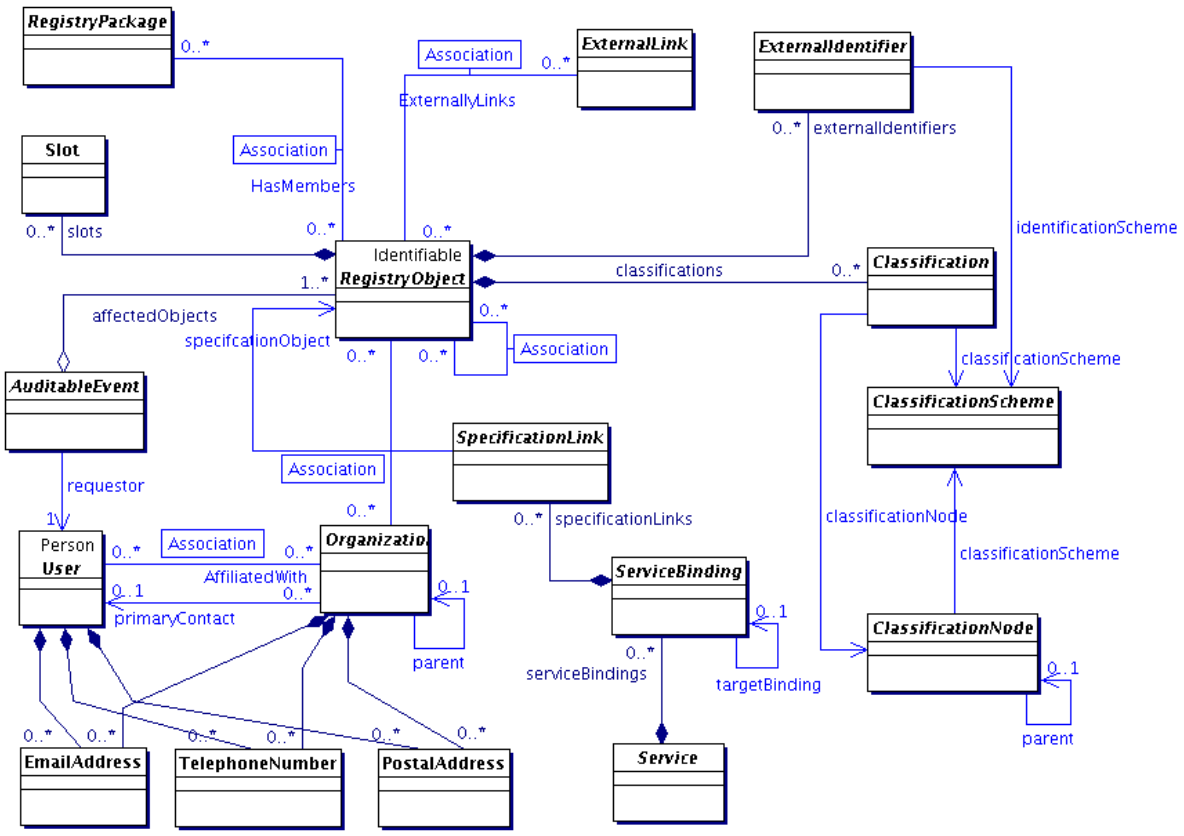

Abbildung 6: Das ebRIM Informationsmodell

[10] [8] ebXML – Wikipedia, 08.10.2010

[11] [9] Cover 16.06.2005

Das Informationsmodell besteht aus folgenden Teilen:

- Die Beschreibung von Personen und Organisationen im unteren linken Teil der Abbildung.
- Die Beschreibung von Services sowie des technischen Zugriffs auf diese (Service, ServiceBinding, SpecificationLink).
- Die Möglichkeit der Klassifizierung von Informationen nach frei definierbaren Schemata (rechter Teil der Abbildung).
- Der Verweis auf externe Ressourcen (ExternalLink)
- Ein generischer Erweiterungsmechanismus (Slot) für die Ablage zusätzlicher Werte. Zusätzlich ist auch die Erweiterung mit beliebigen, selbst definierten Objekten möglich.
- Nicht dargestellt ist in der Abbildung die Möglichkeit, beliebige Dokumente im Repository zu speichern und mit den Objekten der Registry zu assoziieren.
- Beliebige Objekte in der Registry können über Assoziationen miteinander in Verbindung gebracht werden.

Für RegModHarz wird dieses Informationsmodell zunächst auf ein kleines Subset reduziert. Andererseits erlaubt die Vielfältigkeit und Erweiterungsmöglichkeit des Modells in Zukunft einen Ausbau der Rolle der Registry.

5.2. Das RegModHarz Informationsmodell

Prinzipiell stehen mit ebXML die gesamten Möglichkeiten der Registrierung zur Verfügung. Für RegModHarz definieren wir aber zunächst ein Kern-Subset und darauf aufbauend einige optionalen Erweiterungen.

