VERTEILTE SYSTEME

UNIKASSEL
VERSITÄT

Effizientes und generisches Visualisierungswerkzeug für ROS-Anwendungen

Ausarbeitung von

Pavel Fischer

Vorgelegt im

Fachgebiet Verteilte Systeme

Universität Kassel

31. März 2012

Inhaltsverzeichnis

1 Einleitung

Seit dem Jahr 2006 nimmt die Universität Kassel an dem RoboCup[1] als das Team „Carpe Noctem“teil. Das Ziel dieses Wettbewerbes ist die Entwicklung eines autonomen menschenähnlichen Fußball Roboterteams, das im Jahr 2050 das Menschen-Team schlagen soll. Die Spielregeln der MiddleSizeLeague[2], in welcher die Roboter aus der Universität Kassel spielen, und anderen Ligen werden jedes Jahr mehr an die FIFA-Regeln angepasst, somit müssen die Roboter ständig modernisiert und angepasst werden.

Das Spiel verläuft in einer hoch dynamischen Umgebung und ist eine große Herausforderung für das Entwickler-Team. Die Roboter sollten in der Lage sein, Daten von mehreren Sensoren zu verarbeiten, seine Verhaltensweisen der Lage des Spiels anzupassen und gemeinsame Entscheidungen mit Teammitgliedern zu treffen.

Während der Entwicklung von Fußballrobotern ergab sich die Notwendigkeit, eine grafische Kontroll- und Steuerungsumgebung zu implementieren, die beim Modernisierung und beim Überwachen der Sensordaten der Roboter hilft. Seit dem Anfang der Entwicklung im Jahr 2006 waren über die Jahre mehrere Leute damit beschäftigt, die neue Funktionalität in die Umgebung zu integrieren. Diese Änderungen waren oft unsauber integriert, was mit der Zeit zur schlechteren Wartbarkeit des Programms führte, was aber nicht das einzige Problem war. Die Anwendung hatte keine besonderen Technologien für die Fußballfelddarstellung benutzt, was zur höheren CPU-Last führte.

Die Wartbarkeits- und Erweiterungsprobleme könnten mit der Modularisierung der Anwendung gelöst werden. Dafür sollte die ursprüngliche Anwendung auf einzelne Komponenten aufgeteilt und danach neu zusammengefügt werden. Außerdem müssen neue Komponenten geschrieben werden, die die Implementierung von neuen Modulen erleichtern. Höhere CPU-Last ist dadurch bedingt, dass CPU alle Aufgaben für Alpha-Blending

[1] `http://www.robocup.org/`
[2] `http://www.robocup.org/robocup-soccer/middle-size/`

während des Feld- und Objekte-Renderings übernimmt. Die Entwicklung eines neuen OpenGL-basierten Rendering Verfahrens würde die CPU-Last verringern.

Im Kapitel 2 wird die Aufgabenstellung präsentiert. Danach werden im Kapitel 3 die Grundlagen erläutert. Im Kapitel 4 wird der Lösungsansatz und einzelne neue und modifizierte Komponenten vorgestellt. Eine Zusammenfassung der Arbeit und ein Ausblick befinden sich im Kapitel 5.

2 Aufgabenstellung

Während meiner Projektarbeit musste ich die ursprüngliche Anwendung für Überwachung der Sensor-Daten der Roboter modernisieren und modifizieren, um die Probleme der Wartbarkeit und der Leistung zu lösen. Die einzelnen Punkte sind in der Liste dargestellt.

- Implementieren eines flexibles GUI. Das Originalprogramm hat fest vorgeschriebene Positionen für die einzelne Sensor-Daten, was das Einführen der neuen Sensor-Daten erschwert. Außerdem sind nicht immer alle Daten nötig, und es kann bei übermäßiger Anzahl dargestellter Daten dazu führen, dass die Aufmerksamkeit wegen Unübersichtlichkeit abnimmt.
- Umbauen der Struktur für leichtere Erweiterung und Wartbarkeit. Die vorhandene monolithische Struktur verhindert komplexere Änderungen der grundlegenden Strukturen der Anwendung.
- Modifikationen zum Verringern der CPU-Last. Die Anwendung sollte kleinstmögliche CPU-Last zeigen, damit die Simulationsanwendungen die frei gewordene CPU-Zeit nutzen könnten.
- Modifikationen zwecks leichterer Konfigurierbarkeit der Anwendung. Die ursprüngliche Anwendung besitzt keine Mechanismen für Ansicht- und Verhaltenssteuerung des Programms.
- Implementierung einer Erweiterung für schnellere Änderung des Layouts. Diese Erweiterung sollte in der Lage sein, die Positionen einzelner Komponenten zu speichern und bei Bedarf neu zu laden, damit der Nutzer abhängig von seine Teilaufgabe die benötigte Daten besser darstellen könnte und bei Bedarf schnell zwischen den Layouts wechseln kann.

3 Grundlagen

3.1 ROS

Die Fußballroboter verwenden das ROS-Framework[1] (Robotic Operations System) als grundlegendes System. ROS ist ein Middleware System, das Kommunikation zwischen einzelnen Komponenten des Roboters und zwischen Robotern erleichtert. Außerdem besitzt das Framework viele hilfreiche Anwendungen und Bibliotheken, die die Entwicklung von Robotersteuerungssystemen erleichtert.

Die Grundidee von ROS ist der Nachrichtenaustausch. Die Nachrichten werden thematisch in Topics einsortiert. Einzelne Teile der Roboter werden als separate ROS-Komponente erstellt. Beispielsweise benötigt der Roboter eine Komponente für das Wahrnehmen der Umgebung, eine weiter Komponente für das Benutzen von Aktoren zuletzt eine zum Planen. Mittels ROS ist es möglich diese Komponenten lose zu koppeln, und mittels des Nachrichtensystems von ROS zu verbinden. Jede ROS-Komponente (Node in der ROS Terminologie) kann Topics anbieten und andere Topics abonnieren.

3.2 OpenGL

OpenGL[2] ist eine Spezifikation für eine plattform- und programmiersprachenunabhängige Programmierschnittstelle zur Entwicklung von 2D- und 3D-Computergrafik. Der OpenGL-Standard beschreibt etwa 250 Befehle, die die Darstellung komplexer 3D-Szenen in Echtzeit erlauben. Zudem können andere Organisationen (zumeist Hersteller von Grafikkarten) proprietäre Erweiterungen definieren.

[1] `http://www.ros.org/wiki/`
[2] `http://www.opengl.org/`

Die Implementierung des OpenGL-API erfolgt in der Regel durch Systembibliotheken wie Mesa, auf einigen Betriebssystemen auch als Teil der Grafikkarten-Treiber. Diese führen entsprechend Befehle der Grafikkarte aus, insbesondere müssen auf der Grafikkarte nicht vorhandene Funktionen durch die CPU emuliert werden.

