

Moduldokumentation

(MOD)

(TIT10AID, SWE I Praxisprojekt 2011/2012)

Projekt: Multicastor2.0

Auftraggeber: Stuckert/Rentschler

Auftragnehmer: *Eisenhofer Manuel*
Kern Michael
Michelchen Tobias
Schumann Pascal
Scharton Roman

Ansprechpartner: Manuel Eisenhofer

Modul

<XML-Parser>

Inhalt

1.	History	3
2.	Scope	4
3.	Definitionen	5
3.1.	Abkürzungen	5
3.2.	Definitionen	5
3.3.	Terminologien	5
4.	Anforderungen	7
4.1.	Benutzersicht	7
4.2.	Kontext der Anwendung	7
4.3.	Anforderungen	7
5.	Analyse des XML-Parser1.0	7
5.1.1.	XML-Struktur	8
5.1.2.	Architektur	8
5.1.3.	Methoden	9
5.2.	Implementierung	10
6.	Design XML-Parser2.0	12
6.1.1.	XML-Struktur	12
6.1.2.	Architektur	13
6.1.3.	Methoden	14
6.2.	Implementierung	16
7.	Risiken	18
8.	Komponententest	19
8.1.	Komponententestplan	19
8.2.	Komponententestreport	19
9.	Zusammenfassung	20
9.1.	Beurteilung der Komponente	20
9.2.	Ausblick für die Weiterentwicklung	20
10.	Anhang	21
10.1.	Literatur	21
10.2.	Standards	21
10.3.	Diagramme	21
10.4.	Schnittstellendefinitionen	21
10.5.	Code	21
10.6.	Testfälle	21

1. History

Version	Datum	Autor(en)	Kommentare
1.0	20.03.2012	Manuel Eisenhofer	Erstellung des Dokuments und erste Ergänzungen
1.1	29.04.2012	Manuel Eisenhofer	Analyse XML-Parser1.0
1.2	03.01.2012	Manuel Eisenhofer	Änderungen XML-Parser2.0
1.3	10.05.2012	Manuel Eisenhofer	Fertigstellung des Dokuments

2. Scope

The Module Documentation (MOD) describes the architecture, the interfaces and the main features of the module. It also describes the module/component test including the results. It can also serve as a programming or integration manual for the module. If there are some risks related to the module itself, they shall be noted and commented within this document.

Die Moduldokumentation beschreibt die Architektur, die Schnittstellen und die Hauptmerkmale des Moduls. Außerdem werden die Modul bzw. Komponententests einschließlich der Ergebnisse beschrieben und dokumentiert. Die MOD dient bei Bedarf auch als Programmier- oder Integrationshandbuch für das Modul. Wenn bestimmte Risiken direkt mit der Verwendung des Moduls verknüpft sind, so sind sie in diesem Dokument zu benennen und zu kommentieren.

3. Definitionen

3.1. Abkürzungen

CRS	Customer Requirement Specification
MC	Multicast
MCD	Multicast Data
SAX	Simple API for XML
SRS	System Requirement Specification
UID	User Input Data
ULD	User Level Data
XML	Extensible Markup Language

3.2. Definitionen

Reflections: bedeutet, dass ein Programm seine eigene Struktur kennt und diese, wenn nötig, modifizieren kann. Somit ist es möglich, zur Laufzeit, Methoden von Klassen zu erkennen und auszuführen.

XPath: XML Path Language ist eine Abfragesprache, um Knoten/Attribute eines XML-Dokuments adressieren zu können. Sie ist durch das W3-Konsortium entwickelt worden.

Multicast: Nachrichtenübertragung von einem Punkt zu einer Gruppe (Mehrpunktverbindung). Nachrichten können gleichzeitig an mehrere Teilnehmer oder an eine geschlossene Teilnehmergruppe übertragen werden, ohne dass sich beim Sender die Bandbreite mit der Zahl der Empfänger multipliziert. Der Sender braucht beim Multicasting nur die gleiche Bandbreite wie ein einzelner Empfänger. Handelt es sich um paketerorientierte Datenübertragung, findet die Vervielfältigung der Pakete an jedem Verteiler auf der Routerseite statt. Beim Multicast ist eine vorherige Anmeldung bei dem Aussender des Inhaltes vonnöten.

3.3. Terminologien

<SENDER_V4> definiert ob es sich um einen Sender oder Receiver und ob es sich um ein IGMP oder MLD Protokoll handelt

<active> beinhaltet einen Boolean ob der Multicast gestartet oder gestoppt ist

<groupId> beinhaltet die Group-Adresse

<sourceIp> beinhaltet die Source-Adresse

<udpPort> beinhaltet den Port

<packetLength> beinhaltet die Länge des Packetes

<tTl> beinhaltet den Wert für die Lebensdauer (Time to live)

<packetRateDesired> beinhaltet die gewünschte Packetrate

<typ> beinhaltet den Typ des Multicasts

MulticastData: Container für Multicast Konfigurationsdaten.

UserlevelData: Container für Informationen welche Elemente in der GUI angezeigt werden.