Das Kern-Subset muss drei Anforderungen erfüllen:

- Der Anbieter des Services muss identifizierbar sein
- Der Service selbst muss identifizierbar sein
- Der Weg für den Zugriff auf den Service muss dargestellt sein
- Basisinformationen über den Service müssen verfügbar sein

Dafür werden folgende Objekte aus dem Informationsmodell eingesetzt:

- Organization: Identifiziert den Anbieter des Services
- Service: Identifiziert den Service und ist über eine Assoziation mit dem Anbieter des Services verbunden
- ServiceBinding: Spezifiziert den Zugriffsweg zum Service
- SpecificationLink: Spezifiziert den Zugriffsweg auf das Logische Device
- ClassificationScheme: definiert eine mögliche Klassifizierung von Objekten, z.B. „DER Typ“
- ClassificationNode: definiert eine konkrete Klasse, z.B. „Windkraftanlage“
- Classification: ordnet einem Service eine Klassifikation zu
- Slot: ermöglicht die Zuordnung freier Key-Value-Paare

Die eindeutige Identifikation sowohl der Organisation als auch des Services und aller anderen Objekte in der Registry erfolgt über eine UUID.

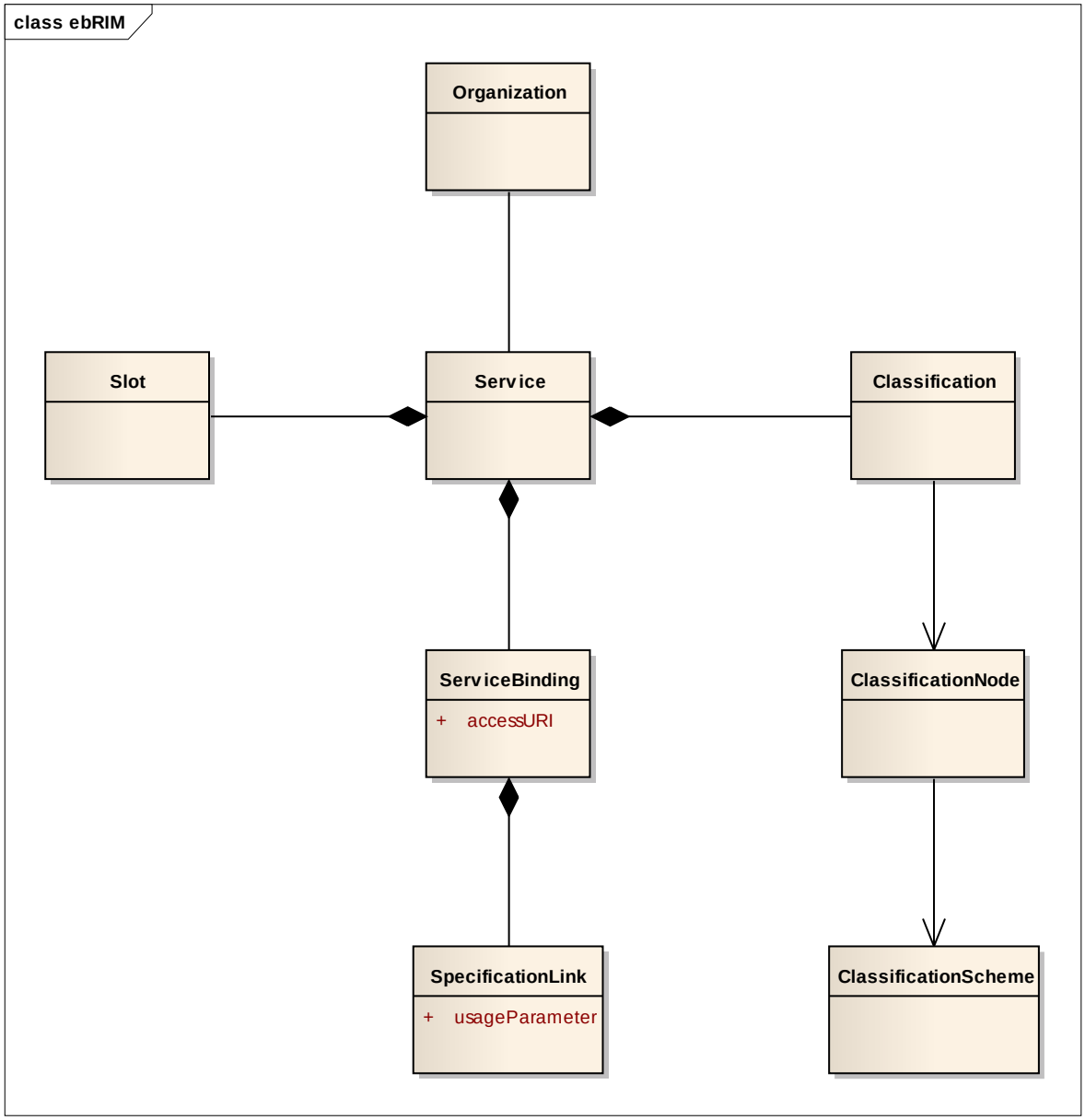

Abbildung 7: Kern-Informationsmodell der Registry

5.3. Klassifizierung

Ein wichtiges Hilfsmittel zur Strukturierung der registrierten Services ist die Klassifizierung der Objekte. ebXML benützt dafür sogenannte ClassificationSchemes, die vom Anwender frei definiert werden können. Ein ClassificationScheme enthält benannte Werte, die ClassificationNodes, die wiederum hierarchisch angeordnet sein können und weitere, untergeordnete ClassificationNodes enthalten können. Beispiele für solche ClassificationSchemes könnten sein:

