# Ein Konfigurierbares World-Interface zur Kopplung von KI-Methoden an Interaktive Echtzeitsysteme

Dennis Wiebusch*, Marc Erich Latoschik*, Henrik Tramberend†

*Intelligent Graphics Group
Universität Bayreuth
95447 Bayreuth
Tel.: +49 (0)921 / 55 - 7601
dennis.wiebusch@uni-bayreuth.de
marc.latoschik@uni-bayreuth.de

† FB Informatik und Medien
Beuth Hochschule für Technik Berlin
13353 Berlin
Tel.: +49 (0)30 / 4504 - 5115
tramberend@beuth-hochschule.de

**Zusammenfassung:** Vorgestellt wird ein konfigurierbares World-Interface für die Kopplung von KI-Verfahren in eine Middlewareplattform für interaktive Echtzeitsysteme der VR, AR und MR. Relevante Ereignisse, deren semantische Repräsentation, sowie die entsprechenden Reaktionen auf die Ereignisse lassen sich zur Laufzeit definieren und für die jeweiligen Anforderungen konfigurieren. Dadurch wird das System auf verschiedene Anwendungskontexte und konkrete Middlewarefunktionalitäten anpassbar. Der Ansatz wird am Beispiel der Kopplung eines Regel-basierten Produktionssystems zur Entwicklung einer Anwendungslogik für eine kleines Computerspiel demonstriert.

**Stichworte:** World-Interface, Intelligente Virtuelle Umgebungen, KI & VR

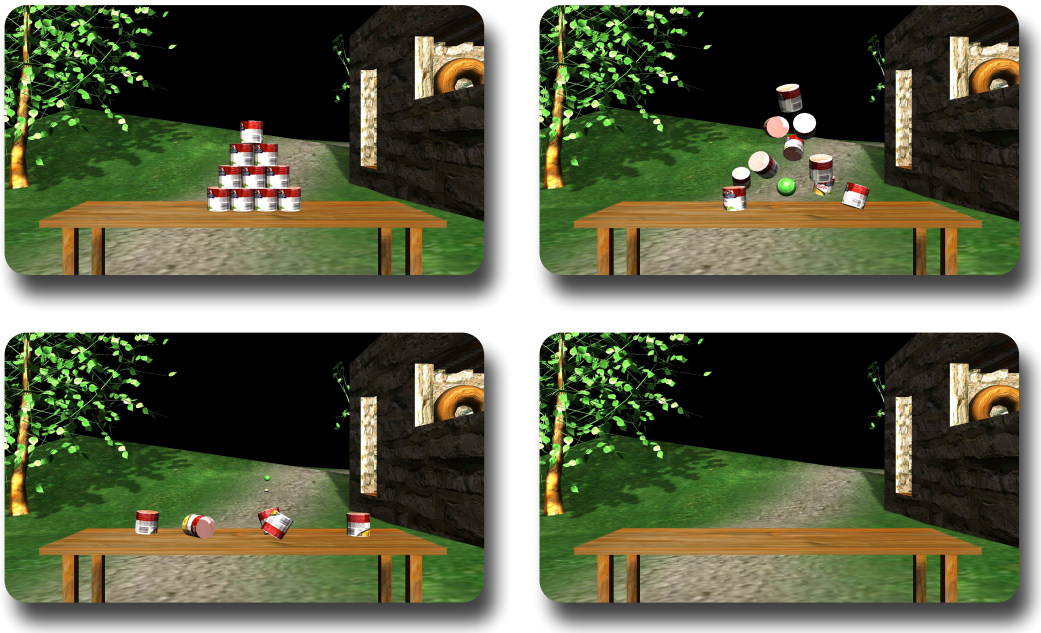

Abbildung 1: *Vier Phasen der Dosenwurf-Simulation: Initialzustand, Benutzerinteraktion, physikalische Simulation und Spielende*