UserInputData: Wird für AutoSave benötigt. Enthält Information über momentan selektierten Tab und eingegebenen Werte.

MulticastController: Ist ein Objekt, welches sich um die Behandlung von Multicasts kümmert.

PanelMulticastConfigNewReceiver: Ist eine Klasse, welche von der JPanel-Klasse erbt und wird auf der GUI des Multicastors verwendet.

4. Anforderungen

4.1. Benutzersicht

Dieses Modul wird benötigt um Konfigurationsdateien zu laden, zu interpretieren und zu speichern. Die Konfigurationsdaten müssen im nachfolgend definierten XML-Format in der Konfigurationsdatei vorliegen.

Das Modul muss auch eine Konfigurationsdatei mit Default-Werten für den Multicast-Protokoll-Handler erzeugen können.

4.2. Kontext der Anwendung

Das Modul wurde implementiert um erzeugte Daten abspeichern und wieder laden zu können. Dadurch müssen Daten nicht bei jedem Start neu eingetragen werden.

4.3. Anforderungen

4.3.1. Funktionale Anforderungen

/UC42/: Einstellungen können in der Konfigurationsdatei gespeichert werden.

/F10/ (/LF10/): Der XML-Parser2.0 muss MMRP Daten abspeichern und laden können

/F90/ (/LF90/): Beim Anlegen eines Multicasts, werden automatisch passende Default-Werte eingetragen, um bei Bedarf schnell und einfach einen Test durchführen zu können.

/F100/ (/LF100/): Default-Werte können im Konfigurationsfile editiert werden.

4.3.2. Nichtfunktionale Anforderungen

/LL20/: Das Starten des Multicastors darf nicht länger wie 5 Sekunden dauern. Somit wurde abgeschätzt, dass das Laden eines Konfigurationsfiles nicht länger wie 2 Sekunden dauern darf.

5. Analyse des XML-Parser1.0

Die Analyse des XML-Parser1.0 befasst sich mit dem Erkennen von Schwachstellen des XML-Parsers, der XML-Struktur und der Implementierung.

Im Nachfolgenden werden die Ergebnisse der Analyse präsentiert.

5.1. XML-Struktur

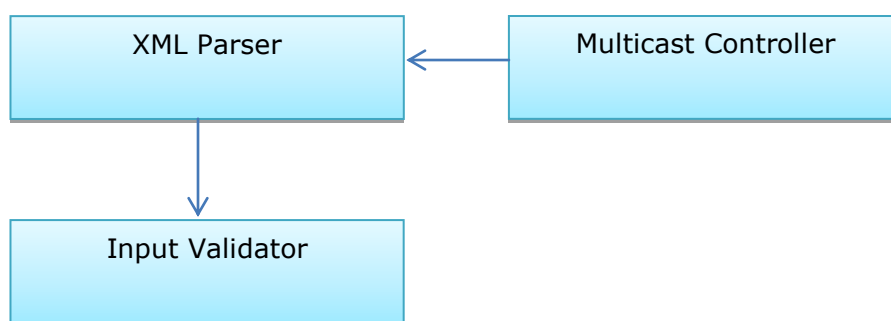
```
<MultiCastor>
  <Multicasts>
    <SENDER_V4> <!--definiert eine Konfiguration für einen Sender vom Typ IGMP-->
      <active>true</active>
      <groupIp>230.0.0.1</groupIp>
      <sourceIp>192.168.2.102</sourceIp>
      <udpPort>1000</udpPort>
      <packetLength>64</packetLength>
      <tTl>7</tTl>
      <packetRateDesired>12</packetRateDesired>
      <typ>SENDER_V4</typ> <!--definiert wiederum den Typ (Redundant)-->
    </SENDER_V4> <!--es wird immer zwischen Sender und Receiver unterschieden-->
  </Multicasts>
</MultiCastor>
```

Schwachstellen:

Im oberen XML-Ausschnitt werden 2 Problematiken aufgezeigt.

Eine Weitere ist, dass dieses XML-File nicht identifiziert werden kann, zum Beispiel durch ein Datum. Ebenso gibt es keine Möglichkeit Daten für das, neu eingebaute, Protokoll MMRP einzufügen.

5.2. Architektur



Schwachstellen:

Es gibt nur einen XML-Parser der für alles, was mit Speichern und Laden von sämtlichen Konfigurationen zu tun hat, zuständig ist. Somit ist der Code in einer einzigen Klasse und somit schwer zu analysieren. Denn der XML-Parser kann UserlevelData, MulticastData, UserInputData und Pfade abspeichern. Für jedes dieser Features besitzt er eine eigene Methode, mit weiteren nicht öffentlichen Methoden.

Diese Häufung von Funktionalität in einer Klasse sollte gesplittet werden.