3.3 Clutter

Clutter[3] ist eine freie Szenengraph-Programmbibliothek, mit der die Software-Entwickler grafische Benutzeroberflächen (GUIs) für Anwendungen erstellen können.

Clutter wurde anfangs von OpenedHand entwickelt. OpenedHand wurde später von Intel gekauft, um auf Basis von Clutter eine Oberfläche für Moblin (heute MeeGo Netbook) zu kreieren. Mittlerweile nutzt eine Reihe von Software-Projekten vor allem GNOME[4] dieses Toolkit.

3.4 Mono und C#

Mono[5] ist eine .NET-kompatible Entwicklungs- und Laufzeitumgebung für plattformunabhängige Software, basierend auf dem CLI-Standard sowie der Programmiersprache C# und ergänzt durch die integrierte Entwicklungsumgebung MonoDevelop. Vorangetrieben wurde das Open-Source-Projekt primär durch Mitarbeiter des Softwareunternehmens Novell, später dann Xamarin[6].

[3] http://www.clutter-project.org/
[4] http://www.gnome.org/
[5] http://www.mono-project.com/Main_Page
[6] http://xamarin.com/

4 Lösungsansatz

Kurz nach dem Start der Entwicklung wurde festgestellt, dass die Aufgabenstellung ohne größere Modifizierungen erweitert werden kann, um die Entwicklung beliebiger Anwendungen für Visualisierung der Daten zu ermöglichen. Um alle Punkte der Aufgabenstellung abzudecken, musste der LebtClient, so heißt die ursprünglichen Anwendung, in einzelne möglichst voneinander unabhängige Komponenten geteilt werden.

Da die einzelnen Komponente von einander unabhängig sein müssen, benötigten sie entweder eigene ROS-Kommunikationsschnittstellen, was aber die Netzwerklast erhöht, weil einige Nachrichten für mehrere Komponenten nötig sind, oder es wird ein neues ROS-Kommunikationsmittel benötigt, um die Netzwerklast nicht zu erhöhen. Diese Kommunikationsschnittstelle wurde als ein Proxy-Server implementiert, der Daten aus ROS speichert, ins ROS versendet und eine größere Menge von Zugriffsmethoden besitzt. Durch diese Methoden könnten die Komponenten unabhängig voneinander auf bestimmte Datenbereiche zugreifen, und so wird das Problem der Netzwerküberlastung verhindert.

Um einzelne Komponenten beliebig im Anwendungsfenster oder aus dem Fenster heraus verschieben zu können, wurde ein Dock-Manager, der unter einer freien Lizenz veröffentlicht wurde, aus `MonoDevelop` integriert. Am Anfang wurde die `libgdl`-Bibliothek aus `Gnome`-Paketen als Dockmanager geplant, aber während der Entwicklung wurde die unzureichende Funktionalität festgestellt, zum Beispiel kann es nur ein Paneel mit minimierten Komponenten existieren und dieses Paneel kann nicht automatisch versteckt werden, wenn es keine minimierten Komponenten gibt.

Der LebtClient wurde auf folgenden Komponenten aufgeteilt: Fußballfeld, RefereeBox, Allgemeine Steuerungsknöpfe und Informationsanzeige mit detaillierter Information zum Roboterstatus. Die interne Struktur dieser Teile wurde so geändert, dass jede Komponente nur die für sie erforderlichen Daten aus dem Gesamtdatensatz herausgreift und als ihre

eigene Kopie speichert.

Alle grundlegenden Komponenten und einige weitere Hilfskomponenten (wie zum Beispiel Einstellungs- und Schnelltastenmanager) wurden in ein eigenes Paket `CNUtils/UniversalGUI` ausgelagert. Dies erlaubt eine leichtere Entwicklung von weiteren Anwendungen mit ähnlicher Funktionalität wie der LebtClient.

4.1 Implementierung

4.1.1 Grundstruktur

In der Abbildung 4.1 ist die Gesamtstruktur der Anwendung dargestellt. Sie besteht aus folgenden Komponenten: `ControlGUI`, `GUIItem`, `HotKeyManager`. `ItemLoader`, `ItemUpdater`,`MenuGenerator`,`SettingsManager`,`Proxy`,`ProxyGenerator`,`FootballField`, `GlobalControl`, `RefereeBox`, `RobotInfoBox`. Der Proxy ist die einzige Komponente, die den direkten Zugriff zum ROS hat. Sie ist durch eine Klasse, die einen IProxy Interface implementiert, realisiert und wird durch einen ProxyGenerator laut der Einstellungsdateien generiert.

Während des Programmstarts wird das Anwendungsfenster mit dem Hauptmenü und einem Dock, der in der ControlGUI Komponente integriert ist, erstellt. Danach wird der SettingsManager initialisiert, der allgemeine Einstellungen aus den Dateien liest. Nach der Initialisierung werden einzelne GUI-Komponenten mittels ItemLoader erstellt und in ControlGUI und ItemUpdater integriert. ItemLoader verbindet auch diese Komponenten mit dem Proxy. Anschließend wird der MenuGenerator erstellt, der auf die Events des SettingsManagers hört und die entsprechenden Punkte des Menüs verändert. Als Letztes wird der ItemUpdater gestartet, der regelmäßig die Update-Methode der GUI-Komponenten ausführt.

4.1.2 Proxy

Der Proxy dient sowohl als eine Verbindungsschicht zwischen der Anwendung und ROS, als auch als Datenzwichenpuffer. Eigentlich existiert keine direkte Verbindung zwischen den GUI-Elementen und dem Proxy. Diese Verbindung wird während des Starts mittels ItemLoader hergestellt. Die genauere Funktionsweise des Itemloaders ist im Ab-

