SMILE - Datenstrukturen
Spezifikation

Holger Siegel / Rachid El Araari

22. November 2006

Inhaltsverzeichnis

Teil I

Übersicht

Kapitel 1

Pakete

In diesem Abschnitt werden die Packages kurz vorgestellt. Für eine detaillierte Beschreibung des Interfaces wird auf Abschnitt II auf Seite 11 verwiesen.

1.1 csm.exceptions

In diesem Paket sind alle Exceptions definiert, die bei der Bearbeitung eines CoreStateMachine auftreten können. Ausgenommen sind die RuntimeExceptions, die beim Laden und Speichern auftreten können, sowie NullPointerExceptions, die immer auf Programmierfehler hindeuten.

Diese Exceptions werden nur von den Methoden der Klassen unterhalb von csm erzeugt.

Ein Wort zur Behandlung von Nullzeigern

Wir, die Datenstruktur-Gruppe, versuchen, die Parameter aller öffentlichen Methoden mit Assertions auf Gültigkeit zu testen.
Es ist unser Fehler, wenn bei eingeschalteten Assertions ein unerlaubter Nullzeiger durchgelassen wird. Es ist euer Fehler, wenn eure Aufrufe unsere Assertions verletzen.
Im Gegenzug garantieren wir, dass die öffentlichen Methoden nur dann Nullzeiger liefern, wenn in der Javadoc-Dokumentation darauf hingewiesen wird.

1.2 semantics

In diesem Paket implementiert die Semantik-Gruppe ihre Funktionalität. Nach außen sichtbar soll nur die Klasse NuGenerator und deren Methode generateNuAutomaton(..) sein.

1.3 nua

In diesem Paket befindet sich die Implementation des ν-Automaten. Dieser wird von der Semantik-Gruppe erzeugt und von der Model-Checking-Gruppe gelesen. Die GUI-Gruppe darf Objekte vom Typ NuAutomaton zwischenspeichern, ohne sie zu auszulesen oder zu verändern.

1.4 csm.statetree

Dieses Paket implementiert den Komponentenbaum, also alle Bestandteile der CoreStateMachine, die auf der Zeichenfläche gezeichnet werden. Die Gruppe GUI darf alle Konstruktoren der nicht-abstrakten Klassen verwenden, um neue Komponenten zu erzeugen. Sie darf alle öffentlichen Methoden verwenden, um den Komponentenbaum darzustellen und zu bearbeiten. Die Semantik-Gruppe soll nur lesend auf den Komponentenbaum zugreifen. Außerhalb des Packages sollen keine Subklassen der hier definierten Klassen definiert werden.

Abbildung 1.1 auf der nächsten Seite zeigt die Klassenstruktur des Komponentenbaums. Abbildung 1.2 auf Seite 6 zeigt die Abhängigkeiten innerhalb des Komponentenbaums.

Das Erzeugen und Löschen von Komponenten

- Alle Komponenten besitzen einen Konstruktor, der als einziges Argument eine Position vom Typ Point entgegennimmt. Diese Position gibt die Position der Komponente innerhalb der sie umgebenden Komponente an.
- mit *p.add(c)* fügt man einer Parent-Komponente p eine Unterkomponente c hinzu. Ist c bereits Unterkomponente irgendeiner Komponente, oder würde c durch das Hinzufügen eine Unterkomponente von sich selbst, dann wird eine Exception *ErrTreeNotChanged* geworfen.
- mit *p.remove(c)* entfernt man die Unterkomponente c wieder aus p. War c keine Unterkomponente von p, dann wird ebenfalls eine Exception *ErrTreeNotChanged* geworfen.

Visitor-Pattern Die Child-Komponenten einer Komponente sind nicht öffentlich zugänglich. Die Methode

```
protected final void visitChildren(CSMComponent component)
```