5.3. Methoden

Um alle gewünschten Funktionalitäten zu gewährleisten wurden folgende nach außen sichtbare (public) Funktionen erstellt:

Return Type	Name	Parameter	Wirft Exceptions	Funktion
void	loadConfig	String Vector<MulticastData> Vector<UserlevelData> Vector<UserInputData> Vector<String>	SAXException,FileNotFoundException, IOException	Lese eine Konfiguration aus einer XML Datei ein.
void	loadConfig	String Vector<MulticastData> Vector<UserlevelData>	SAXException,FileNotFoundException, IOException	Lese eine Konfiguration aus einer XML Datei ein.
void	saveConfig	String Vector<MulticastData> Vector<UserlevelData> Vector<UserInputData> Vector<String>	IOException	Speichere eine Konfiguration ab.
void	saveConfig	String Vector<MulticastData> Vector<UserlevelData>	IOException	Speichere eine Konfiguration ab.
void	loadDefaultULD	Vector<MulticastData> Vector<UserlevelData>	SAXException, IOException	Lade die beiden default Userlevel „Beginner“ und „Expert“

Tabelle 1: XmlParser Funktionen

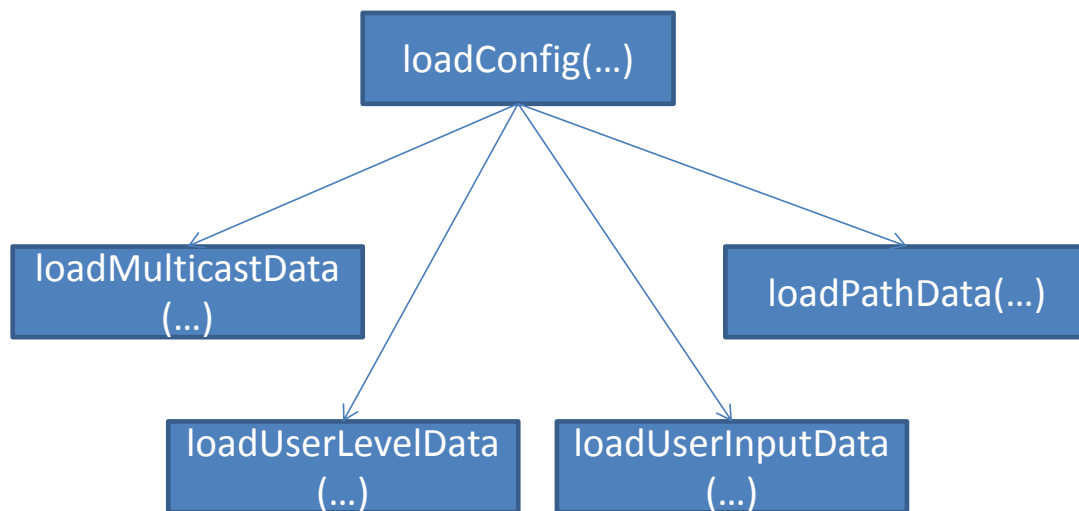


Diagramm 1 Methoden XML-Parser1.0

Wenn in der Klasse XmlParser die Methode loadConfig(Vector,Vector,...)aufgerufen wird, wird für jeden übergebenen Vector eine extra private (nicht sichtbare) Funktion aufgerufen.

Das Selbe wird in der Methode saveConfig(...) ausgeführt, mit dem Unterschied, dass in saveConfig die Methoden saveMulticastData(...), saveUserInputData(...) usw. aufgerufen werden.

Dadurch, dass der XML-Parser verkleinert bzw. auf mehrere Klassen aufgeteilt werden soll, werden nicht alle Methoden in der Klasse bleiben.

5.4. Implementierung

Parser:

Im Dokument [1]wird beschrieben, dass zum Einlesen der XML-Dateien ein SAXParser verwendet wird. Bei der Analyse des Xml-Parsers1.0 ist jedoch aufgefallen, dass ein DOMParser verwendet wurde.

Der DOMParser ist gut für kleinere XML-Dateien geeignet, da er das XML-File komplett in den Arbeitsspeicher lädt und man dadurch zwischen den einzelnen Knoten beliebig wechseln kann.

Defaultdaten:

Es gibt keine Implementierung für das Laden von Default-Daten.

Falls weitere Informationen über die alte Implementierung, Architektur oder Ähnliches gewünscht sind wird auf die Quelle [1]verwiesen.