- Typ der Anlage: PV, Biogas, Wasser, Wind, BHKW, …
- Anschlusspunkt der Anlage: Niederspannung, Mittelspannung, Hochspannung
- Gemeinde des Standortes
- Leistungsbereich: < 1kW, 1-10kW, 10-100kW, …

Wichtig ist zunächst die Klassifizierung nach Anlagentyp. Hierzu wird ein ClassificationScheme und dazugehörige ClassificationNodes gemäß folgender Liste verwendet:

Typ	URN	code
ClassificationScheme	urn:regmodharz:classification:DERType	
ClassificationNode	urn:regmodharz:DERType:none	0
ClassificationNode	urn:regmodharz:DERType:mixed	1
ClassificationNode	urn:regmodharz:DERType:Motor	2
ClassificationNode	urn:regmodharz:DERType:FuelCell	3
ClassificationNode	urn:regmodharz:DERType:Photovoltaic	4
ClassificationNode	urn:regmodharz:DERType:CHP	5
ClassificationNode	urn:regmodharz:DERType:WindPower	101
ClassificationNode	urn:regmodharz:DERType:Biogas	102
ClassificationNode	urn:regmodharz:DERType:PumpStorage	103
ClassificationNode	urn:regmodharz:DERType:StoragePower	104
ClassificationNode	urn:regmodharz:DERType:ElectricCar	105
ClassificationNode	urn:regmodharz:DERType:ControllableLoad	106

Die weiteren in RegModHarz verwendeten ClassificationSchemes werden in einem weiteren Dokument festgelegt.

5.4. Slots

Zur Beschreibung weiterer Eigenschaften eines beliebigen Objektes in der Regsitry kann das Slot-Konzept eingesetzt werden. Es ermöglicht die Zuordnung beliebiger Key-Value-Paare zu einem Objekt in der Registry.

Für RegModHarz wird der Service mit folgendem Slot weiter beschrieben:

Name: MaxWLim

Value: Maximalleistung der Anlage [W]

Der Wert wird aus dem Datenmodell der Anlage entnommen und entspricht dem Wert von DRCT1.MaxWLim. Kann der Wert nicht aus dem Datenmodell ermittelt werden, so ist der Slot nicht vorhanden oder enthält den Wert 0.0.

5.5. Registrierung

Die PowerBridge sendet bei jedem Hochlauf, bei einer Änderung der Kommunikationsverbindung oder auf expliziten Wunsch des Anwenders einen „Submit Objects“ Request an die Registry und übergibt dabei die Daten, die in die Registry eingetragen werden sollen. Durch Konfiguration der PowerBridge kann eingestellt werden, ob die Registrierung automatisch erfolgt oder ob sie über die Benutzeroberfläche des Gateways angestoßen werden muss.

5.6. Benachrichtigung

Um über Änderungen der Daten in der Registry informiert zu werden, kann ein Anwender ein Subscription Objekt in die Registry eintragen. Dieses Objekt enthält im Wesentlichen:

- Den Verweis auf eine Suchanfrage in Form eines SQL-Ausdrucks und
- eine Benachrichtigungsaktion.

Sobald in der Registrierung ein Ereignis auftritt, auf das die Suchanfrage passt, wird der Anwender über die festgelegte Benachrichtigungsaktion informiert. Dies kann entweder eine e-Mail oder ein Web Service Aufruf sein.

Inhalt der Benachrichtigung ist entweder eine Liste der UUIDs der betroffenen Objekte oder aber direkt der Inhalt der entsprechenden Objekte. Weitere über Assoziationen verbundene Objekte werden jedoch nicht mitgeliefert und müssen bei Bedarf über eine entsprechende Abfrage ermittelt werden.

In RegModHarz benutzen wir die Benachrichtigung über Web Services. Dazu muss der Anwender einen NotificationListener Service implementieren. Ein Code Beispiel dazu findet sich in Kapitel 7.3.

6. Spezifische Festlegungen für RegModHarz

Für den konkreten Einsatz der in dieser Spezifikation beschriebenen Protokolle und Schnittstellen ist es erforderlich, weitere einschränkende Festlegungen zu treffen, die den spezifischen Erfordernissen des Projektes entsprechen. Dieses Kapitel beschreibt nun die in den Datenmodellen spezifizierten DataSets, die mit ACSI-Aufrufen gemäß Kapitel 2.2.4 angesprochen werden können. Es beschreibt weiterhin die Verwendung von Fahrplänen oder direkten Steuerkommandos für die Beeinflussung angeschlossener dezentraler Energieanlagen.