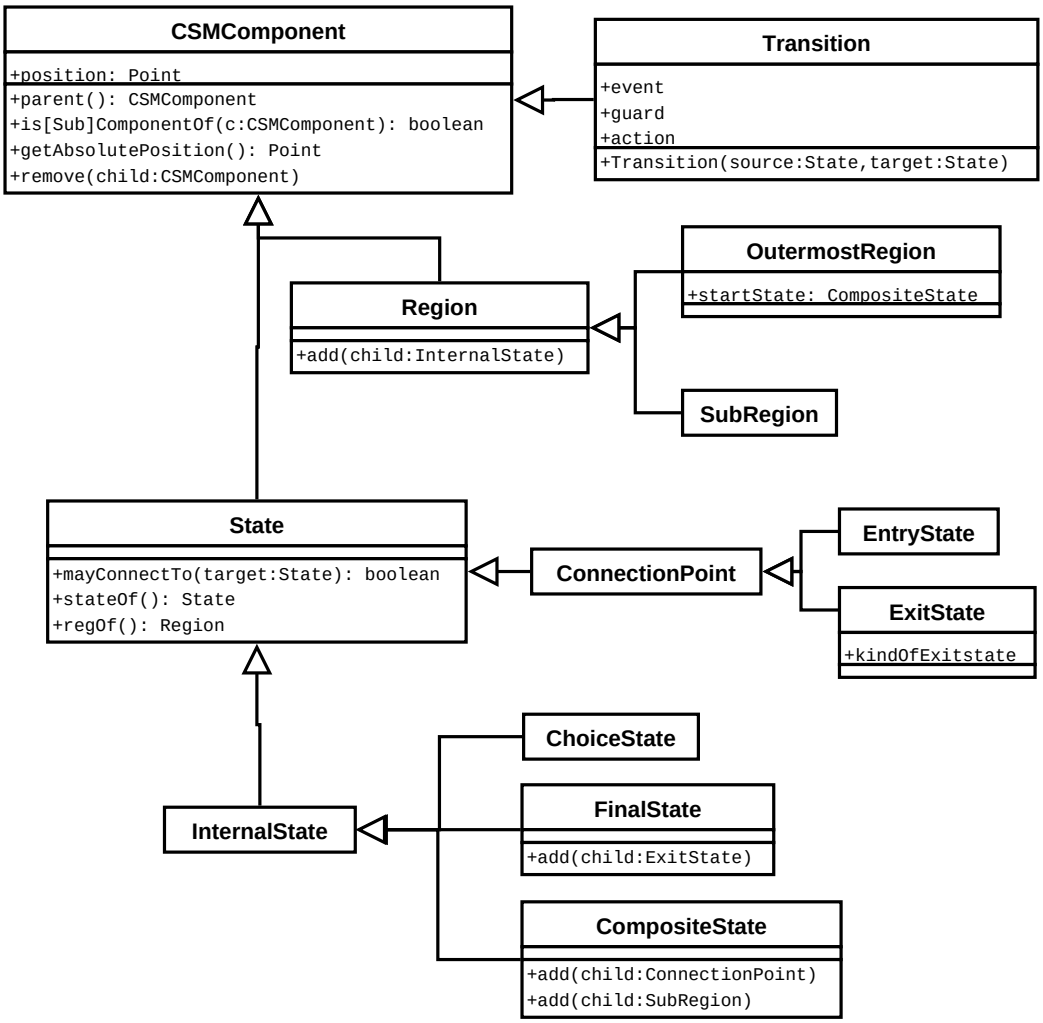

Abbildung 1.1: Die Klassenstruktur des Komponentenbaums

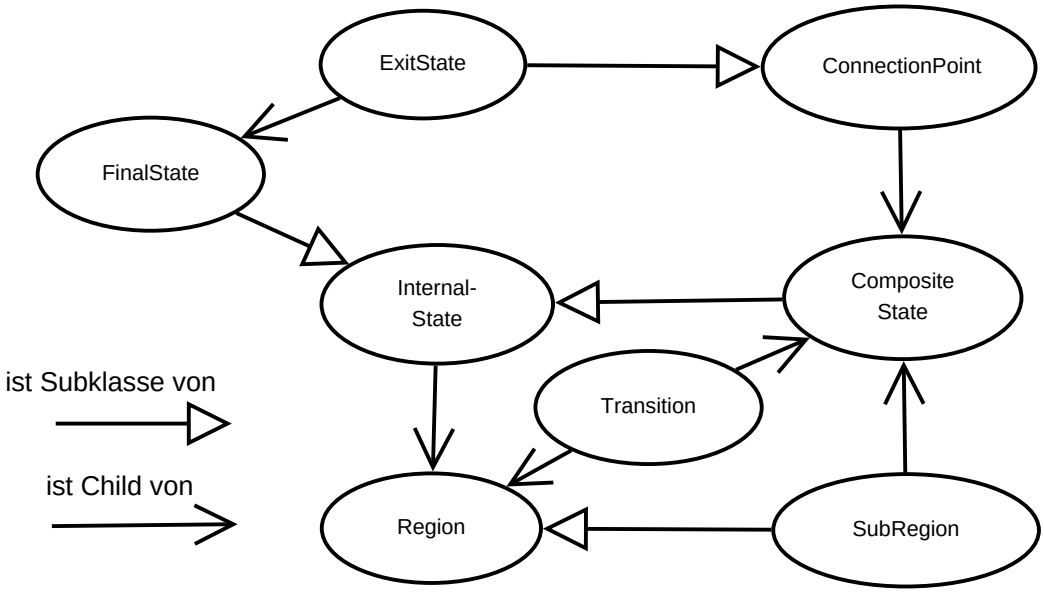

Abbildung 1.2: Übersicht über die Abhängigkeiten zwischen den Bestandteilen des Komponentenbaums

der Klasse csm.statetree.Visitor ermöglicht es, auf die Unterkomponenten einer Komponente zuzugreifen. Sie ruft für jede Unterkomponente von component die entsprechende visit-Methode des Visitor-Objektes auf. Durch Überschreiben dieser visit-Methoden kann man Subklassen von Visitor implementieren, die nahezu beliebige Operationen auf dem Komponentenbaum ausführen.

1.5 csm

Dieses Paket implementiert die CoreStateMachine sowie Listen für Events und Variablen. Die Gruppe GUI darf die Konstruktoren von CoreStateMachine, Event und Variable verwenden. Sie darf alle öffentlichen Methoden verwenden, um die CoreStateMachine darzustellen und zu bearbeiten. Die Semantik-Gruppe soll nur lesend auf die CoreStateMachine zugreifen. Außerhalb des Packages sollen keine Subklassen der hier definierten Klassen definiert werden.

1.6 csm.expression

In diesem Paket sind die abstrakten Syntaxbäume für Guards und Actions implementiert. Es enthält ein Unterpaket csm.expression.parser, in dem mit JavaCC ein Parser für Guards und Actions implementiert wird. Alle Syntaxbäume stammen von der Klasse

```
Expression<Ergebnistyp>
```

ab. Diese stellt eine Methode prettyprint() zur Verfügung, die den Ausdruck in einen String umwandelt. Es wird garantiert, dass beim Parsen eines mit prettyprint() ausgegebenen Syntaxbaums ein zu diesem semantisch äquivalenter Syntaxbaum entsteht.

Action-Terme bestehen aus Unterklassen der Klasse

```
Action extends Expression<ExpressionEnvironment>
```

Die Unterklassen von Expression und Action sowie die zugrundeliegende Syntax sind vollständig im Paket csm.expression gekapselt. Außerhalb dieses Paketes sollen daher keine Annahmen über die Syntax oder die interne Klassenstruktur gemacht werden.

Jede Expression<Result> hat eine Methode

```
public Result evaluate(ExpressionEnvironment env)
```

Diese Methode wertet die Expression in dem Environment env aus.

Kapitel 2

Verwendung

2.1 Gruppe GUI

Die Gruppe GUI verwendet die Klassen in den Paketen unterhalb von csm, um Objekte vom Typ CoreStateMachine zu erstellen und zu bearbeiten.

Zum Erstellen stellt die Klasse CoreStateMachine einen Konstruktor sowie eine statische Methode loadCSM(..) zur Verfügung.

Um GUI-spezifische Daten in der CoreStateMachine zu speichern, stehen dort die Methoden setGuiMetadata und getGuiMetadata zur Verfügung.