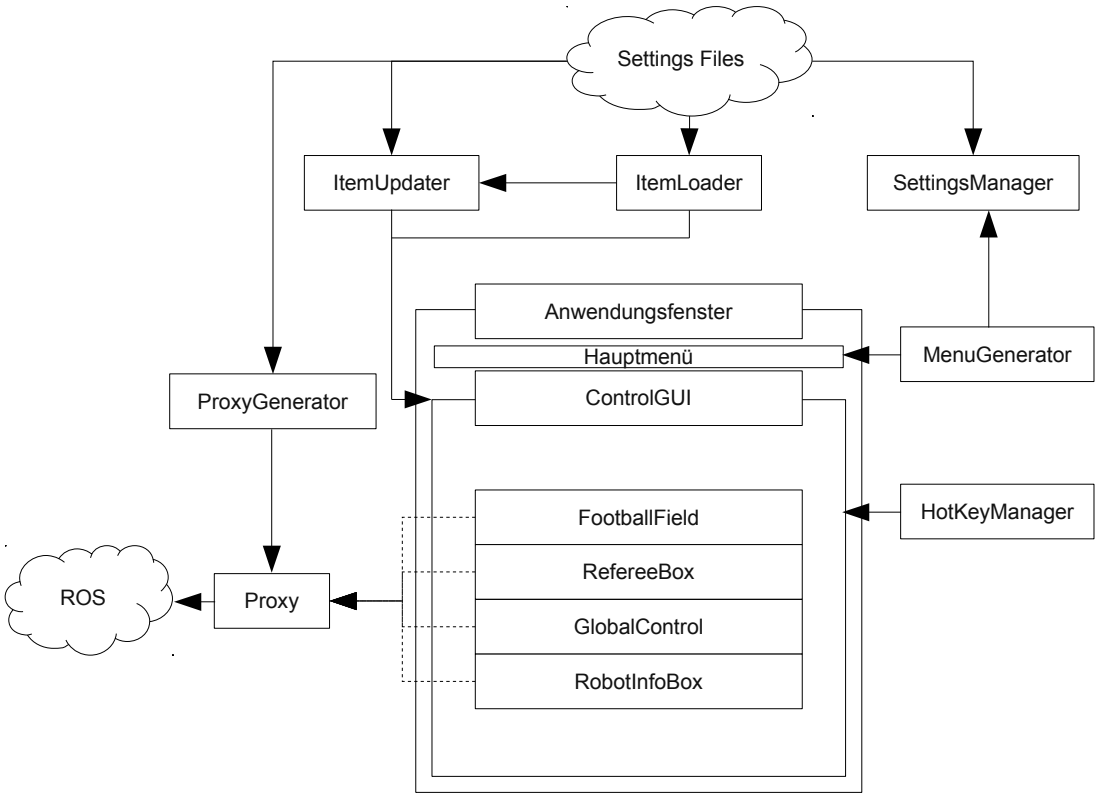

Abbildung 4.1: Gesamtstruktur der Anwendung

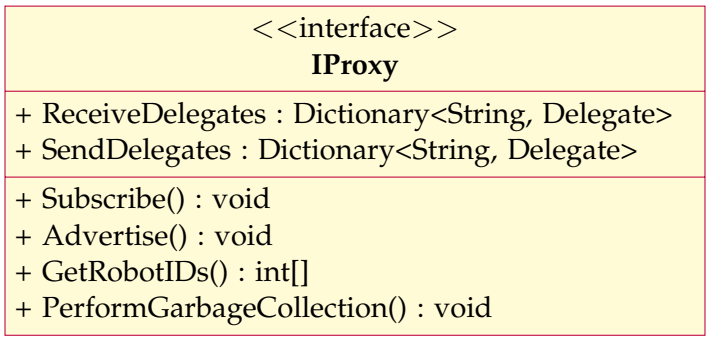

Abbildung 4.2: Interface IProxy

schnitt 4.1.6 beschrieben. Die Proxy-Klasse soll einen IProxy Interface implementieren, um die korrekte Funktionsweise des ItemLoaders zu gewährleisten.

Das Interface IProxy ist in der Abbildung 4.2 dargestellt. `ReceiveDelegates` und `SendDelegates` Wörterbücher enthalten Delegaten, die bestimmte Teile der Datenstruktur zurückliefern, beziehungsweise bestimmte Nachrichten verschicken. `Subscribe` und `Advertise` Methoden verbinden interne Datenstruktur mit ROS. Die Methode `GetRobotIDs` liefert ein Array mit den IDs der Roboter, wessen Daten momentan im Proxy gespeichert sind. Dieser Array wird von GUI-Elementen benutzt, um festzustellen, ob ein neuer Roboter hinzugefügt oder entfernt wurde. Die letzte Methode, `PerformGarbageCollection`, startet interne Bereinigungsfunktion, die Daten löscht, die während des gegebenen Zeitintervalls nicht aktualisiert wurden. Außerdem soll diese Funktion auch die Verweise auf die Roboter löschen, zu denen keine aktuelle Information mehr existiert.

Die Klasse, die die Proxy-Komponente realisiert, wird mittels einer anderen Komponente ProxyGenerator aus dem Paket „UniversalGUI“generiert. Der Generator erstellt anhand der Daten aus einer `.config`-Datei und eines Schablonenordners ein Proxy. Das Erstellen erfolgt automatisch während der Programmkompilierung. Außerdem prüft dieser Befehl, ob die Einstellungsdatei geändert wurde, und erstellt das Proxy neu, falls erforderlich.

Auf dem jetzigen Entwicklungsstand ist der ProxyGenerator in der Lage, Proxys zu generieren, die mit ROS arbeiten und nur vorgegebene Template-Variablen enthalten.

In der Einstellungsdatei wird gespeichert, welche Nachrichten die Anwendung annehmen beziehungsweise verschicken will. Die Struktur der Datei ist in der Tabelle 4.1 dargestellt.

Für das Generieren des Proxy muss der ProxyGenerator die Struktur der Nachrichten wis-

Tabelle 4.1: Struktur der Proxy Einstellungsdatei

Parameter	Bezeichnung
Namespace	Name des Namensraumes mit Punkt am Anfang
IDProperty	Name des Attributes, der die Roboteridentifikationsnummer enthält
HistoryAmount	Anzahl der Nachrichten aus dem rosout-Topic, die gespeichert werden
[Subscribe]	In dieser Section werden die abonnierten Topics aufgelistet
[Topic Name]	Für den Entwickler verständliche Bezeichnung des Topics
Topic	Name des ROS-Topics
MessageType	Voller Name der C#-Klasse mit der Nachrichtendefinition aus diesem Thema
UseFilter	Boolesche Variable, definiert, ob die Bereinigung von diesen Topic erlaubt ist oder nicht
TimeOutInterval	Zeitintervall in `ms` seit der letzten Aktualisierung, nach der das Topic bereinigt wird
[!Topic Name]	Ende der Topics Section
[!Subscribe]	Ende des Subscribe Section
[Advertise]	In dieser Section werden die angebotenen Topics aufgelistet
[Topic Name]	Für den Entwickler verständliche Bezeichnung des Topics
Topic	Name des ROS-Topics
MessageType	Voller Name der C#-Klasse mit der Nachrichtendefinition aus dieser Thema
[!Topic Name]	Ende der Topics Section
[!Advertise]	Ende der Advertise Section

sen. Dafür lädt er anwendungsbezogenen „`.Communications`“Assembly und alle ihre referenzierten Assemblies, die „.Messages“im Namen enthalten. Außerdem werden die Dateien aus dem Schablonenordner und die Einstellungsdatei geladen. Danach werden die Markierungen in der `Main`-Schablone sukzessiv mit dem Text, der aus der entsprechenden Schablonendatei generiert wurde, ersetzt. Die komplexen Nachrichten, die weitere Nachrichten enthalten, werden rekursiv bearbeitet.