6.1. DataSets für RegModHarz

Um den Zugriff der Leitwarte auf die Daten in der PowerBridge zu vereinfachen, werden in allen PowerBridges drei DataSets konfiguriert:

- **Description:** Beinhaltet prinzipielle beschreibende Grunddaten der dezentralen Energieanlage
- **Measure:** Beinhaltet aktuelle Messwerte der dezentralen Energieanlage
- **DVER:** Beinhaltet einen Satz von Messwerten und Einstellgrößen aus dem Logical Node DVER, der die Variabilität einer dezentralen Energieanlage beschreibt

Das erste DataSet wird von der Leitwarte bei der Kontaktaufnahme abgefragt. Die beiden anderen werden zyklisch abgefragt, um stets einen aktuellen Satz von Messwerten und Zustandsvariablen zu erhalten.

Die Definition der DataSets ist wie folgt:

```
<scl:DataSet name="Description">
    <scl:FCDA fc="DC" lnClass="LPHD" lnInst="1" doName="PhyNam" daName="name" />
    <scl:FCDA fc="DC" lnClass="LPHD" lnInst="1" doName="PhyNam" daName="latitude" />
    <scl:FCDA fc="DC" lnClass="LPHD" lnInst="1" doName="PhyNam" daName="longitude" />
    <scl:FCDA fc="DC" lnClass="LPHD" lnInst="1" doName="PhyNam" daName="mrID" />
    <scl:FCDA fc="SP" lnClass="DRCT" lnInst="1" doName="MaxWLim" daName="setMag" />
    <scl:FCDA fc="SP" lnClass="DRCT" lnInst="1" doName="DERtyp" daName="setVal" />
</scl:DataSet>
<scl:DataSet name="Measure">
    <scl:FCDA fc="ST" lnClass="LLN" lnInst="0" doName="Health" daName="stVal" />
    <scl:FCDA fc="ST" lnClass="LLN" lnInst="0" doName="Health" daName="q" />
    <scl:FCDA fc="ST" lnClass="LLN" lnInst="0" doName="Health" daName="t" />
    <scl:FCDA fc="MX" lnClass="MMXU" lnInst="1" doName="TotW" daName="mag" />
    <scl:FCDA fc="MX" lnClass="MMXU" lnInst="1" doName="TotW" daName="q" />
    <scl:FCDA fc="MX" lnClass="MMXU" lnInst="1" doName="TotW" daName="t" />
    <scl:FCDA fc="ST" lnClass="MMTR" lnInst="1" doName="SupWh" daName="actVal" />
    <scl:FCDA fc="ST" lnClass="MMTR" lnInst="1" doName="SupWh" daName="q" />
    <scl:FCDA fc="ST" lnClass="MMTR" lnInst="1" doName="SupWh" daName="t" />
</scl:DataSet>
<scl:DataSet name="DVER">
    <scl:FCDA fc="SP" lnClass="DVER" lnInst="1" doName="ContPwrCtl" daName="setVal" />
    <scl:FCDA fc="SP" lnClass="DVER" lnInst="1" doName="PwrCtlSteps" daName="numPts" />
    <scl:FCDA fc="SP" lnClass="DVER" lnInst="1" doName="PwrCtlSteps" daName="crvPts" />
    <scl:FCDA fc="SP" lnClass="DVER" lnInst="1" doName="MaxWExp" daName="setMag" />
    <scl:FCDA fc="SP" lnClass="DVER" lnInst="1" doName="MaxWImp" daName="setMag" />
    <scl:FCDA fc="SP" lnClass="DVER" lnInst="1" doName="StartTm" daName="setTm" />
    <scl:FCDA fc="SP" lnClass="DVER" lnInst="1" doName="TargetTm" daName="setTm" />
    <scl:FCDA fc="SP" lnClass="DVER" lnInst="1" doName="TargetCont" daName="setMag" />
    <scl:FCDA fc="SP" lnClass="DVER" lnInst="1" doName="MinStorLvl" daName="numPts" />
    <scl:FCDA fc="SP" lnClass="DVER" lnInst="1" doName="MinStorLvl" daName="time" />
    <scl:FCDA fc="SP" lnClass="DVER" lnInst="1" doName="MinStorLvl" daName="val" />
    <scl:FCDA fc="SP" lnClass="DVER" lnInst="1" doName="StorCap" daName="setMag" />
    <scl:FCDA fc="MX" lnClass="DVER" lnInst="1" doName="StorCont" daName="mag" />
    <scl:FCDA fc="MX" lnClass="DVER" lnInst="1" doName="StorCont" daName="q" />
    <scl:FCDA fc="MX" lnClass="DVER" lnInst="1" doName="StorCont" daName="t" />
    <scl:FCDA fc="SP" lnClass="DVER" lnInst="1" doName="StorEff" daName="setMag" />
    <scl:FCDA fc="SP" lnClass="DVER" lnInst="1" doName="StorInflow" daName="setMag" />
    <scl:FCDA fc="SP" lnClass="DVER" lnInst="1" doName="StorInflowCrv" daName="numPts" />
    <scl:FCDA fc="SP" lnClass="DVER" lnInst="1" doName="StorInflowCrv" daName="time" />
    <scl:FCDA fc="SP" lnClass="DVER" lnInst="1" doName="StorInflowCrv" daName="val" />
    <scl:FCDA fc="SP" lnClass="DVER" lnInst="1" doName="StorInflowCrv" daName="rmpTyp" />
</scl:DataSet>
```

Für weitere Informationen bezüglich der Datenmodelle sei auf die Spezifikation der Datenmodelle[12] verwiesen.