Verwendung der Datenstrukturen

csm.exceptions die Exceptions in diesem Paket werden ausschließlich von den Methoden des Datenmodells erzeugt. Sie werden geworfen, wenn eine Änderung des Modells schiefgegangen ist. Es ist Aufgabe der GUI, auf diese Fehler zu reagieren.

semantics die statische Methode generateNuAutomaton(..) der Klasse NuGenerator soll von der GUI verwendet werden, um aus einer CoreStateMachine einen ν-Automaten zu erstellen. Es ist darauf zu achten, dass die CoreStateMachine während der Erstellung nicht verändert wird. Andernfalls ist eine korrekte Umsetzung nicht gewährleistet.

nua die Klasse NuAutomaton repräsentiert einen ν-Automaten; mit den anderen Klassen dieses Pakets hat die GUI-Gruppe nichts zu tun. Die GUI soll keine Methoden von NuAutomaton verwenden.

csm.statetree alle öffentlichen Klassen, Methoden und Konstruktoren in diesem Paket dürfen verwendet werden, um den Komponentenbaum

darzustellen und zu verändern. Von den Klassen in diesem Paket sollen keine Unterklassen abgeleitet werden. Um Informationen über den Komponentenbaum zu erhalten, sollen Subklassen von der Klasse csm.statetree.Visitor abgeleitet werden.

csm Alle Konstruktoren von CoreStateMachine, Event, Variable, GuiMetadata dürfen verwendet werden. Darüber hinaus dürfen alle öffentlichen Methoden dieser Klassen verwendet werden, um die CoreStateMachine darzustellen und zu verändern. Von den Klassen in diesem Paket sollen keine Unterklassen abgeleitet werden.

csm.expression Ausdrücke, also Actions und Guards, werden intern in den Setter-Methoden von CompositeState und Transition geparst. Deshalb soll die GUI von der Klasse Expression und ihren Subklassen nur die Methode prettyprint(..) verwenden. Darüber hinaus sollen keine Annahmen über die Subtypen von Expression<Integer>, Expression<Boolean> und Action getroffen werden.

2.2 Gruppe Semantik

Die Gruppe *Semantik* darf lesend auf alle Methoden aller Klassen unterhalb von csm zugreifen. Daher muß die GUI sicherstellen, dass während der Generierung des ν-Automaten nicht schreibend auf die CoreStateMachine zugegriffen wird.

Zum Erstellen des ν-Automaten dürfen alle Methoden aller Klassen unterhalb von nua verwendet werden.

Verwendung der Datenstrukturen

csm.exceptions weil die CSM nicht verändert wird, treten bei der semantischen Analyse keine dieser Exceptions auf.

semantics in diesem Paket muss die statische Methode NuGenerator.generateNuAutomaton(..) vorhanden sein. Alles andere steht der Gruppe frei.

nua alle Methoden aller Klassen dürfen verwendet werden, um einen ν-Automaten zu erstellen.

csm.statetree Alle lesenden Zugriffe sind erlaubt. Methoden, die den Zustand des Komponentenbaums ändern können, dürfen nicht aufgerufen werden. Um Informationen über den Komponentenbaum zu erhalten,

sollen Subklassen von der Klasse csm.statetree.Visitor abgeleitet werden.

csm Alle lesenden Zugriffe sind erlaubt. Methoden, die den Zustand der CoreStateMachine ändern können, dürfen nicht aufgerufen werden.

csm.expression für alle Objekte, die von Expression<Integer>, Expression<Boolean> und Action abstammen, soll die Methode evaluate(..) verwendet werden, um Ausdrücke auszuwerten. Darüber hinaus sollen keine Annahmen über die Subtypen von Expression<Integer>, Expression<Boolean> und Action getroffen werden. Die Methode prettyprint() soll nur dazu verwendet werden, den Benutzer im Fehlerfall zu informieren.

2.3 Gruppe Model Checking

Die Gruppe *Model Checking* verwendet ausschließlich die Klassen im Paket nua. Sie soll nur lesend auf die darin definierten Klassen zugreifen.

Teil II

API-Spezifikation

Contents

Chapter 1

Package csm.exceptions

Package Contents *Page*

Classes

1.1 Classes

1.1.1 Class **CSMEditException**

Oberklasse der Exceptions, die anzeigen, dass das Bearbeiten einer CoreStateMachine gescheitert ist.

Declaration

public abstract class CSMEditException
extends java.lang.Exception

Constructors

- *CSMEditException*
  `public` **CSMEditException**`( java.lang.String` **message** `)`
- *CSMEditException*
  `public` **CSMEditException**`( java.lang.String` **message**`, java.lang.Throwable` **cause** `)`

1.1.2 Class **ErrAlreadyDefinedElement**

Zeigt an, dass versucht wurde, einem Dictionary einen Wert hinzuzufuegen, der dort bereits vorhanden ist.

Declaration

public final class ErrAlreadyDefinedElement
extends csm.exceptions.CSMEditException

Serializable Fields

- public final String name
  - der Name, der bereits im Dictionary enthalten ist

Fields

- public final String name
  - der Name, der bereits im Dictionary enthalten ist

CONSTRUCTORS

- *ErrAlreadyDefinedElement*
  `public` **ErrAlreadyDefinedElement**`( java.lang.String` **name** `)`
  - **Usage**
    - Erzeugt eine Exception mit dem Text "$name is already defined", wobei fuer $name der uebergebene Name eingesetzt wird
  - **Parameters**
    - `name` - der Name, der bereits im Dictionary enthalten ist

1.1.3 CLASS **ErrMayNotConnect**

zeigt an, dass der Versuch, zwei States mit einer Transition zu verbinden, gescheitert ist

DECLARATION

public class ErrMayNotConnect
extends csm.exceptions.CSMEditException

CONSTRUCTORS

- *ErrMayNotConnect*
  `public` **ErrMayNotConnect**`( )`
  - **Usage**
    - Zeigt an, dass versucht wurde, eine unerlaubte Transition zwischen zwei States zu erzeugen. Der Text dieser Exception ist lediglich "illegal transition".

1.1.4 CLASS **ErrStillConnected**

DECLARATION

public class ErrStillConnected
extends csm.exceptions.CSMEditException

SERIALIZABLE FIELDS

- public final Transition transition
  –

FIELDS

- public final Transition transition
  –

CONSTRUCTORS

- *ErrStillConnected*
  `public` **ErrStillConnected**`( csm.statetree.Transition  t )`
  - **Parameters**
    * `message` -

1.1.5 CLASS ErrSyntaxError

Zeigt an, dass beim Parsen eines Ausdrucks ein Syntaxfehler aufgetreten ist

DECLARATION

public final class ErrSyntaxError
extends csm.exceptions.CSMEditException

CONSTRUCTORS

- *ErrSyntaxError*
  `public` **ErrSyntaxError**`( java.lang.String  msg )`
  - **Parameters**
    * `msg` - die Fehlermeldung des Parsers

1.1.6 CLASS ErrTreeNotChanged

Zeigt an, dass eine aenderung des Komponentenbaums gescheitert ist.

