Moduldokumentation

(MOD)

(TIT10AID, SWE I Praxisprojekt 2011/2012)

Projekt:	MultiCastor-Tool V2.0
Auftraggeber:	Markus Rentschler Andreas Stuckert
Auftragnehmer:	TIT10AID – Team 3 Rotebühlplatz 41/1 70178 Stuttgart
Ansprechpartner:	Roman Scharton (Teamleiter) Michael Kern

Modul

<XmlParserWorkbench>

Inhalt

1. History 3
2. Scope 4
3. Definitionen 5
   3.1. Abkürzungen 5
   3.2. Definitionen 5
   3.3. Terminologien 5
4. Anforderungen 6
   4.1. Benutzersicht 6
   4.2. Kontext der Anwendung 6
   4.3. Anforderungen 6
5. Analyse 7
   5.1. Voruntersuchungen 7
   5.2. Systemanalyse 7
6. Design 9
7. Risiken 10
8. Implementierung 12
9. Komponententest 16
   9.1. Komponententestplan 16
   9.2. Komponententestreport 16
10. Zusammenfassung 19
   10.1. Beurteilung der Komponente 19
   10.2. Ausblick für die Weiterentwicklung 19
11. Anhang 20
   11.1. Literatur 20
   11.2. Standards 20
   11.3. Diagramme 20
   11.4. Schnittstellendefinitionen 20
   11.5. Code 20
   11.6. Testfälle 20

1. History

Version	Datum	Autor(en)	Kommentare
1.0	16.04.2012	Michael Kern	Erstellung des Dokuments
1.1	13.05.2012	Michael Kern	Dokumentation von Komponententest
1.2	14.05.2012	Michael Kern	Überarbeitung und Formatierung

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

DOM	Document Object Model
GUI	Graphical User Interface
MVC	Model View Controller

3.2. Definitionen

Workbench	Als Workbench wird der der gesamte Frame des Tools, mit dessen beinhaltenden Panels bezeichnet.
Multicast	Eine Nachrichtenübertragung von einem Punkt zu einer Gruppe. Nachrichten können gleichzeitig an mehrere Teilnehmer oder eine geschlossene Teilnehmergruppe gesendet werden.

3.3. Terminologien

-

4. Anforderungen

4.1. Benutzersicht

Das Modul XmlParserWorkbench wird beim Starten und Schließen des MultiCastor-Tools verwendet, um den Zustand der Oberfläche zu speichern bzw. wiederherzustellen. Die erfassten Daten werden in einer Xml-Datei abgelegt.
Das Laden und Sichern der zuletzt geöffneten MultiCast-Ströme ist Teil des AutoSave-Moduls und somit nicht Teil des XmlParserWorkbench-Moduls.

4.2. Kontext der Anwendung

Das Modul XmlParserWorkbench wurde vom XmlParser abgeleitet, da dieses Modul Funktionalitäten zur Verfügung stellt, die auch im XmlParserWorkbench ihre Anwendung finden. Durch das Speichern und Wiederherstellen eines Oberflächenzustands, soll die Zeit bis zur Inbetriebnahme des MultiCastor-Tools minimiert werden. Der Benutzer findet das Tool nach dem Start in einem Zustand wieder, in dem er es geschlossen hat.

4.3. Anforderungen

/F140/ (/LF140/): Der MultiCastor wird den Zustand, in dem er beim Schließen ist, nach einem erneuten Start wiederherstellen. [2]

Die Komponente XmlParserWorkbench wurde entwickelt um die Wiederaufnahme der Arbeit mit dem MultiCastor effizient zu gestalten. Die Notwendigkeit des Wiederherstellen der Workbench stellte sich im Verlauf der Entwicklung des MultiCastor heraus.
Nach einer Sprachänderung muss das Tool neu gestartet werden. Damit der Benutzer seine Einstellung nicht verliert, wurde das Modul XmlParserWorkbench implementiert.

5. Analyse

Die Analyse beschäftigt sich mit dem Lokalisieren der benötigten Daten sowie der Art, wie die Daten gespeichert werden sollen. Des Weiterem werden Fragen der Implementierungsmöglichkeiten reflektiert.

5.1. Voruntersuchungen

Die zu erfassenden Daten werden in einer XML-Struktur gespeichert. Für die Verarbeitung von XML-Strukturen stellt Java verschiedene Möglichkeiten zur Verfügung. Für das Modul XmlParserWorkbench wurde der DOM-Parser verwendet. Dies bietet den Vorteil, dass das gesamte XML-Dokument zur Laufzeit im Arbeitsspeicher zur Verfügung steht.