# 1 Einleitung

Die Kopplung zwischen Simulationskomponenten einer RIS-Middleware (RIS: Realtime Interactive Systems) und Verfahren der Künstlichen Intelligenz (KI) stellt besondere softwaretechnische Anforderungen an Systemarchitekturen. Konzeptionell bestehen zwischen KI- und Simulationskomponenten starke Konsitenzabhängigkeiten auf Daten- und Prozessebene. Auf der Datenebene ist dies etwa die Beziehung zwischen der Geometrie der dargestellten Szene und einer existierenden Graphrepräsentation eines Suchverfahrens. Auf der Prozessebene betrifft es zum Beispiel die Einkopplung von geometrischen Wahrnehmungsfunktionen, etwa einer *Line-of-Sight* Berechnung für die Interpretation von Benutzereingaben oder die Realisierung der Eingabe einer Entscheidungsfindung eines Agenten.

In der Computerspielentwicklung hat sich der Begriff des *World-Interface* für die Schnittstelle zwischen der internen Datenrepräsentation sowie den die Simulation realisierenden Softwarekomponenten und einer von der technischen Realisierung abstrahierenden Vermittlungsschicht etabliert [MF09]. Wir betrachten insbesondere vier verschiedene Operationen als zentral für ein World-Interface:

1. **Benachrichtigung über Änderungsereignisse** *relevanter* Simulationszustände.

2. **Ausführen von Aktionen** zur Änderung *relevanter* Simulationszustände.

3. **Verarbeitung von Anfragen** an *relevante* und beliebige Simulationszustände.

4. **Konfiguration** *relevanter* Ereignisse und möglicher Aktionen.

Ein zentraler Aspekt bei dem hier vorgestellten Ansatz ist Operation 4: Konfiguration. Das erste Problem: Für eine RIS-Middleware kann im Gegensatz zu vielen Spieleengines im Vorfeld nicht definiert werden, welche Zustandsänderungen konkret relevant sind. Ob und für welche Objekte etwa ein Kollisionsereignis registriert ist hängt zum einen von der jeweiligen Anwendung ab, zum anderen natürlich von den verfügbaren Komponenten (hier einer Kollisionserkennung). Das zweite Problem: Viele symbolische KI-Verfahren benötigen eine semantische Repräsentation der Simulationsobjekte (der *Entitäten*), um etwa Eigenschaften und Beziehungen modellierbar zu machen.

Die Besonderheit des im Folgenden vorgestellten Ansatzes besteht darin, je nach Anwendungskontext und Konfiguration der RIS-Middleware sowohl die relevanten Ereignisse, deren semantische Repräsentation, als auch die entsprechenden Reaktionen dynamisch konfigurieren zu können. Als Zielarchitektur kommt dabei eine in der Entwicklung befindliche RIS-Middleware (Arbeitstitel: Simulator I) zum Einsatz. Die Tragfähigkeit des Ansatzes wird am Beispiel der Entwicklung und Kopplung eines Regelinterpreters für die Definition einer Anwendungslogik im Kontext eines einfachen Spiels demonstriert.

### 1.1 Beispielanwendung

Bei dem zur Demonstration des Ansatzes verwendeten Spiel handelt es sich um eine Dosenwurf-Simulation. In dieser koppelt das hier vorgestellte World-Interface eine Rendering-Komponente, eine Physik-Komponente, den entwickelten Regelinterpreter sowie eine Eingabekomponente zur Einbindung verschiedener Eingabegeräte (hier 6-DOF Tracking und Wiimote). Der Regelinterpreter übernimmt die Verwaltung der unterschiedlichen Simulationszustände, von denen vier in Abbildung 1 visualisiert sind.

Das Spiel kann in einer CAVE oder als Desktopapplikation dargestellt werden. In letzterem Fall wird statt dem Tracking- eine Maus-/Keyboard-Eingabekomponente angebunden. Beide sind aufgrund der verwendeten Middlewarearchitektur sowie der Funktionalität des präsentierten World-Interface problemlos austauschbar.

