CARL VON OSSIETZKY universität OLDENBURG

DEPARTMENT FÜR INFORMATIK
SYSTEMSOFTWARE UND VERTEILTE SYSTEME

Entwurf und Implementierung eines Sensornetzwerk-Monitors

Bachelorarbeit

15. September 2010

Carolin Wiechert
Akazienallee 4a
49661 Cloppenburg

Erstprüfer Prof. Dr.-Ing. Oliver Theel
Zweitprüfer Dipl.-Inform. Felix J. Oppermann

In der vorliegenden Arbeit wird ein System vorgestellt, das Daten aus drahtlosen Sensornetzwerken verarbeitet und zur Verfügung stellt. Dabei lässt es sich durch die leichte Anpassbarkeit für jedes Anwendungsgebiet nutzen. Eine benutzerfreundliche Oberfläche bietet eine Reihe von Funktionen mit denen der Nutzer die Daten unterschiedlich darstellen kann.

This paper presents a system which can work with data from wireless sensor networks and provides them for the user. It can be used in every application area, as it's easily adaptable. A userfriendly interface offers a set of features for the different display of the data.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

Sensornetzwerke finden in immer mehr Gebieten Anwendung. Einzelne Komponenten werden jedes Jahr kleiner und günstiger und somit interessanter für Bereiche wie beispielsweise das Gesundheitswesen, die Industrie oder in unterschiedlichen Lebensräumen [CES04]. Um die vom Sensornetzwerk erfassten Daten zugänglich zu machen, werden Benutzeroberflächen benötigt, die die Daten in einer geeigneten Form darstellen. Viele Sensornetzwerksysteme haben dafür ein speziell auf sie zugeschnittenes Graphical User Interface (GUI), das jedoch einen zusätzlichen Entwicklungs- und Implementierungsaufwand bedeutet.
Die vorliegende Arbeit stellt ein System vor, das Daten aus Sensornetzwerken verarbeitet und dem Nutzer bereitstellt. Dabei kann es ohne großen Aufwand an die unterschiedlichen Anwendungsgebieten angepasst werden. Das System stellt kein Sensornetzwerk zur Verfügung sondern lediglich eine Möglichkeiten mit diesem über Schnittstellen zu kommunizieren.
Kapitel 2 der Arbeit führt den Begriff des Sensornetzwerks ein und beschreibt einige Anwendungsgebiete genauer um die Vielfältigkeit des Einsatzes zu verdeutlichen. Darüber hinaus werden Projekte vorgestellt, die den gleichen Anspruch, wie das in dieser Arbeit vorgestellte System, erheben, nämlich unabhängig von einem bestimmten Sensornetzwerk zu arbeiten. Diese werden anschließend miteinander verglichen. In Kapitel 3 werden die Anforderungen für das System festgelegt, die als Grundlage für den Entwurf in Kapitel 4 dienen. Kapitel 4 betrachtet den gesamten Aufbau des Systems und die Abhängigkeit einzelner Komponenten in Form von Packages. Dabei wird die Beziehung zwischen Client und Server verdeutlicht. Das folgende Kapitel 5 beschreibt die einzelnen Komponenten detaillierter mit ihren Aufgaben und Funktionen. Dazu werden Klassendiagramme zur Verdeutlichung hinzugezogen. Im anschließenden Teil 6 wird das System auf Fehler geprüft und insbesondere die Anforderungen evaluiert. Das abschließende Kapitel 7 zieht einen Vergleich zwischen dem System und den in Kapitel 2 vorgestellten bestehenden Projekten. Zusätzlich gibt es einen Ausblick in Form von Verbesserungsmöglichkeiten des Systems.

2 Grundlagen

Damit im Folgende von dem gleichen Verständnis eines Sensornetzwerks ausgegangen werden kann, wird dieser mit denen für diese Arbeit wichtigen Aspekten definiert.

2.1 Begriffsdefinition

Ein Sensornetzwerk[1] besteht aus vielen Sensorknoten, die drahtlos miteinander verbunden sind und mit denen Messungen durchgeführt werden [Sto05]. Ein einzelner Sensorknoten beinhaltet einen oder mehrere Sensoren, eine Energiequelle, einen Speicher, einen Prozessor und ein Kommunikationsmittel. Die Sensoren führen die eigentlichen Messungen durch und sind beispielsweise Temperatur- , Licht- und Feuchtigkeitssensoren. Als Energiequelle werden in der Regel Batterien verwendet, es sind aber auch Alternativen wie Solarenergie denkbar. Das Kommunikationsmittel, ein Transceiver, ist für das Senden und Empfangen der gemessenen Daten zuständig. Die Übertragung erfolgt meistens über Funk, andere Möglichkeiten wären Schall oder Laser [RM04]. Die Größe der Knoten kann sich nach Anwendung und Kosten unterscheiden [CES04].
Sensornetzwerke weisen keine feste Struktur auf, da Knoten neu hinzukommen, ausfallen oder versetzt werden können. Jeder Sensorknoten kann mit mehreren anderen Sensorknoten in Funkkontakt stehen und so Informationen hop-by-hop, d.h. von Knoten zu Knoten, weitergeben. Ein Sensornetzwerk kann entweder homogen, d.h. es besteht aus gleichen Knoten, oder heterogen sein, d.h. die Knoten unterscheiden sich hinsichtlich ihrer Komponenten, insbesondere der Art und Anzahl der Sensoren.
Sensornetzwerke können in vielen unterschiedlichen Bereichen wie dem Gesundheitswesen, der Überwachung von unterschiedlichen Lebensräumen und in der Industrie eingesetzt werden. Smart Condo [SCB+09] ist ein Projekt aus dem Gesundheitswesen, das eine Wohnanlage mit einem Sensornetzwerk ausstattet. Dadurch soll älteren Menschen ermöglicht werden ihre Unabhänigkeit zu erhalten und trotzdem die Sicherheit zu haben, dass jemand automatisch alarmiert wird, wenn gesundheitsschädliche Ereignisse auftreten. FireWxNet [HHSH06] überwacht die Wetterbedingungen in Umgebungen mit Waldbränden. Dadurch können Vorhersagen über das Verhalten des Feuers getroffen werden und bessere Sicherheitsvorkehrungen getroffen werden. Ein weiterer Einsatz von Sensornetzwerken ist bei Überwachung von Kühlketten von Lebensmitteln denkbar [RV04]. Durch die permanente Speicherung der Temperaturen kann sichergestellt werden, ob die Kühlkette während des Transports vom Hersteller über den Laden zum Kunden unterbrochen wurde oder nicht.

2.2 Bestehende Projekte

Es wurden bereits einige Systeme entwickelt, die nicht auf ein konkretes Sensornetzwerk zugeschnitten sind, sondern für unterschiedliche eingesetzt werden können. Drei dieser Systeme werden im Folgenden, in Bezug auf ihre Funktionen und ihren Aufbau, betrachte.

[1] Damit ist ein drahtloses Sensornetzwerk gemeint, im Folgenden wird nur der Begriff Sensornetzwerk verwendet. Oft findet sich auch die Abkürzung WSN (wireless sensor network).

2.2.1 MoteView

MoteView [Tur05] [Cro07] ist ein Teil des von Crossbow[2] entwickelten MoteWorks und hat die Aufgabe Daten zu analysieren und darzustellen[3]. In mehreren Registerkarten (Tabs) werden die Daten in einer Tabelle und als unterschiedliche Graphen angezeigt. Die Tabelle zeigt für jeden Knoten eine Zeile an, die Sensoren werden als Spalten umgesetzt. Der Nutzer hat die Möglichkeit die Daten eines bestimmten Zeitpunkts aufzurufen, indem ein Scrollbalken bewegt wird. Alle vorliegenden Daten können automatisch abgespielt werden. Weitere Darstellungsarten liegen in Form von Histogrammen [4] (Histogram), Streudiagrammen [5] (Scatterplots) und Graphen für einzelne Sensorarten vor. Für diese Anzeige lassen sich Knoten auswählen, die miteinbezogen werden sollen. Bei der Erstellung der Graphen ist zusätzlich eine Auswahl von Sensoren möglich. Darüber hinaus können alle Daten in XML exportiert werden. Bei einer weiteren Exportart in CSV können die Daten auf einen bestimmten Knoten und Zeitraum beschränkt werden. Außerdem hat der Nutzer die Möglichkeit die Graphen als Bild im JPG-Format abzuspeichern. In dem Topology-Tab werden alle Knoten, die per Drag and Drop[6] verschoben werden können, und ihre Verbindungen zueinander auf einer Ebene angezeigt. Bei dem Drüberfahren mit dem Mauszeiger über einen Knoten werden die Werte der einzelnen Sensoren zu dem auf der Leiste ausgewählten Zeitpunkt an der rechten Fensterseite angezeigt. Ein Bild in unterschiedlichen Formaten lässt sich als Gebäudeplan[7] im Hintergrund hochladen. Darüber kann eine Schicht gelegt werden, die beispielsweise die Temperatur farblich visualisiert. Knoten können zur Benutzeroberfläche hinzugefügt oder von dieser entfernt werden. MoteWorks setzt sich aus drei Ebenen zusammen:

- Die Sensorschicht (sensor tier) enthält das Sensornetzwerk mit dem von Crossbow entwickelten Knoten und läuft mit XMesh, der ebenfalls von Crossbow entwickelten Software für Sensornetzwerke.
- Die Server-Schicht (server tier) ist für das automatische Erfassen und Übersetzen (Parsen) der Messdaten sowie deren Abspeicherung in einer relationalen Datenbank zuständig. Die verwendete Software ist das von Crossbow entwickelte XServe.
- Die Client-Schicht (client tier) verwaltet und visualisiert die Daten für den Nutzer und besteht aus dem schon erwähnten MoteView. MoteView wird in vier weitere Ebenen unterteilt, auf die hier nicht eingegangen wird, da sie für den weiteren Verlauf der Arbeit irrelevant sind.

2.2.2 jWebDust

jWebDust [CMN05] ist eine webbasierte Java-Anwendung, die mit mehreren Sensornetzwerken gleichzeitig arbeiten kann. Diese werden in jWebDust als ein Sensornetzwerk interpretiert, unabhängig davon ob sie heterogen sind und aus welchen Anwendungsgebieten sie stammen. Dementsprechend werden alle Daten in einer Datenbank gespeichert und die einzelnen Sensornetzwerke durch einen Identifikator (Id) gekennzeichnet. Die Knoten registrieren sich selbstständig mit der Information welche Sensoren sie beinhalten bei jWebDust und werden in die Datenbank geschrieben. Knoten können während das System läuft ausgetauscht werden. Anfragen an das Sensornetzwerk werden periodisch oder eventbasiert gestellt (siehe 5.2.4 für Anfragetypen). Der Nutzer kann Knoten festlegen, an die die Anfrage gehen soll. Darüber hinaus können Attribute in der Anfrage angegeben werden, die an das komplette Sensornetzwerk geschickt werden.

[2] Crossbow ist eine Hersteller von Sensorsystemen.

[3] Die hier vorgestellten Funktionen sind nur eine Auswahl aus MoteView, die im Hinblick auf das in dieser Arbeit vorgestellte System ausgewählt wurden.

[4] Histogramme stellen die Häufigkeitsverteilung graphisch dar.

[5] Streudiagramme werden für die graphische Darstellung zweier Merkmale verwendet.

[6] *Drag and Drop* bedeutet, dass ein Element mit dem Mauszeiger von einer Position auf eine andere Position gezogen (drag) und dort fallengelassen (drop) wird.

[7] Ein Gebäudeplan meint eine Ansicht der Umgebung in der das Sensornetzwerk platziert ist.

Von den Knoten, auf die diese Attribute passen, werden Antworten gesendet. Neben diesen Werten werden auch Daten über die Knoten abgespeichert. Besteht keine Verbindung zum Sensornetzwerk, dann werden die Antworten auf Seiten des Netzwerks und die Anfragen auf Seiten des Systems gespeichert. Sobald die Verbindung wieder hergestellt ist, werden diese Informationen ausgetauscht.
Die Multifunktionalität findet sich auch in anderen Bereichen wieder, da mehrere Nutzer das System gleichzeitig verwenden können. jWebDust erlaubt verschiedene Sichten auf das System: die des Administrators, der control center jedes Sensornetzwerks und des Supervisors. Neue Funktionen lassen sich auf allen Ebenen ohne großen Aufwand in das System implementieren.
jWebDust besteht aus dem Sensornetzwerk und dem in Java implementierten Teil des Systems, das die Daten verarbeitet. Das komplette System gliedert sich in fünf Ebenen:

- Die Sensorschicht (sensor tier) enthält ein oder mehrere Sensornetzwerke.
- Die Kontrollschicht (control tier) funktioniert als Gateway[8] zwischen der Datenschicht und der Sensorschicht und leitet die Daten weiter. Dort finden sich alle Kontrollstationen (control center) der einzelnen Sensornetzwerke.
- Die Datenschicht (data tier) speichert die Daten, die aus den Sensornetzwerken kommen, in einem relationalen Datenbanksystem.
- Die Logikschicht (middle tier) ist für die Verarbeitung der Daten zuständig und generiert u.a. die Statistiken. Sie leitet die Strukturen und Daten an die Präsentationsschicht weiter.
- Die Präsentationsschicht (presentation tier) hat die Aufgabe die Eingabe des Nutzers zu sammeln und die Daten aus den Sensornetzwerken darzustellen.

2.2.3 Feit Wireless Sensor Network

Feit Wireless Sensor Network (im Folgenden: Feit) [SD09] ist eine webbasierte Anwendung und kann für die Überwachung unterschiedlicher Lebensräume angepasst werden. Die zuletzt gemessenen Daten werden in einer Tabelle angezeigt, die einen Knoten pro Zeile und die Sensoren als Spalten anzeigt. Zusätzlich können die Daten auch in Echtzeit abgespielt werden. Eine weitere Darstellung erfolgt als Graph in der Chart-Ansicht. Der Nutzer kann einen Sensor und mehrere Knoten auswählen aus denen der Graph gezeichnet wird. Darüber hinaus kann ein Zeitpunkt angegeben werden, ab dem die Werte in den Graphen miteinfließen sollen. Bleibt dieses Feld leer, werden die zuletzt gemessenen Werte verwendet. Der Graph wird dynamisch erzeugt und ändert sich jede Sekunde. Außerdem kann er als Bild abgespeichert werden. In der Statistic-Ansicht werden Maximum, Minimum und Durchschnitt für einen bestimmten Sensor und ausgewählte Knoten berechnet. Auch dort ist eine Zeitbeschränkung mit einem Startzeitpunkt möglich. Die Topology wird mit einem Gebäudeplan und den vorhandenen Knoten erstellt. Knoten beziehungsweise ihre Verbindungen können verschoben, hinzugefügt oder gelöscht werden. Zu den einzelnen Knoten werden Metadaten angezeigt, die vom Nutzer geändert werden können.
Feit zeichnet sich durch leichte Erweiterbarkeit der Funktionen aus. Ein wichtiger Punkt ist außerdem, dass alle mit dem Internet verbundenen Geräte die Anwendung nutzen können.
Das System gliedert sich in drei Ebenen:

- Die Sensorschicht (sensor tier) enthält das Sensornetzwerk und läuft mit der von Crossbow entwickelten Software XMesh.
- Die Server-Schicht (server tier) arbeitet mit dem von Crossbow entwickelten XServe.
- Die Client-Schicht (client tier) wird als Webapplikation umgesetzt und verwendet u.a. Silverlight von Microsoft.

[8] Durch ein *Gateway* können Netzwerke miteinander kommunizieren, die auf unterschiedlichen Protokollen basieren.