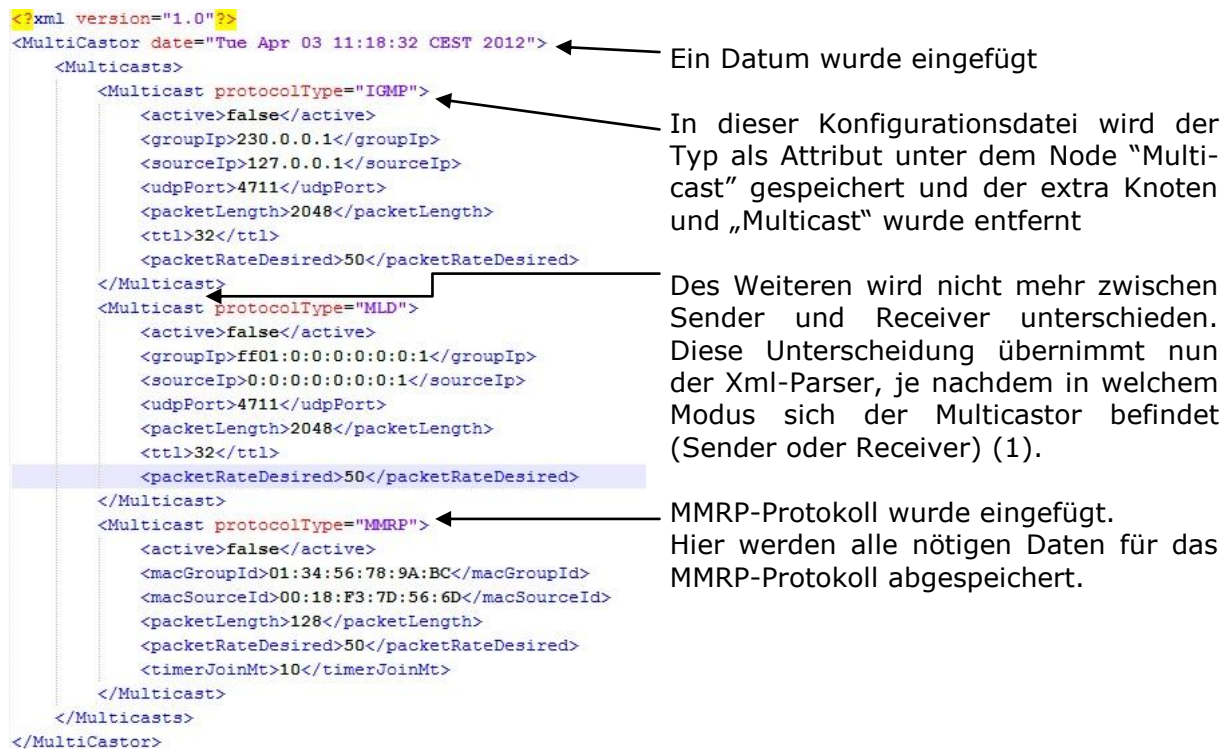
6. Design XML-Parser2.0

6.1. Erweiterungen XML-Parser 2.0

Der neue XML-Parser2.0 soll folgende Erweiterungen realisieren:

- 1) Erstellungsdatum im Header
- 2) Entfernen der UserLevelData
- 3) Auslagern der UserInputData aus dem Modul <XML-Parser> in das Modul <XML-ParserWorkbench>
- 4) Änderung des XML-Datenmodells
 - a. MMRP-Protokolldaten hinzufügen
 - b. Behandlung von redundanten Tags

6.2. XML-Struktur



```
<?xml version="1.0"?>
<MultiCastor date="Tue Apr 03 11:18:32 CEST 2012">
  <Multicasts>
    <Multicast protocolType="IGMP">
      <active>false</active>
      <groupIp>230.0.0.1</groupIp>
      <sourceIp>127.0.0.1</sourceIp>
      <udpPort>4711</udpPort>
      <packetLength>2048</packetLength>
      <tTtl>32</tTtl>
      <packetRateDesired>50</packetRateDesired>
    </Multicast>
    <Multicast protocolType="MLD">
      <active>false</active>
      <groupIp>ff01:0:0:0:0:0:0:1</groupIp>
      <sourceIp>0:0:0:0:0:0:0:1</sourceIp>
      <udpPort>4711</udpPort>
      <packetLength>2048</packetLength>
      <tTtl>32</tTtl>
      <packetRateDesired>50</packetRateDesired>
    </Multicast>
    <Multicast protocolType="MMRP">
      <active>false</active>
      <macGroupId>01:34:56:78:9A:BC</macGroupId>
      <macSourceId>00:18:F3:7D:56:6D</macSourceId>
      <packetLength>128</packetLength>
      <packetRateDesired>50</packetRateDesired>
      <timerJoinMt>10</timerJoinMt>
    </Multicast>
  </Multicasts>
</MultiCastor>
```

Ein Datum wurde eingefügt

In dieser Konfigurationsdatei wird der Typ als Attribut unter dem Node "Multicast" gespeichert und der extra Knoten und „Multicast“ wurde entfernt

Des Weiteren wird nicht mehr zwischen Sender und Receiver unterschieden. Diese Unterscheidung übernimmt nun der Xml-Parser, je nachdem in welchem Modus sich der Multicastor befindet (Sender oder Receiver) (1).

MMRP-Protokoll wurde eingefügt. Hier werden alle nötigen Daten für das MMRP-Protokoll abgespeichert.

Abbildung 1 XML-Struktur XML-Parser2.0

(1) Das Abspeichern ohne anzugeben ob es sich um einen Sender oder Receiver handelt hat folgenden Vorteil. Wenn ein Multicaststrom im Sender angelegt und die Konfigurati-

onsdatei abgespeichert wird, kann später die Konfiguration als Receiver geladen werden. Dies ist möglich, da der Sender alle nötigen Informationen enthält, die auch ein Receiver benötigt. Nur bei MMRP würde dies nicht funktionieren, da der MMRP-Receiver ein timer-JoinMt-Attribut benötigt. Hier wird jedoch beim Abspeichern als Sender ein Default-Wert 10 in die Konfigurationsdatei geschrieben. Somit können alle Sender-Konfigurationen auch als Receiver geladen werden.