DECLARATION

public class ErrTreeNotChanged
extends csm.exceptions.CSMEditException

Constructors

- *ErrTreeNotChanged*
  `public` **ErrTreeNotChanged**`( java.lang.String` **message** `)`
  - **Parameters**
    - `message` - ein kurzer Text, der anzeigt, warum die Aenderung gescheitert ist

1.1.7 Class ErrUndefinedElement

Zeigt an, dass versucht wurde, aus einem Dictionary einen Wert zu lesen, der darin nicht vorhanden ist.

Declaration

public final class ErrUndefinedElement
extends csm.exceptions.CSMEditException

Serializable Fields

- public final String name
  - der Name des Elementes, das im Dictionary nicht gefunden wurde

Fields

- public final String name
  - der Name des Elementes, das im Dictionary nicht gefunden wurde

Constructors

- *ErrUndefinedElement*
  `public` **ErrUndefinedElement**`( java.lang.String` **name** `)`
  - **Usage**
    - Erzeugt eine Exception mit dem Text "$name is not defined", wobei fuer $name der uebergebene Name eingesetzt wird.
  - **Parameters**
    - `name` - der Name, der nicht gefunden wurde

1.1.8 Class ErrValueOutOfBounds

Zeigt an, dass versucht wurde, einen Zahlenwert zu setzen, der ausserhalb der erlaubten Grenzen liegt. Wird zum Beispiel von der Klasse Variable verwendet, damit Minimal-, Maximal- und Initialwert konsistent sind.

Declaration

public final class ErrValueOutOfBounds
extends csm.exceptions.CSMEditException

Constructors

- *ErrValueOutOfBounds*
  `public` **ErrValueOutOfBounds**`( java.lang.String` **message** `)`
  - **Parameters**
    - `message` - ein kurzer Text, aus dem hervorgeht, welche Grenze verletzt wurde

Chapter 2

Package semantics

Package Contents *Page*

Classes

2.1 Classes

2.1.1 Class **NuGenerator**

Declaration

public class NuGenerator **extends** java.lang.Object

Constructors

- *NuGenerator*
  `public` **NuGenerator**`( )`

Methods

- *generateNuAutomaton*
  `public static NuAutomaton` **generateNuAutomaton**`( csm.CoreStateMachine` **csm** `)`
  - **Usage**
    - erzeugt einen “nu-Automaten aus einer gegebenen CSM

Chapter 3

Package nua

Package Contents *Page*

Classes

3.1 Classes

3.1.1 Class **NuAutomaton**

Der “nu-Automat gemaess Definition 1 der Aufgabenbeschreibung

Declaration

public class NuAutomaton **extends** java.lang.Object

Fields

- public final Set states

  –

- public final Set rootStates

  –

- public final Set transitions

  –

Constructors

- *NuAutomaton*
  `public` **NuAutomaton**`( )`

Methods

- *isConcrete*
  `public boolean` **isConcrete**`( )`

3.1.2 Class **NuState**

Ein State eines “nu-Automaten

Declaration

public class NuState **extends** java.lang.Object

FIELDS

- public final NuAutomaton nua
  –

CONSTRUCTORS

- *NuState*
  `public` **NuState(** `nua.NuAutomaton` **nua)**
  - **Usage**
    * Erzeugt einen neuen State und traegt ihn im “nu-Automaten ein.

- *NuState*
  `public` **NuState(** `nua.NuAutomaton` **nua,** `boolean` **isRootState)**
  - **Usage**
    * Erzeugt einen neuen State und traegt ihn im “nu-Automaten ein. Ist das Argument isRootState true, dann wird er auch in der Liste der Rootstates des “nu-Automaten eingetragen.
  - **Parameters**
    * `nua` - der “nu-Automat, von dem dieser State ein Teil ist
    * `isRootState` - gibt an, ob der State als Root-State im “nu-Automaten eingetragen werden soll.

3.1.3 CLASS **NuTransition**

eine Transition des “nu-Automaten

DECLARATION

public class NuTransition **extends** java.lang.Object

FIELDS

- public final NuAutomaton nua
  –
- public final NuState source
  –
- public final Event action
  –
- public final Set targets
  –

Constructors

- *NuTransition*
  `public` **NuTransition**`( nua.NuAutomaton` **nua**`, nua.NuState` **source**`, csm.Event` **action**`, java.util.Set` **targets** `)`
  - **Usage**
    * erzeugt eine neue Transition und traegt sie im “nu-Automaten ein

Chapter 4

Package csm.statetree

Package Contents *Page*

Classes

4.1 Classes

4.1.1 Class **ChoiceState**

im Paper als S_choice bekannt

Declaration

public final class ChoiceState
extends csm.statetree.InternalState

Constructors

- *ChoiceState*
  `public` **ChoiceState**`( java.awt.Point` **position** `)`

4.1.2 Class **CompositeState**

im Paper als S_com bekannt

Declaration

public final class CompositeState
extends csm.statetree.InternalState

Constructors

- *CompositeState*
  `public` **CompositeState**`( java.awt.Point` **position** `)`

Methods

- *add*
  `public final void` **add**`( csm.statetree.ConnectionPoint` **child** `)`
  - **Usage**
    - Fuegt den Child-States dieser Komponente einen ConnectionPoint hinzu
  - **Parameters**
    - `child` - der hinzuzufuegende State
  - **Exceptions**
    - `csm.exceptions.ErrTreeNotChanged` - wenn der ConnectionPoint schon das Child irgendeiner Komponente ist

- *add*
  `public final void` **add**`( csm.statetree.SubRegion` **child** `)`
  - **Usage**
    - Fuegt den SubRegions dieser Komponente eine SubRegion hinzu. Die Stelle, an der sie zwischen den anderen SubRegions eingefuegt wird, ist dabei nicht festgelegt. Es ist Sache der einfuegenden Methode, die Positionen der SubRegions zu setzen.
  - **Parameters**
    - `child` - die hinzuzufuegende SubRegion
  - **Exceptions**
    - `csm.exceptions.ErrTreeNotChanged` - wenn 1. die Region schon das Child irgendeiner Komponente ist, oder wenn 2. versucht wurde, eine Komponente zu ihrer eigenen Unterkomponente zu machen

---

- *getDeferredEvents*
  `public final LinkedList` **getDeferredEvents**`( )`
  - **Usage**
    - gibt die in diesem State als deferred markierten Events zurueck
  - **Returns** - eine neue Kopie der Eventliste

- *getDoAction*
  `public final Action` **getDoAction**`( )`
  - **Usage**
    - Jeder CompositeState enthaelt doAction, das ein Objekt vom Typ csm.action.Action enthaelt
  - **Returns** - die doAktion oder null, wenn der Zustand keine Do-Action hat