2.2.4 Fazit

MoteView und Feit gleichen sich vom Aufbau stark, da sie die gleichen Schichten haben, die die gleichen Aufgaben in dem System erfüllen. Darüber hinaus wird, bis auf die Client-Schicht, die gleiche Software, nämlich die von Crossbow entwickelte, verwendet. jWebDust hingegen gliedert sich in fünf Schichten, auf die die Aufgaben des Systems verteilt sind. Die Verarbeitung und Anzeige der Daten findet beispielsweise in der Logik- und Präsentationsschicht statt, bei den anderen beiden Systemen liegen diese Aufgaben in der Client-Schicht. Alle drei Systeme bieten eine Reihe von Funktionen wobei MoteView die meisten bereitstellt, Feit und jWebDust aber einfach um Funktionalitäten erweitert werden können. Trotz ihrer Vielfältigkeit weisen alle drei Systeme Schwächen auf. Der größte Nachteil an MoteView ist, dass es nur mit den von Crossbow entwickelten Knoten arbeitet und somit nicht ohne Aufwand an alle Sensornetzwerke angepasst werden kann. Hinzu kommt, dass MoteView windowsbasiert und nicht auf anderen Plattformen verwendbar ist[9][Cro07]. Bei jWebDust und Feit ist der größte Mangel, dass beide Programme nicht verfügbar sind. Darüber hinaus arbeitet Feit in der Sensornetzwerk- und Serverschicht mit der von Crossbow entwickelten Software und unterliegt damit den gleichen Beschränkungen wie MoteView. jWebDust legt im Gegensatz zu MoteView und Feit, die eine gut durchdachte Benutzeroberfläche bereitstellen, das Hauptaugenmerk auf die Server- und Sensornetzwerkschicht.

[9] Dies gilt für das Client-Programm, XServe ist in Java geschrieben und somit plattformunabhängig.

3 Anforderungen

Anforderungen sind bestimmte Bedingungen, die ein System erfüllen muss. Dabei wird zwischen funktionalen und nichtfunktionalen Anforderungen unterschieden. Funktionale Anforderungen geben Auskunft über die Interaktion des Systems mit der Umgebung, d.h. mit den Nutzern und anderen Systemen. Dabei ist entscheidend was das System können soll und nicht wie es umgesetzt wird. Nichtfunktionale Anforderungen beschreiben das System in Hinblick auf unterschiedliche Aspekte, die keine Funktionen sind, sondern Aussagen über die Qualität des Systems machen. In Abbildung 3.1 ist ein Überblick über die Anwendungsfälle gegeben, die die Funktionalitäten des Systems beschreiben.

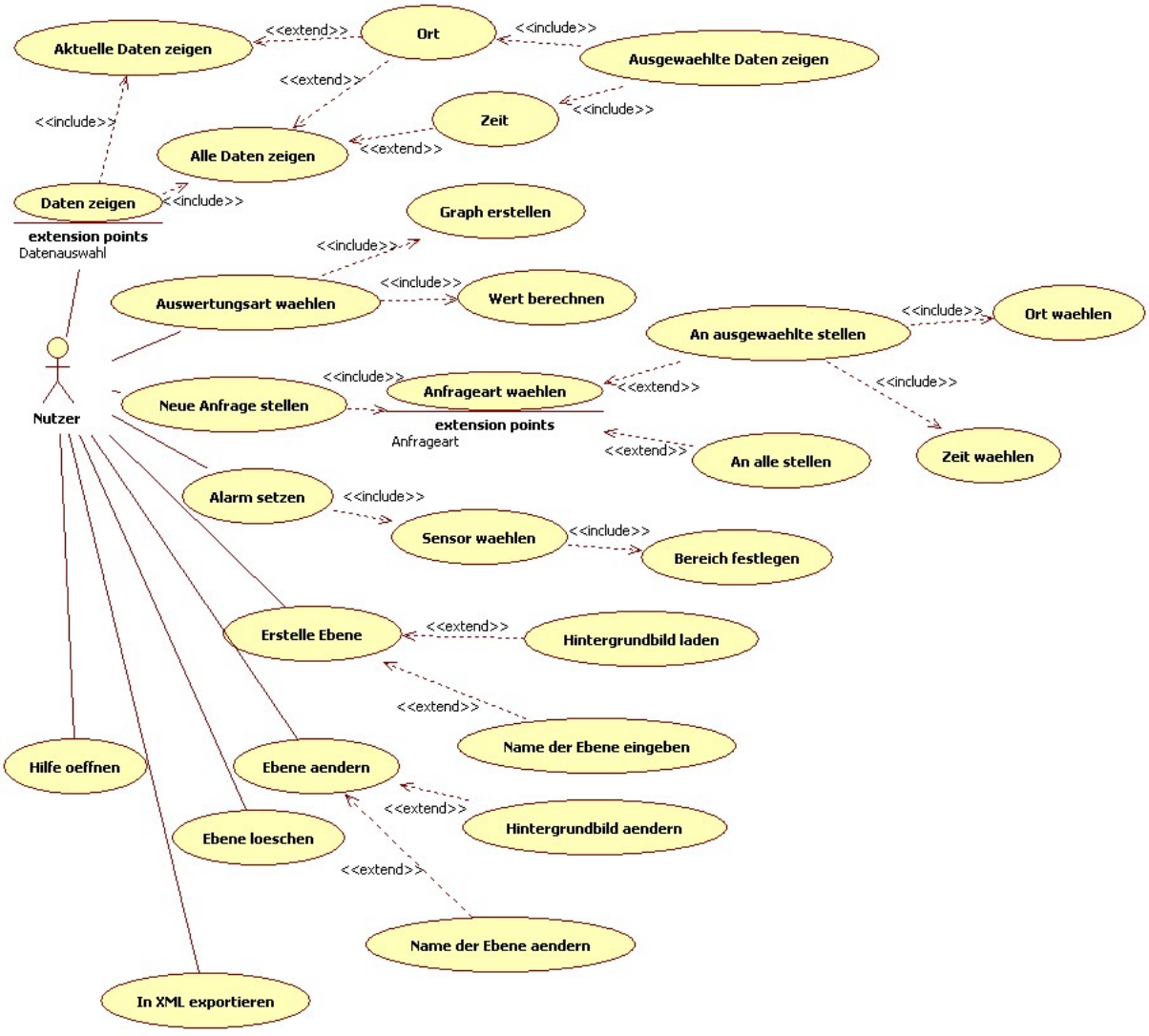

Abbildung 3.1: Anwendungsfalldiagramm

3.1 Funktionale Anforderungen

Die funktionalen Anforderungen sind:

- Anzeige der aktuellen Daten:
  Der Nutzer kann sich die aktuellen Daten, d.h. die zuletzt gemessenen und gespeicherten Werte, anzeigen lassen. Da ein Datensatz neben dem Wert aus dem Ort der Messung und der Zeit der Erfassung besteht, kann die Auswahl auf einen bestimmten Bereich beschränkt werden.

- Anzeige der archivierten Daten:
  Der Nutzer kann sich die archivierten Daten, d.h. alle gespeicherten Werte, anzeigen lassen. Da ein Datensatz neben dem Wert aus dem Ort der Messung und der Zeit der Erfassung besteht, kann die Auswahl auf einen bestimmten Bereich und einen bestimmten Zeitraum beschränkt werden.

- Anfrage an das Sensornetzwerk:
  Der Nutzer kann Anfragen an das Sensornetzwerk senden, um Werte zu erhalten. Dabei kann die Anfrage entweder an eine Auswahl von Sensoren und Knoten oder an das gesamte Sensornetzwerk gestellt werden. Außerdem kann eine Anfrage punktuell sein, d.h. die Daten werden zu einem Zeitpunkt einmal angefordert, oder ein Zeitraum angegeben werden, in dem die Daten in einem bestimmten Abstand gesammelt werden sollen.

- Auswertung der Daten:
  Da die Daten ohne Analyse nur einen begrenzten Aussagewert haben, werden sie zusätzlich ausgewertet. Dazu werden Graphen erstellt sowie Mittel-, Maximal- und Minimalwert berechnet. Der Nutzer hat dort ebenfalls die Möglichkeit die Daten auf Zeit und Ort zu beschränken.

- Alarm bei kritischen Werten:
  Wenn die gemessenen Werte eines Sensors einen kritischen Punkt erreichen, dann wird ein Alarm angezeigt. Den kritischen Bereich legt der Nutzer eigenständig fest.

- Anzeige des Sensornetzwerkzustandes:
  Bei der Anzeige des Sensornetzwerkzustandes kann der Nutzer die Anordnung der Knoten über mehrere Ebenen auf einem Gebäudeplan visualisieren und sich zusätzlich Statusdaten zu den Knoten angezeigen lassen.

- Statusdaten/Status des Netzwerks:
  Zu den einzelnen Knoten werden die Sensoren und ihre zuletzt gemessenen Werte, die benachbarten Knoten und der aktuelle Energiezustand angezeigt.

- Exportieren in XML:
  Der Nutzer hat die Möglichkeit die Daten in das XML-Format zu exportieren. Zusätzlich können die Daten auf Zeit und Ort beschränkt werden. Durch dieses Format werden die Daten strukturiert dargestellt und sind plattformunabhängig.

3.2 Nichtfunktionale Anforderungen

Die nichtfunktionalen Anforderungen sind:

- Flexibilität:
  Das System lässt sich leicht an unterschiedliche Sensornetzwerke anpassen, die aus verschiedenen Anwendungsgebieten stammen.
- Wartung/Anpassbarkeit:
  Der Programmcode wird kommentiert und hält die JavaDoc-Konventionen ein. Außerdem wird das System mit Methoden des Software Engineerings geplant, um es übersichtlich zu halten.
- Bedienbarkeit:
  Die Benutzeroberfläche soll für den Nutzer intuitiv sein und keine lange Einarbeitungszeit erfordern. Aus diesem Grund und da Sensornetzwerke eine große Menge von Daten erfassen, die dem Nutzer präsentiert werden müssen, wird das GUI einfach gehalten. Eine Hilfefunktion soll den Umgang mit dem System unterstützen und Nutzern mit wenig Computererfahrung helfen sich leicht in das System einzuarbeiten.
- Robustheit:
  Alle Benutzereingaben werden auf Fehler geprüft, so dass unzulässige Daten nicht weitergegeben werden.
- Leistungsanforderung:
  Lange Wartezeiten sollen, beispielsweise beim Laden aus der Datenbank oder bei der Verarbeitung von Nutzereingaben, vermieden werden.

4 Entwurf

Sensorznetzwerk-Systeme werden in der Regel in drei Teile aufgeteilt: Client-Schicht, Server-Schicht und Sensornetzwerkschicht [Tur05]. Im vorliegenden System findet die Sensornetzwerkschicht nur am Rande Beachtung, da keine Implementierung durchgeführt wird. Das in dieser Arbeit beschriebene System lässt sich darum als eine Client-Server-Architektur mit Schnittstellen zum Sensornetzwerk beschreiben.
Eine Client-Server-Architektur wird verwendet, wenn Daten zentral in einer Datenbank gespeichert werden. Der Client hat dabei die Aufgabe Benutzereingaben entgegen zu nehmen und Datenbanktransaktionen zu initiieren. Der Server ist dafür zuständig, dass diese Transkationen durchgeführt werden und die Daten vollständig sind [BD04]. Im vorliegenden System findet sich diese Aufgabenteilung wieder.
Abbildung 4.1 gibt einen Gesamtüberblick über das System. Der Server ist für die Verteilung der Daten zuständig. An das Sensornetzwerk gestellte Anfragen werden in das `server.wsn`-Package gegeben und von dort weitergeleitet. Die Antworten aus dem Sensornetzwerk werden in diesem Package an ihre Bestimmungsorte verteilt. Das `server.wsn`-Package enthält außerdem die Schnittstellen für die Kommunikation mit dem Sensornetzwerk. Das `server.database`-Package ist für das Lesen, Schreiben und Ändern der Daten in der Datenbank zuständig. Die Schnittstellen für die Kommunikation zwischen Client und Server liegen in dem `server.interfaces`-Package.
Der Client holt die Daten über den Server und zeigt sie dem Nutzer in geeigneter Form an. Die dort enthaltenen Packages umfassen zusammengehörige Klassen, die die Funktionen des Systems umsetzen.
Durch dieses Vorgehen wird die lose Kopplung und die enge Kohäsion unterstützt. Lose Kopplung meint, dass zwischen einzelnen Subsystemen nur geringe Abhängigkeiten bestehen, um die Auswirkung von Änderungen eines Subsystems auf ein anderes möglichst gering zu halten. Die Schnittstelle zwischen Client und Server sowie Sensornetzwerk und Server sind Beispiele dafür. Durch diese ist es möglich das Sensornetzwerk auszutauschen, was ein angestrebtes Ziel des vorliegenden Systems ist. Ebenso ist es möglich die Datenbank durch eine andere zu ersetzen oder eine andere Benutzeroberfläche zu entwickeln. Eine starke Kohäsion besteht, wenn die Klassen innerhalb eines Subsystems stark voneinander abhängen d.h. wenn sie in einer Assoziationsbeziehung zueinander stehen.

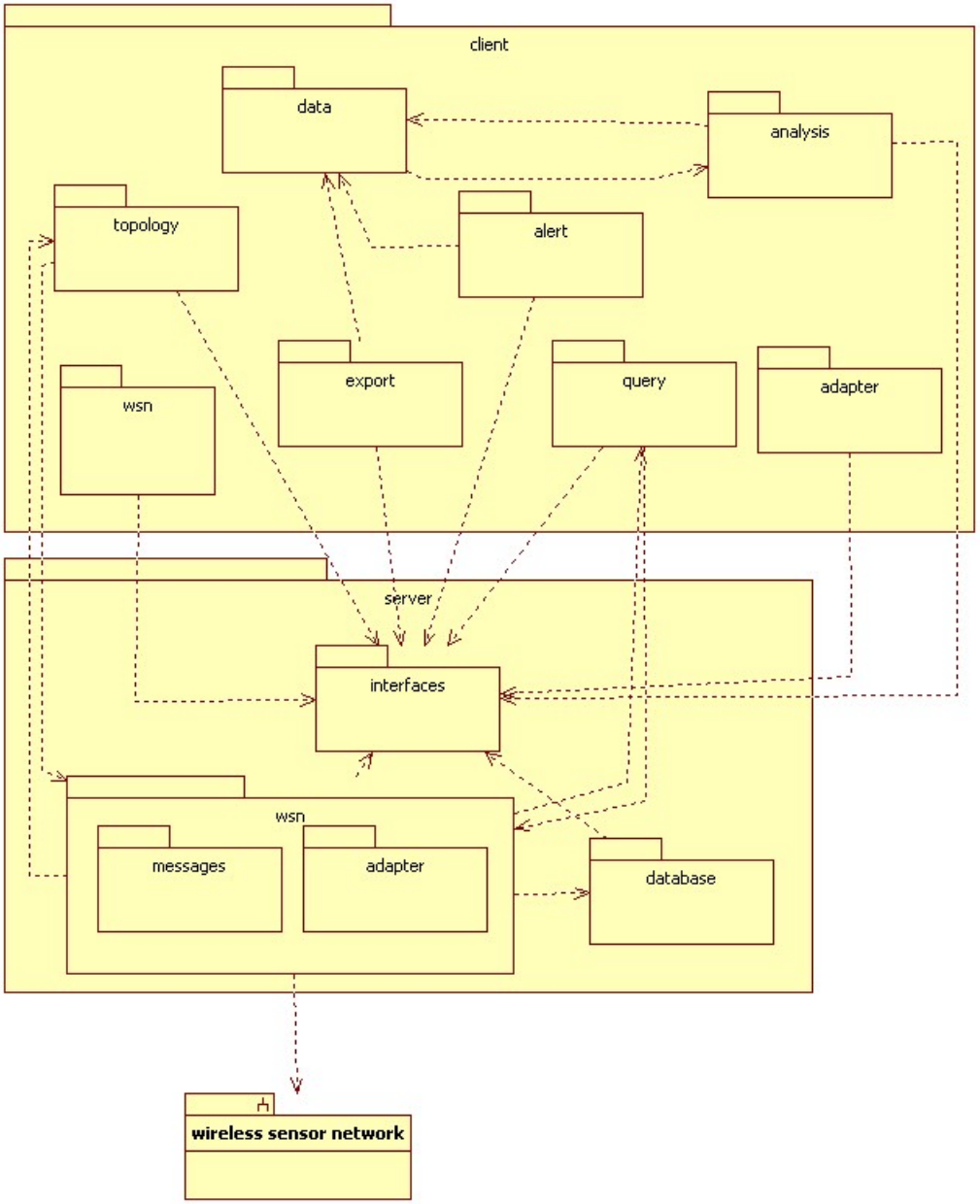

Abbildung 4.1: Gesamtüberblick über das System

5 Feinentwurf und Implementierung

Das vorliegende System wird als Java-Applikation umgesetzt, da dadurch eine Plattformunabhängigkeit gewährleistet werden kann. Die Benutzeroberfläche ist mit der von Java bereitgestellten Oberflächenbibliothek Swing realisiert. Das gesamte System wurde in englischer Sprache entwickelt.