## 2 Stand der Forschung

Intelligente Virtuelle Umgebungen (*Intelligent Virtual Environments – IVEs*) [AL00] verbinden KI-Verfahren mit interaktiven Systemen der Virtual, Augmented und Mixed Reality (VR, AR und MR). Beispielszenarien sind neuartige multimodale Mensch-Maschine Schnittstellen, wissensbasierte Simulation pseudophysikalischen Objektverhaltens, intelligente Virtuelle Agenten mit autonomen Wahrnehmungs-, Entscheidungs- und Aktionsmöglichkeiten sowie der technisch eng mit der VR/AR/MR verwandte Bereich der Computerspiele.

Ein erster wesentlicher Aspekt dieser Systeme ist eine semantische Repräsentation der Anwendungs- und Szeneninhalte [Sot97, SA02, PS03, LS03, KCM06, LC07], um einen wissensbasierten Zugriff auf die Entitäten der Szene zu ermöglichen. Diese stellt die Voraussetzung für eine ganze Reihe symbolischer KI-Methoden dar.

Ein zweiter zentraler Aspekt ist die Kopplung zwischen KI-Verfahren und der RIS-Middleware [SA02, LS03, KCM06]. Viele Systeme stellen zur technischen Realisation eines World-Interface proprietäre Eventsysteme zur Verfügung [BLRS98, Tra99, Hag96, SF04, AGL+04]. Die Eventdistribution folgt oft einem vorgegebenen *Execution-Scheme*. Objektorientierte Ableitungs- oder Callback-Mechanismen realisieren dabei die reaktive Anwendungslogik. Die zeitliche Ausführung unterliegt in diesem Fall dem zugrundeliegenden Prozessfluss innerhalb des Ausführungsschemas. Objektmodell und Prozessfluss sind dabei eng gekoppelt. Das Zusammenspiel mit etwaigen anderen Komponenten muss im Vorfeld implementiert werden. In existierenden Spieleengines, wie der Unreal Engine 3 oder der Quake3 Engine, sind sowohl die verfügbaren Komponenten (und damit deren Funktionalität) als auch die verfügbaren World-Interfaces vordefiniert und nur in geringem Maße konfigurierbar.

Die existierenden Ansätze zur Kopplung von KI-Verfahren und einer RIS-Middleware erweisen sich unter den Aspekten aktueller Anforderungen an Parallelisierung und Softwarequalitätsmerkmale wie Erweiterbarkeit und Übertragbarkeit von Verfahren als ungenügend. Zentrale Weltzustands- und Entitätsmodelle mit fest daran geknüpften *Execution-Threads* erzwingen entweder weitreichende Zugriffsschutzmethoden oder eine Serialisierung. Beides

ist nachteilig für eine optimale Ausnutzung paralleler Architekturen und – im ersten Fall – mit einer deutlichen Erhöhung der Codekomplexität verbunden.

# 3 Kopplung des World-Interface

Als Grundlage zum Verständnis des World-Interface-Entwurfs werden zuerst die dabei wichtigen Architekturprinzipien und -elemente der eingesetzten RIS-Middleware erläutert. Der Entwurf folgt dem Paradigma der Kopplungsminimierung und Kohäsionsmaximierung.

## 3.1 Simulator I Architekurprinzipien

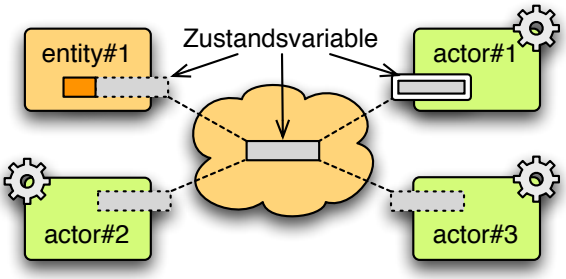


Abbildung 2: *Eine semantisch entity#1 zugeordnete Zustandsvariable wird von actor#1 kontrolliert. actor#2 und #3 greifen über ein asynchrones Nachrichtensystem auf den Zustand zu.*