Weitere Vorteile die daraus resultieren:

- ➔ Redundante Typangabe wurde entfernt
- ➔ Datum zur Wiedererkennung wurde eingefügt
- ➔ MMRP-Daten können geladen und gespeichert werden

6.3. Architektur

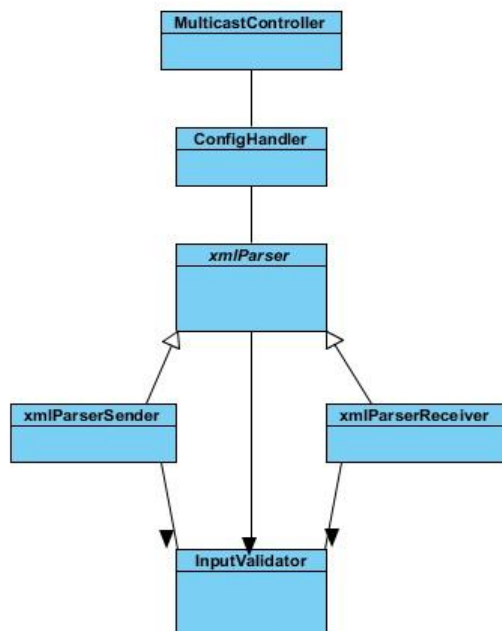


Diagramm 2 Xml-Parser2.0 Architektur

- ➔ Es gibt 4 Xml-Parser
 - xmlParser (abstrakt)
 - xmlParserSender (erbt von xmlParser)
 - xmlParserReceiver (erbt von xmlParser)
 - der vierte XML-Parser ist ein eigenes Modul und wird in Quelle [2]näher erklärt
- ➔ MulticastController greift nicht mehr direkt auf den Xml-Parser zu, sondern geht über einen ConfigHandler

Für die Validierung der Daten wurde eine extra Klasse erstellt. Diese heißt InputValidator. Mit Hilfe dieser Klasse werden die Daten in den Xml-Parsern überprüft.

Da die Konfigurationsdatei umgestellt wurde, wird nicht mehr zwischen Sender- und Receiver-Daten, im Konfigurationsfile, unterschieden. Dafür ist nun der Parser zuständig. Je nachdem ob man sich im Modus Sender oder Receiver befindet, müssen verschiedene Validierungskriterien beim Laden einer Konfigurationsdatei angewendet werden. Dies bedeutet, dass es zwei XML-Parser gibt. Einen für den Fall, dass man sich im Modus Sender befindet und den anderen für den Modus Receiver.

Damit der Multicast-Controller diese Änderung ignorieren kann, wurde ein Config-Handler eingefügt. Dieser trifft Unterscheidungen welcher der beiden Parser aufgerufen werden soll. Somit ermöglicht er, dass der Multicast-Controller, trotz der Änderung, die public Methoden load(...) bzw. save(...) aufrufen kann.

Dieses Design wurde nach der Vorgabe des Facade Design Pattern erstellt.

Die folgende Abbildung beschreibt näher was die Vorteile daraus sind.

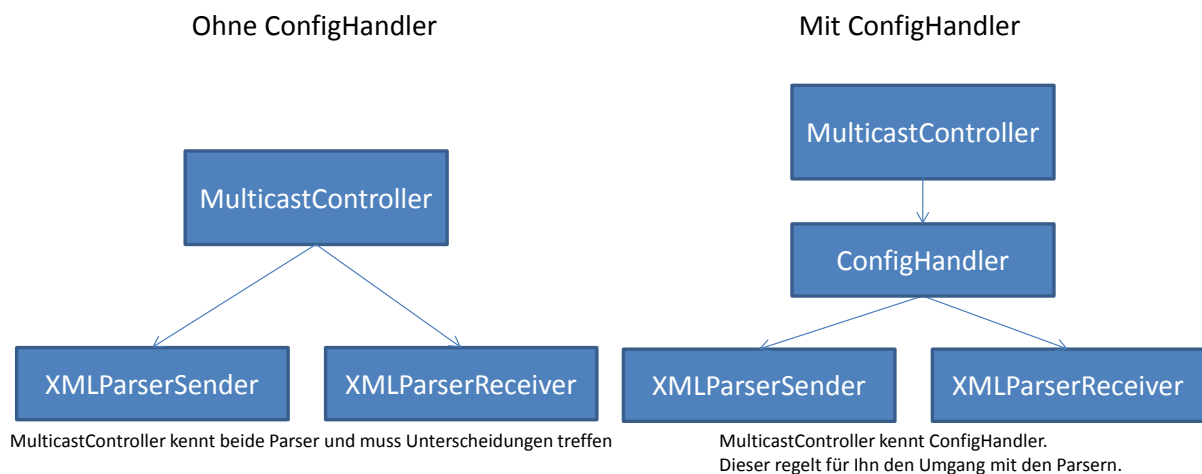


Diagramm 3 ConfigHandler

Durch diese Verbesserung wird Logik aus dem MulticastController entfernt und zentral im ConfigHandler gehalten. Somit könnten auch andere Klassen auf den ConfigHandler zugreifen und müssten ebenso keine weiteren Entscheidungen zw. Sender- und Receiver-Parser vornehmen.