6.2. Beeinflussung von Energieanlagen

Über die PowerBridge können geeignete dezentrale Energieanlagen auf zwei Weisen beeinflusst werden:

- Durch direkte Steuerung der Wirk- und/oder Blindleistung
- Durch einen Fahrplan für die Wirkleistung

Die direkte Steuerung erfolgt dabei über die Datenobjekte **DRCC1.OutWSet** (Wirkleistung) bzw. **DRCC1.OutVarSet** (Blindleistung). Dabei wird jeweils das Datenattribut **ctlVal** gesetzt.

Zur Quittierung spiegelt die PowerBridge den gesetzten Wert im Datenattribut **mxVal** des Datenobjektes. Liegt dieser Wert jedoch außerhalb des von der PowerBridge angebotenen Bereichs von **DVER1.MaxWImp** bis **DVER1.MaxWExp**, so wird in **mxVal** lediglich ein Wert angegeben, der der jeweiligen Bereichsgrenze entspricht. Auf diese Weise kann die Leitwarte noch einmal überprüfen, ob die angeschlossene Anlage bereit ist, den eingestellten Wert auch tatsächlich zu realisieren.

Die Steuerung über einen Fahrplan erfolgt über das Datenobjekt **DSCH1.SchdAbsTm** und einige weitere, nachfolgend beschriebene Objekte. Das Vorgehen ist wie folgt:

1. Übertragen des Fahrplans (Liste von Zeit-/Leistungswertepaaren) an das Datenobjekt **DSCH1.SchdAbsTm**
2. Aktivieren des Fahrplans, indem das Datenattribut **DSCC1.ActWSchd.ctlVal** auf 1 gesetzt wird
3. Die Aktivierung wird zunächst in **DSCC1.ActWSchd.stVal** quittiert. Zusätzlich wird auch der Fahrplan im Attribut **DSCH1.SchdSt.stVal** als aktiv gekennzeichnet.

Der Fahrplan muss explizit wieder deaktiviert werden, bevor er geändert wird.

Auch wenn die Steuerung der Anlage über einen Fahrplan erfolgt, überprüft die PowerBridge die Einhaltung der oben genannten Bereichsgrenzen. Der jeweils aktuell aus dem Fahrplan entnommene Leistungswert wird nach **DRCC1.OutWSet.ctlVal** übertragen. Der von der PowerBridge akzeptierte Wert kann aus **DRCC1.OutWSet.mxVal** entnommen werden. Die tatsächliche Reaktion der Energieanlage kann über die Messwerte im Logical Node MMXU beobachtet und kontrolliert werden.

[12] [3] RegModHarz

7. Code Beispiele

7.1. IEC 61850 Web Service Client

7.1.1. An- und Abmelden: Associate und Release

Programmbeispiel Associate:

```
private String associate() {
    AssociateRequest req = new AssociateRequest();
    req.setMaxMessageSize(4096);
    req.setUserName("username");
    req.setPassword("pasword");
    req.setUUID(UUID.randomUUID().toString());

    AssociateResponse resp = service.associate(req);

    if (resp.getServiceError() != null) {
        // TODO error handling
    }

    return resp.getAssocID();
}
```

Programmbeispiel Release: Die Fehlerbehandlung wird nicht erneut gezeigt. Zu sehen ist auch der in Kapitel 2.2.1 angesprochene Fehler im WSDL File: das Ergebnis ist eine AbortResponse statt einer ReleaseResponse.

```
private void release(String associd) {
    ReleaseRequest req = new ReleaseRequest();
    req.setAssocID(associd);
    req.setUUID(UUID.randomUUID().toString());

    AbortResponse resp = service.release(req);
}
```

7.1.2. Verzeichnisdienst: GetServerDirectory

Programmbeispiel GetServerDirectory:

```
private List<String> getServerDirectory(String associd) {
    GetServerDirectoryRequest req = new GetServerDirectoryRequest();
    req.setAssocID(associd);
    req.setUUID(UUID.randomUUID().toString());
    req.setObjClass(TObjectClass.LD);

    GetServerDirectoryResponse resp = service.getServerDirectory(req);

    return resp.getLDRef();
}
```

7.1.3. Verzeichnisdienste: GetLogicalDeviceDirectory

Programmbeispiel GetLogicalDeviceDirectory:

```
private List<String> getLogicalDeviceDirectory(String associd, String ld) {
    GetLogicalDeviceDirectoryRequest req
        = new GetLogicalDeviceDirectoryRequest();
    req.setAssocID(associd);
    req.setUUID(UUID.randomUUID().toString());
    req.setLDRef(ld);

    GetLogicalDeviceDirectoryResponse resp
        = service.getLogicalDeviceDirectory(req);

    return resp.getLNRef();
}
```