5.2. Systemanalyse

Analyse der Rahmenbedingungen

Der XmlWorkbenchParser ist für das Einlesen und die Validierung der Konfigurationsdatei *workbench.xml* verantwortlich. Ebenfalls sind Default Werte als statische Konstanten in dieser Klasse implementiert. Innerhalb des Moduls werden folgende Attribute gehalten:

Attribut	**Beschreibung**
window-size	Hält die Größe des Hauptframes.
mode	Modus des MultiCastor-Tools (Sender oder Receiver)
active-rb	Enthält Informationen, welcher Radiobutton im Panel MulticastConfig aktiviert werden muss (IGMP, MLD, MMRP).
graph-console	Hält die Information, ob der Graph oder die Konsole sichtbar ist.
locale	Wird verwendet um die Sprache des Tools festzulegen.
position	Hält die Information über die Position des Hauptframes auf dem Bildschirm.
config	Enthält die Dateipfade für die 4 zuletzt geladenen Konfigurationsdateien

Tabelle 1: Attribute der Workbench

Die Aufgabe des WorkbenchHandler ist es, die eingelesenen Werte auf die Workbench zu übertragen bzw. die Werte für das Speichern der Workbench zusammen zutragen und in eine XML-Struktur zu fassen. Die erstellte XML-Struktur nutzt der XmlParserWorkbench, um die Workbencheinstellungen persistent im Dateisystem in der Datei *workbench.xml* abzulegen.

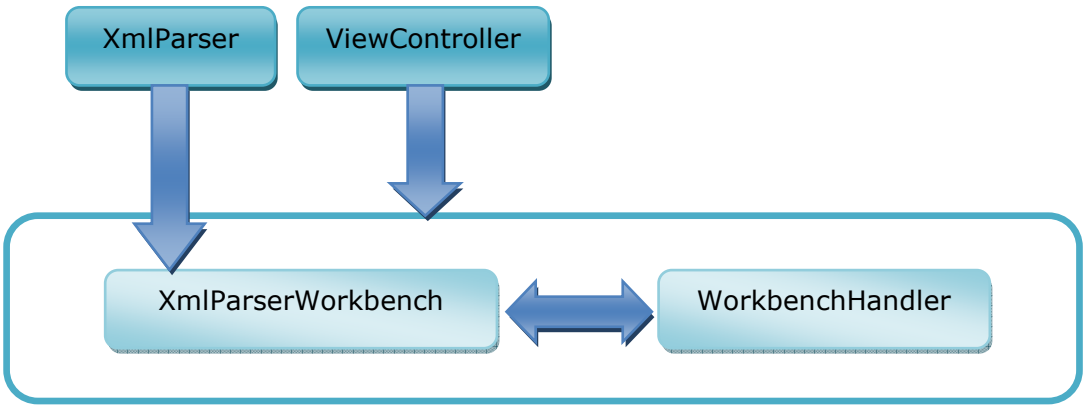
Analyse der Problemstellung

Der XmlWorkbenchParser dient dem Speichern und Laden der unter „Analyse der Rahmenbedingungen" beschriebenen Attribute. Desweitern werden die Attribute über den WorkbenchHandler auf die Workbench übertragen bzw. für das Speichern zusammen getragen.

Architektur und Gliederung der Komponenten

Das Modul XmlParserWorkbench setzt sich aus den Klassen XmlParserWorkbench und WorkbenchHandler zusammen. Die Klasse XmlParserWorkbench erbt von der Klasse XmlParser. Damit stehen die Methoden des XmlParser auch dem XmlWorkbenchParser zur Verfügung. Die Funktionalitäten des Moduls <XmlParser> wird näher in der Quelle [1] beschrieben.