5.1 Server

Wie bereits erwähnt, ist der Server für das Weiterleiten der Daten aus und zu der Datenbank sowie dem Sensornetzwerk zuständig. Das `server`-Package setzt sich aus den drei Teilen Kommunikation zwischen Client und Server, Kommunikation des Systems mit dem Sensornetzwerk und Datenbank zusammen.

5.1.1 Schnittstellen zwischen Client und Server

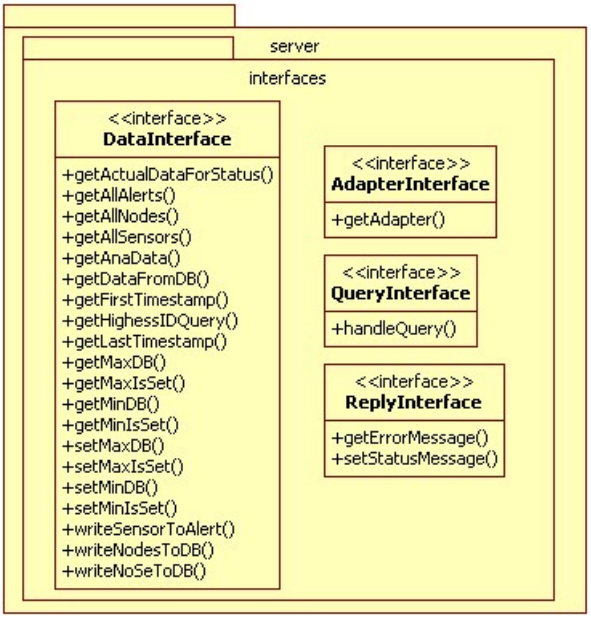

Abbildung 5.1: Package server.interfaces

Im `server.interfaces`-Package (Abbildung 5.1) liegen die Schnittstellen für die Kommunikation zwischen Client und Server. Es werden insgesamt vier Interfaces definiert, um ihre Aufgaben voneinander zu trennen.

- Das `DataInterface` ist dafür zuständig die Anfragen an die Datenbank aus dem Client an den Server weiterzugeben und anschließend die Daten zurück an den Client zu leiten. Dieses Interface wird im `server.database`-Package von den einzelnen Klassen implementiert.
- Das `QueryInterface` sendet die Anfragen, die im Client gemacht werden, an den Server, der diese weiter verarbeitet. Dieses Interface wird von der `WsnAndDb`-Klasse im `server.wsn`-Package implementiert.

- Das `ReplyInterface` gibt die Antworten aus dem Sensornetzwerk, die nicht persistent gespeichert werden, an den Client weiter. Dabei handelt es sich um die Statusdaten, die in der Topology-Ansicht verwendet werden, und die Fehlermeldungen, die im Client in der `Reply`-Klasse des `client.query`-Packages verarbeitet werden. Dieses Interface wird von der `StatusData`-Klasse in dem `client.topology`-Package und der `Reply`-Klasse im `client.query`-Package implementiert.
- Das `AdapterInterface` gibt den im Client ausgewählten und gespeicherten Adapter an den Server. Das Interface wird von der `Adapter`-Klasse im `client.adapter`-Package implementiert.

5.1.2 Sensornetzwerk

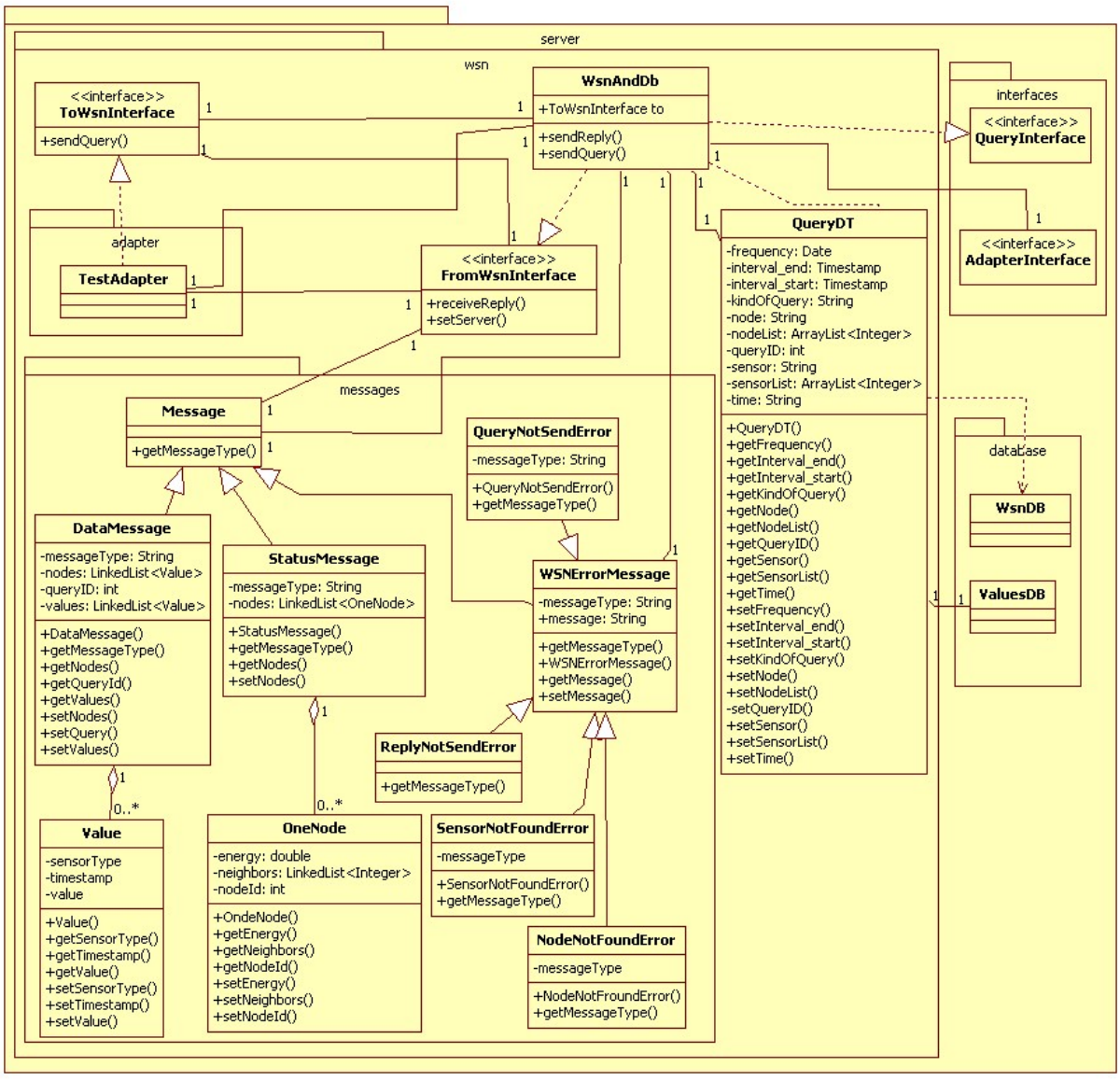

Abbildung 5.2: Package server.wsn

Das server.wsn-Package (Abbildung 5.2) enthält alle das Sensornetzwerk betreffenden Klassen insbesondere die für die Kommunikation mit dem Sensornetzwerk. Diese wird mittels des Delegate-Entwurfsmusters umgesetzt, welches besagt, dass beim Erhalten einer Aufgabe, diese an ein anderes Objekt delegiert wird. Im server.wsn-Package kommt das Delegate-Entwurfsmuster an zwei Stellen zum Einsatz: Wenn eine Anfrage an das Sensornetzwerk gestellt wird und wenn eine Antwort aus dem Sensornetzwerk kommt. Diese Umsetzung wird der anderen Möglichkeit, nämlich dem Polling, vorgezogen. Beim Polling wird nach einem bestimmten Zeitabstand abgefragt ob neue Daten vorliegen und somit unnötig Ressourcen verschwendet. Beim Delegieren hingegen werden Aktionen nur durchgeführt, wenn eine neue Aufgabe vorliegt.
Für die Umsetzung des Delegate-Entwurfsmusters werden zwei Interfaces bereitgestellt: Das ToWsnInterface mit der sendQuery-Methode, das die Anfrage an das Sensornetzwerk weiterleitet, und das FromWsnInterface mit der sendReply-Methode um die Antwort an den Client weiterzugeben.
Die Klasse WsnAndDb ist für die serverseitige Verarbeitung der Sensornetzwerkdaten zuständig. Sie implementiert das FromWsnInterface und hält eine Referenz auf das ToWsnInterface. Die Anfragen aus dem Client werden über das QueryInterface an die WsnAndDb-Klasse gegeben und von dort an das server.database-Package und das Sensornetzwerk weitergeleitet. Letzteres funktioniert mithilfe der ToWsnInterface-Referenz und eines Adapters. Der ausgewählte Adapter wird über das AdapterInterface aus dem Client geholt (siehe 5.2.7). Eine Antwort aus dem Sensornetzwerk wird in der receiveReply-Methode verarbeitet, in der der vorliegende Messagetyp seinem Bestimmungsort zugewiesen wird.
Die QueryDT-Klasse erzeugt für jede Anfrage aus dem Client ein Objekt. Eine Datenanfrage wird mit *data* in der kindOfQuery-Variable gekennzeichnet. Dies ist der Standardwert der QueryDT-Klasse und muss nicht explizit gesetzt werden. Die QueryId wird auf Grundlage der höchsten in der Datenbank gespeicherten QueryId berechnet. Für die Auswahl der Knoten und Sensoren spielen die Schlüsselwörter *all* und *select* eine Rolle. Werden Knoten und Sensoren vom Nutzer festgelegt, dann werden diese als Listen im QueryDT-Objekt abgespeichert und die node- bzw. sensor-Variable auf *select* gesetzt. Soll die Anfrage an das gesamte Sensornetzwerk gesendet werden, wird *all* gesetzt. Wird eine punktuelle Anfrage gewählt, enthalten die Variablen interval_start, interval_end und frequency keine Werte, sie stehen also auf null, andernfalls wird der gewählte Wert der Variable zugewiesen. Diese Abgrenzungen sind auch für die Speicherung in der Datenbank wichtig (siehe 5.1.3). Eine Statusanfrage wird durch *status* in der kindOfQuery-Variable gekennzeichnet. Diese wird an das gesamte Sensornetzwerk gesendet und eine Beschränkung auf Knoten wird im vorliegenden System nicht beachtet.
Der Adapter ist die Verbindungsstelle zu dem eigentlichen Sensornetzwerk und muss an dieses angepasst werden. Damit das System mit dem Adapter, und somit auch mit dem Sensornetzwerk, arbeiten kann, muss dieser das FromWsnInterface implementieren und in dem server.wsn.adapter-Package liegen. Zusätzlich hält der Adapter eine Referenz auf die WsnAndDb-Klasse, die mit der setServer-Methode gesetzt wird. Zur Verdeutlichung findet sich ein Test-Adapter in dem server.wsn.adapter-Package in der Abbildung 5.2 [1].
Das Package server.wsn.messages enthält die Nachrichtentypen, die von dem Adapter für die Antworten genutzt werden. Es wird dabei zwischen drei Arten unterschieden:

- DataMessage:
  Antworten, die den Messwert eines Sensors transportieren, werden über ein DataMessage-Objekt an das System weitergegeben. Die Daten aus dem Sensornetzwerk können zusammengefasst werden, aber auch als einzelne Werte gesendet werden. Da das Sensornetzwerk Antworten schicken kann, ohne damit auf eine Anfrage zu reagieren, steht die QueryId standardmäßig auf 0. Die Werte aus der values-Liste werden allen Knoten aus der nodes-Liste zugeordnet. Ein Value enthält dabei

[1] Jeder Adapter steht in einer Assoziationsbeziehung zu allen Message-Typen. Aufgrund der Übersichtlichkeit wurden diese Verbindungen nicht in das Diagramm eingezeichnet.

den Sensortyp, den Wert sowie den Zeitpunkt der Messung. Damit ist es möglich, Werte aus dem Sensornetzwerk zusammenzufassen, aber auch einzelne Werte an das System zu geben.

- `StatusMessage`:
  Als Antwort wird der aktuelle Status jedes einzelnen Knotens sowie seiner benachbarten Knoten an das System gegeben. Die `StatusMessage` wird im vorliegenden System in der Topology-Ansicht benötigt.

- `WSNErrorMessage`:
  Bei Fehlern aus dem Sensornetzwerk wird eine der `WSNErrorMessage`-Klassen aufgerufen. Neben der allgemeinen Klasse, die eine beliebige Fehlermeldung aufnehmen kann, werden auch vordefinierte Meldungen angeboten: `NodeNotFoundError`, wenn ein Knoten nicht gefunden werden konnte, `SensorNotFoundError`, wenn ein Sensor nicht gefunden werden konnte. Bei beidem kann die Ursache ein ausgefallener Knoten sein. Weitere Fehler sind `ReplyNotSendError` und `QueryNotSendError`.

5.1.3 Datenbank

Die persistente Speicherung der Daten erfolgt in einer MySQL-Datenbank. Für die Kommunikation zwischen dem System und der Datenbank wird ein JDBC-Driver verwendet. Dieser kann mit Java und unterschiedlichen Datenbanken u.a. MySQL arbeiten. Die Entscheidung für die MySQL-Datenbank erfolgte, weil sie eine Open Source Software ist und auf unterschiedlichen Betriebssystemen verwendet werden kann. Die Datenbank kann bei Bedarf durch eine andere ersetzt werden, wobei diese mit JDBC kompatibel sein muss.

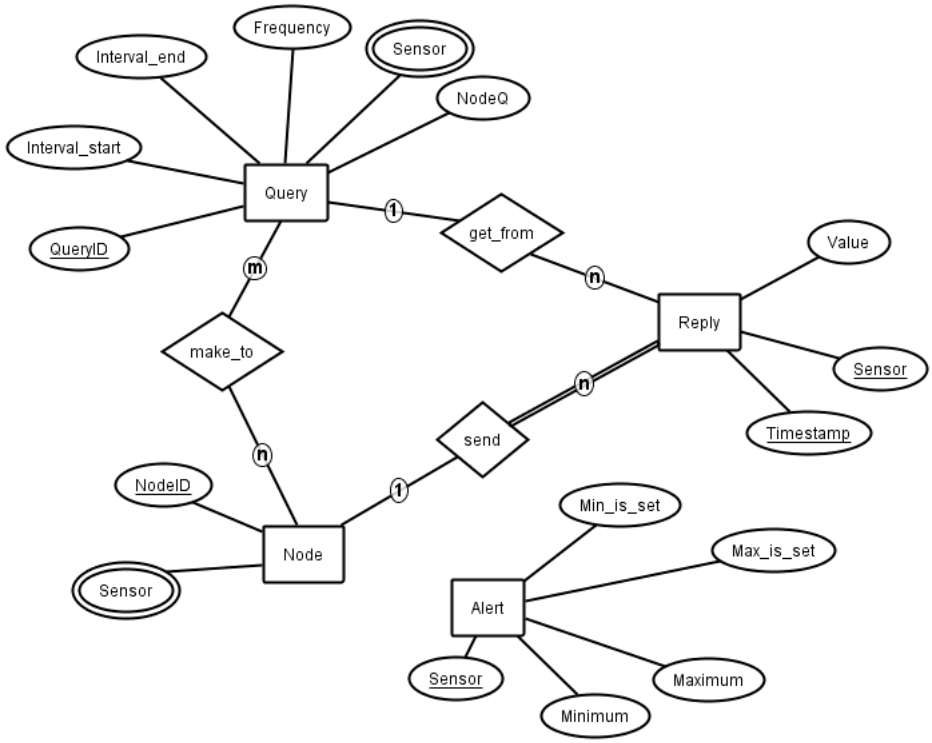

Abbildung 5.3: ER-Diagramm

Das in Abbildung 5.3 dargestellte Datenbankschema setzt sich aus den folgenden Teilen zusammen.
Entitäten:

- Node:
  Ein Node repräsentiert einen Knoten im Sensornetzwerk. Neben der NodeId, die den Knoten eindeutig identifiziert, werden alle Sensoren, die der Knoten enthält, abgespeichert.