4.1.3 Settings Manager

Der SettingsManager dient zur zentralisierten Verwaltung und Speicherung der Einstellungen. Er speichert jetzige und existierende Ansichtsschemas, Position und Größe des Anwendungsfensters und Einstellungen für einzelne Komponenten oder andere Objekte

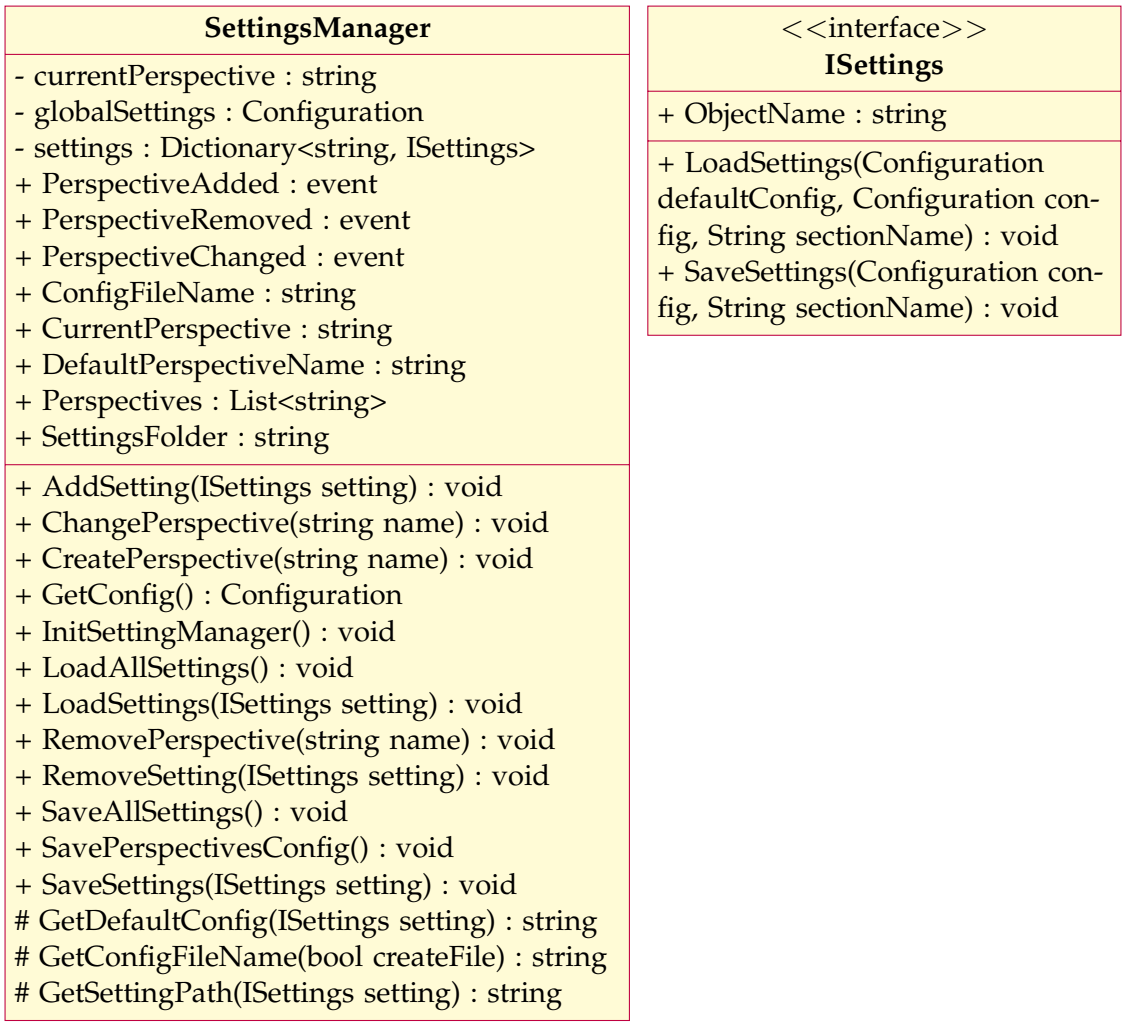

Abbildung 4.3: Klasse SettingsManager und Interface ISettings

zum Beispiel für die Roboter. Die Klasse SettingsManager und das ISettings Interface sind in der Abbildung 4.3 dargestellt. Einzelne Einstellungsdateien werden im vorgegebenen Ordner als Castor-Dateien gespeichert. Castor ist ein Paket aus ROS für Speicherung der Einstellungen.

Bevor man den Manager nutzen kann, müssen die Properties `ConfigFileName` und `SettingsFolder` eingestellt und die Methode `InitSettingManager` aufgerufen werden. Das `ConfigFileName`-Property enthält den Dateinamen mit den globalen Einstellungen und im Property `SettingsFolder` wird der Pfad zu dieser Datei vorgegeben. Die Struktur der Datei ist in der Tabelle 4.2 dargestellt.

Defaultwerte für die Einstellungen müssen in der entsprechend genannten Datei gespeichert werden. Der Name dieser Datei ist der Name der Klasse, die das `ISettings`-Interface implementiert. Die Werte aus der Default-Datei werden nur dann gelesen, wenn die Einstellungsdatei für das gegebene Ansichtsschema keinen Wert für diesen Parameter enthält.

Tabelle 4.2: Struktur der Datei mit globalen Einstellungen

Parameter	Bezeichnung
[SettingsManager]	In dieser Section werden die Parameter aufgelistet
DefaultPerspective	Name des Default Ansichtsschemas
Perspectives	Liste aller Schemanamen, getrennt durch Semikolon
CurrentPerspective	Name des aktuellen Ansichtsschemas
WindowSize	Breite und Höhe des Anwendungsfensters, getrennt durch Semikolon
[!SettingsManager]	Ende der SettingsManager Section

Für das automatische Laden und Speichern von Einstellungen nach der Änderung der Ansichtsschemas sollten die einzelnen Einstellungsobjekte mittels Methode `AddSetting` zu der Liste der Einstellungen hinzugefügt werden. Der Manager selbst kann keine Einstellungen serialisieren oder de-serialisieren. Er übernimmt nur die Lokalisierungsaufgabe, das heißt er weißt, wo die Daten gespeichert werden sollten aber nicht wie. Das Speichern und Wiederherstellen übernimmt die Einstellungsklasse selbst.

4.1.4 Menügenerator

Der Menügenerator ist verantwortlich für das Generieren von `Settings`-, `View`- und `Perspective`-Punkten des Hauptmenüs und ihre Anpassung während der Änderung der Ansichtsschemas. Der Menügenerator ist vom Einstellungsmanager abhängig, somit muss er nach der Initialisierung des Einstellungsmanagers initialisiert werden. Während der Initialisierung abonniert der Generator alle Ereignisse des Einstellungsmanagers, und die Änderungen in den Ansichtsschemas richtig zu behandeln. Außerdem werden Menüeinträge für das Erstellen, Löschen und Ändern des Ansichtsschemas sowie Einträge für Einblenden und Ausblenden der einzelnen GUI-Elemente generiert. Die Klasse MenuGenerator und das IMenuSupport-Interface sind in der Abbildung 4.4 dargestellt.