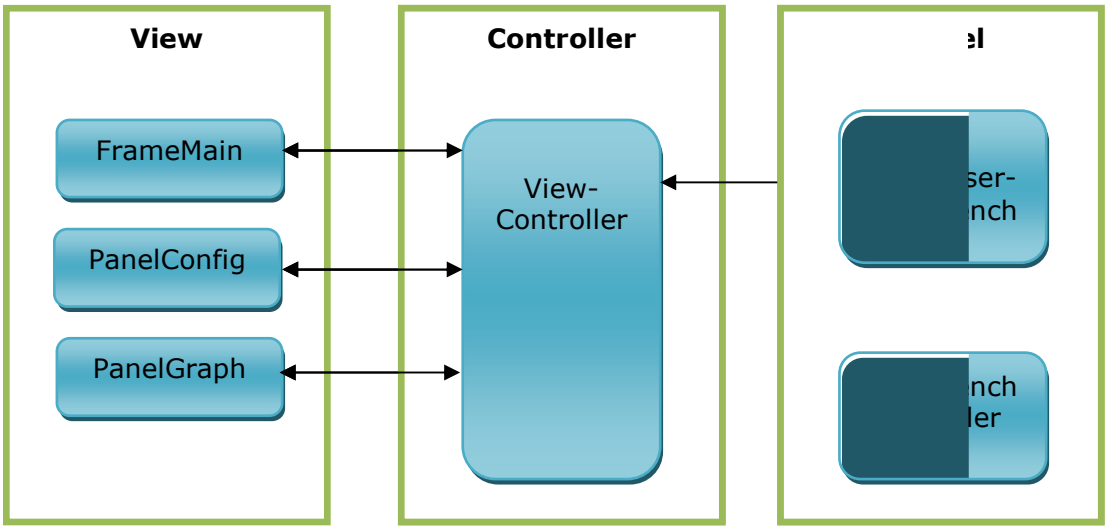
Interaktionsanalyse / Abhängigkeiten



Der XmlParser stellt Funktionalitäten für die Xml-Verarbeitung zur Verfügung und wird deshalb von dem Modul XmlParserWorkbench geerbt.
Das Modul XmlParserWorkbench wird von dem ViewController aufgerufen und übergibt eine Referenz auf sich selbst. Dadurch wird das MVC-Prinzip eingehalten. Diese Referenz wird benötigt um den WorkbenchHandler den Zugriff auf Oberfläche Elemente zu ermöglichen.
Die Validierung der eingelesenen Attribute erfolgt innerhalb des XmlParserWorkbench. Eine Auslagerung der Validierung wurde aufgrund der Trivialität nicht in Betracht gezogen.

6. Design

Die Komponente benutzt die im gesamten Tool verwendete MVC-Struktur:



Die Komponente XmlParserWorkbench greift über den ViewController auf die View-Elemente zu, um die Workbench-Einstellungen zu setzen bzw. zu speichern.

Der Konstruktor der Klasse XmlParserWorkbench erwartet eine Instanz des ViewController sowie einen Logger. Für das Laden und Speichern der Xml-Datei wurden die öffentlichen Methoden *saveWorkbench()* und *loadWorkbench()* implementiert. Die Validierung der Einzelnen Attribute erfolgt über private Methoden.

```
public void XmlParserWorkbench(Logger logger, ViewController ctrl)
public void saveWorkbench(String filepath)
public void loadWorkbench(String filepath)
```

Der WorkbenchHandler wird über die Methoden setWorkbenchAfter() und setWorkbenchBefore() aufgerufen. Die Methode setWorkbenchAfter() wurde von der Methode setWorkbenchBefore() getrennt, da das Setzen der Sprache vor der Instanziierung des FrameMain erfordert. Auch die zuletzt geladenen Konfigurationsdateien müssen davor zur Verfügung stehen.

```
public void setWorkbenchBefore()
public void setWorkbenchAfter()
```

XML-Struktur

Der Aufbau der Xml-Datei wird nach folgenden Schema erfolgen. Ein konkretes Beispiel ist im Kapitel „Implementierung" -> „Aufbau der Datei workbench.xml" zu finden.

```
<xs:simpleType name="workbench">
      <xs:element name="window-size" height="xs:integer"
                                          width="xs:integer"/>
      <xs:element name="mode" type="xs:string"/>
      <xs:element name="active-rb" type="xs:string"/>
      <xs:element name="graph-console" type="xs:string"/>
      <xs:element name="locale" country="xs:string" language="xs:string"/>
      <xs:element name="position" posx=" xs:integer" posy=" xs:integer"/>
      <xs:element name="config">
            <xs:complexType mixed="true">
                  <xs:sequence>
                        <xs:element name="path" type="xs:string">
                        <!—Weitere Elemente -->
                  </xs:sequence>
            </xs:complexType>
      </xs:element>
</xs:simpleType>
```