- Query:
  Eine Anfrage wird eindeutig durch die QueryId identifiziert, die vom System auf Grundlage der höchsten in der Datenbank gespeicherten Id berechnet wird. Das Sensor-Attribut ist mehrwertig, um eine Auswahl der Sensoren bei den Anfragen aufnehmen zu können. Das NodeQ-Attribut nimmt die Information auf, ob die Anfrage an alle oder eine Auswahl an Knoten gesendet werden soll. Daneben hat der Nutzer die Möglichkeit, den Zeitraum der Anfrage in Interval_start und Interval_End festzulegen sowie eine Frequency zu setzen nach deren Ablauf ein neuer Wert aus dem Sensornetzwerk geholt und abgespeichert werden soll.

- Reply:
  Eine Antwort aus dem Sensornetzwerk wird eindeutig durch die NodeId der Node-Entität, den Sensor und den Timestamp identifiziert. Darüber hinaus wird der gemessene Wert gespeichert. Eine Antwort kann auf eine Anfrage erfolgen. Es ist aber auch möglich, dass das Sensornetzwerk Werte ohne Anfrage sendet, die ebenfalls über Reply abgespeichert werden.

- Alert:
  Die Festlegung der Alarmwerte für die Sensoren wird in Alert gespeichert. Der Nutzer hat die Möglichkeit einen Maximal- sowie einen Minimalwert abzuspeichern. Die Attribute Min_is_set und Max_is_set zeigen an, ob ein Feld gesetzt ist (das Feld ist 1) oder nicht (das Feld ist 0).

Beziehungen:

- make_to:
  Die Verbindung zwischen Node und Query ordnet einer Anfrage mehrere Knoten und einem Knoten mehrere Anfragen zu. Sie spielt eine wichtige Rolle bei dem Speichern einer Anfrage (siehe Beschreibung der `WsnDB`-Klasse).

- send:
  Ein Knoten kann mehrere Antworten senden, wobei eine Antwort immer einem Knoten zugeordnet sein muss.

- get_from:
  Zu einer Anfrage können mehrere Antworten gesendet werden.

Das `server.database`-Package (Abbildung 5.4) ist das Verbindungspackage zwischen dem System und der Datenbank. Es ist für das Lesen, Ändern und Speichern der Daten verantwortlich und SQL-Statements werden ausschließlich in diesem Package ausgeführt.
Die Klasse `Database` stellt die Verbindung zur Datenbank her und holt sich dafür die vom Nutzer eingegeben Verbindungsdaten aus der `MyProperties`-Klasse (siehe 5.2.8). Bis auf die `ParseTimeForDB`-Klasse, sind die anderen Klassen für den Zugriff auf die Daten zuständig. Um die einzelnen Klassen nicht zu groß werden zu lassen, wurde der Programmcode auf mehrere Klassen verteilt. Jede dieser Klassen implementiert das `DataInterface` und überschreibt die benötigten Methoden [2].

[2] In der Abbildung 5.4 werden in den Klassen nur die überschriebenen Methoden angezeigt. Implementiert werden natürlich alle Methoden des `DataInterface`.

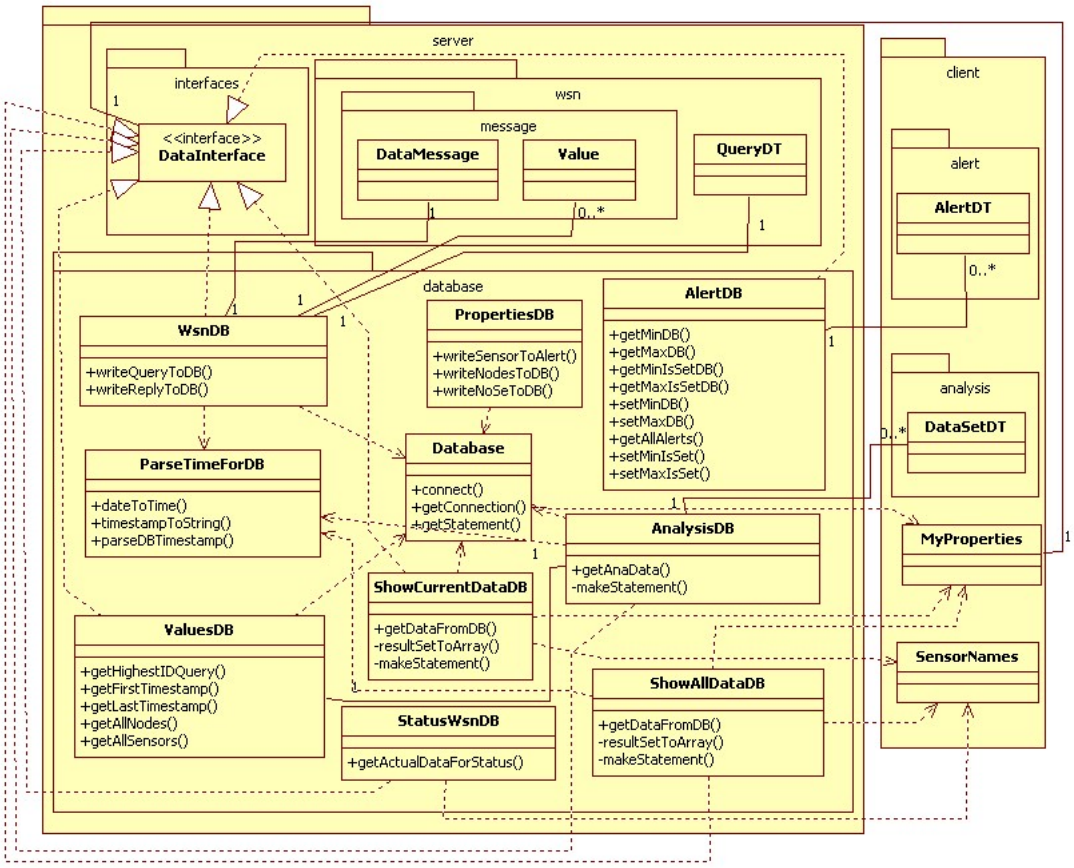

Abbildung 5.4: Package server.database

- Die `AlertDB`-Klasse liest die Daten aus der Alert-Tabelle, wenn sie von dem Alert Settings Dialog (siehe 5.2.5) angefordert werden und schreibt die aktualisierten Werte in diese Tabelle.
- Die `AnalysisDB`-Klasse holt die Daten aus der Reply-Tabelle für die Auswertung in der Analysis-Ansicht im Client. In der `makeStatement`-Methode werden die in dem Client gemachten Einschränkung mit in das Statement einbezogen.
- Die `PropertiesDB`-Klasse synchronisiert die Daten aus der Datenbank mit denen aus der Properties-Datei bei dem entsprechenden Befehl (siehe 5.2.9).
- `ShowAllDataDB` liest alle Daten aus der Reply-Tabelle für die Darstellung der Daten in der Tabelle in der Data-Ansicht und für den Export. In der `makeStatement`-Methode werden die in dem Client gemachten Einschränkungen mit in das Statement einbezogen.
- `ShowCurrentDataDB` liest die aktuellen Daten zu jedem Sensor von jedem Knoten aus der Reply-Tabelle für die Darstellung in der Data-Ansicht und für den Export. Die `makeStatement`-Methode bezieht die Beschränkung auf Sensoren und Knoten mit in das Statement ein.
- Die `StatusWsnDB`-Klasse holt die zuletzt gemessenen Daten der Sensoren eines Knotens aus der Reply-Tabelle, um sie in der Topology-Ansicht anzeigen zu können.
- Die `ValuesDB`-Klasse holt die Werte, die im System zur Auswahl bereitgestellt werden. Dazu

gehören alle Werte für Sensoren und Knoten sowie der frühste und späteste Timestamp. Darüber hinaus wird die höchste QueryId für ein neues `QueryDT`-Objekt aus der Query-Tabelle gelesen.

- Die `WsnDB`-Klasse verarbeitet die Anfragen an und die Antworten aus dem Sensornetzwerk. Für das Speichern der Anfragen in der Query-Tabelle wird ein `QueryDT`-Objekt in die `writeQueryToDB`-Methode gegeben. Ist der Inhalt der `sensor`- bzw. `node`-Variable in diesem Objekt *select*, so besitzt die `sensorList`- bzw. `nodeList`-Variable Einträge. Die entsprechenden Spalten in der Query-Tabelle werden auf *select* gesetzt um anzuzeigen, dass die Werte aus den Listen in die make_to-Tabelle (für die Knoten) und die SensorQ-Tabelle ausgelagert werden. Das Schlüsselwort *all* gibt an, dass an das gesamte Netzwerk gesendet wird und keine Werte mit der zugehörigen QueryId in den beiden Tabellen zu finden sind. Diese Umsetzung reduziert die Daten in der Datenbank. Die Werte in den Variabeln für die Zeit, `interval_start`, `interval_end` und `frequency`, werden in ihre entsprechenden Felder in die Query-Tabelle geschrieben. `null` bedeutet, dass dort kein Wert für die Anfrage angegeben und eine punktuelle Anfrage angefordert wurde. Eine Antwort aus dem Sensornetzwerk liegt als `DataMessage`-Objekt vor und wird in der `writeReplyToDB`-Methode verarbeitet.

5.2 Client

Im Client werden die Daten verarbeitet und dem Nutzer über eine Benutzeroberfläche angezeigt. Diese besteht aus drei Hauptfenstern, die als Tabs umgesetzt sind: Data-, Analysis- und Topology-Ansicht. Im Menü befinden sich wichtige Funktionen, die auf allen Ebenen verwendet werden. Gleiches gilt für die Toolbar, wobei der Nutzer dort auf häufig verwendete Funktionen zurückgreifen kann. Für die Einstellung der Alarmwerte, des Adapters und das Stellen einer neuen Anfrage öffnen sich nach Betätigen der Schaltflächen zusätzliche Dialoge.

Das `client`-Package (Abbildung 5.5) enthält, neben den Subpackages, Klassen, die in mehreren Teilen des Client benötigt werden. Die Hauptklasse, in der alle Teile der Benutzeroberfläche zusammenkommen, ist die `Mainframe`-Klasse. Sie enthält den Code für das Grundgerüst des GUI insbesondere das Menü und die Toolbar, deren Funktionen ebenfalls in dem `Mainframe` aufgerufen werden.
Die `Place`- und die `Time`-Klasse werden in dem `client.data`-, `client.analysis`- und `client.query`-Package benötigt. Die `Place`-Klasse repräsentiert die Auswahl der Sensoren und Knoten, die auf der Benutzeroberfläche als jeweils eine Liste mit Checkboxen, die beim Programmstart ausgewählt sind, dargestellt werden. Die Sensoren und Knoten werden dabei aus der Datenbank gelesen und beim Starten des Programms neu erzeugt beziehungsweise nach einer Synchronisation (siehe 5.2.9) neu gesetzt. Die `Time`-Klasse repräsentiert die Auswahl des Zeitintervalls in dem die Daten liegen sollen. Dieses Intervall hat einen Startzeitpunkt (from) und einen Endzeitpunkt (to), die sich jeweils aus Jahr, Monat, Tag und der Uhrzeit (Stunden, Minuten, Sekunden) zusammensetzen und auf der Benutzeroberfläche als Spinner[3] dargestellt werden. Dadurch soll dem Nutzer die Auswahl der Werte erleichtert werden, da er die Teile getrennt einstellen kann. Die Startwerte in der Data- und in der Analysis-Ansicht sind der frühste und der späteste Timestamp, die sich in der Reply-Tabelle der Datenbank befinden. Damit wird sichergestellt, dass alle Daten erfasst werden, wenn die Spinner vom Nutzer noch nicht betätigt wurden. Für den Query-Dialog werden die Startwerte auf das aktuelle Datum und die Uhrzeit beim Öffnen des Dialogs gesetzt. Darüber hinaus sind Place und Time in der Data- und in der Analysis-Ansicht miteinander verknüpft, so dass, unabhängig davon in welcher Ansicht die Auswahl getroffen wird, diese auch in der anderen erfolgt. Damit soll dem Nutzer ermöglicht werden, die Daten ohne unnötigen Aufwand, d.h. dass die Daten nicht erneut ausgewählt werden müssen, auf unterschiedliche Art anzuzeigen.
Der `InputValidator` überprüft die Eingaben des Nutzers auf Gültigkeit bevor sie in das System gegeben werden. So wird beispielsweise bei den Zeitwerten geprüft, ob die Startzeit kleiner ist als die Endzeit.

[3] *Spinner* sind graphische Steuerelement, die dem Nutzer erlauben die Werte mittels Pfeiltasten in zwei Richtungen zu verändern.

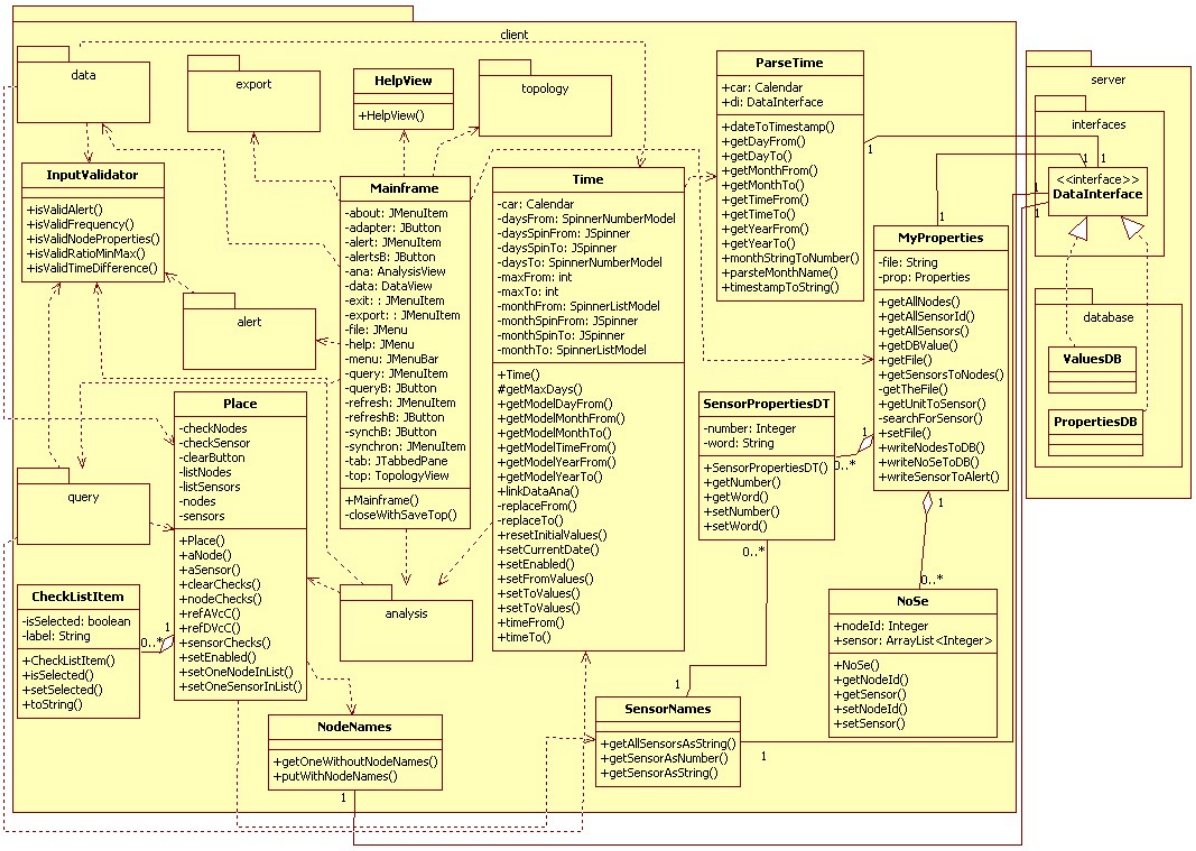

Abbildung 5.5: Package client

Andernfalls wird die Eingabe zurückgewiesen um Fehler im System zu vermeiden.