Der Inhalt der Menüpunkte „View “und „Perspective“wird automatisch angepasst, ohne das gesonderte Befehle dazu benötigt werden. Einzelne Menüeinträge ins „Settings“-Menü

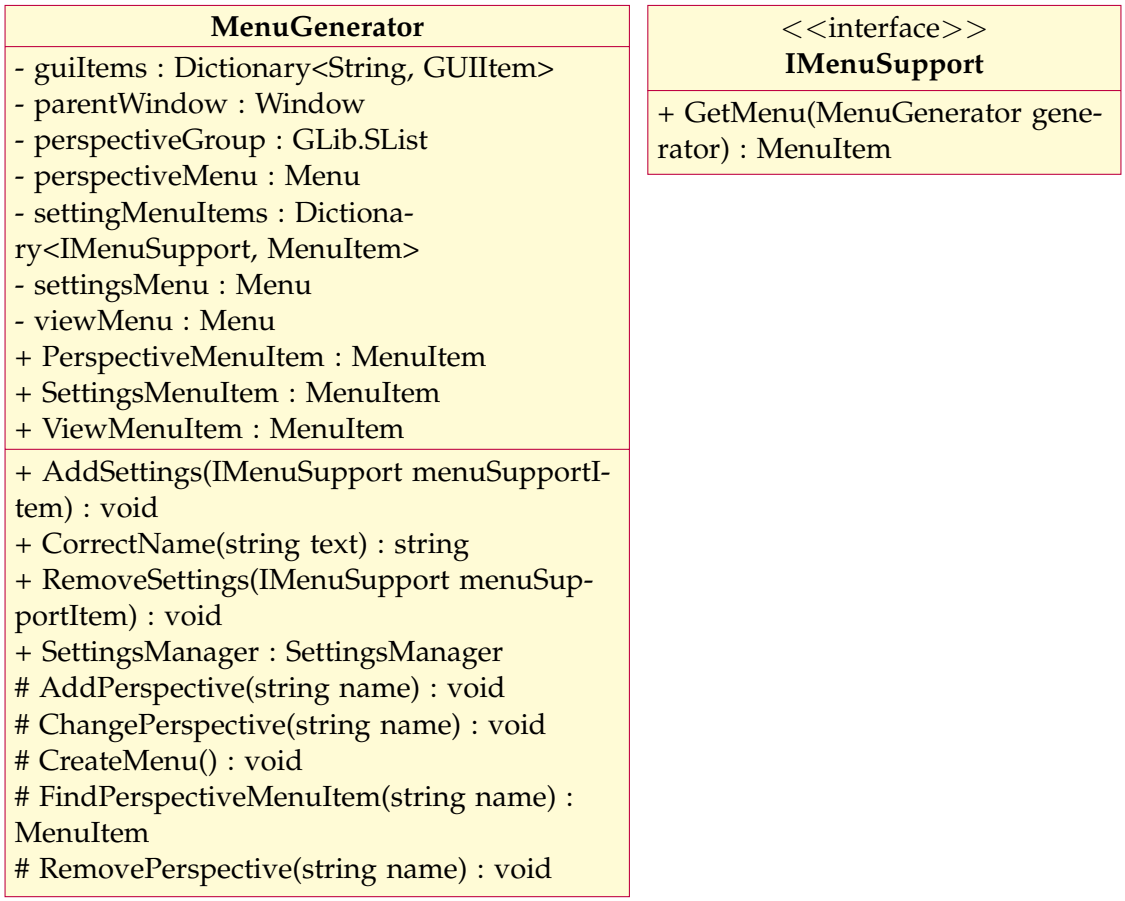

Abbildung 4.4: Klasse MenuGenerator und Interface IMenuSupport

werden mittels Methoden `AddSetting` und `RemoveSettings` gesteuert. Die entwickelte Version vom Menügenerator unterstützt weder Vereinigung verschiedener Einträge in Gruppen noch alphabetische Sortierung der Menüeinträge.

4.1.5 HotKey Manager

Aufgrund der Tatsache das GTK# nur ein Binding von GTK+ für C# Sprache ist, ist nicht die ganze Funktionalität des GTK+ in C# nutzbar. Ein Beispiel dafür sind „HotKeys“. Die Funktion, die ein anwendungsglobales Shortcut erstellt, benötigt einen CallBack-Funktionspointer als einen Parameter. Das Aufrufen eines Managed-Code aus Unmanaged-Code ist sehr schwierig realisierbar und außerdem sehr instabil, weil der GarbageCollector von C# nicht wissen kann, wann er die mit Unmamaged-Code verbundenen Objekte löschen kann.

Der HotKey Manager implementiert eine sehr vereinfachte Version von den globalen Shortcuts, indem alle GUI-Komponente in einem einzelnen EventBox gepackt werden, der `KeyPressed`- und `KeyReleased`-Ereignisse filtert und vorgegebene Funktionen aufruft. Die Klasse HotKeyManager und die Struktur HotKeyEntry sind in der Abbildung 4.5 dargestellt.

Die einzelnen HotKeys werden mittels Methoden `AddKeyPressListener`, `AddAllKeyPressListeners` (beziehungsweise `AddKeyReleaseListener`, `AddAllKeyReleaseListeners`) hinzugefügt und mittels Methoden `RemoveKeyPressListener`, `RemoveAllKeyPressListeners` (beziehungsweise `RemoveKeyReleaseListener`, `RemoveAllKeyReleaseListeners`) entfernt.

4.1.6 Item Loader

Item Loader ist eine Hilfsklasse, die einzelne GUI Elemente aus der Einstellungsdatei `UIItems.conf` lädt und sie mit dem Proxy verbindet. Die Struktur der Datei ist in der Tabelle 4.3 dargestellt.

Der Loader bearbeitet sequenziell alle Sections aus der Datei. Zuerst werden die `ID`, `IconName` und `ClassName` Werte gelesen und eine neue Instanz von den gelesenen Klasse erstellt. Danach wird das Bild, wenn eines existiert hat, aus der Assembly-Ressourcen gelesen und im erstellten Objekt, sowie die `ID` gespeichert. Danach werden die Einstellun-