---

- *setDeferredEvents*
  `public final void` **setDeferredEvents**`( java.util.LinkedList` **events** `)`
  - **Usage**
    - setzt die Liste der in diesem State als deferred markierten Events
  - **Parameters**
    - `events` - eine Liste von Events
  - **Exceptions**
    - `csm.exceptions.ErrUndefinedElement` - wenn auf der Liste ein Event ist, der noch nicht in der CSM definiert ist

---

- *setDoAction*
  `public final void` **setDoAction**`( csm.expression.Action` **action** `)`
  - **Parameters**
    - `action` - die dem State zugeordnete DoAction oder null, wenn der State keine DoAction enthaelt.
  - **Exceptions**
    - `csm.exceptions.ErrUndefinedElement` - wenn die Action auf Variablen verweist, die in der zugeordneten CSM nicht definiert sind

---

- *setDoAction*
  `public final void` **setDoAction**`( java.lang.String` **action** `)`
  - **Parameters**
    - `action` - die dem State zugeordnete DoAction oder null, wenn der State keine DoAction enthaelt.
  - **Exceptions**
    - `csm.exceptions.ErrUndefinedElement` - wenn die Action auf Variablen verweist, die in der zugeordneten CSM nicht definiert sind

4.1.3 Class **ConnectionPoint**

Abstrakte Oberklasse der Entry- und ExitStates

Declaration

public abstract class ConnectionPoint
extends csm.statetree.State

Methods

- *stateOf*
  `public final InternalState` **stateOf**`( )`

4.1.4 Class **CSMComponent**

CSMComponent ist die Oberklasse der States und Regions. Sie besitzt die im Paper unter Definition 1 angegebenen Funktionen parent und '>' (hier: isComponentOf), durch die die Baumstruktur der CSM definiert ist.

Als Gegenstueck zum Parent-Attribut besitzt jede Komponente eine Liste ihrer Child-Komponenten – also derjenigen Komponenten, deren Parent-Attribut auf diese Komponente verweist. Welche Typen von Komponenten als Children einer Komponente eingetragen werden duerfen, bestimmen die add-Methoden dieser Komponente. Beim Eintragen einer Komponente muss man also deren konkreten Typ angeben.

Die entsprechende remove-Methode testet, ob die zu entfernende Komponente tatsaechlich eine Child-Komponente ist.

Daneben besitzt jede CSM-Komponente eine zweidimensionale Koordinate position, die die Position der Komponente relativ zu ihrer parent-Komponente angibt, sowie eine Funktion getAbsolutePosition, die die Position relativ zur root-Komponente angibt. Wie diese Position zu interpretieren ist, ist Sache der GUI.

(Anmerkung: Damit entscheiden wir, dass die Position der Komponenten ein Teil des Modells ist. Das erspart uns erstens, die Objekte automatisch plazieren zu muessen, und zweitens, die komplette Komponenten-Hierarchie in der View spiegeln zu muessen)

Declaration

public abstract class CSMComponent
extends java.util.Observable

Methods

- *getAbsolutePosition*
  `public final Point` **getAbsolutePosition**`( )`
  - **Usage**
    - ermittelt die Position relativ zur aeussersten (root)-Komponente
  - **Returns** - die Position

- *getCSM*
  `public CoreStateMachine` **getCSM**`( )`
  - **Usage**
    - ermittelt die CSM, in der diese Komponente verbaut ist.
  - **Returns** - entweder die CoreStateMachine oder null, wenn die Komponente in keiner CSM verbaut ist

---

- *getName*
  `public final String` **getName**`( )`

- *getParent*
  `public final CSMComponent` **getParent**`( )`
  - **Usage**
    - die Funktion parent gemaess Definition 1 des Skripts
  - **Returns** - die CSM-Komponente, in der diese Komponente enthalten ist

---

- *getPosition*
  `public final Point` **getPosition**`( )`

- *isComponentOf*
  `public final boolean` **isComponentOf**`( csm.statetree.CSMComponent` **possibleParent** `)`
  - **Usage**
    - die Funktion >= gemaess Def. 1 des Skriptes, also die reflexiv-transitive Huelle der parent-Funktion
  - **Parameters**
    - `possibleParent` - die Komponente, in der moeglicherweise this enthalten ist
  - **Returns** - true, wenn dieses Objekt ein Unterobjekt von possibleParent oder possibleParent selbst ist

---

- *isSubComponentOf*
  `public final boolean` **isSubComponentOf**`( csm.statetree.CSMComponent` **possibleParent** `)`

  - **Usage**
    * die Funktion >gemaess Definition 1 des Skripts, also die transitive Huelle der parent-Funktion
  - **Parameters**
    * `possibleParent` - die Komponente, in der moeglicherweise this enthalten ist
  - **Returns** - true, wenn dieses Objekt ein Unterobjekt des Parameters ist

---

- *remove*
  `public final void` **remove**`( csm.statetree.CSMComponent` **child** `)`

- *setName*
  `public final void` **setName**`( java.lang.String` **name** `)`

  - **Usage**
    * Name oder Kommentar, muss nicht eindeutig sein, kann auch null sein
  - **Parameters**
    * `name` - irgendein String oder null

- *setPosition*
  `public final void` **setPosition**`( java.awt.Point` **position** `)`

  - **Usage**
    * setzt die Koordinate dieser Komponente
  - **Parameters**
    * `position` - muss ungleich null sein

4.1.5 Class **EntryState**

im Paper als S_entry bekannt

Declaration

public final class EntryState
extends csm.statetree.ConnectionPoint

Constructors

- *EntryState*
  `public` **EntryState**`( java.awt.Point` **position** `)`

4.1.6 Class **ExitState**

im Paper als S_exit bekannt

DECLARATION

public final class ExitState
extends csm.statetree.ConnectionPoint

CONSTRUCTORS

- *ExitState*
  `public` **ExitState**`( java.awt.Point` **position** `)`
  - **Usage**
    - erzeugt einen neuen ExitState mit kindOfExitstate=PR.

METHODS

- *getKindOfExitstate*
  `public ExitState.KindOfExitstate` **getKindOfExitstate**`( )`
  - **Returns** - der Typ des Exitstates
- *setKindOfExitstate*
  `public void` **setKindOfExitstate**`( csm.statetree.ExitState.KindOfExitstate` **kindOf** `)`
  - **Usage**
    - setzt den Typ des Exitstates
  - **Parameters**
    - `kindOf` - muss ungleich null sein

4.1.7 CLASS ExitState.KindOfExitstate

Der Typ des ExitStates.