5.2.1 Data

Das `client.data`-Package (Abbildung 5.6) beinhaltet alle Klassen für die Darstellung der Daten aus dem Sensornetzwerk. Sie werden in einer Tabelle angezeigt, die die NodeId, den Sensor, den Wert, die Messeinheit des Wertes sowie den Zeitpunkt der Messung in Form des Timestamps beinhaltet. Dabei werden, bis auf die Messeinheiten, die aus den Properties (siehe 5.2.8) geholt werden, alle Daten über das `DataInterface` aus der Datenbank geladen. Die Art des Sensors wird als Inhalt in der Sensor-Spalte angezeigt. Eine andere Möglichkeit wäre die Sensoren als Spalten mit dem Wert als Inhalt darzustellen. Diese wurde ausgeschlossen, da das System mit heterogenen Sensornetzwerken arbeitet und die Daten in der Datenbank mit der Sensorart als Inhalt vorliegen. Für jeden Sensor müsste beim Darstellen in der Tabelle überprüft werden, ob er zu dieser Spalte passt. Da viele Datensätze vorliegen können, würde dies zu Perfomance-Problemen führen. Um Unübersichtlichkeit bei der gewählten Tabellenstruktur zu vermeiden, hat der Nutzer die Möglichkeit der Auswahl der Knoten auch eine Beschränkung auf die Sensoren hinzuzufügen.
In der Data-Ansicht kann der Nutzer zwischen zwei Modi wählen: *Current* bezieht sich auf die Auswahl der aktuellsten Daten jedes Sensors von jedem Knoten. Diese können weiter auf einzelne oder mehrere Knoten sowie Sensoren eingeschränkt werden. *All* beinhaltet die archivierten Daten, die ebenfalls auf diese Art beschränkt werden können. Hinzu kommt eine Einschränkung auf einen bestimmten Zeitabschnitt.

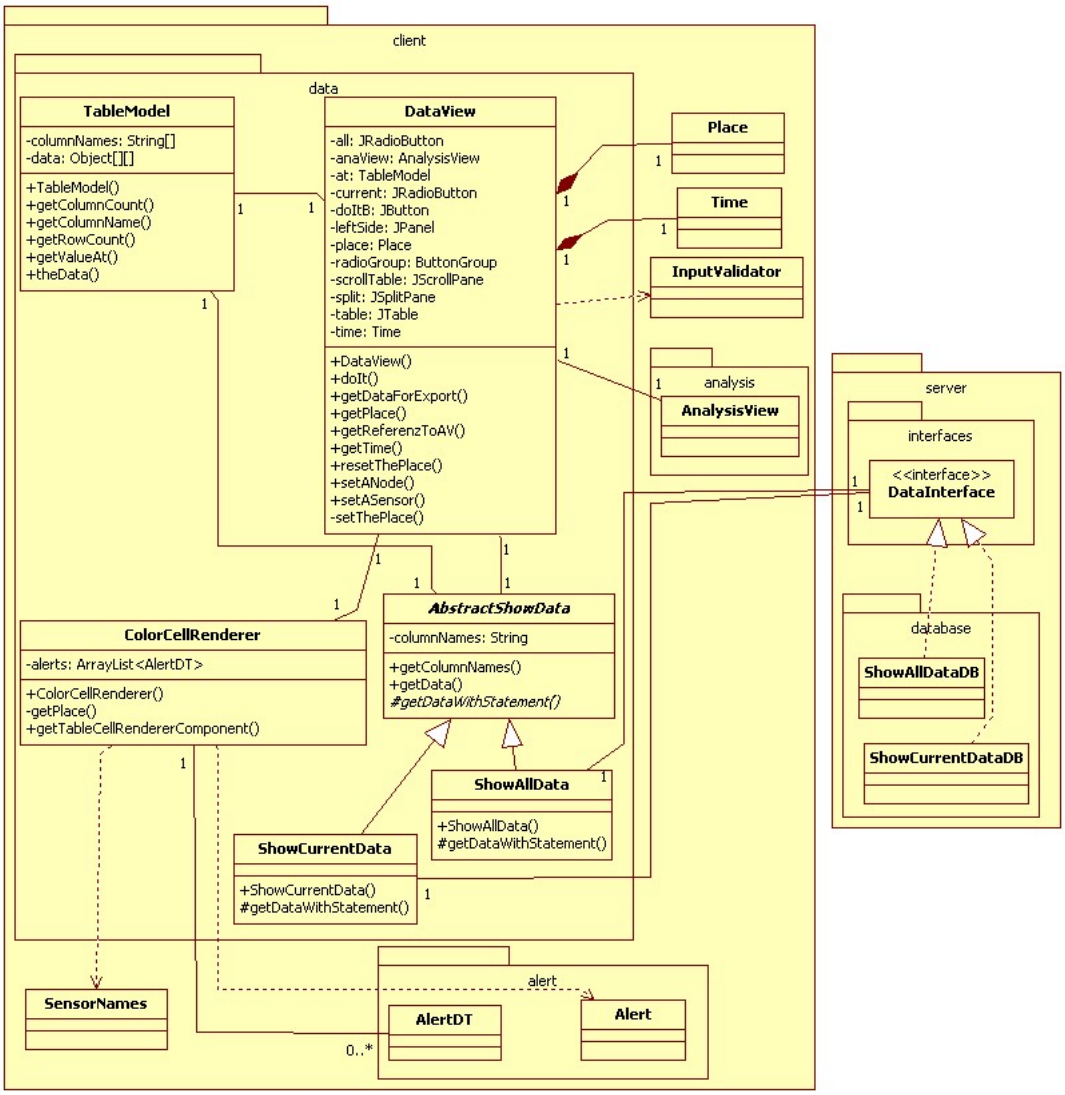

Abbildung 5.6: Package client.data

Bei beiden Modi werden, nach dem Ausführen mit dem Do-It-Button, die ausgewählten Daten an das `server.database`-Package weitergegeben. Die angeforderten Werte werden in einer neuen Tabelle bereit gestellt. Wie in der Abbildung zu sehen ist, wird zwischen aktuellen (`ShowCurrentData`) und archivierten Daten (`ShowAllData`) unterschieden, die eine gemeinsame Oberklasse (`AbstractShowData`) besitzen. Diese Trennung wurde aufgrund der unterschiedlichen SQL-Statements, die im `server.database`-Package erzeugt werden, gewählt und um die Möglichkeit offen zu halten, andere gewünschte Beschränkungen einfach in das System zu bringen ohne den bestehenden Code verändern zu müssen.
Die `DataView`-Klasse ist die wichtigste Klasse in dem Package. Sie hält den Code für die Data-Ansicht und ist die zentrale Anlaufstelle für alle Klassen aus diesem Package. Sie weiß ob die aktuellen oder alle Daten angefordert wurden und gibt die Beschränkung an die entsprechende Klasse weiter.
In der Tabelle werden ebenfalls die kritische Werte angezeigt (zum Setzen von Alarmerten siehe 5.2.5). Dazu implementiert die `ColorCellRender`-Klasse das `TabelCellRenderer`-Interface, das für das Zeichnen der Zellen in einer Tabelle zuständig ist. In jedem Datensatz d.h. in jeder Zeile, wird der Wert in Abhänigkeit zum Sensor mit den gespeicherten Alarmwerten abgeglichen. Liegt ein Wert über dem

Maximal- oder unter dem Minimalwert, erhält dieser eine rote Markierung.

5.2.2 Analysis

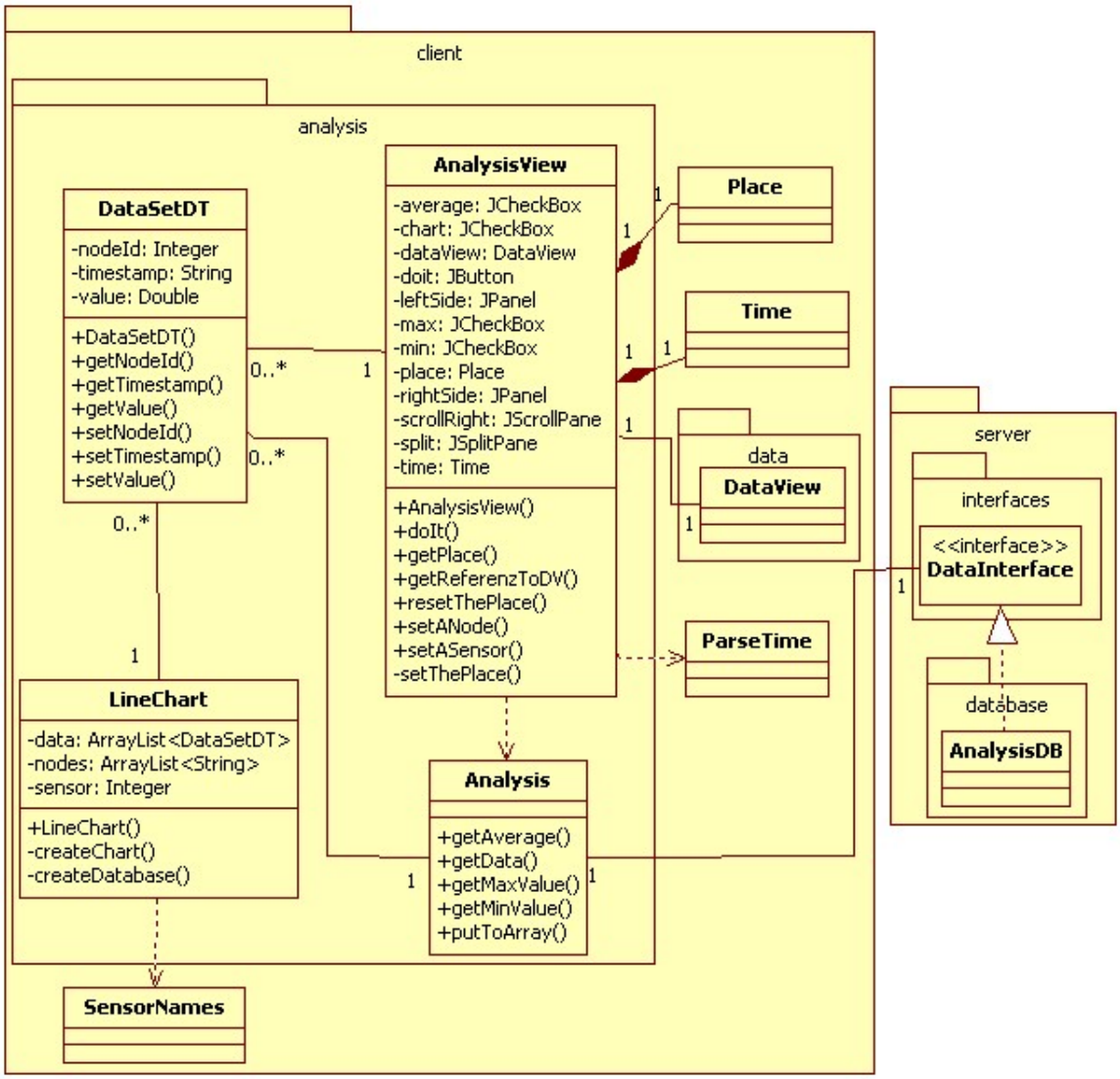

Abbildung 5.7: Package client.analysis

Die Auswertung der Daten erfolgt in dem `client.analysis`-Package (Abbildung 5.7), in der vier Auswertungsarten gegeben sind, aus denen der Nutzer wählen kann. Alle Auswertungen werden dabei nach Sensoren gruppiert. Ein Beispiel: Ist der Sensor *Light* und die Knoten *Node 1*, *Node 3* und *Node 4* ausgewählt, so werden die Werte des Sensors *Light* dieser drei Knoten für die Berechnung verwendet. Ist ein Knoten gewählt, der den Sensor nicht enthält, findet er in der Berechnung keine Beachtung. Die vier Auswertungsarten sind:

- Chart (Graph): Die Graphen werden mit JFreeChart, einem Framework für die Erstellung von unterschiedlichen Diagrammen, generiert. In der Klasse `LineChart` wird aus den ausgewählten Daten mithilfe der speziellen Klassen von JFreeChart ein Graph erstellt. Die Werte der Sensoren werden

auf der y-Achse angezeigt, die Zeit der Messung ist der Maßstab der x-Achse. Darüber hinaus kann der Graph durch JFreeChart als Bilddatei abgespeichert werden.

- Min (Minimalwert): Der Minimalwert wird aus allen gewählten Knoten zu einem Sensor ermittelt. Dazu wird ein Array mit den Werten an die `getMinValue`-Methode in der `Analysis`-Klasse gegeben. Zum einen wird der Minimalwert jedes einzelnen Knotens berechnet, zum anderen wird der Minimalwert aus allen Knoten ermittelt.
- Max (Maximalwert): Der Maximalwert wird, mit der `getMaxValue`-Methode in der `Analysis`-Klasse, nach dem gleichen Prinzip wie bei den Minimalwerten berechnet.
- Average (Durchschnittswert): Der Durschschnittswert wird, mit der `getAverage`-Methode in der `Analysis`-Klasse, nach dem gleichen Prinzip wie der Minimal- und der Maximalwert berechnet.

Durch das Auflisten der Ergebnisse für die einzelnen Knoten soll dem Nutzer die Möglichkeit gegeben werden, die Werte besser miteinander zu vergleichen. Alle Ergebnisse, insbesondere bei dem Durchschnittswert, werden nicht gerundet um die Genauigkeit möglichst zu erhalten. Da die Werte als Double vorliegen erfolgt eine Beschränkung nur durch dessen Wertebereich.
Wie in der Data-Ansicht hat der Nutzer die Möglichkeit die Werte auf Sensoren, Knoten und einen bestimmten Zeitraum zu beschränken, die zur Auswertung herangezogen werden.
Die `AnalysisView`-Klasse enthält den Programmcode für die Analysis-Ansicht. Dort wird auch die Auswertung erstellt, wobei eng mit der `LineChart`-Klasse, die für den Graphen zuständig ist, und der `Analysis`-Klasse, die neben der Berechnung von Minimal-, Maximal- und Durchschnittswert auch die Daten über das `DataInterface` anfordert, zusammengearbeitet wird.