7.1.4. Verzeichnisdienste: GetLogicalNodeDirectory

Programmbeispiel GetLogicalNodeDirectory:

```
private List<String> getLogicalNodeDirectory(String associd, String ln) {
    GetLogicalNodeDirectoryRequest req = new GetLogicalNodeDirectoryRequest();
    req.setAssocID(associd);
    req.setUUID(UUID.randomUUID().toString());
    req.setLNRef(ln);
    req.setIEMcls(TIEMcls.DATA);

    GetLogicalNodeDirectoryResponse resp = service.getLogicalNodeDirectory(req);

    return resp.getDATAname();
}
```

7.1.5. Daten lesen: GetDataValues

Programmbeispiel GetDataValues:

```
private List<TDataAttributeValue> getDataValues(
    String associd, String dataref, TFC fc) {
    GetDataValuesRequest req = new GetDataValuesRequest();
    req.setAssocID(associd);
    req.setUUID(UUID.randomUUID().toString());

    TFcdFcdaType ref = new TFcdFcdaType();
    ref.setRef(dataref);
    ref.setFC(fc);

    req.setRef(ref);

    GetDataValuesResponse resp = service.getDataValues(req);

    return resp.getDataAttrVal();
}

private MV getMeasuredValue(String associd, String dataref) {
    // Helper method to extract a measured value from a
    // list of TDataAttributes

    List<TDataAttributeValue>  data = getDataValues(associd, dataref, TFC.MX);
    return new MV(dataref, data);
}
```

7.1.6. Daten schreiben: SetDataValues

Programmbeispiel SetDataValues:

```
private void setDataValues(
    String associd, String dataref, TFC fc, List<TDataAttributeValue> data) {
    SetDataValuesRequest req = new SetDataValuesRequest();
    req.setAssocID(associd);
    req.setUUID(UUID.randomUUID().toString());
    TFcdFcdaType fcd = new TFcdFcdaType();
    fcd.setFC(fc);
    fcd.setRef(dataref);
    req.setRef(fcd);
    req.getDataAttrVal().addAll(data);

    SetDataValuesResponse resp = service.setDataValues(req);
}

private void setING(String associd, ING ing) {
    // Helper method to set an Integer setpoint
    List<TDataAttributeValue> data = new ArrayList<TDataAttributeValue>();
    String dataref = ing.getDataRef();
    ing.addTo(dataref, data);

    setDataValues(associd, dataref, TFC.SP, data);
}
```

7.1.7. Datengruppe lesen: GetDataSetValues

Programmbeispiel GetDataSetValues:

```
private List<TDataAttributeValue> getDataSetValues(
    String associd, String datasetref) {
    GetDataSetValuesRequest req = new GetDataSetValuesRequest();
    req.setAssocID(associd);
    req.setUUID(UUID.randomUUID().toString());
    req.setDSRef(datasetref);

    GetDataSetValuesResponse resp = service.getDataSetValues(req);

    return resp.getDataAttrVal();
}
```

7.1.8. Ereignisse beobachten: AddSubscription

Programmbeispiel AddSubscription:

```
private void addSubscription(
    String associd, String rcbref, String rptid, String datasetref) {
    AddSubscriptionRequest req = new AddSubscriptionRequest();
    req.setAssocID(associd);
    req.setUUID(UUID.randomUUID().toString());
    req.setRCBRef(rcbref);
    req.setRCBType(TRCBType.BRCB);
    req.setRptID(rptid);

    TTrgCond trgCond = new TTrgCond();
    trgCond.setDchg(true);
    req.setTrgOp(trgCond);
    req.setDatSet(datasetref);

    AddSubscriptionResponse resp = service.addSubscription(req);
}
```

7.1.9. Ereignisse beobachten: RemoveSubscription

Programmbeispiel RemoveSubscription:

```
private void removeSubscription(String associd, String rcbref) {
    RemoveSubscriptionRequest req = new RemoveSubscriptionRequest();
    req.setAssocID(associd);
    req.setUUID(UUID.randomUUID().toString());
    req.setRCBRef(rcbref);

    RemoveSubscriptionResponse resp = service.removeSubscription(req);
}
```

7.1.10. Ereignisse abfragen: Report

Programmbeispiel Report:

```
private void report(String associd, String rcbref) {
    ReportRequest req = new ReportRequest();
    req.setAssocID(associd);
    req.setUUID(UUID.randomUUID().toString());

    DatatypeFactory datatypeFactory;

    try {
        datatypeFactory = DatatypeFactory.newInstance();
    } catch (DatatypeConfigurationException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
        return;
    }

    Duration maxRespTime = datatypeFactory.newDuration(60000);
    req.setMaxResponseTime(maxRespTime );

    Duration minRespTime = datatypeFactory.newDuration(1000);
```

```
        req.setMinResponseTime(minRespTime);

        while (true) {
            ReportResponse resp = service.report(req);

            List<TReportFormat> reports = resp.getReportFormat();

            for (TReportFormat report : reports) {
                // TODO handle the reported event
            }

            Duration minRequestTime = resp.getMinRequestTime();
            Date date = new Date();

            try {
                Thread.sleep(minRequestTime.getTimeInMillis(date));
            } catch (InterruptedException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
    }
```