7. Risiken

Für den Fall das die Konfigurationsdatei nicht wohlgeformt ist bzw. invalide Daten enthält oder nicht vorhanden ist, sind in der Klasse XmlParserWorkbench statische Konstanten definiert, die an Stelle der korrupten Daten gesetzt werden. Die Standardwerte wurden wie folgt definiert:

Attribut	**Wert**	**Beschreibung**
DEF_WIDTH	850 (px)	Breite des Hauptframes
DEF_HEIGHT	694 (px)	Höhe des Hauptframes
DEF_POSX	100	X-Position des Hauptframes
DEF_POSY	100	Y-Position des Hauptframes
DEF_MODE	„sender“	Modus des MultiCastors
DEF_GoC	„graph“	Graph oder Konsole anzeigen
DEF_LANGUAGE	„en“	Sprachauswahl
DEF_COUNTRY	„us“	Landesauswahl
DEF_RB	IGMP	Selektierter RadioButton im PanelMulticastConfig

Tabelle 2: Standardwerte

Sollte ein Wert oder sogar die gesamte Datei invalide sein, wird dies in der Logdatei vermerkt und die konstanten Default-Einstellungen werden zum Laden verwendet.

8. Implementierung

Der XmlParserWorkbench wird in der Klasse ViewController in den Methoden:

- initialize() -> starten des Programmes
- restart() -> neustarten des Programmes
- closeProgram() -> beenden des Programmes

aufgerufen.

Laden

Die Methode zum Laden der Wokbench ist loadWorkbench(). Sie wird in der Methode initialize() in der Klasse ViewController aufgerufen.

```
public void initialize(MulticastController p_mc) {
      XmlParserWorkbench workbench = new XmlParserWorkbench(logger, this);
      workbench.loadWorkbench(FileNames.WORKBENCH);
      workbench.setWorkbenchBefore();
      messages = ResourceBundle
                  .getBundle("zisko.multicastor.resources.i18n.messages");
      mcController = p_mc;
      prefs = Preferences.userRoot().node("multicastor/viewController");
      if (f == null) {
            f = new FrameMain(this);
            workbench.setWorkbenchAfter();
      }
}
```

Hier wird zunächst eine neue Instanz von XmlParserWorkbench erstellt. Anschließend wird über den Aufruf der Methode loadWorkbench() die XML-Datei eingelesen. Danach wird mit der Methode setWorkbenchBefore() alle Attribute gesetzt, die vor der Instanziierung des FrameMain gesetzt werden müssen (z.B. Spracheinstellungen). Nachdem der FrameMain instanziiert ist, werden mit der Methode setWorkbenchAfter() alle übrigen Attribute auf den FameMain übertragen.

Die Methode loadWorkbench() ist wie folgt aufgebaut:

```
public void loadWorkbench(String filepath){
      try {
            doc = parseDocument(filepath);
            NodeList workbench = doc.getElementsByTagName("workbench");

            // Validierung der XML Nodes
            if(_validateNodeLength(workbench, "workbench")){
                  _validateWindowSize(doc.getElementsByTagName("window-size"));
                  …
            }
      } catch …
}
```

Nachdem die Xml-Struktur aus dem übergebenen Dateipfad geparst wurde, werden verschiedenen Validierungsmethoden für die einzelnen Nodes ausgeführt. Nach der Validierung stehen die Attribute als Klassenvariablen zur Verfügung. Diese werden in den Methoden setWorkbenchBefore() und setWorkbenchAfter() an den WorkbenchHandler übergeben, der diese wiederum auf den FrameMain überträgt.

Dem WorkbenchHandler wird bei dem Aufruf des Konstruktors eine Referenz des ViewController übergeben. Diese ist notwendig um die eingelesenen Daten auf die grafische Oberfläche zu übertragen. Der WorkbenchHandler besitzt folgende Methoden um die Workbench zu steuern:

Methode	**Beschreibung**
setGraphOrConsole(String gOc)	Graphtab oder Konsolentab im GraphPanel anzeigen.
setLocale(String language, String country)	Setzen der Spracheinstellung.
setPosition(int posx, int posy)	Setzen der FrameMain Position auf dem Bildschirm.
setPaths(NodeList paths)	Zuletzt geladene Dateipfade setzen.
setMode(String mode)	Setzen des Modus von dem MultiCastor-Tool.
setRB(ProtocolType type)	Setzen des aktiven RadioButtons.
setWindowSize(int width, int height)	Setzen der FrameMain Größe.