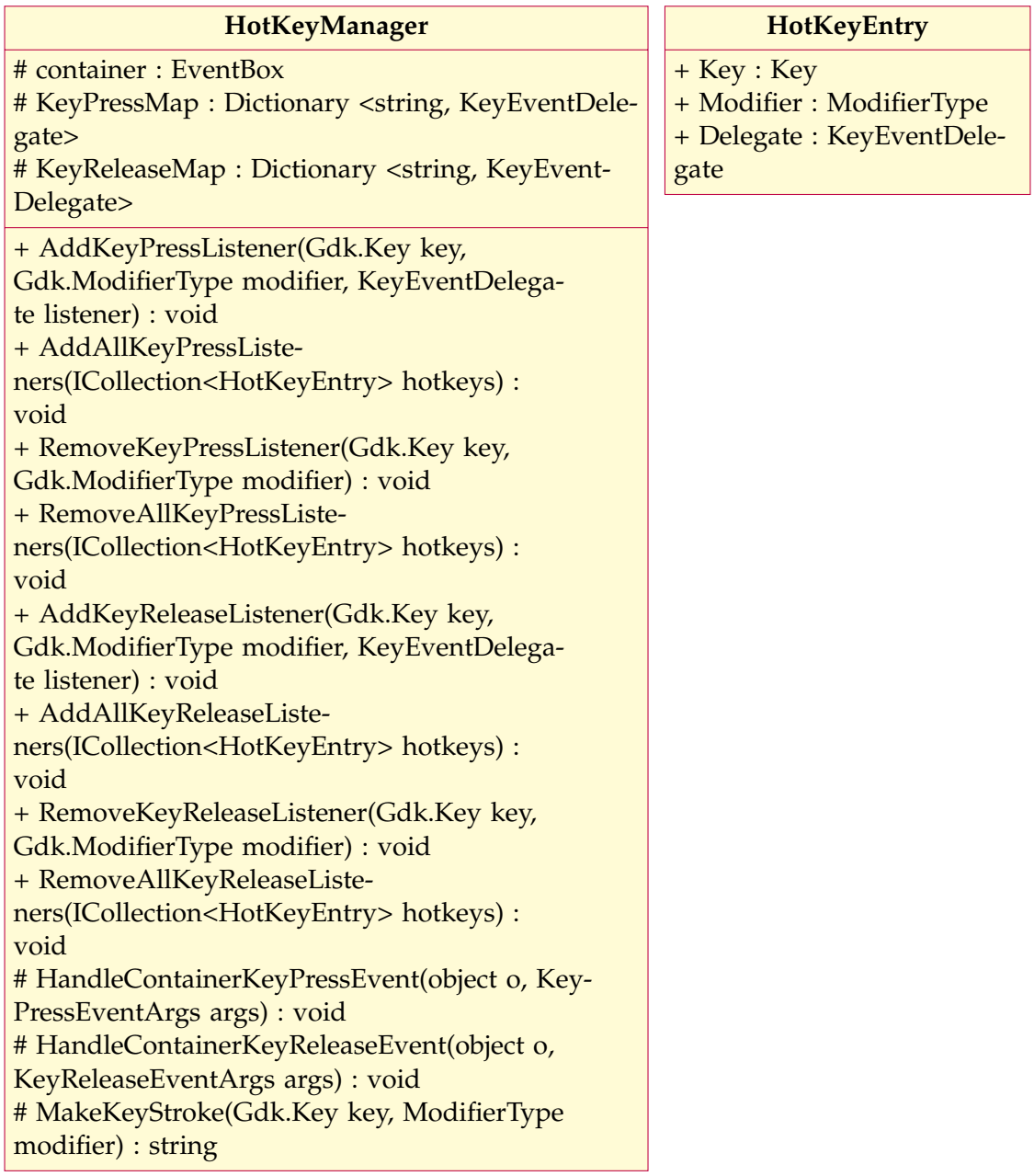

Abbildung 4.5: Klasse HotKeyManager und Struktur HotKeyEntry

Tabelle 4.3: Struktur der Datei mit globalen Einstellungen

Parameter	Bezeichnung
[UIItems]	Section mit Einstellungen
[ItemName]	Für den Entwickler verständliche Bezeichnung der Komponente
Id	Eindeutige Bezeichnung der Komponente. Muss eindeutig sein
Label	Name, der auf der Dock-Leiste dargestellt wird
RobotID	Liste der Roboter, für die diese Komponente zuständig isr, oder 'ALL'
ClassName	Vollständiger Name der Komponentenklasse
UpdateInterval	Zeitintervall zwischen zwei Aktualisierungen in ms.
UseOwnUpdater	Boolescher Wert, der zeigt, ob diese Komponente eine eigene Methode zur Aktualisierung besitzt
[Subscribe]	Section mit den abonnierten Topics
TopicName	Name des ROS-Topics
DelegateName	Name des Delegate-Property der Komponente
[!Subscribe]	Ende der Subscribe Section
[Advertise]	Section mit den angebotenen Topics
TopicName	Name des ROS-Topics
DelegateName	Name des Delegate-Property der Komponente
[!Advertise]	Ende der Advertise Section
[!ItemName]	Ende der Komponente Section
[!UIItems]	Ende der UIItems Section

gen für ItemUpdater und die Liste der RoboterID, die diese Komponente bearbeiten muss, gelesen. Wenn die Komponente für alle möglichen Roboter zuständig ist, sollte dieses Feld den `'all'` Text enthalten, ansonsten stehen dort mit Komma getrennte Zahlen, die die ID's der Roboter bezeichnen.

Der letzte Schritt ist die Verbindung dieser Komponente mit dem Proxy. Hier werden die in den Sections `Subscribe` und `Advertise` aufgelistete Delegaten der Komponente mit den entsprechenden Methoden des Proxies initialisiert.

4.1.7 Item Updater

Diese Komponente dient zur regulären Aktualisierung aller Komponenten. Damit die Anwendung während der Aktualisierung weiter uneingeschränkt benutzt werden kann, läuft

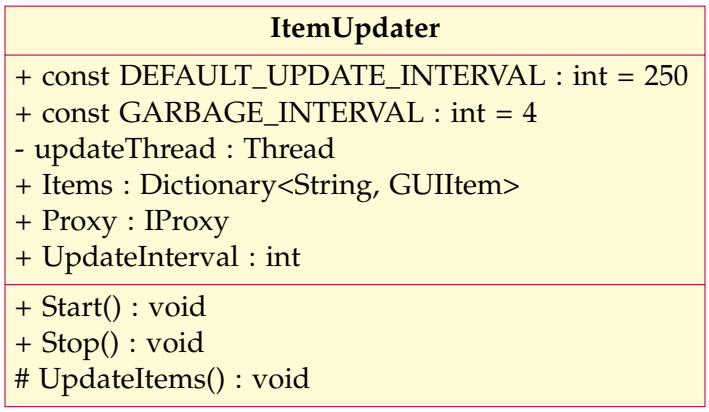

Abbildung 4.6: Klasse ItemUpdater

diese Komponente in einem anderen Thread. Aus diesem Grund müsste die Komponente die GTK+ Threadsicherheit sich selbst gewährleisten. Außerdem werden einzelne Aktualisierungen mittels „`try catch`“-Block geschützt. Der Zeitintervall zwischen zwei Aktualisierungsserien ist in einem Property `UpdateInterval` gespeichert (Defaultwert ist 250 `ms`). Nach der in der Konstante `GARBAGE_INTERVAL` vorgegebenen Anzahl der Serien wird die Methode `PerformGarbageCollection` des Proxy aufgerufen, um veraltete Daten zu löschen. Die Klasse ItemUpdater ist in der Abbildung 4.6 dargestellt.