DECLARATION

public static final class ExitState.KindOfExitstate
extends java.lang.Enum

FIELDS

- public static final ExitState.KindOfExitstate PR
  –
- public static final ExitState.KindOfExitstate NPR

–

- public static final ExitState.KindOfExitstate CP

  –

Methods

- *valueOf*
  `public static ExitState.KindOfExitstate` **valueOf**`( java.lang.String` **name** `)`
- *values*
  `public static final ExitState.KindOfExitstate` **values**`( )`

4.1.8 Class **FinalState**

im Paper als S_final bekannt

Declaration

public final class FinalState **extends** csm.statetree.InternalState

Constructors

- *FinalState*
  `public` **FinalState**`( java.awt.Point` **position** `)`

Methods

- *add*
  `public final void` **add**`( csm.statetree.ExitState` **child** `)`
  - **Usage**
    - Fuegt den Child-States dieser Komponente einen ExitState hinzu
  - **Parameters**
    - `child` - der hinzuzufuegende ExitState
  - **Exceptions**
    - `csm.exceptions.ErrTreeNotChanged` - wenn der ExitState schon das Child irgendeiner Komponente ist

4.1.9 Class **InternalState**

DECLARATION

public abstract class InternalState
extends csm.statetree.State

METHODS

- *stateOf*
  `public final InternalState` **stateOf**`( )`

4.1.10 CLASS **OutermostRegion**

eine spezielle Region, die keine Parent-Komponente hat. Sie hat zusaetzlich einen Start-State und einen (unveraenderlichen) Verweis auf die CoreStateMachine, deren Teil sie ist.

im Paper als SubRegion “epsilon bekannt

DECLARATION

public final class OutermostRegion
extends csm.statetree.Region

CONSTRUCTORS

- *OutermostRegion*
  `public` **OutermostRegion**`( csm.CoreStateMachine` **containingCSM** `)`
  - **Parameters**
    - `containingCSM` - die CoreStateMachine, zu der diese Region gehoert; muss ungleich null sein

METHODS

- *enumerateStates*
  `public void` **enumerateStates**`( )`
  - **Usage**
    - alle Komponenten mit neuen, eindeutigen uniqueIds versehen
- *getCSM*
  `public CoreStateMachine` **getCSM**`( )`
- *getStartState*
  `public CompositeState` **getStartState**`( )`

  - **Returns** - der StartState der CoreStateMachine oder null, wenn kein StartState definiert ist

---

- *setStartState*
  `public void` **setStartState**`( csm.statetree.CompositeState` **state** `)`
  - **Usage**
    * setzt den Start-State, wenn er 1. ein Composite-State und 2. direkt in dieser aeussersten Region enthalten ist.
  - **Parameters**
    * `state` - der Start-State oder null
  - **Exceptions**
    * `csm.exceptions.ErrTreeNotChanged` - wenn der neue Start-State nicht direkt in dieser aeussersten Region liegt.

4.1.11 Class **Region**

Abstrakte Oberklasse aller Regionen

Declaration

public abstract class Region **extends** csm.statetree.CSMComponent

Methods

- *add*
  `public final void` **add**`( csm.statetree.InternalState` **child** `)`
  - **Usage**
    * Fuegt den Child-States dieser Region einen InternalState hinzu
  - **Parameters**
    * `child` - der hinzuzufuegende InternalState
  - **Exceptions**
    * `csm.exceptions.ErrTreeNotChanged` - wenn der InternalState schon das Child irgendeiner Komponente ist, oder wenn er durch das Hinzufuegen ein Substate seiner selbst wuerde

4.1.12 Class **State**

Abstrakte Oberklasse aller States

Declaration

public abstract class State **extends** csm.statetree.CSMComponent

Methods

- *getUniqueId*
  `public int` **getUniqueId**`( )`
  - **Usage**
    - Damit beim Laden der CSM die Zuordnung der Connections zu ihren Source- und Target-States erhalten bleibt, erhaelt jeder State eine eindeutige ID. Diese ID wird beim Modifizieren von States nicht geupdated. Deshalb wird der Statetree bei jedem Speichern neu durchnummeriert.

      Wer die uniqueId ausserhalb des csm-Pakets verwendet, ist selbst schuld.
  - **Returns** - die uniqueId dieses States

- *mayConnectTo*
  `public boolean` **mayConnectTo**`( csm.statetree.State` **target** `)`
  - **Returns** - true, wenn eine Transition diesen State mit dem angegebenen Target-State verbinden darf

---

- *regOf*
  `public final Region` **regOf**`( )`
  - **Usage**
    - gemaess Paper, Definition 1: stateOf(parent(this))
  - **Returns** - die umgebende SubRegion dieses States

- *stateOf*
  `public abstract InternalState` **stateOf**`( )`
  - **Usage**
    - gemaess Paper, Definition 1
  - **Returns** - fuer einen ConnectionPoint der enthaltende InternalState, fuer einen InternalState diesen selbst

4.1.13 Class **SubRegion**

eine Unterregion eines Composite State

Diese Klasse kann als Unterregion in CompositeStates eingetragen werden. Ansonsten ist sie mit der Region identisch.

Declaration

public final class SubRegion **extends** csm.statetree.Region

CONSTRUCTORS

- *SubRegion*
  `public` **SubRegion**`( java.awt.Point` **position** `)`

4.1.14 CLASS **Transition**

DECLARATION

public final class Transition **extends** csm.statetree.CSMComponent

CONSTRUCTORS

- *Transition*
  `public` **Transition**`( java.awt.Point` **location**`, csm.statetree.State` **source**`, csm.statetree.State` **target** `)`
  - **Usage**
    - Erzeugt eine neue Transition und traegt sie im Komponentenbaum ein. Wenn es nicht moeglich ist, Source- und Target-State zu verbinden, wird eine Exception geworfen, und der Komponentenbaum bleibt unveraendert.
  - **Parameters**
    - `source` - der Source-State dieser Transition
    - `target` - der Target-State dieser Transition

METHODS

- *getAction*
  `public final Action` **getAction**`( )`
- *getEvent*
  `public final NamedObject` **getEvent**`( )`
- *getGuard*
  `public final Expression` **getGuard**`( )`
- *setAction*
  `public final void` **setAction**`( csm.expression.Action` **action** `)`
  - **Usage**
    - Setzt die dieser Transition zugeordnete Aktion. Enthaelt die Aktion Verweise auf Variablen, die in der CSM, zu der diese Transition gehoert, nicht definiert sind, dann bleibt die Transition unveraendert, und es wird eine Exception geworfen.
  - **Parameters**
    - `action` - eine Aktion oder null, falls dieser Transition keine Aktion zugeordnet ist