Speichern

Die Methode zum Speichern der Workbench ist die Methode saveWorkbench(). Sie wird in den Methoden restart() und closeProgram() der Klasse ViewController aufgerufen. Die Methode saveWorkbench() bindet die einzelnen Workbench-Attribute in eine Xml-Struktur. Diese wird mit Hilfe von Methoden aus dem Modul XmlParser erstellt.

```
public void saveWorkbench(String filepath){
      File xmlFile = new File(filepath);
      DocumentBuilder builder;
      DocumentBuilderFactory factory;
      DOMImplementation impl;
      Document doc;

      try {
            …

            // Document erstellen
            doc = handler.buildXmlDocument(doc, xmlFile);

            // XmlParser: Transformer erstellen
            Transformer transformer = setupTransformer();

            // XmlParser: Erstelle einen String aus dem XML Baum
            String xmlString = XMLtoString(doc, transformer);

            …

      } catch …
}
```

Die Methode buildXmlDocument() der Klasse WorkbenchHandler erstellt ein Document Objekt, dass eine Xml-Struktur repräsentiert. Der Transformer wird über die Methode setupTranformer() des XmlParser erstellt. Er wird benötigt um die Xml-Struktur des Document Objekts in einen einfachen String zu konvertieren. Dies geschieht in der Methode XMLtoString().

Validierung

Die Validierung der eingelesenen Daten ist Aufgabe des XmlParserWorkbench. Hierzu stehen ihn die folgenden Methoden zur Verfügung:

Methode	Beschreibung
_validateActiveRB()	Validierung des zu setzenden RadioButton
_validateGraphOrConsole()	Validierung des GraphPanels
_validateLocale()	Validierung der Spracheinstellungen
_validateMode()	Validierung des Modus
_validateNodeLength	Prüft auf: NodeLength == 1
_validatePaths()	Validierung der zuletzt geladenen Konfigurationdateien
_validatePosition()	Validierung der Frame-Position
_validateWindowSize()	Validierung der Frame-Größe

Die Validierung erfolgt für jedes Node einzeln. An die Methoden werden die jeweiligen Werte übergeben. Sollte der übergebene Wert valide sein, wird die Klassenvariable des entsprechenden Wertes mit dem übergebenen Wert überschrieben. Ist der Wert invalide wird er nicht überschrieben und er zuvor zugewiesene Default-Wert ist gültig.

Aufbau der Datei workbench.xml

Eine gültige Workbenchkonfiguration könnte folgender Maßen aussehen:

```
<?xml version="1.0"?>
<workbench>
    <window-size height="694" width="850"/>
    <mode>sender</mode>
    <active-rb>MLD</active-rb>
    <graph-console>console</graph-console>
    <locale country="US" language="en"/>
    <position posx="313" posy="108"/>
    <config>
        <path>C:\Documents and Settings\kern\Desktop\beispiel2.xml</path>
        <path>C:\Documents and Settings\kern\Desktop\beispiel1.xml</path>
    </config>
</workbench>
```

9. Komponententest

Bei den Komponententests handelt es sich um Unit Tests, die mit Hilfe von JUnit ausgeführt wurden.

9.1. Komponententestplan

TestNo	Feature ID	Test Specification (Description or TCS)
001	testLoadWorkbenchDefault	Testen des korrekten Ladens von Default-Werten.
002	testLoadWorkbench	Testen des korrekten Laden von validen Werten.
003	testLoadWorkbenchWithInvalideSchema	Testen des Laden bei invaliden Xml-Schema

Tabelle 3: Komponententestplan

9.2. Testdaten

Die Testdaten beziehen sich auf den Inhalt der Datei workbench.xml.

XmlParserWorkbench – 001

Alle Daten in der Testdatei wurden bewusst invalide gewählt. Die Erwarteten Werte stammen aus der Default-Konfiguration.

Variable	Wert	Erwarteter Wert	Pass/Fail
Country	US	US	Pass
Language	En	En	Pass
Mode	Sender	Sender	Pass
RadioButton	IGMP	IGMP	Pass
Graph or Console	Graph	Graph	Pass
Pos X	100	100	Pass
Pos Y	100	100	Pass
Height	694	694	Pass
Width	850	850	Pass

Tabelle 4: Test XmlParserWorkbench 001

XmlParserWorkbench – 002

Alle Daten in der Testdatei wurden valide gewählt.