Das oben stehende Beispiel realisiert eine Endlosschleife für das Pollen von Reports. Zwischen den einzelnen Abfragen wird mindestens die minRequestTime gewartet, die der Server angefordert hat. Als maximale Antwortzeit werden 60 sec vorgegeben. Stehen innerhalb dieser Zeit keine Report Daten zur Verfügung, so gibt der Aufruf eine leere Liste zurück.

7.2. SOAP/HTTPS

Transportiert werden die Web Service Aufrufe über SOAP und https. Dies ermöglicht die Verwendung standardisierter Web Service Clients, wie sie z.B. im Java 6 enthalten sind. Die Kenntnis der URL des Services reicht dann aus, um die Methoden des Services aufzurufen.

Die URL der IEC 61400-Schnittstelle der PowerBridge hat folgenden Aufbau:

```
http://<hostname>:<port>/61400?wsdl        oder
https://<hostname>:<port>/61400?wsdl
```

Unter dieser Adresse würde man mit einem Web-Browser zunächst die WSDL des Services herunterladen können. Auch der in Java 6 enthaltene Client benutzt zunächst diese URL, um die WSDL zu erhalten und daraus weitere Informationen zum Service zu entnehmen.

Mit der Service URL sowie dem Namespace des Services kann man auf Client-Seite einen Service Port erzeugen, der dann für Web Service Aufrufe benutzt wird. Ein Beispiel dazu zeigt der folgende Code-Ausschnitt:

```
    private ServicePortType getService() {
        // create connection to Web Service
        URL wsdl;

        try {
            wsdl = new URL("http://<host>:<port>/61400?wsdl");
        } catch (MalformedURLException e) {
            e.printStackTrace();
            return null;
        }

        IECXMLService locator;
        ServicePortType srv;

        try {
            locator = new IECXMLService(wsdl, new QName(
                    "http://iec.ch/61400/ews/1.0/", "IECXMLService"));
            srv = locator.getIECXMLServicePort();
        } catch (WebServiceException e) {
            e.printStackTrace();
            return null;
        }

        return srv;
    }
```

Das zurückerhaltene `ServicePortType` Objekt dient nun für die Aufrufe des Services, wie in den Beispielen im vorhergehenden Kapitel gezeigt.

Wird eine https-URL und Port benutzt, so ist lediglich der Protokolltyp „https:“ statt „http:“ zu benutzen. Damit der Client den Server-Zertifikaten der PowerBridge vertraut, muss außerdem das CA Zertifikat der Certificate Authority zum Trust Store des Clients hinzugefügt werden.

In der Praxis gibt es dabei bei einer Standard Sun Java Installation zwei Wege:

- Man fügt das CA Zertifikat mit dem `keytool` zum Standard Trust Store hinzu. Dieser wird mit der Java JRE mitgeliefert und liegt dort im Verzeichnis jre6/lib/security mit dem Namen cacerts
- Man verwendet einen getrennten Trust Store und macht diesen der virtuellen Maschine über folgende Parameter bekannt:

```
-Djavax.net.ssl.trustStore=<filename> -Djavax.net.ssl.trustStorePassword=<passwd>
```

Die erste Vorgehensweise hat den Nachteil, dass die Datei cacerts bei jedem Update der Java Laufzeitumgebung überschrieben wird, da sie mit dem JRE mitgeliefert wird. Die zweite Vorgehensweise ist daher zu bevorzugen.

7.3. ebXML NotificationListener mit Java 6

Nachfolgend ist ein einfaches Beispiel eines NotificationListener mit Java 6 gezeigt. Die WSDL-Datei NotificationListenerServices.wsdl aus dem Oasis-Standard[13] wurde dazu zunächst mit `wsimport` in Java Klassen umgewandelt. Für das so generierte Interface `NotificationListenerPortType` wurde dann eine Implementierung erstellt, wobei alle Annotationen des Interfaces auch in die Klasse übernommen werden müssen.

```
import javax.jws.Oneway;
import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.jws.WebService;
import javax.jws.soap.SOAPBinding;
import javax.xml.bind.annotation.XmlSeeAlso;
import javax.xml.ws.Endpoint;

import oasis.names.tc.ebxml_regrep.xsd.rim._3.NotificationType;
import your.urn.goes.here.NotificationListenerPortType;

@WebService(
      name = "NotificationListenerPortType",
      targetNamespace = "urn:oasis:names:tc:ebxml-
regrep:wsdl:NotificationListener:interfaces:3.0")
@SOAPBinding(parameterStyle = SOAPBinding.ParameterStyle.BARE)
@XmlSeeAlso( { oasis.names.tc.ebxml_regrep.xsd.rim._3.ObjectFactory.class,
      oasis.names.tc.ebxml_regrep.xsd.rs._3.ObjectFactory.class,
      oasis.names.tc.ebxml_regrep.xsd.query._3.ObjectFactory.class })
public class Notificator implements NotificationListenerPortType {

   /**
    * @param args
    */
   public static void main(String[] args) {
      Endpoint endpoint = Endpoint
            .publish(

   "http://localhost:8081/NotificationListener/notificationListener",
                  new Notificator());
   }

   /**
    * Delivers a Notification from registry to NotificationListener
    *
    * @param body
    */
```