Die Simulator I Middlewarearchitektur zerlegt den Weltzustand in einzelne Zustandsvariablen. Verschiedene Execution-Threads werden über das *Actor-Model* realisiert. Konzeptionell können Zustandsvariablen bestimmten Entitäten (*Entities*) zugeordnet werden, um so ein objektzentriertes Weltmodell zu erhalten. In Abbildung 2 wird die Zugriffskontrolle einer mit entity#1 assoziierten Zustandsvariable über actor#1 realisiert. Lese/Schreibzugriffe von anderen Aktoren und damit Execution-Threads, wie hier den Aktoren actor#2 und actor#3, werden transparent über ein asynchrones Nachrichtensystem an den kontrollierenden actor#1 delegiert. Ein Event-basierter Zugriff erlaubt darüber hinaus die Bereitstellung reaktiven Anwendungsverhaltens. Als Implementierungssprache kommt Scala zum Einsatz.

## 3.2 Registrierung von Architekturelementen

Um Ereignisse, Aktionen, Anfragen und Konfigurationen als Hauptoperationen zu realisieren, stellt das World-Interface eine lose koppelnde Registraturkomponente zur Verfügung. Diese ist, dem Komponentenmodell der Middleware folgend, ebenfalls als Aktor realisiert (siehe Abbildung 3). Die Registratur ermöglicht das Anmelden sämtlicher beteiligter Architekturelemente. Dazu zählen die Zustandsvariablen, die Entitäten sowie die Aktoren. Im letzteren Fall natürlich auch die als Aktor realisierten Simulationskomponenten (Graphikrenderer, Physiksimulation, KI, etc).

Die Registratur am World-Interface etabliert eine Assoziation der Architekturelemente mit symbolischen Bezeichnern. Dies vermeidet die Notwendigkeit einer Benutzung absoluter Referenzen auf Elemente der Anwendung und entkoppelt Konzept von Implementierung. Unter Angabe desselben Symbols mit dem die Registrierung vorgenommen wurde, kann ein Objekt zu einem späteren Zeitpunkt an einer beliebigen Stelle des Programms und in einem beliebigen Execution-Thread referenziert werden. Die Registrierung von Simulationskomponenten unterstützt die notwendigen Initialisierungsoperationen. Sie stellt darüber hinaus die Grundlage für spätere dynamische Komponentenkopplungen auf Basis einer semantischen Funktionsbeschreibung dar.

Die Adressierung von Architekturelementen erfolgt daraufhin über einen symbolischen Zugriff. Intern wird Eindeutigkeit für alle Elemente – auch über Prozessgrenzen hinweg – durch die Verwendung von so genannten *UUIDs* (Universally Unique Identifiers) gewährleistet. Die zusätzliche Indirektion ist dabei zu vernachlässigen, da es Nutzern dieses Service freisteht, diesen nur für initiale Zuordnungen einzusetzen und anschließend Zugriffe mit Hilfe von Caching-Funktionen zu realisieren. Die Verwendung von Symbolen dient jedoch einem zusätzlichen Zweck: Zum einen unterstützen menschenlesbare eindeutige Benennungen den Entwicklungsprozess in verteilten Szenarien und ermöglichen Objektidentifikationen zur Laufzeit. Zum anderen ermöglichen sie eine Kopplung symbolischer KI-Verfahren, für welche die Lesbarkeit für den Aufbau und die Wartung der Wissensbasen notwendig ist.