Variable	Wert	Erwarteter Wert	Pass/Fail
Country	DE	DE	Pass
Language	De	De	Pass
Mode	Receiver	Receiver	Pass
RadioButton	MLD	MLD	Pass
Graph or Console	Console	Console	Pass
Pos X	20	20	Pass
Pos Y	30	30	Pass
Height	1000	1000	Pass
Width	1200	1200	Pass

Tabelle 5: Test XmlParserWorkbench 002

XmlParserWorkbench – 003

Alle Daten in der Testdatei wurden bewusst invalide gewählt. Die Erwarteten Werte stammen aus der Default-Konfiguration.

Variable	Wert	Erwarteter Wert	Pass/Fail
Country	US	US	Pass
Language	En	En	Pass
Mode	Sender	Sender	Pass
RadioButton	IGMP	IGMP	Pass
Graph or Console	Graph	Graph	Pass
Pos X	100	100	Pass
Pos Y	100	100	Pass
Height	694	694	Pass
Width	850	850	Pass

Tabelle 6: Test XmlParserWorkbench 003

9.3. Komponententestreport

TestNo	Pass/Fail	If failed: Test Result	Date	Tester
001	Pass		13.05.2012	Kern
002	Pass		13.05.2012	Kern
003	Pass		13.05.2012	Kern

Tabelle 6: Komponententestreport

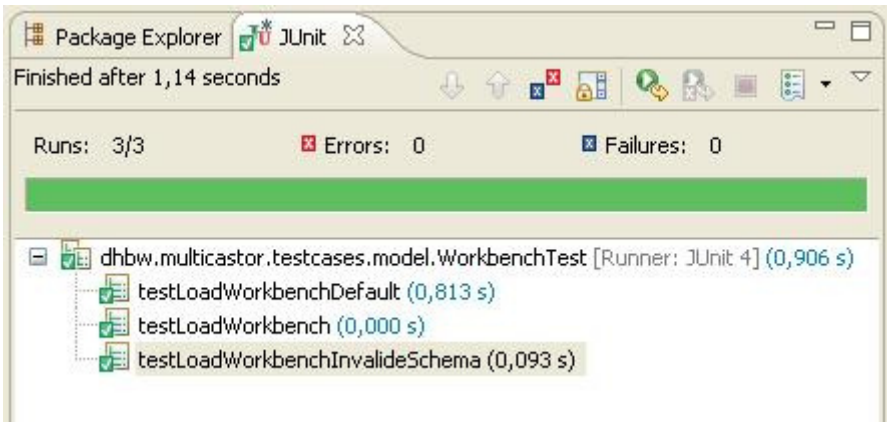


Abbildung 1: Komponententestreport

Zusammenfassung

9.4. Beurteilung der Komponente

Die Methoden der Komponente XmlParserWorkbench sind speziell für die Anforderungen an den MultiCastor angepasst und damit nur schwer auf andere Komponenten übertragbar. Damit das Programm in einem stabilen Zustand läuft, wurden die Default-Werte fest im Programmcode implementiert. In einer weiteren Optimierungsschritt könnten die Default-Werte in eine extra Datei ausgelagert werden. Dazu müssten zusätzliche Validierungsmethoden implementiert werden, die einen stabilen Zustand garantieren.
Da die Komponente einen schnellen Wiedereinstieg in die Arbeit mit dem MultiCastor-Tool ermöglichen soll, könnte das abspeichern weiterer Oberflächenelemente zu einer weiteren Verbesserung führen. Beispielsweise könnte die Reihenfolge der Tabellenspalten mit gespeichert werden.
Die Komponente ist fehlertolerant. Dies bezieht sich auf die Nutzung von Default-Werten. Bei invaliden Daten wird lediglich eine Warnung geloggt. Das Tool startet unter der Verwendung eines Default-Wertes und kann ohne Einschränkung verwendet werden.

9.5. Ausblick für die Weiterentwicklung

Es sind keine konkreten Weiterentwicklungen geplant.
Aufgrund der übersichtlichen Struktur der Komponente sind Weiterentwicklungen und Anpassungen einfach umsetzbar.

10. Anhang

10.1. Literatur

[1] "TIT10AID_MOD_XMLParser_Team_3_0v1.docx"
[2] "TIT10AID_SRS_Multicastor_Team_3_1v5.docx"
[3] "TIT10AID_STP_Multicastor_Team_3_0v1-2.doc"

10.2. Standards

-

10.3. Diagramme

-

10.4. Schnittstellendefinitionen

-

10.5. Code

-

10.6. Testfälle

siehe [3] und Kapitel Komponententest