5.2.3 Topology

Das `client.topology`-Package (Abbildung 5.8) ist für die Visualisierung des Sensornetzwerkzustandes zuständig. Die Knoten können per Drag and Drop auf mehreren Ebenen platziert werden, so dass der Nutzer die Struktur des Sensornetzwerks erstellen kann. Die Ebenen werden durch ein `TopologyLevel` repräsentiert und als Tabs in der Topology-Ansicht angezeigt. Auf jeder Ebene kann der Nutzer einen Gebäudeplan in Form eines JPG-Bildes als Hintergrund laden. Übertritt die Größe des Bildes den Tab, werden Scrollbalken gesetzt, so dass sich der Tab an das Bild anpasst und der Gebäudeplan nicht verzerrt dargestellt wird. Das Bild und der Name der Ebene können im Nachhinein geändert werden. Dafür wird der Change-Level-Button betätigt, der Änderungen in dem ausgewählten Tab ermöglicht. Der Delete-Level-Button wird ebenfalls auf den offenen Tab angewendet. Beim Löschen einer Ebene werden alle darauf platzierten Knoten wieder in das Node Home verschoben.
Ein Knoten wird durch ein 30x30 großes `NodePanel` mit der NodeId zum Identifizieren, repräsentiert. Beim Starten des Programms wird für jeden Knoten aus der Datenbank ein Node Panel anhand der NodeId erstellt. Die `NodeHomePanel`-Klasse hält die Node Panel, wenn sie keiner Ebene zugeordnet sind. Die Größe wird in Abhängigkeit aller Knoten berechnet, so dass alle Knoten dort Platz finden.
Beim Verschieben eines Knotens wird geprüft ob die Fläche, auf die er gesetzt werden soll, diese Komponente aufnehmen kann. In dem System sind das in der Topology-Ansicht alle Ebenen und das Node Home, an allen anderen Stellen wird ein Drop zurückgewiesen. Wenn der Drop akzeptiert wird, wird das Node Panel an der gewünschten Stelle platziert. Da das Node Panel durch ein einfaches JPanel realisiert ist, das nicht von vornherein eine Drag source ist, muss es nach jedem Versetzen erneut zu einer Drag source gemacht werden, um den Knoten ein weiteres Mal verschieben zu können. Anschließend wird mit einer `cleanup`-Methode, die in der `TopologyView` liegt, das NodePanel von seiner Ausgangsstelle entfernt. Dazu wird auf allen Ebenen nach dem soeben gesetzten NodePanel mit der NodeId gesucht und das ursprüngliche gelöscht.
Der Status eines einzelnen Knotens setzt sich zusammen aus den zuletzt gemessenen Werten der Sensoren mit Messzeitpunkt, dem Energiezustand und den Nachbarschaftsbeziehung, wobei die beiden zuletzt

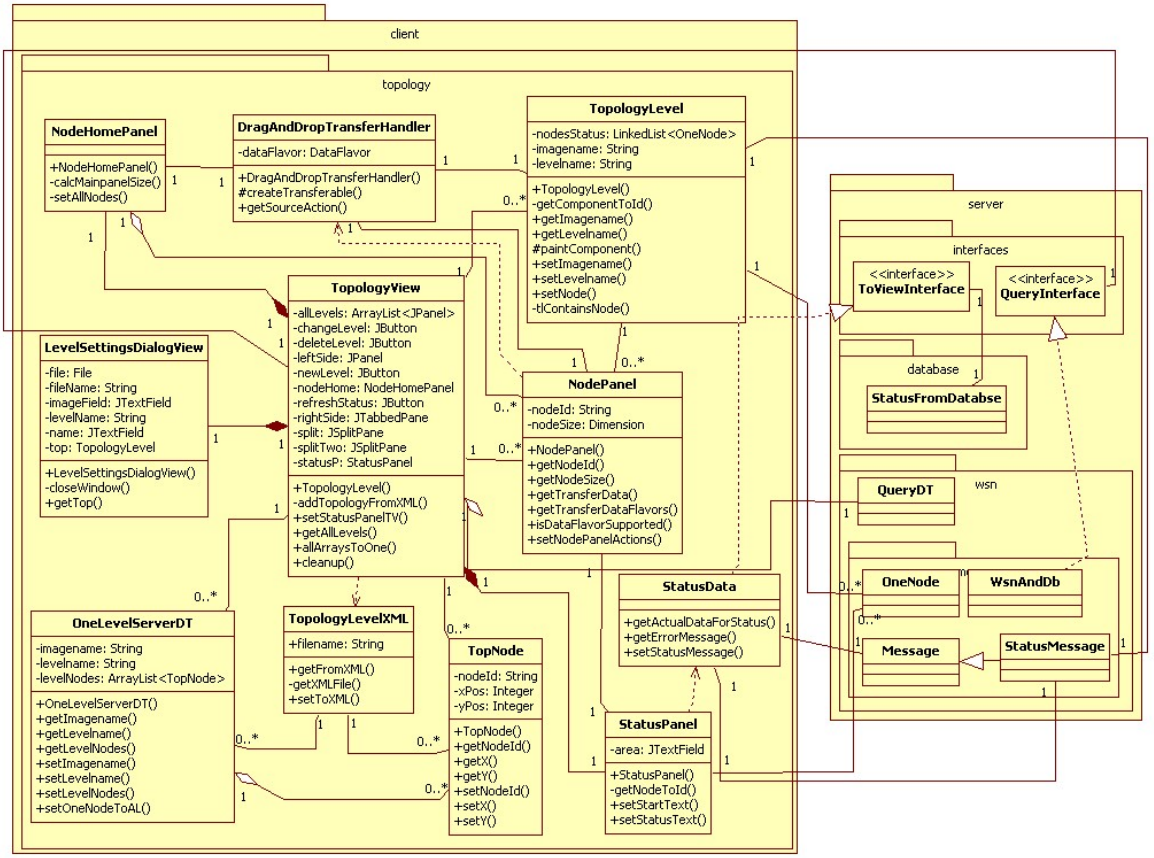

Abbildung 5.8: Package client.topology

genannten Teil der Status Message sind. Fährt der Nutzer mit dem Mauszeiger über ein Node Panel, dann wird in der rechten Seite der Topology-Ansicht die dem Knoten entsprechende NodeId sowie die Statusdaten angezeigt. Dafür wird jedesmal eine neue Anfrage an die Datenbank für die Daten zu den Sensoren gestellt. Der Energiezustand wird ebenfalls in der rechten Seite angezeigt und aus der zuletzt angeforderten Status Message geholt. Neben dem Energiezustand sind die Nachbarschaftsbeziehungen Teil der Status Message. Zwei Knoten sind benachbart, wenn sie in Funkkontakt zueinander stehen. Die Anzeige erfolgt direkt auf den Ebenen. Dazu werden benachbarte Knoten, wenn sie auf derselben Ebene liegen, mit einer grünen Linie verbunden, die auch beim Verschieben innerhalb dieser Ebene erhalten bleibt.
Mit dem Refresh-Status-Button wird eine neue Status Message aus dem Sensornetzwerk angefordert. Die Nachbarschaftsbeziehungen und der Energiezustand werden nicht persistent abgespeichert, da sie in dem System lediglich in der Topology verwendet und keiner Auswertung unterzogen werden. Aus diesem Grund liegen beim Programmstart keine Informationen zu diesem Teil des Status vor, sondern müssen erst angefordert werden. Die Werte der Sensoren werden hingegen direkt aus der Datenbank geladen und sind somit von den Anfragen über den Query-Dialog aktualisierbar.
Um die Topology beim erneuten Starten des Programms in der zuletzt gesetzten Form anzeigen zu können, wird sie über die `TopologyLevelXML`-Klasse lokal in einer XML-Datei gespeichert. Die XML-Form sieht folgendermaßen aus:

```
1  <?xml version="1.0" encoding="UTF-8" standalone="no"?>
2 <topology>
3   <level imagename="C:\images\testbild.JPG" levelname="One">
4     <node id="1">
5       <xpos>129</xpos>
6       <ypos>167</ypos>
7     </node>
8     <node id="7">
9       <xpos>323</xpos>
10      <ypos>288</ypos>
11    </node>
12  </level>
13 </topology>
```

Jede Ebene wird in den *level*-Tags eingeschlossen und enthält den Namen der Ebene in *levelname* und den Pfad des Bilds im *imagename*. Die auf der Ebene platzierten Knoten werden mit ihrer NodeId sowie der x- und der y-Position gespeichert. Beim Starten des Programms werden die Werte mit der `getFromXML`-Methode aus der XML-Datei ausgelesen und reproduziert. Die gespeicherten Knoten werden beim Start zwar direkt auf der Ebene platziert und nicht im Node Home, aber trotzdem in die Berechnung der Größe des NodeHome-JPanels miteinbezogen.

5.2.4 Query

Das `client.query`-Package ist dafür zuständig, die Anfragen an das Sensornetzwerk von dem Nutzer entgegenzunehmen und an den Server weiterzuleiten. Dabei lassen sich mehrere Anfragetypen nach bestimmten Kriterien unterscheiden.

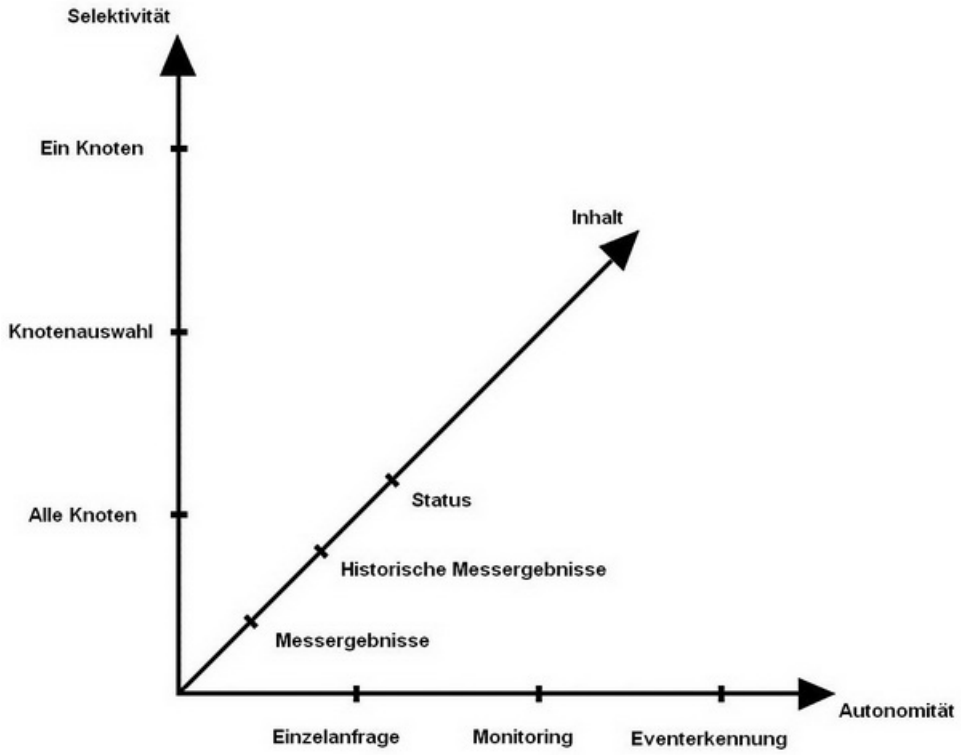

Abbildung 5.9: Anfragetypen

Die Selektivität besagt an welche Menge von Knoten eine Anfrage gestellt wird, wobei zwischen drei Möglichkeiten unterschieden wird: Die Anfrage geht an alle Knoten, an eine Auswahl oder an einen einzelnen Knoten. Von dem System werden alle drei Arten unterstützt, wobei die Auswahl mehrerer Knoten für jeden Knoten sowie Sensoren getroffen wird und nicht für bestimmte Bereiche.
Die Autonomität beschreibt wie die Anfrage an bzw. die Antwort aus dem Sensornetzwerk gesendet wird. Dabei wird zwischen drei Arten unterschieden:

- Einzelanfrage:
  Die wenigste Autonomität ist vorhanden, wenn eine Einzelanfrage an das Sensornetzwerk gestellt wird d.h. dass zu einem bestimmten Zeitpunkt der Wert einmal abgefragt wird. Dieser Vorgang kann nur von einem Nutzer ausgelöst werden.

- Monitoring:
  Das Sensornetzwerk sendet über einen bestimmten Zeitraum in einem bestimmten Abstand Werte von den Knoten. Der Nutzer muss dazu den Zeitraum und den Abstand angeben, danach arbeitet das Sensornetzwerk selbstständig.

- Eventerkennung:
  Die meiste Autonomität liegt bei der Eventerkennung vor, da das Sensornetzwerk eigenständig eine Antwort sendet, wenn ein bestimmtes Ereignis eintritt.

Das System unterstützt die Einzelanfrage und das Monitoring.
Es können verschiedene Inhalte vom Sensornetzwerk angefragt werden. Messergebnisse bezieht sich auf den aktuellsten Wert, der gemessen wurde. Historische Messergebnisse meint die von dem Knoten gespeicherten Werte. Der Status gibt den momentanen Status des Knotens an beispielsweise den Energiezustand. Im vorliegenden System wird der Status und das aktuellste Messergebniss unterstützt. Die historischen Messergebnisse werden aus der Datenbank geholt und waren ursprünglich aktuelle Messergebnisse.

Der `QuerySettingsDialog` (Abbildung 5.10), der für die Einstellungen einer Anfrage verwendet wird, kann im gesamten System über die Toolbar oder das Menü als modaler Dialog[4] geöffnet werden. Die Eingabemaske für eine Anfrage bietet mehrere Auswahlmöglichkeiten: Die Anfrage kann entweder über *All* an alle Knoten und alle Sensoren des Sensornetzwerks gesendet werden oder über *Select* an die von dem Nutzer ausgewählten. Dabei muss in beiden Listen mindestens jeweils ein Haken gesetzt sein. Ist das nicht der Fall, dann kann die Anfrage nicht gesendet werden, da sie nicht vollständig ist. Ebenso hat er die Möglichkeit seine Anfrage über *Select* zeitlich zu begrenzen indem ein Start- und ein Endzeitpunkt angegeben wird. Zusätzlich kann eine Frequenz eingegeben werden d.h. ein Abstand mit dem die Anfrage erneuert durchgeführt werden soll. Die andere Zeitauswahl über *One* ist eine punktuelle Anfrage d.h. dass die Anfrage ein einziges Mal an das Sensornetzwerk gesendet wird. Nach Abschluss der Eingabe wird die Anfrage in einem `QueryDT`-Objekt an die `WsnAndDb`-Klasse im Server weitergegeben. Durch das Setzen einer Variable wird verhindert, dass eine neue Anfrage gestellt werden kann, wenn die vorherige Anfrage noch nicht in die Datenbank geschrieben ist. Dadurch wird das gleichzeitige Schreiben in die Datenbank verhindert und es wird sichergestellt, dass, beim Erstellen eines neuen `QueryDT`-Objekts, die richtig QueryId gesetzt wird.
Die `Reply`-Klasse verarbeitet lediglich die aus dem Sensornetzwerk kommenden Fehlermeldungen d.h. die Nachrichten vom Typ `WSNErrorMessage` und alle Unterklassen. Sie werden dem Nutzer mit ihrem entsprechenden Error-Message-Text angezeigt.
Die Antworten zu Datenanfragen werden direkt in die Datenbank geschrieben ohne sie gleichzeitig an den Client zu geben, die Statusdaten werden in der Topology verarbeitet (siehe 5.2.3).

[4] Ist ein modaler Dialog geöffnet, dann kann keine Aktion in den anderen Fenstern durchgeführt werden.