### 3.3 Eventhandling

Während Simulationskomponenten meist statische Einheiten einer Anwendung darstellen, bietet der Einsatz von dynamisch verwendbaren Eventhandlern die Möglichkeit auf die im Simulationskontext relevanten Ereignisse zu reagieren. Die Simulator I Architektur erlaubt die Wertänderung einer Zustandsvariable als Ereignis aufzufassen, bei dessen Auftreten ein registrierter Eventhandler aufgerufen wird. Die Registrierung erfolgt dabei unter Angabe des Eventnamens, welcher wiederum mit dem Bezeichner einer Zustandsvariablen verknüpft wird. Innerhalb eines solchen Handlers können Berechnungen ausgeführt, Zustandsvariablen nachrichtenbasiert Werte zugewiesen und Nachrichten an Aktoren versendet werden. Um die Wertänderungen mehrerer Zustandsvariablen mit demselben Eventhandler verarbeiten zu können, ist die Indirektion über einen Eventnamen notwendig. Listing 1 demonstriert die Registrierung und Definition eines solchen Handlers am Beispiel des World-Interface.

```
1 WorldInterface.addValueChangeTrigger(variable_name, entity_name, event_name)
2
3 WorldInterface.registerHandler({
4   case WorldInterfaceEvent(event_name, (variable_name, entity_name, value))
5     => { /* do event handling here */ }
6 })
```

Listing 1: *Definition eines Eventhandlers für das Überwachen einer Zustandsvariable*

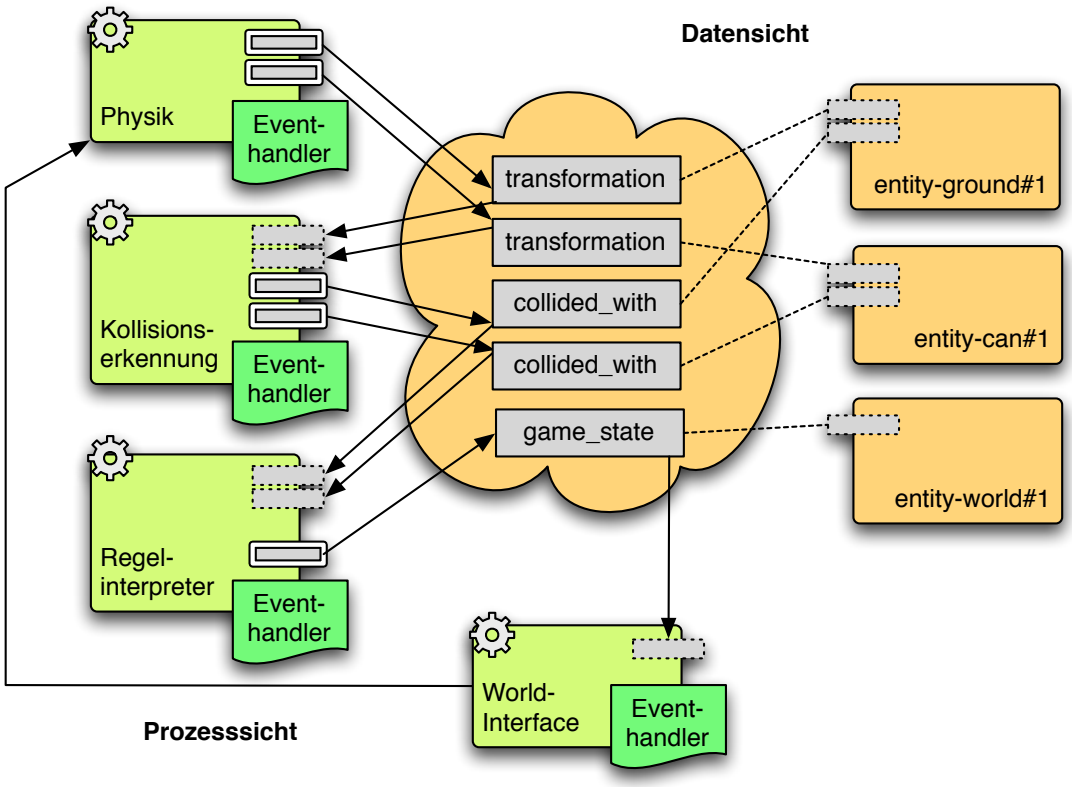


Abbildung 3: *Vereinfacht dargestellte Architektur der einleitend beschriebenen Anwendung. Der abgebildete Ausschnitt stellt die Verarbeitung der Kollision einer Dose mit dem Boden, die dazu notwendigen Anwendungselemente sowie deren vom Entwickler gewählten Bezeichner dar.*

Die Verarbeitung der Events erfolgt im Execution-Thread des jeweiligen Aktors, so dass die Eventdistribution keinem vorgegebenen Schema unterliegt. Statt einem allgemeinen Prozessfluss unterworfen zu sein werden Events zur Zeit ihres Auftretens verarbeitet.