6.4. Methoden

Return Type	Name	Parameter	Wirft Exceptions	Funktion
void	loadConfig	String Vector<MulticastData>	SAXException,FileNotFoundException,IOException	Lese eine Konfiguration aus einer XML Datei ein.
void	saveConfig	String Vector<MulticastData>	IOException	Speichere eine Konfiguration ab.

void	loadDefault-MulticastData	String	WrongConfigurationException, SAXException, IOException	Lade die Default-Werte aus einer Konfigurationsdatei
------	---------------------------	--------	--	--

Tabelle 2: XmlParser2.0 Methoden

Beim Vergleich der Tabelle 1 mit Tabelle 2 kann man erkennen, dass es anstatt 5 Methoden nur noch 3 gibt. Dafür gibt es eine neue Klasse „ConfigHandler“ mit weiteren Methoden.

Return Type	Name	Parameter	Wirft Exceptions	Funktion
void	loadConfig	MulticastController	---	Ruft die load-Methode des ConfigHandlers auf mit load(„“, true, MulticastController)
void	loadConfig	String Boolean MulticastController	---	Ruft die load-Methode eines XML-Parsers auf und behandelt die Exceptions. Der Boolean wird benötigt um unterscheiden zu können ob die MulticastDefaultdatei geladen werden soll oder nicht.
void	load-DefaultMulticastData	void	---	Ruft die load-DefaultMulticastData-Methode eines XML-Parsers auf und behandelt alle Exceptions
void	loadConfig-WithoutGUI	String MulticastController	FileNotFoundException SAXException IOException	Ruft die load-Methode eines XML-Parsers auf, jedoch ohne eine Updatefunktion für die GUI.
void	saveConfig	String Boolean Vector<MulticastData>	---	Ruft save-Methode eines XML-Parsers auf und behandelt die Exceptions

Tabelle 3: ConfigHandler Methoden

Die in Tabelle 3 beschriebenen Methoden, sind dafür da, die Unterscheidung zwischen einem Methodenaufruf über den xmlParserSender oder den xmlParserReceiver zu tätigen.

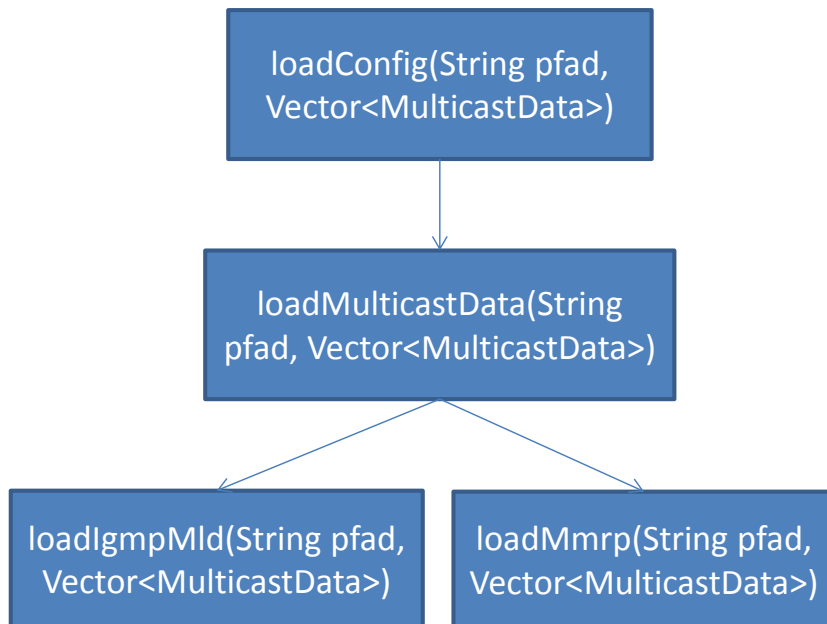


Diagramm 4 loadConfig Xml-Parser2.0

Wenn die Methode `loadConfig(...)` des XML-Parsers, aus Diagramm 4, aufgerufen wird, wird geprüft, ob der übergebene Parameter ungleich „null“ ist. Danach wird die private `loadMulticastData(...)` Methode aufgerufen, welche alle IGMP, MLD und MMRP-Daten, mit Hilfe der privaten Methoden `loadIgmpMld(...)` und `loadMmrp(...)`, lädt.

Dasselbe Verfahren wird für `saveConfig(...)` angewendet.