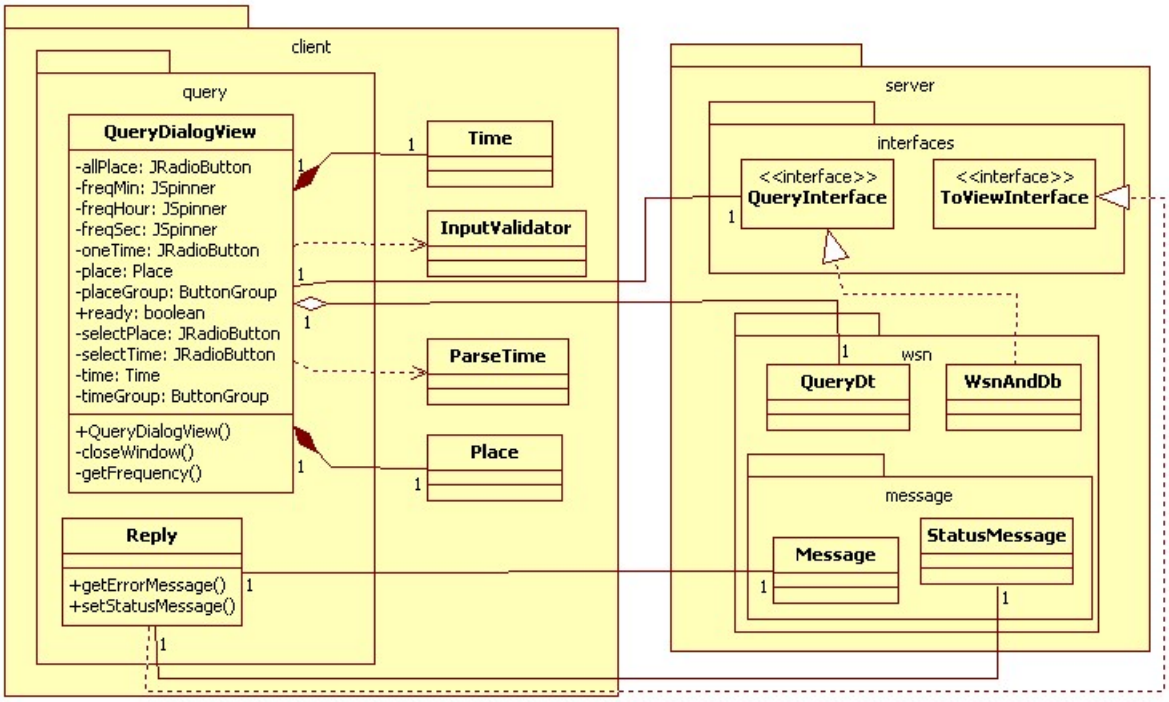

Abbildung 5.10: Package client.query

5.2.5 Alert

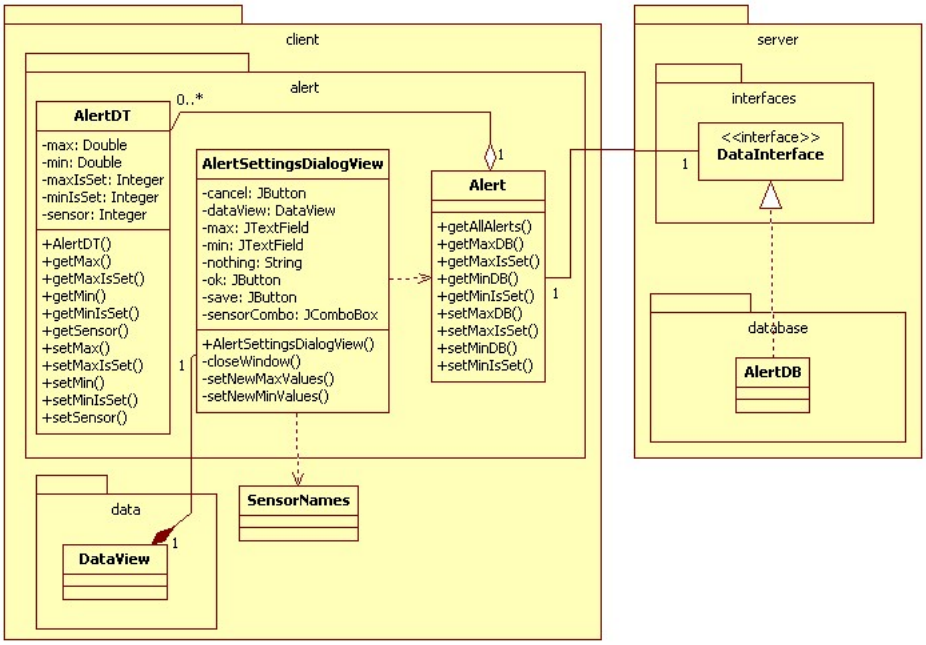

Abbildung 5.11: Package client.alert

Die Eingabe von einem kritischen Wertebereich für den Alarm erfolgt über die Alert Settings, die als modaler Dialog über die Toolbar und das Menü aufgerufen werden können. Zu jedem Sensor kann ein Minimalwert und ein Maximalwert eingegeben werden. Liegen Werte unter dem Minimalwert oder über dem Maximalwert, so sind sie nicht mehr in dem akzeptierten Bereich und werden in der Datentabelle markiert (siehe 5.2.1). Ist das min- oder max-Feld leer, dann ist an dieser Stelle kein Wert gesetzt. In der Datenbank werden dazu min_is_set bzw. max_is_set auf 0 gesetzt, bei einem gesetzten Wert stehen die Felder auf 1. Die `Alert`-Klasse (Abbildung 5.11) fordert die Daten aus der Datenbank an und setzt die geänderten Werte. Gibt der Nutzer einen Wert ein und speichert diesen nicht ab, wird beim Betätigen des OK-Buttons nachgefragt, ob der Wert gespeichert werden soll. Die Werte werden nach dem Speichern direkt auf die Tabelle angewendet, so dass kritische Werte für den Nutzer sofort sichtbar sind.

5.2.6 Export

Das Exportieren ist in XML über das `client.export`-Package möglich. Nach Auswahl des Export-Befehls über den Menüpunkt Export kann der Nutzer wählen wo die Datei abgespeichert werden soll. Die Daten für den Export werden auf die gleiche Art geholt wie für die Tabelle in der Data-Ansicht und in dieser Tabellenform exportiert. Darüber hinaus wird die Beschränkung auf Sensoren, Knoten und Zeit, die dort angeben werden auf den Export angewendet.
Die XML-Daten liegen in der folgenden Form vor.

```
1 <data>
2   <dataset>
3     <nodeID>
4       1
5     </nodeID>
6     <sensor>
7       Temperature
```

```
 8       </sensor>
 9       <value>
10         1
11       </value>
12       <unit>
13         Celsius
14       </unit>
15       <time>
16         2010-06-20 14:47:32.0
17       </time>
18     </dataset>
19 </data>
```

Die gesamten Daten werden mit dem *data*-Tag eingeschlossen. Jeder einzelne Datensatz entspricht einer Zeile in der Tabelle und wird durch den *dataset*-Tag abgegrenzt. Darin sind die einzelnen Werte eines Datensatzes in ihren eigenen Tags enthalten.

5.2.7 Adapter über Client

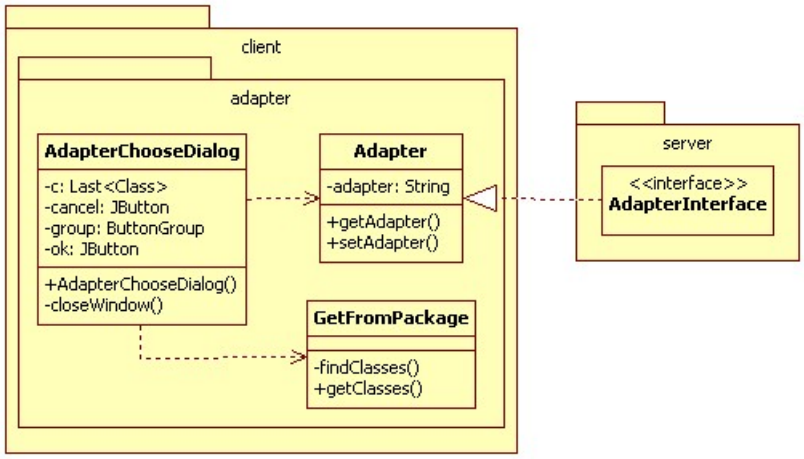

Abbildung 5.12: Package client.adapter

Das `client.adapter`-Package (Abbildung 5.12) ist für die Auswahl des Sensornetzwerk-Adapters zuständig. In dem modalen Choose Adapter Dialog, der sich über die Toolbar oder das Menü öffnen lässt, werden alle im `server.wsn.adapter`-Package liegenden Adapter zur Auswahl bereit gestellt. Die `GetFromPackage`-Klasse holt dafür alle Klassen aus dem Package und zeigt sie mit ihrem Klassennamen im Dialog an. Nach Bestätigung der Auswahl wird der gewählte Adapter in der `Adapter`-Klasse abgespeichert und kann vom Server bei Bedarf abgefragt werden. Der gewählte Adapter wird nicht persistent gespeichert und muss nach dem Programmstart neu gesetzt werden. Da es möglich ist mehrere Adapter zu halten, soll damit erreicht werden, dass der Nutzer jedesmal explizit angeben muss, welchen Adapter er ansprechen will. Um zu verhindern, dass Anfragen gesendet werden, ohne dass ein Adapter ausgewählt ist, wird der Nutzer bei der Wahl des Query-Dialogs oder beim Betätigen des Refresh-Status-Button in der Topology-Ansicht darauf hingewiesen, dass keine Verbindung zum Sensornetzwerk besteht.

5.2.8 Properties

In der config.properties-Datei liegen die Verbindungsdaten für die Datenbank und die Informationen zum Sensornetzwerk. Sie müssen von dem Nutzer direkt in die Datei geschrieben werden. Die Möglichkeit die Daten in eine XML-Datei zu schreiben wurde ausgeschlossen, da keine Eingabemaske in dem GUI vorliegt und die Form von Properties-Dateien für den Nutzer übersichtlicher und weniger fehleranfällig bei der Eingabe ist. Ein Properties-Datensatz besteht aus einem eindeutigen Key, einem Gleichheitszeichen und einem Value.
Damit das System mit den Daten arbeiten kann, müssen sie in einer bestimmten Form vorliegen:
Für die Verbindung zur Datenbank werden lediglich die Werte ergänzt und zwar den Namen der Datenbank, den Namen des Servers, den Nutzer und das Passwort der Datenbank. Die eingegebenen Daten werden in der Database-Klasse verwendet um die Verbindung herzustellen (siehe 5.1.3).

```
dbName=wsnba
serverName=localhost
user=root
password=
```

Die Informationen über das Sensornetzwerk bestehen aus vier Teilen, die durch bestimmte Schlüsselwörter zu Beginn des Keys gekennzeichnet werden. Diese sind notwendig um beim Auslesen in der `MyProperties`-Klasse die korrekten Werte identifizieren und zuordnen zu können. Sind diese Wörter nicht vorhanden, dann wird der Wert nicht erfasst. Neben dem Schlüsselwort wird ein weiterer Wert in dem Key angegeben, der zum einen die Eindeutigkeit des Keys herstellen soll und zum anderen eine weitere Funktion erfüllt.
Für jeden Knoten aus dem Sensornetzwerk wird ein Eintrag mit dem Schlüsselwort *node* im Key vorgenommen. Die Zahl hinter dem Gleichheitszeichen ist eine ganze Zahl, die der NodeId entspricht und als diese in die Datenbank geschrieben wird. Die Zahl direkt an *node* ist die einzige, die keine zusätzliche Funktion erfüllt und aus Gründen der Übersichtlichkeit der NodeId entspricht.

```
node1=1
node2=2
```

Jede Sensorart erhält einen Eintrag mit dem Schlüsselwort *sensor*. Die Zahl, die zusätzlich im Key steht, gibt die SensorId an, die auch in der Datenbank gespeichert wird. Im Value findet sich die Art des Sensors.

```
sensor1=Temperature
sensor2=Light
sensor3=Sound
```

Die Einheiten der Sensoren werden mit *unit* gekennzeichnet. Zu jeder Sensorart wird eine Einheit gespeichert. Die Zahl hinter *unit* gibt die SensorId an, der die Einheit zugeordnet wird. So ist im Beispiel der Sensor mit der SensorId 1 ein Temperatursensor und misst die Werte in Celsius.

```
unit1=Celsius
unit2=Candela
unit3=Decibel
```

Da ein heterogenes Sensornetzwerk vorliegen kann, müssen die Sensoren den Knoten zugeordnet werden. Dazu muss das Schlüsselwort *nose* (eine Zusammensetzung aus *node* und *sensor*) eingefügt werden. Die Zahl im Key gibt die NodeId an. Der Value enthält alle zum Knoten gehörenden Sensoren, mit ihrer Id, die durch ein Semikolon getrennt werden. Im Beispiel hat der Knoten mit der NodeId 1 die Sensoren Temperature (SensorId 1), Light (SensorId 2) und Sound (SensorId 3).

```
nose1=1;2;3;
nose2=3;
```

Die Knoten und die Zuordnung der Sensoren zu den Knoten wird in der Datenbank gespeichert und zwar bei einer Synchronisation mit dem entsprechenden Button. Die Art des Sensors sowie seine Einheit werden beim Laden der Daten aus der Datenbank für die Tabelle in der Benutzeroberfläche und den Export hinzugefügt.

5.2.9 Weitere Funktionen

Bei der Synchronisation werden die Sensoren und Knoten in der Data- und in der Analysis-Ansicht aktualierst. Außerdem werden die Knoten im Node Home angepasst. Dazu wird für jeden Knoten, der aus der Properties-Datei gelesen wurde, überprüft, ob er bereits auf einer der Ebenen liegt. Diese werden nicht in das Node Home gelegt. Außerdem werden auf den Ebenen liegende Knoten, die nicht mehr in der Properties-Datei sind, gelöscht.
Der Refresh-Button ist für die Aktualisierung der Ansichten, insbesondere nach einer neuen Antwort aus dem Sensornetzwerk, zuständig. Davon sind die Tabelle in der Data-Ansicht und die Berechnung in der Analysis-Ansicht betroffen wobei die momentane Auswahl der Knoten und Sensoren bestehen bleibt. Zusätzlich werden die Zeitwerte erneut auf den frühsten und den spätesten Timestamp gesetzt. Die Alternative zu dem Refresh-Button wäre, dass sich die Ansichten automatisch nach Eingang neuer Antworten aktualisieren. Diese Möglichkeit wurde jedoch ausgeschlossen um dem Nutzer die Kontrolle zu überlassen und Unruhe im Programm zu verhinden, da auch Antworten ohne Anfrage aus dem Sensornetzwerk gesendet werden können.

6 Test und Evaluation

Für das Testen des Systems wurde eine Sensornetzwerk in Form einer Java-Klasse geschrieben, das bei Anfragen Zufallswerte generiert und zurückgibt. Dieses kann über den Adapter *TestWsn* angesprochen werden. Das Test-Sensornetzwerk besteht aus acht Knoten und vier Sensoren, die wie folgt zugeordnet sind:

- Knoten 1: Temperatur, Licht, Sound
- Knoten 2: Sound
- Knoten 3: Temperatur, Licht
- Knoten 4: Licht
- Knoten 5: Feuchtigkeit
- Knoten 6: Sound, Feuchtigkeit
- Knoten 7: Temperatur, Licht
- Knoten 8: Licht, Feuchtigkeit

Das Testen eines Systems läuft in der Regel in vier Teststufen ab, die unterschiedliche Aspekte in den Vordergrund stellen. Der Komponententest prüft einzelne Einheiten unabhängig voneinander um Fehler auf diese Einheiten beschränken zu können. Der Integrationstest testet das Zusammenspiel zwischen Komponenten und betrachtet die Schnittstellen. Diese Teststufe erfolgt nach dem Komponententest und arbeitet mit geprüften Komponenten, so dass Fehler, die auftreten, der Kommunikation zwischen den geprüften Komponenten zugewiesen werden können. Fehler, die beim Integrationstest auftreten können sind beispielsweise, dass Komponenten die Daten unterschiedlich interpretieren oder dass Daten falsch oder verspätet übergeben werden. Der Systemtest prüft das Gesamtsystem in Hinblick auf die spezifizierten Anforderungen um festzustellen ob diese korrekt und vollständig sind. Die letzte Teststufe ist der Abnahmetest an dem der Kunde beteiligt ist. Diese Stufe wird im folgenden nicht miteinbezogen.
Da es nicht möglich ist das komplette System ausreichend zu testen und alle Fehler aufzudecken, beschränken sich die Tests auf die wichtigsten und fehleranfälligsten Teile des Systems. Dabei ist es wichtig, dass keine Fehler auftreten, die zu einen Systemabsturz führen und dass wesentliche Funktionen keine fehlerhaften Aktionen verursachen [SL05]. Das Hauptaugenmerk beim Testen des vorliegenden Systems liegt auf dem Systemtest. Dort wird auch das Zusammenspiel mit dem Test-Sensornetzwerk überprüft.

6.1 Komponenten- und Integrationstest

Der Komponenten- und Integrationstest wird mit JUnit, einem Framework zum Testen von Java-Programmen, durchgeführt. Die Testklassen dazu finden sich in dem `test`-Package des Programms.
Die geprüften Komponenten sind:
Input Validator: Der Input Validator prüft die Nutzereingaben auf Korrektheit und ist ein wesentlicher Kriterium für das richtig Arbeiten des Systems. Darum wird getestet, ob durch den Input Validator falsche Eingaben abgefangen und richtige Eingaben an das System weitergegeben werden. Für jede Methode des Input Validator liegt eine Testklasse mit mehreren Testfällen vor.
Auswertung: Für die Auswertung wird die korrekte Berechnung des Minimal-, Maximal- und Durchschnittswerts durch den Test sichergestellt.