  - **Exceptions**
    * `csm.exceptions.ErrUndefinedElement` - wenn in der Action undefinierte Variablen referenziert werden

---

- *setAction*
  `public final void` **setAction**`( java.lang.String` **action** `)`
  - **Usage**
    * die Action parsen und eintragen
  - **See Also**
    * `csm.statetree.Transition.setAction(Action)`

- *setEvent*
  `public final void` **setEvent**`( csm.Event` **event** `)`
  - **Parameters**
    * `event` - ein Event oder null, falls dieser Transition kein Event zugeordnet ist
  - **Exceptions**
    * `csm.exceptions.ErrUndefinedElement` - wenn der Event nicht definiert ist

- *setGuard*
  `public final void` **setGuard**`( csm.expression.Expression` **guard** `)`
  - **Usage**
    * Setzt den Guard der Transition.
  - **Parameters**
    * `guard` - the guard to set
  - **Exceptions**
    * `csm.exceptions.ErrUndefinedElement` - wenn im Guard undefinierte Variablen referenziert werden

---

- *setGuard*
  `public final void` **setGuard**`( java.lang.String` **guard** `)`
  - **Usage**
    * den Guard parsen und eintragen
  - **Parameters**
    * `guard` - the guard to set
  - **Exceptions**
    * `csm.exceptions.ErrUndefinedElement` - wenn im Guard undefinierte Variablen referenziert werden

4.1.15 Class **Visitor**

ein Visitor-Pattern fuer CSM-Komponentenbaeume.

Die Methode visitChildren(CSMComponent) besucht alle Children der Komponente. Das sollte ausreichen, um die meisten Operationen auf Komponentenbaeumen zu implementieren. Ein Beispiel findet sich unter OutermostRegion#enumerateStates() , ein anderes unter csm.CSMSaver

Declaration

public class Visitor
extends java.lang.Object

Constructors

- *Visitor*
  `public` **Visitor**`( )`

Chapter 5

Package csm

Package Contents *Page*

Classes

5.1 Classes

5.1.1 Class **CoreStateMachine**

die CoreStateMachine

Declaration

public final class CoreStateMachine
extends java.util.Observable

Fields

- public final OutermostRegion region
  - die Komponenten, Regionen und Transitionen der CSM als Baumstruktur
- public final Dictionary events
  - eine Liste aller Events, die von der CSM verwendet werden
- public final Dictionary variables
  - die Liste aller Variablen, die von der CSM verwendet werden

Constructors

- *CoreStateMachine*
  `public` **CoreStateMachine**`( )`

Methods

- *getGuiMetadata*
  `public final GuiMetadata` **getGuiMetadata**`( java.lang.String` **guiId** `)`
  - **Usage**
    - gibt die Metadaten einer gegeben GUI zurueck (oder null, wenn fuer diese GUI-Id keine Metadaten existieren)
- *loadCSM*
  `public final CoreStateMachine` **loadCSM**`( java.io.Reader` **reader** `)`
  - **Usage**
    - eine CSM im XML-Format laden
  - **Exceptions**
    - `java.io.IOException` - auftretende IO-Exceptions werden nicht behandelt, sondern an den Aufrufer weitergegeben
    - `csm.exceptions.CSMEditException` - wenn die Datei eine fehlerhafte Maschine beschreibt

- *saveCSM*
  `public void` **saveCSM**`( java.io.Writer` **writer** `)`
  - **Usage**
    - die CSM im XML-Format speichern
  - **Exceptions**
    - `java.io.IOException` - auftretende IO-Exceptions werden nicht behandelt, sondern an den Aufrufer weitergegeben

- *setGuiMetadata*
  `public final void` **setGuiMetadata**`( csm.GuiMetadata` **guiData** `)`
  - **Usage**
    - setzt die Metadaten fuer eine GUI

5.1.2 Class **Dictionary**

excute the list of the events and the list of the variable which will be used from the csm the hash map contents coulden't be managed from the outside of the package

Declaration

public final class Dictionary **extends** java.util.Observable

Constructors

- *Dictionary*
  `public` **Dictionary**`( )`

Methods

- *contains*
  `public boolean` **contains**`( java.lang.String` **key** `)`
  - **Usage**
    - Returns true if the map contents contains a mapping for the specified key

- *get*
  `public NamedObject` **get**`( java.lang.String` **key** `)`
  - **Usage**
    - Returns the elem to which the specified key is mapped in this identity hash map or throws the exception ErrUndefinedElement if the map contents contains no mapping for this key.
  - **Parameters**

    - `key` - to look for
  - **Exceptions**
    - `ErrUndefinedElement, -`

- *getInitials*
  `public HashMap` **getInitials**`( )`
  - **Returns** - eine neues Array, das die Initialwerte aller definierten Variablen enthaelt

---

- *mayNotContain*
  `public void` **mayNotContain**`( csm.NamedObject` **elem** `)`
  - **Usage**
    - throws an exception if the map contents already contains a mapping for the specified elem.getName
  - **Parameters**
    - `elem` - to look for
  - **Exceptions**
    - `csm.exceptions.ErrAlreadyDefinedElement` -

- *mayNotContain*
  `public void` **mayNotContain**`( java.lang.String` **key** `)`
  - **Usage**
    - throws an exception if the map contents already contains a mapping for the specified key
  - **Parameters**
    - `key` - to look for
  - **Exceptions**
    - `csm.exceptions.ErrAlreadyDefinedElement` -

- *mustContain*
  `public void` **mustContain**`( csm.NamedObject` **elem** `)`
  - **Parameters**
    - `elem` - to look for
  - **Exceptions**
    - `csm.exceptions.ErrUndefinedElement` - if the map contents dosen't contains a mapping for the specified elem.getName

---

- *mustContain*
  `public void` **mustContain**`( java.lang.String` **key** `)`
  - **Usage**
    - throws an exception if the map contents dosen't contains a mapping for the specified key
  - **Parameters**
    - `key` - to look for
  - **Exceptions**
    - `csm.exceptions.ErrUndefinedElement` -

- *remove*
  `public NamedObject` **remove**`( java.lang.String` **key** `)`
  - **Usage**
    - removes the mapping for this key from this map if present.otherwise throws an exception