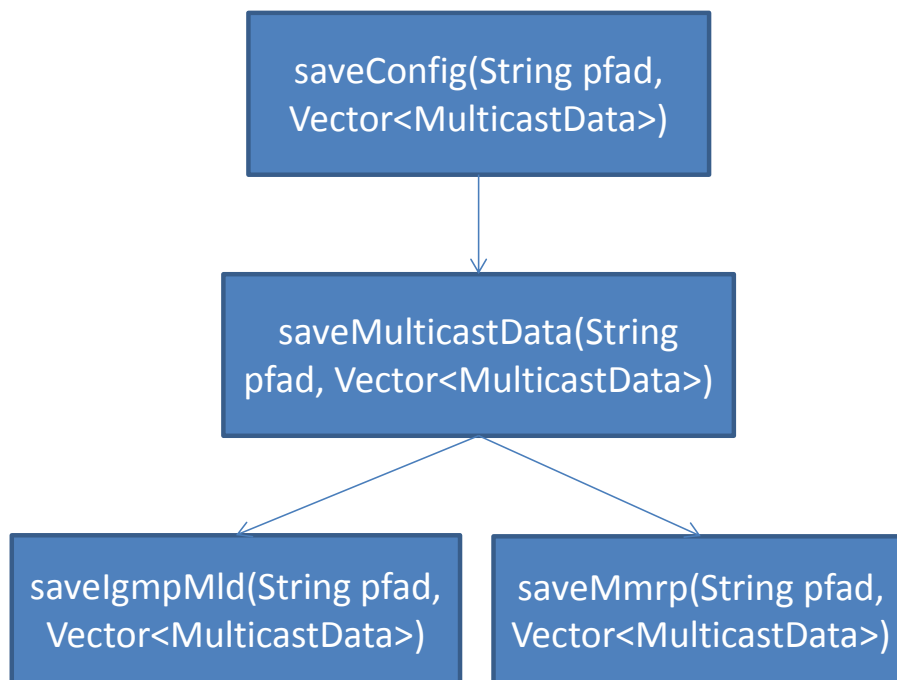


Diagramm 5 saveConfig Xml-Parser2.0

6.5. Implementierung

Parser:

In 5.4 wird beschrieben, welche Vorteile der DOMParser besitzt. Aufgrund dieser, wird im XML-Parser2.0 wiederum der DOMParser verwendet.

Suchen der Knoten:
Die Knoten, im XML-File, werden per XPath adressiert.

Default-Daten:

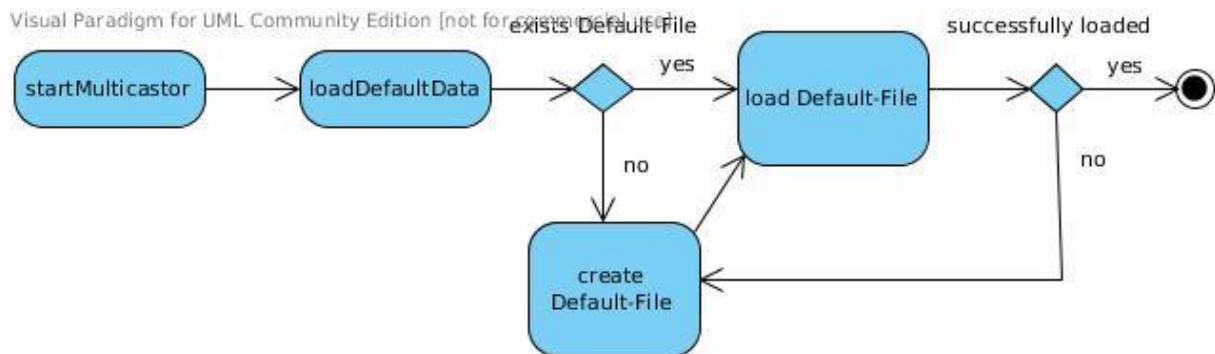


Diagramm 6 load Default Data

Diagramm 6 zeigt, wie das Laden von Default-Daten realisiert wurde. Nachdem der Multicaster gestartet wurde, wird versucht ein Default-File, welches sich im Ordner des Multicasters befinden muss, zu öffnen. Falls dieses nicht vorhanden ist, wird ein Default-File erstellt und im Ordner des Multicasters abgespeichert.

Falls das File schon existiert hat oder erstellt wurde, wird versucht die Datei zu laden. Wenn das Default-File schon existiert hat, kann es Fehler enthalten. Deswegen wird nach dem Laden abgefragt ob Fehler aufgetreten sind. Falls es Fehler gab, wird das Default-File neu erstellt und geladen.

Der Fall, dass es zu einem Deadlock kommt, wurde durch entsprechende Implementierung beseitigt.

Die Methode loadDefaultMulticastData(..) kann im ConfigHandler aufgerufen werden.

Für die Abspeicherung der Default-Daten wurde eine Klasse DefaultMulticastData erstellt. Diese enthält die in Diagramm 7 dargestellten Klassenmethoden. Somit kann von überall auf die Default-Daten zugegriffen werden.

Zugriffspunkte sind im XML-Parser, um die Default-Daten einzutragen, und im PanelMulticastConfigNew- Receiver/Sender, um Default-Daten auf die Oberfläche zu schreiben.

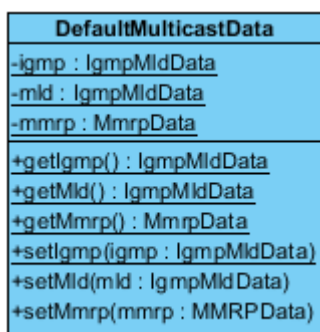


Diagramm 7 Klassendiagramm DefaultMulticastData

7. Risiken

Risiko1:

Eintragung falscher Daten als Multicastdaten.