### 3.4 Konfiguration

Anders als in den meisten bekannten Systemen kann das Zusammenspiel der verwendeten Komponenten der vorgestellten Architektur dynamisch festgelegt werden. Hierzu müssen lediglich die von den Komponenten verwendeten Bezeichner einheitlich definiert werden, damit die Registratur etwaiger Objekte am World-Interface möglich ist. Die Kenntnis dieser Bezeichner ist ausreichend, um eine Kommunikation der Komponenten untereinander zu gewährleisten, da diese hauptsächlich über die Wertänderungen von Zustandsvariablen, selten auch direkt über das Versenden von Nachrichten erfolgt.

Zur Veranschaulichung des erläuterten Konzepts zeigt Abbildung 3 den Ausschnitt einer möglichen Konfiguration der einleitend vorgestellten Anwendung. Die auf Seite der Datensicht abgebildeten Entitäten sind dabei über die abgebildeten Bezeichner am World-Interface

registriert. Die in der Prozesssicht dargestellten Simulationskomponenten können, unter Angabe der symbolischen Bezeichner der zentral dargestellten Zustandsvariablen, Anfragen an den Simulationszustand stellen. Konkret bedeutet dies, dass die Komponenten initial die von ihnen kontrollierten, mit Entitäten assoziierten Zustandsvariablen unter Verwendung der abgebildeten Namen am World-Interface registrieren. Daraufhin melden sich ihre Aktoren für Benachrichtigungen über Wertänderungen der für sie relevanten Variablen an. Diese werden dabei über die jeweiligen Bezeichner referenziert. Des Weiteren wird ein Eventhandler für Wertänderungen der Zustandsvariable `game_state` angemeldet, welcher die Physikkomponente im Falle der Beendigung des Spiels ist durch den Versand einer Nachricht notifiziert. Der sich aus dieser Konfiguration ergebende Informationsfluss (schwarze Pfeile) ist mittels asynchronem Nachrichtenversand realisiert. Die lose Kopplung der Komponenten ermöglicht hier beliebige Ausführungsschemata.

Die Flexibilität, welche das präsentierte World-Interface hinsichtlich der Anwendungskonfiguration bietet, wird durch eine Umstellung auf folgende alternative, in sämtlichen nachfolgend genannten Beispielen verwendete, Systemkonfiguration illustriert: Unter der Annahme, dass keine kollisionserkennende Komponente zur Verfügung steht, muss die notwendige Anwendungslogik auf Basis alternativer Ereignisse realisiert werden. Eine Analyse der verfügbaren Komponenten identifiziert die Physikkomponente als Quelle von Positionsänderungen. Diese werden jetzt als die signifikanten Zustände verwendet, indem Eventhandler registriert werden, welche die Position der Dosen beobachten. Befindet sich eine Dose unterhalb der Ebene der Tischplatte, wird dem Regelinterpreter mitgeteilt, dass diese Dose gefallen ist. Weitere Anpassung der Anwendung entfallen. Diese Beispielkonfiguration ist hinsichtlich der Performanz natürlich nachteilig. Entscheidend ist hier, dass die Anwendungsfunktionalität erhalten bleibt obwohl sich die Systemkonfiguration wechselnden Voraussetzungen anpassen muss.

# 4 Integration von Objektsemantik

Die Referenzierung von Architekturelementen wird mit Hilfe von Symbolen durch vorherige Registrierung ermöglicht. Durch die Verwendung von Objektsemantik reflektierenden Bezeichnern stellt sich die, von KI-Komponenten benötigte, semantische Repräsentationsschicht als direkt in das World Interface integriert dar. Da die Wahl der Bezeichner den Anwendungs- und Komponentenentwicklern freigestellt ist, kann deren Eindeutigkeit auf technischer Ebene nicht ohne Weiteres garantiert werden. Aus diesem Grund wird im Folgenden eine Richtlinie für die Auswahl von Bezeichnern vorgeschlagen.