[13] [10] OASIS ebXML Registry TC

```
    @WebMethod(action = "urn:oasis:names:tc:ebxml-
regrep:wsdl:NotificationListener:bindings:3.0:NotificationListenerPortType:onObject
RefsNotification")
    @Oneway
    public void onNotification(
            @WebParam(
                    name = "Notification",
                    targetNamespace = "urn:oasis:names:tc:ebxml-regrep:xsd:rim:3.0",
                    partName = "body")
            NotificationType body) {

        System.out.println("Notification recieved");
    }
}
```

Sehr lange Zeilen sind in obigem Listing umgebrochen. Diese müssen wieder zu einer Zeile zusammengefügt werden.

Die Klasse definiert eine Methode `onNotification`, die aufgerufen wird, wenn eine Benachrichtigung zugestellt werden soll. Der Web Service kann über `Endpoint.publish` wie gezeigt publiziert werden. Er ist dann unter der angegebenen URL erreichbar.

Für den Ablauf des gezeigten Code-Beispiels ist kein zusätzlicher Web Service Container notwendig. Alle notwendigen Klassen sind in Java 6 enthalten.

7.4. Abfrage der Registry

Die Registry kann sehr flexibel über SQL-Anweisungen per SOAP abgefragt werden. Eine typische Abfrage hat folgenden Aufbau:

```
<?xml version="1.0" encoding="UTF-8"?>

<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    <soap:Body>
        <query:AdhocQueryRequest
            xmlns:lcm="urn:oasis:names:tc:ebxml-regrep:xsd:lcm:3.0"
            xmlns:rim="urn:oasis:names:tc:ebxml-regrep:xsd:rim:3.0"
            xmlns:rs="urn:oasis:names:tc:ebxml-regrep:xsd:rs:3.0"
            xmlns:xml="http://www.w3.org/XML/1998/namespace"
            xmlns:query="urn:oasis:names:tc:ebxml-regrep:xsd:query:3.0"
            xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
            <query:ResponseOption returnType="LeafClass" />
            <rim:AdhocQuery
                id="">
                <rim:QueryExpression
                    queryLanguage="urn:oasis:names:tc:ebxml-regrep:QueryLanguage:SQL-
92">
                    ${QUERY}
                </rim:QueryExpression>
            </rim:AdhocQuery>
        </query:AdhocQueryRequest>
    </soap:Body>
</soap:Envelope>
```

8. Literaturverzeichnis

[1] Designing the architecture for Web services - slide "Service-Oriented Architecture" (2003). W3C. Online verfügbar unter http://www.w3.org/2003/Talks/0521-hh-wsa/slide5-0.html, zuletzt aktualisiert am 22.05.2003, zuletzt geprüft am 22.12.2010.

[2] Ratz, Markus (2010): DKE Normungsroadmap SmartGrid. DKE. Online verfügbar unter http://www.e-energy.de/documents/DKE_Roadmap_Smart_Grid_230410_Deutsch.pdf, zuletzt aktualisiert am 13.04.2010, zuletzt geprüft am 14.01.2011.

[3] RegModHarz: IEC61850-basierte Datenmodelle. RMH_AP2.1_D0149_V03.doc. RegMod-Harz.

[4] OASIS ebXML Registry TC. Online verfügbar unter http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=regrep, zuletzt geprüft am 21.12.2010.

[5] IEC 61400-25-4, 28.08.2008: Wind turbines - Part 25-4: Communications for monitoring and control of wind power plants - Mapping to communication profile.

[6] RegModHarz: Algorithmen im IKT Gateway. RMH_AP3.1_D0210_V01.doc. RegModHarz.

[7] IEC 61850-6, 17.12.2009: Communication networks and systems for power utility automation - Part 6: Configuration description language for communication in electrical substations related to IEDs.

[8] ebXML – Wikipedia (2010). Online verfügbar unter http://de.wikipedia.org/wiki/EbXML, zuletzt aktualisiert am 08.10.2010, zuletzt geprüft am 20.12.2010.

[9] Cover, Robin (2005): Cover Pages: OASIS Approves ebXML Registry Version 3.0 Committee Draft for Public Review. OASIS, Organization for the Advancement of Structured Information Standards. Online verfügbar unter http://xml.coverpages.org/ni2005-02-14-a.html, zuletzt aktualisiert am 16.06.2005, zuletzt geprüft am 19.12.2010.

[10] OASIS ebXML Registry TC. Online verfügbar unter http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=regrep, zuletzt geprüft am 22.12.2010.