Die maximalen Tage: Die Zeitwerte liegen auf der Benutzeroberfläche in Spinnern vor. Die maximalen Tage eines Monats werden gesondert berechnet. Diese Berechnung erfolgt in der `getMaxDays`-Methode der `Time`-Klasse und wird ebenfalls einem JUnit-Test unterzogen.

Durch den Integrationstest geprüfte Teile sind:
MyProperties: Damit das System korrekt funktioniert, müssen Werte in die Properties-Datei geschrieben und über die `MyProperties`-Klasse ausgelesen werden. Diese Umsetzung ist sehr fehleranfällig, da die Werte keiner Beschränkung unterliegen. Darum wird das Zusammenspiel dieser Klasse mit der Datei und dem Input Validator überprüft indem Testproperties verwendet werden.

6.2 Systemtest

Die funktionalen Anforderungen werden an unterschiedlichen Stellen des Systems umgesetzt.

- Anforderung: Anzeige der aktuellen Daten
  Die aktuellen Daten werden in einer Tabelle in der Data-Ansicht angezeigt. Über die an der linken Fensterseite gegebenen Auswahlmöglichkeiten ist eine Beschränkung auf Sensoren sowie Knoten möglich.
- Anforderung: Anzeige der archivierten Daten
  Die archivierten Daten werden ebenfalls in der Tabelle in der Data-Ansicht angezeigt, wenn der entsprechende Modus ausgewählt ist. Die Beschränkung erfolgt wie bei den aktuellen Daten und kann sich auf Sensoren sowie Knoten beziehen. Darüber hinaus kann der Nutzer einen Zeitbereich angeben in dem die Daten angezeigt werden sollen.
- Anforderung: Anfrage an das Sensornetzwerk
  Die Anfragen an das Sensornetzwerk erfolgen über den Query-Dialog, der über Menü oder die Toolbar aufgerufen werden kann. Dort kann der Nutzer festlegen ob die Anfrage an das gesamte Sensornetzwerk gesendet werden soll oder an ausgewählte Sensoren und Knoten und ob die Anfrage eine punktuell sein oder sie einer bestimmten Zeitbeschränkung unterliegen soll.
- Anforderung: Auswertung der Daten
  In der Analysis-Ansicht werden die Daten als Graph, Minimal-, Maximal- und Mittelwert ausgewertet. Auf der linken Fensterseite kann der Nutzer eine Beschränkung auf Sensoren und Knoten angeben sowie einen bestimmten Zeitbereich auswählen in dem die Auswertung getätigt werden soll. Er kann ebenfalls die Art der Auswertung festlegen.
- Anforderung: Alarm bei kritischen Werten
  Der Bereich für kritische Werte wird über den Alert-Dialog eingestellt, der über das Menü oder die Toolbar aufgerufen werden kann. Dort kann der Nutzer zu jedem Sensor einen minimal sowie einen maximalen Wert eingeben. Diese werden dann auf die Tabelle in der Data-View angewendet und Werte, die außerhalb des Bereichs liegen, werden farbig markiert. Bei Bedarf kann der Bereich verändert oder gelöscht werden.
- Anforderung: Anzeige des Sensornetzwerkzustandes
  In der Topology-Ansicht kann der Nutzer mehrere Ebenen erstellen und einen Gebäudeplan als Hintergrundbild hochladen. Die Knoten können per Drag and Drop angeordnet werden.
- Anforderung: Statusdaten
  Die Sensoren und der Energiezustands werden beim Anwählen eines Knoten an der rechten Seite des Fensters angezeigt. Die benachbarten Knoten werden auf den Ebenen durch Linien gekennzeichnet.

- Anforderung: Exportieren in XML
  Der Export in XML erfolgt über das Menü und den Menüpunkt Export. Zusätzlich wird die Beschränkung der Daten in der Data-Ansicht miteinbezogen.

Darüber hinaus wird ein besonderes Augenmerk auf das Testen der nichtfunktionalen Anforderungen gelegt. Der wichtigste Punkt ist die Flexibilität um das System für viele Sensornetzwerke nutzbar zu machen. Um das System an ein Sensornetzwerk anzupassen, müssen die Informationen in die Properties-Datei geschrieben werden. Außerdem muss der Adapter in dem `server.wsn.adapter`-Package liegen und kompatibel mit dem vorliegenden System sein. Die Bereitstellung der Sensoren und der Knoten auf der Benutzeroberfläche erfolgt in Abhängigkeit der in der Datenbank liegenden Informationen. So werden die Auswahl in der Data-, in der Analysis-Ansicht und in dem Query-Dialog aus den Datenbankwerten erstellten. Gleiches gilt für die Knoten, die im Node Home der Topology-Ansicht liegen und die Sensoren, die für die Einstellung der Alarmbereiche ausgewählt werden. Eine Beschränkung auf eine bestimmte Anzahl erfolgt dabei nicht, da sich die Benutzeroberfläche an die gegeben Werte anpasst. Die Berechnung der Auswertung in der Analysis-Ansicht erfolgt ebenfalls flexibel, da sie abhängig von den ausgewählten Werten durchgeführt wird. Ein weiterer Punkt sind die Ebenen in der Topology-Ansicht, die vom Nutzer mit einem selbstgewählten Gebäudeplan und Ebenennamen, erstellt werden können. Es gibt keine Beschränkung auf eine bestimmte Ebenenanzahl und der Gebäudeplan wird vom Nutzer gewählt. Dadurch kann das System für die unterschiedlichen Anwendungsgebiete genutzt werden.
Die Verifikation der Robustheit erfolgt durch die Überprüfung des Systemverhaltens bei Fehleingaben.
Die Benutzerfreundlichkeit ist durch die einfache Benutzeroberfläche gegeben. Außerdem werden bei unzulässigen Eingaben Meldungen an den Nutzer gegeben.
Da das vorliegende System die Daten aus Sensornetzwerken auf geeignete Weise über eine Benutzeroberfläche visualisieren soll, ist die Zufriedenheit des Nutzers ein wesentlich Kriterium für das Gelingen des Projekts. Aus diesem Grund wurde das System zusätzlich durch externe Nutzer getestet.

7 Fazit und Ausblick

Das System erfüllt alle gestellten Anforderungen und lässt sich durch seine an vielen Stellen umgesetzte Flexibilität leicht in unterschiedlichen Anwendungsgebieten nutzen. Für die Umsetzung wurden einige Ideen der im Kapitel 2 vorgestellten bestehenden Systeme übernommen und angesprochenen Kritikpunkte versucht zu umgehen.
Da MoteView als einziges der Systeme verfügbar ist, lässt sich dort der Vergleich am Einfachsten ziehen. Es wurde die Tab-Struktur und ihre Reihenfolge - die Daten als Tabelle finden sich im ersten Tab, die Topology im Letzten - übernommen, wobei das vorliegende System mit weniger Tabs auskommt. Die Visualisierung der Daten erfolgt in beiden Systemen auf mehrere Arten: In einer Tabelle, in der auch archivierte Daten zu finden sind, und als Graph(en). In der Topology wurde die Idee eine Bilddatei als Gebäudeplan im Hintergrund zu verwenden von MoteView übernommen ebenso wie das Platzieren der Knoten per Drag and Drop. Weitere positive Aspekte, die Eingang in das vorliegende System gefunden haben, sind die Darstellung der Nachbarschaftsbeziehungen von Knoten durch Linien und der Status an der rechten Fensterseite, der beim Fahren über einen Knoten angezeigt wird.
Die Hauptkritikpunkte, die in Kapitel 2 genannt wurden, werden vom vorliegenden System umgangen. Durch die schon erwähnte Flexibilität kann prinzipiell mit allen Sensornetzwerken gearbeitet werden, MoteView beschränkt sich hingegen auf die von Crossbow entwickelten Knoten. Ebenso besteht durch die Verwendung von Java, insbesondere Swing, eine Plattforumunabhängigkeit. Ein Unterschied zwischen den beiden Systemen ist die Darstellung der Tabelle. In MoteView wird jeder Sensor durch eine Spalte repräsentiert, im vorliegenden System sind alle Sensoren Inhalt der Sensor-Spalte. In der Topology von MoteView gibt es nur eine Ebene auf der die Knoten platziert werden können, während das vorliegende System die Möglichkeit bietet, mehrere Ebenen hinzuzufügen. Ebenso lassen sich die Knoten bei MoteView nicht von dieser Topology entfernen. Wird ein Knoten gelöscht, wird er aus dem gesamten System entfernt. In dem in dieser Arbeit vorgestellten System besteht die Möglichkeit, die Knoten im Node Home abzulegen, wenn sie nicht benötigt werden. Eine Schwäche von MoteView ist der Export aller Daten als XML. Da keine Beschränkung auf bestimmte Bereiche möglich ist, wird der Nutzer mit einer großen und unüberschaubaren Datenmenge konfrontiert. Diese Schwäche wird in dem vorliegenden System durch die Möglichkeit der Auswahl über die Data-Ansicht behoben.
Da jWebDust sein Hauptaugenmerk auf die Multifunktionalität legt und zu der Benutzeroberfläche nicht viele Informationen zu finden sind, sind von diesem System am wenigsten Ideen übernommen worden. Für beide Systeme gilt, dass sie mit heterogenen Sensornetzwerken arbeiten und der Nutzer in der Benutzeroberfläche keine weiteren Einstellungen vornehmen muss. Bis auf die gleiche Programmiersprache, nämlich Java, treten keine weiteren nennenswerten Gemeinsamkeiten auf.
Die Benutzeroberfläche von Feit hat zu der Idee geführt, auf der Startseite die zuletzt gemessenen Werte anzuzeigen. Ebenso wurde übernommen was aus den Daten berechnet werden soll, nämlich der Minimal-, Maximal- und Durchschnittswert. Die Beschränkung der Auswahl der Sensoren und Knoten wurde ebenfalls in Ansätzen übernommen, wobei dies im vorliegenden System durch die Bestimmung eines Start- und Endzeitpunkts sowie die Auswahl mehrere Sensoren, detaillierter ist. Feit stellt die Daten in der Tabelle, ebenso wie MoteView, mit den Sensoren als Spalten dar, das im vorliegenden System, wie bereits erwähnt, umgangen wird. Die Topology von Feit ist ebenfalls mit einem Gebäudeplan ausgestattet und die Knoten können darauf verschoben werden. Die Daten zu einem Knoten werden rechts angezeigt. Feit hat trotz einiger Unterschiede viel zu dem vorliegen System beigetragen und liefert wenig Kritikpunkte, die verbessert werden könnten. Der Punkt, dass aufgrund der von Crossbow entwickelten Software nicht alle Sensornetzwerke verwendet werden können, wird, wie schon erwähnt, von dem vorliegenden System umgangen.

Das System wurde bei der Entwicklung auf die wichtigsten Grundfunktionen reduziert um einen sinnvollen Umgang mit Sensornetzwerkdaten gewährleisten zu können. Darüber hinaus lassen sich an vielen Stellen Verbesserungen und Erweiterungen einfügen.
Der Client lässt sich alternativ auch als Webanwendung umsetzen und kann somit für mehrere Nutzer über das Internet bereitgestellt werden. Ein wichtiger Punkt ist die Eingabe der Informationen über das Sensornetzwerk, die zurzeit über das config.properties-Datei erfolgt, über eine benutzerfreundliche Eingabemaske zu realisieren. Dadurch wird der Nutzer besser bei der Eingabe unterstützt indem Fehleingaben schon währenddessen mitgeteilt werden.
In der Analysis-Ansicht ist es denkbar, die Auswertung ebenfalls in einem geeigneten Format abzuspeichern um dem Nutzer die Möglichkeit zu geben, diese Werte miteinander zu vergleichen.
Einige wichtige Erweiterungen lassen sich auch in dem Query-Teil durchführen. Zurzeit wird eine nicht an das Sensornetzwerk gesendete Anfrage in der Datenbank gespeichert ohne die Möglichkeit, diese erneut zu schicken. Dem Nutzer sollte eine Ansicht zur Verfügung gestellt werden, die alle laufenden und nicht gesendeten Anfragen anzeigt und die darüber hinaus Buttons für das Abbrechen oder erneute Senden einer Anfrage bereitstellt. Es ist ebenfalls sinnvoll den Anfragen Prioritäten zuordnen zu können, wenn beispielsweise mehrere Anfragen zu einem gleichen Zeitpunkt gestellt werden.
Die Markierung von kritischen Werten kann zusätzlich in der Analysis-Ansicht in den Graphen und in den Werten erfolgen. Darüber hinaus ist eine weitere Differenzierung bei der Eingabe von Alarmwerten sinnvoll z.B. die Beschränkung auf bestimmte Knoten.
Um dem Nutzer eine noch bessere Visualisierung in der Topology zu geben, können die Verbindungen zwischen benachbarten Knoten in unterschiedlichen Farben je nach Verbindungsstärke dargestellt werden.
Das Exportieren ist im vorliegenden System nur in XML möglich. Denkbar sind jedoch andere Formate wie beispielsweise CSV.

Trotz dieser Verbesserungsmöglichkeiten bietet das vorliegende System ausreichend Funktionen um vernünfigt mit diesem arbeiten zu können. Der Nutzer kann sich die Daten auf unterschiedliche Art anzeigen lassen und somit das System für die Arbeit mit einem Sensornetzwerk nutzen.

Literaturverzeichnis

[BD04] BRÜGGE, B. ; DUTOIT, A.H.: *Objektorientiere Softwaretechnik mit UML, Entwurfsmustern und Java*. Pearson Studium, 2004

[CES04] CULLER, D. ; ESTRIN, D. ; SRIVASTAVA, M.: Overview of Sensor Networks. In: *IEEE Computer* 37 (2004), S. 41–49

[CMN05] CHATZIGIANNAKIS, I. ; MYLONAS, G. ; NIKOLETSEAS, S.: jWebDust: A Java-based Generic Application Environment for Wireless Sensor Networks. In: *First IEEE International Conference (DCOSS 2005)*, 2005

[Cro07] CROSSBOW: *MoteView Users Manual*, 2007

[HHSH06] HARTUNG, C. ; HAN, R. ; SEIELSTAD, C. ; HOLBROOK, S.: FireWxNet: A Multi-Tiered Portable Wireless System for Monitoring Wearther Conditions in Wildland Fire Environments. In: *MobiSys '06: Proceedings of the 4th international conference on Mobile systems, applications and services*, 2006

[RM04] RÖMER, K. ; MATTERN, Friedemann: The Design Space of Wireless Sensor Networks. In: *IEEE Wireless Communications* 11 (2004), S. 54–61

[RV04] RIEM-VIES, R.: Cold Chain Management using an Ultra Low Power Wireless Sensor Network. In: *WAMES 2004*, 2004

[SCB+09] STROULIA, E. ; CHODOS, D. ; BOERS, N.M. ; HUANG, J. ; GBURZYNSKI, P. ; NIKOLAIDIS, I.: Software-Engineering for Healt Education and Care Delivery Systems: The Smart Condo Project. In: *2009 ICSE Workshop on Software Engineering in Health Care*, IEEE Computer Society, 2009

[SD09] STOJKOSKA, B. ; DAVCEC, D.: Web Interface for Habitat Monitoring using Wireless Sensor Network. In: *2009 Fifth International Conference on Wireless and Mobile Communications*, 2009, S. 157–162

[SL05] SPILLNER, A. ; LINZ, T.: *Basiswissen Softwaretest*. dpunkt.verlag, 2005

[Sto05] STOJMENOVIC, I.: *Handbook of Sensor Networks: algorithms and architectures*. Wiley-Interscience, 2005

[Tur05] TURON, M.: MOTE-VIEW: A Sensor Network Monitoring and Management Tool. In: *The 2nd IEEE Workshop on Embedded Network Sensors*, IEEE Computer Society, 2005

Erklärung zur Urheberschaft

Hiermit versichere ich, dass ich diese Arbeit selbständig verfasst und keine anderen als die angegebenen Hilfsmittel und Quellen benutzt habe.

Cloppenburg, den 15. September 2010

Carolin Wiechert