## 4.1 Wahl eines Benennungsschemas

Auf dem Gebiet der Wissensrepräsentation, speziell im Bereich der Beschreibungslogiken wird für gewöhnlich zwischen terminologischem ($TBox$) und assertionalem ($ABox$) Wissen unterschieden. Konzeptionell stellen Entitäten und Aktoren ABox-Elemente dar und müssen

daher zwecks Referenzierung anwendungsweit eindeutige Bezeichner erhalten. Um eine einheitliche und nachvollziehbare Benennung zu gewährleisten wird folgendes Schema zur Wahl ihrer Bezeichner angewendet:

$$Bezeichner := Typname - Instanzbezeichnung \# Nummerierung$$

Im Gegensatz dazu erfolgt der Zugriff auf Zustandsvariablen über die sie referenzierenden Entitäten, weshalb für ihre Bezeichner nur entitätsweite Eindeutigkeit erforderlich ist. Aufgrund der Containerfunktionalität von Entitäten ist es sinnvoll, terminologische Bezeichner zu wählen, welche die Semantik des enthaltenen Wertes reflektieren.

Listing 2 zeigt zur Veranschaulichung das folgende, aus dem vorgestellten Spiel entnommene Beispiel[1]: Ein Eventhandler sendet dem als `actor-rules#1` registrierten Aktor des Regelinterpreters die Nachricht, dass die Entität `entity-can#1` als gefallen zu markieren ist, da ihre, in der Zustandsvariable `transformation` gespeicherte Position, als niedriger als die Tischposition festgestellt wurde.

```
1 WorldInterface.registerEntity('entity-can#1, canEntity)
2 WorldInterface.addValueChangeTrigger('transformation, 'entity-can#1,
3                                      'canDownEvent);
4 WorldInterface.registerHandler({
5   case WorldInterfaceEvent('canDownEvent, (variable, entity, transform) )
6     if ( transform isLowerThan tableTop ) =>
7       WorldInterface.forwardMsg('actor-rules#1, Fact('can, entity, "down"))
8 })
```

Listing 2: *Eventhandler für das Überwachen der Dosenpositionen. Im Vorfeld wurde die Zustandsvariable transformation in der Entität canEntity durch die Physik-Engine angelegt.*

### 4.2 Semantische Repräsentationsschicht

Die in das World-Interface durch die Wahl Semantik reflektierender Bezeichner eingebettete semantische Repräsentationsschicht ermöglicht eine direkte Kopplung zwischen KI-Verfahren und RIS-Middleware. Somit ist eine künstliche hergestellte Verbindung von KI-Komponenten und Anwendung nicht erforderlich. Gleichzeitig werden damit mögliche Inkonsistenzen zwischen Datenebene und Wissensbasis ausgeschlossen.

Längerfristig betrachtet ist in diesem Zusammenhang die Erstellung und Nutzung einer Ontologie sinnvoll, um eine einheitliche Benennung von Anwendungselementen forcieren und Benennungsfehler gegebenenfalls frühzeitig erkennen zu können. Außerdem ergeben sich durch die Verwendung einer Ontologie weitere Möglichkeiten hinsichtlich der Anbindung von KI-Komponenten, da nicht nur semantische Funktionsbezeichner sondern auch deren Beziehungen untereinander reflektiert werden können.

[1]Mit ' begonnene Zeichenketten bezeichnen in diesem und den folgenden Listings Symbole. Die in den Code-Beispielen verwendete Scala-Version ist 2.8.0.

# 5 Kopplung des Regelinterpreters

Die Kopplung einer Komponente an das World-Interface wird im Folgenden anhand der Implementierung und Anbindung eines Regelinterpreters erläutert. In seiner Eigenschaft als KI-Komponente stellt dieser ebenso ein Beispiel für die Verwendung der semantischen Repräsentationsschicht dar.