      Diese Methode entfernt ein Element auch dann, wenn es woanders noch gebraucht wird. Deshalb sollte sie niemals verwendet werden!
  - **Parameters**
    - `key` -
  - **Exceptions**
    - `csm.exceptions.ErrUndefinedElement` - wenn das Dictionary kein Element dieses Namens enthaelt

---

- *rename*
  `public void` **rename**`( java.lang.String` **key**`, java.lang.String` **newkey** `)`
  - **Usage**
    - * renams the elem to which the specified key is mapped in this hash map and throws an exception if the map contents haven't got the specified key

      Diese Methode benennt ein Element nur im Dictionary um, aber nicht an den Stellen, an denen es verwendet wird. Deshalb sollte sie niemals aufgerufen werden!
  - **Parameters**
    - `key` -
    - `newkey` -
  - **Exceptions**
    - `csm.exceptions.ErrUndefinedElement` -
    - `csm.exceptions.ErrAlreadyDefinedElement` -

5.1.3 Class **Event**

Events sind "irgendwelche" benannte Objekte, die sich von anderen Objekten unterscheiden.

Declaration

public final class Event
extends csm.NamedObject

Constructors

- *Event*
  `public` **Event**`( csm.Dictionary` **parent**`, java.lang.String` **name** `)`

5.1.4 Class **ExpressionEnvironment**

eine Variablenbelegung

Variablenbelegungen werden nur in der semantischen Analyse verwendet. Die initiale Variablenbelegung wird in den Variablen-Objekten gesetzt.

neben den aktuellen Werten enthaelt ein VarAssignment den Namen und den Wert des im letzten Schritt gesendeten Events: Wenn eine Aktion keinen Event sendet, muss sie den Namen sendEventName auf null setzen. wenn sie einen Event senden will, muss sie sendEventName und sendEventValue setzen.

Declaration

public final class ExpressionEnvironment
extends java.lang.Object

Fields

- public String sendEventName
  - der Name des in der letzten Aktion gesendeten Events ode null, wenn kein Event gesendet wurde
- public int sendEventValue
  -
- public Boolean nab
  -
- public Boolean wla
  -

Constructors

- *ExpressionEnvironment*
  `public` **ExpressionEnvironment**`( csm.Dictionary` **dictionary** `)`
  - **Usage**
    - erzeugt ein initiales Variablen-Assignment aus der Variablenliste der CoreStateMachine. Dabei werden alle Variablen auf den in deren Variablenliste eingetragenen Initialwert gesetzt.
  - **Parameters**
    - `variableList` - die Liste der Variablen,

- *ExpressionEnvironment*
  `public` **ExpressionEnvironment**`( csm.ExpressionEnvironment` **e** `)`
  - **Usage**
    - erzeugt eine Kopie des Environments, in der sendEventName auf null gesetzt ist

Methods

- *lookupVar*
  `public int` **lookupVar**`( java.lang.String` **varname** `)`
  - **Usage**
    - ermittelt den Integer-Wert, der in diesem VarAssignment an einen Variablennamen gebunden ist. Ist dem uebergebenen Variablennamen kein Wert zugeordnet, dann ist das Verhalten der Funktion undefiniert.
  - **Parameters**
    - `varname` - der Variablenname

- *setVar*
  `public void` **setVar**`( java.lang.String` **n**`, int` **value** `)`
  - **Usage**
    - Setzt den einem Variablennamen zugeordneten Integer-Wert. Dabei werden keinerlei Ueberpruefungen des Gueltigkeitsbereichs vorgenommen. Ist an den uebergebenen Variablennamen bei der Erzeugung des VarAssignment-Objektes kein Wert gebunden worden, dann ist das Verhalten der Funktion undefiniert.

5.1.5 Class **GuiMetadata**

Mithilfe dieses Objektes koennen Metadaten fuer eine graphische Oberflaeche gespeichert werden.

Declaration

public class GuiMetadata **extends** java.lang.Object

Fields

- public final String guiId
  –
- public final HashMap data
  –

Constructors

- *GuiMetadata*
  `public` **GuiMetadata**`( java.lang.String` **guiId** `)`

5.1.6 Class **NamedObject**

ein benanntes Objekt, das in einem Dictionary verwaltet werden kann

Der Name kann nicht von ausserhalb des Packages gesetzt werden, da sonst nicht sicherzustellen ist, dass in einem Dictionary jeder Name nur einmal vorkommt.

Declaration

public abstract class NamedObject
extends java.util.Observable

Fields

- public final Dictionary parent
  –

Methods

- *getName*
  `public final String` **getName**`( )`

5.1.7 Class **Variable**

Declaration

public final class Variable
extends csm.NamedObject

Constructors

- *Variable*
  `public` **Variable**`( csm.Dictionary` **parent**`, java.lang.String` **name** `)`
  - **Usage**
    - setzt die Minimal-, Maximal- und Initialwerte auf 0
  - **Parameters**
    - `name` - ungleich `null`

- *Variable*
  `public` **Variable**`( csm.Dictionary` **parent**`, java.lang.String` **name**`, int` **i**`, int` **min**`, int` **max** `)`

  - **Parameters**
    * `i` - Initialwert
    * `min` - Minimalwert
    * `max` - Maximalwert

METHODS

- *getInitialValue*
  `public int` **getInitialValue**`( )`
- *getMaxValue*
  `public int` **getMaxValue**`( )`
- *getMinValue*
  `public int` **getMinValue**`( )`
- *setInitialValue*
  `public void` **setInitialValue**`( int` **initialValue** `)`
  - **Exceptions**
    * `csm.exceptions.ErrValueOutOfBounds` - wenn der Initialwert ausserhalb der Grenzen liegt

  ---
- *setMaxValue*
  `public void` **setMaxValue**`( int` **maxValue** `)`
  - **Exceptions**
    * `csm.exceptions.ErrValueOutOfBounds` - wenn der Maximalwert ausserhalb der Grenzen liegt

  ---
- *setMinValue*
  `public void` **setMinValue**`( int` **minValue** `)`
  - **Exceptions**
    * `csm.exceptions.ErrValueOutOfBounds` - wenn der Minimalwert ausserhalb der Grenzen liegt

  ---
- *setValues*
  `public void` **setValues**`( int` **i**`, int` **min**`, int` **max** `)`
  - **Usage**
    * setzt alle drei Werte auf einmal
  - **Parameters**
    * `i` - Initialwert
    * `min` - Minimalwert
    * `max` - Maximalwert
  - **Exceptions**
    * `csm.exceptions.ErrValueOutOfBounds` -