4.1.8 Neues Fußballfeld

Das Fußballfeld sollte neu implementiert werden, um die CPU-Last zu verringern. Es wurde mittels Clutter Toolkit realisiert. Jedes Objekt auf dem Feld und das Feld selbst heißen Actors in der Clutter-Terminologie. Das Klassendiagramm des neuen Fußballfeldes mit nur den wichtigsten Elementen ist in der Abbildung 4.7 dargestellt.

Die Klasse `FootballFieldItem` ist die Hauptklasse. Sie enthält das Fußballfeld selbst, eine Sammlung aus Instanzen der RobotObject-Klasse, die Roboter und alle von ihm wahrgenommene Gegenstände implementiert und eine Instanz der ActorsFactory-Klasse. Diese Klasse enthält die Methoden zum Darstellen der Texturen, sowie verschiedene Konvertierungsmethoden für die Anpassung der Koordinaten. Zusätzlich enthält sie einen Verweis auf die Instanz der RobotCollection-Klasse, die als zentralisierte Sammlung der Robotereinstellungen dient. Diese Sammlung ist deswegen nötig, weil die Einstellungen auch in der RobotInfoBox-Komponente geändert werden können.

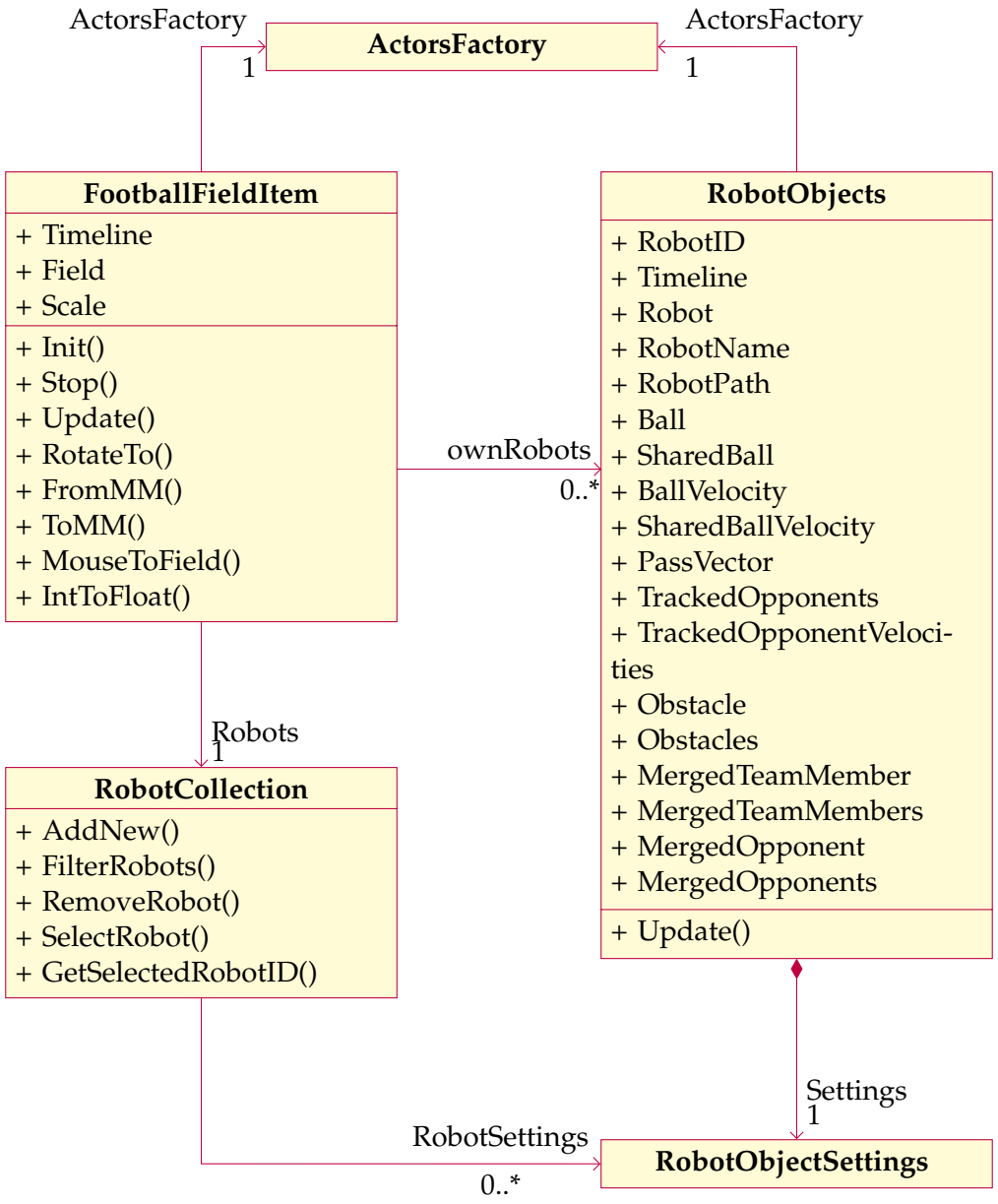

Abbildung 4.7: Klassendiagramm des neuen Fußballfeldes

Das Feld besitzt einen eigenen Aktualisierungsmechanismus. Der Grund dafür ist die Threadsicherheit von Clutter. Dieser Mechanismus ist als Timeline realisiert und dient gleichzeitig als Zeitfunktion für die Animationen. Die `Update`-Methode der Komponente funktioniert folgendermaßen: Zuerst wird geprüft, ob die Größe des Fensters geändert wurde und dementsprechend alle Feldobjekte skaliert werden. Danach wird der Fenstertitel aktualisiert, wenn der RemoteControl aktiviert ist. Anschließend wird die Information über den Roboter auf den aktuellen Stand gebracht. Alte Roboter werden gelöscht und neue hinzugefügt. Als Letztes werden die Roboter aktualisiert, was schon mittels Klasse RobotObjects gemacht wird.

Die RobotObjects-Klasse prüft, ob die verschiedenen von den Robotern wahrgenommenen Hindernisse angezeigt werden müssen, ändert ihre Positionen und passt die Geschwindigkeitsvector-Texturen an. Die genaueren Abläufe werden an einem Beispiel im Abschnitt 4.2.3 gezeigt. Jedes Objekt auf dem Feld ist eine Instanz der AnimatedCairoTexture-Klasse. Diese Klasse enthält Hilfsmethoden für deklarative Animation, was die Entwicklung erleichtert, und passt die Position AnchorPoint bei der Skalierung der Textur automatisch so an, dass sie in der Mitte der Textur liegt.

Besonders werden aber die Obstacles, MergedOpponents und MergedTeamMembers behandelt. Da der Roboter nur die Koordinaten dieser Objekte liefert, werden sie nicht einzeln, sondern als Clones dargestellt. Die Clones haben einen gemeinsamen VertexBuffer, aber eigene Position, Rotation und Skalierung.

4.2 Tutorials

In folgenden Abschnitten wird erläutert, wie die Anwendung erweitern werden kann. Hier werden nacheinander alle dafür benötigten Schritte aufgelistet, um der Proxy zu erweitern, eine neue Komponente oder ein neues Objekt zum Fußballfeld hinzuzufügen.