Risikominderung:

Einbauen eines InputValidator, welcher Eingabedaten aus dem XML-File validiert und das Ergebnis dem XML-Parser zurückgibt.

Risiko2:

Laden einer XML-Datei, welche Strukturfehler besitzt oder falsche Tag-Namen enthält.

Risikominderung:

Vor dem Laden der XML-Datei wird eine Überprüfung gestartet, die auf das korrekte XML-Schema prüft. Wenn sich Strukturfehler in der Datei befinden, wird die Datei nicht geladen. Wenn einzelne Multicastdaten Fehler enthalten, werden nur die korrekten Multicastdaten geladen.

8. Komponententest

Zu Vorgehensweise und Aufbau des Tests siehe [3].

8.1. Komponententestplan

TestNo	Test Specification (Description or TCS)
<TC-003-001>	Konfiguration Laden: Prüfung auf gültiges XML-Schema
<TC-003-002>	Konfiguration Laden: Prüfung auf gültiges XML-Format
<TC-003-003>	Konfiguration Laden: Prüfung der korrekten Dateieindung des Konfigurationsmediums
<TC-003-004>	Konfiguration Laden: Konfiguration mit 200 Streams laden
<TC-003-005>	Laden: Nicht vorhandene Network-Schnittstellen
<TC-003-006>	Load-Incremental / not Incremental
<TC-003-007>	ungültige Werte in der Konfigurationsdatei
<TC-003-008>	Laden: Datei mit gültigen und ungültigen Daten
<TC-003-009>	Laden: Fehler in der Defaultdaten
<TC-003-010>	Speichern: ungültiger Dateinamen mit und ohne Dateinamen

Tabelle 4 Test Specification

8.2. Komponententestreport

TestNo	Pass/Fail	If failed: Test Result	Date	Tester
<TC-003-001>	pass	-	26.04.2012	Eisenhofer
<TC-003-002>	pass	-	26.04.2012	Eisenhofer
<TC-003-003>	pass	-	26.04.2012	Eisenhofer
<TC-003-004>	pass	-	26.04.2012	Eisenhofer
<TC-003-005>	pass	-	26.04.2012	Eisenhofer
<TC-003-006>	pass	-	26.04.2012	Eisenhofer
<TC-003-007>	pass	-	26.04.2012	Eisenhofer
<TC-003-008>	pass	-	26.04.2012	Eisenhofer
<TC-003-009>	pass	-	26.04.2012	Eisenhofer
<TC-003-010>	pass	-	26.04.2012	Eisenhofer

Tabelle 5 Komponententestreport

9. Zusammenfassung

9.1. Beurteilung der Komponente

Diese Komponente wurde gegenüber der Vorgängerversion vereinfacht, da Methoden in eine neue Komponente verlagert wurden. Dadurch wurde der Xml-Parser übersichtlicher und ein schnelleres Einarbeiten wurde ermöglicht.

Hinzufügen oder Löschen von Tags/Attributen.

Es muss eine Änderungen an zwei verschiedenen Stellen erfolgen:

- Im jeweiligen (imgpMld/mmrp -attributes)-Tag Enum
- In der save Methode wird es durch Reflections automatisch hinzugefügt
- In der Methode loadIgmpMld oder loadMmrp

9.2. Ausblick für die Weiterentwicklung

Eine mögliche Erweiterung wäre, dass die Architektur so verändert wird, dass beliebige Dateiformate abgespeichert werden können. Beispielsweise als Excel-Sheet oder im CSV-Format. Hierfür könnte eine Abstract Factory verwendet werden.

10. Anhang

10.1. Literatur

- [1] „TIT09AIB_ModDoc_XmlParser_Multicastor_1v0.doc,“ 2011.
- [2] „TIT10AID_MOD_XmlParserWorkbench_Team_3_0v1.doc,“ 2012.
- [3] „TIT10AID_STP_Multicastor_Team_3_0v1-2.doc,“ 2012.

10.2. Standards

-

10.3. Diagramme

Diagramm 1 Methoden XML-Parser1.0	10
Diagramm 2 Xml-Parser2.0 Architektur	13
Diagramm 3 ConfigHandler	14
Diagramm 4 loadConfig Xml-Parser2.0	16
Diagramm 5 saveConfig Xml-Parser2.0	16
Diagramm 6 load Default Data	17
Diagramm 7 Klassendiagramm DefaultMulticastData	17

10.4. Schnittstellendefinitionen

-

10.5. Code

-

10.6. Testfälle

10.6.1. Funktionale Anforderungen

Siehe [3] und 8. Komponententest.

10.6.2. Nichtfunktionale Anforderungen

/LL20/: Damit festgestellt werden konnte, ob diese Anforderung erfüllt wurde, musste ein Zeitstopper in das Programm mit eingebaut werden. Dieser Stopper wurde nach dem Test wieder entfernt.

Das Ergebnis war, dass der XML-Parser ca. 550ms benötigt um ein durchschnittliches XML-File zu laden. Somit ist die Anforderung erfüllt.