4.2.1 Topics

Das Einfügen eines neuen Topics ist sehr einfach. Dafür muss nur die Konfigurationsdatei des Proxy anpasst werden und das Programm mittels `make` Befehl neu kompiliert werden. Hier wird ein komplexeres Beispiel betrachtet.

Angenommen, wir möchten eine neue Komponente in die bereits vorhandene Abwendung integrieren. Diese neue Komponente sollte Nachrichten aus dem Topic `NewTopic` mit dem Typ `TopicType` erhalten. Diese Nachrichten stammen aus einem ROS-Paket `NewComponent`. Für die Integration werden folgende Schritte benötigt:

1. In der Datei `manifest.xml` das benötigte Paket hinzufügen.
   `<depend package="NewComponent"/>`
2. In der Datei `CMakeLists.txt` neue Nachrichten in der Liste hinzufügen.
   `rosbuild_gen_cs_msg(... NewComponent/)`
3. In der Proxy-Konfigurationsdatei im Subscribe-Abschnitt neue Section hinzufügen.

```
[NewTopic]
 Topic=NewComponent/NewTopic
 MessageType=RosCS.NewComponent.NewTopic
 UseFilter=True
 TimeOutInterval=5000
[!NewTopic]
```

4. Die Anwendung mittels `make` Befehl neu kompilieren.

4.2.2 Komponenten

Wenn die neue Komponente Daten aus noch nicht im Proxy vorhandenen Topic benötigt, sollte zuerst das Topic in den Proxy hinzugefügt werden. Somit werden alle benötigte Delegatentypen generiert, was die Codevervollständigung erlaubt.

Um eine neue Komponente zu der Anwendung hinzuzufügen, werden folgende Schritte benötigt:

1. Implementieren neuer Klasse, Unterklasse der Klasse UIItem, die diese Komponente beschreibt.
2. In der Datei `UIItems.conf` eine neue Section mit der Komponente und den Proxyverbindungen hinzufügen.

```
[NewComponent]
 Id = new_component_item
```

```
 Label = New Componet
 RobotID = all
 ClassName = NameSpace.ComponentClass,AssemplyName
 UpdateInterval = 250
 UseOwnUpdater = false
 [Subscribe]
  TopicName = NewTopicName
  DelegateName = GetDataFromNewTopic
 [!Subscribe]
[!NewComponent]
```

3. Die Anwendung mittels `make` Befehls neu kompilieren.

4.2.3 Objekte im Fußballfeld

Das Einfügen des neuen Objekts in Fußballfeld ist ein bisschen komplizierter, als vorher beschriebener Ablauf, weil es alle diese Schritte benötigen könnte. Im allgemeinen werden folgende Schritte benötigt:

- Einfügen eines neuen Topics in Proxy falls notwendig.
- Einfügen eines neuen Topics in Fußballfeld-Section der `UIItems.conf` Datei falls notwendig.
- Implementieren eine Methode in der Klasse `ActorsFactory`, die für den gegebenen Vergrößerungsfaktor die Größe der für den Objekt benötigten Textur liefert.
- Implementieren eine Methode in dieselbe Klasse, die eine Textur für diesen Objekt zeichnet.
- Anlegen von einer Instanz der Klasse `AnimatedCairoTexture`, die ein neues Objekt enthalten wird, in der Klasse `RobotObjects`.
- Anpassen der Methoden `CreateObjects`, die den Objekt initialisiert und den Actor zu dem Fußballfeld hinzufügt, `HideAllObjects`, die alle Objekte versteckt, `RemoveAllObjects`, die alle Objekte aus der Feld entfernt und wenn nötig `ScaleObjects`, die die Objekte skaliert.

- Anpassen der Methode `Update`. Hier sollten die neuen Daten aus dem Topic gelesen und Position, Rotation oder weitere Eigenschaften des Objekts geändert werden.
- Anpassen der Klasse `RobotObjectSettings` falls notwendig, um die Einstellungen für diesen Objekt hinzuzufügen, und die Defaultwerte in der Konfigurationsdatei.

5 Zusammenfassung

Seit dem Jahr 2006 nimmt die Universität Kassel an dem RoboCup teil. Während des Spiels generieren die Fußballroboter große Mengen an Daten, die mittels Anwendung LebtClient visualisiert wurden. Diese Anwendung wurde im Laufe des Jahres von mehreren Leuten entwickelt. Neue Funktionalität wurde oft unsauber implementiert und integriert, was dazu führte, dass das LebtClient sich seit einigen Jahren sehr schwer zu erweitern lässt und eine hohe CPU-Last zeigt.

Im Rahmen meines Projektes wurde ein effizientes und generisches Visualisierungswerkzeug entwickelt, das LebtClient modernisiert und auf Basis des entwickelten Werkzeugs als neues Paket LebtMonitor implementiert. Die grundlegenden Komponenten des Werkzeugs (Proxy, MenuGenerator, ItemLoader und so weiter) sind generisch aufgebaut und als ein selbständiges Paket namens `UniversalGUI` angelegt, was die Entwicklung neuer Anwendungen für Visualisierung der Daten auf Basis dieses Rahmenwerks erlaubt.

Die Ziele der Modernisierung wurden erreicht. Die Anwendung ist konfigurierbar, leicht erweiterbar und wartbar geworden. Die CPU-Last auf einem `Core2Duo E8400` Prozessor ist von 15% auf 8% gesunken.

Das Rahmenwerk bietet viele Möglichkeiten für Weiterentwicklung und Modernisierung. Zum Beispiel könnten alle Komponenten der Anwendung in zwei Klassen geteilt werden: sichtbare und unsichtbare. Sichtbare sind Hauptmenü, Statusleiste, Dock, Fußballfeld und so weiter. Unsichtbare sind Einstellungsmanager, Proxy. Der ItemLoader kann so erweitert werden, dass er alle Komponenten lädt und sie anschließend nach der topologischen Sortierung initialisiert. Dafür ist noch die Erweiterung einiger Konfigurationsdateien um die Liste erforderlicher Komponenten nötig. Somit werden die einzelnen Komponenten mehr voneinander unabhängig und verfolgen die Plugin-Philosophie.

Literaturverzeichnis

[1] The clutter cookbook. Webseite.
http://docs.clutter-project.org/docs/clutter-cookbook/1.0/.

[2] Cmake 2.8 docs. Webseite.
http://www.cmake.org/cmake/help/cmake-2-8-docs.html.

[3] Gtk+ 2 reference manual. Webseite.
http://developer.gnome.org/gtk/.

[4] Latex-kompendium. Webseite.
http://de.wikibooks.org/wiki/LaTeX-Kompendium.

[5] Ros wiki. Webseite.
http://www.ros.org/wiki/.

[6] Stackoverflow q&a. Webseite.
http://stackoverflow.com/.