## 5.1 Funktionsweise und Implementierungsdetails

Prinzipiell generiert der Regelinterpreter Fakten, indem er Regeln auf eine Wissensbasis anwendet. Des Weiteren stellt er zu invalidierende Fakten fest und aktualisiert anschließend die Wissensbasis.

Eine ein Faktum repräsentierende Instanz der Klasse `Fact` ist durch einen Bezeichner (Subjekt) sowie eine Menge von Variablen beliebigen Typs (Prädikaten) definiert. Jeder Regel sind die Bezeichner der Fakten, auf welche sie angewendet werden kann bekannt. Drei verschiedene Regelklassen (Simple-, Normal- und ComplexRules) erlauben dem Entwickler ein unterschiedliches Maß an Einflussnahme auf die Wissensbasis.

Listing 3 zeigt eine, der einleitend beschriebenen Anwendung entnommene, Regel. Diese überprüft die Wissensbasis auf die Existenz der Fakten `cans_down` und `can_count`. Bei Gleichheit derer Prädikate wird die Zustandsvariable `state` der Entität `entity-game#1` auf den Wert „ended“ gesetzt. Es ist zu sehen, dass sich die semantische Repräsentationsschicht sich trotz ihrer großen Flexibilität nahezu nahtlos in das Programm einfügt.

```
1 class GameEndRule extends ComplexRule(List('cans_down, 'can_count)){
2   val apply = {
3     case List(Fact('cans_down, numDown : Int), Fact('can_count, numMax : Int))
4       if ( numDown equals numMax ) => {
5         WorldInterface.setStateValue('state, 'entity-game#1, "ended")
6         /* effect on KB: */ noEffect
7       }
8   }
9 }
```

Listing 3: *Definition einer Regel zur Feststellung des Spielzustandes*

Bei Anwendung einer Regel werden aus der zugrundeliegenden Wissensbasis sämtliche Kombinationen von Fakten generiert, welche diese überprüfen kann. Permutationen dieser Menge werden nicht betrachtet. Die Zahl der zu überprüfenden Kombinationen reduziert sich hierdurch in der Regel von $n^k$ auf $\binom{n-m}{k}$, wobei $n$ die Anzahl der Fakten in der Wissensbasis, $m$ die Kardinalität der Menge der nicht betrachteten Fakten und $k$ die Anzahl der Vorbedingungen der Regel ist. Trifft die Regel zu, wird die Wissensbasis der Regeldefinition entsprechend verändert.

### 5.2 Komponentenintegration am Beispiel des Regelinterpreters

Die Anbindung einer Simulationskomponente an eine existierende Anwendung, ist unter Verwendung der beschriebenen Methoden ohne großen Aufwand möglich. Der Regelinterpreter stellt eine solche Komponente dar. Ihr Aktor und beliebig viele Zustandsvariablen, mittels welcher neu erzeugte Fakten publiziert werden, werden am World-Interface registriert. Alternativ ist es dem Regelinterpreter möglich mit anderen Aktoren zu kommunizieren. Eingaben erhält er auf gleiche Weise über beobachtete Zustandsvariablen oder über registrierte Eventhandler.

```
1 WorldInterface.addValueChangeTrigger('state, 'entity-game#1, 'gameStateEvent)
2 WorldInterface.registerHandler({
3   case WorldInterfaceEvent('gameStateEvent, (variable, entity, value) )
4     if ( value equals "ended" ) =>
5       WorldInterface.forwardMsg('actor-physics#1, ResetMessage)
6 })
```

Listing 4: *Definition eines Handlers für das Überwachen der Zustandsvariable gamestate*

Die Anbindung des Regelinterpreters ist in den Listings 2 - 4 dargestellt: Die Dosen werden am World-Interface gemäß des vorgestellten Benennungsschemas registriert und ein Eventhandler an ihrer Transformationsvariable angemeldet. Sobald eine Dose unterhalb der Tischplatte registriert wird, wird dies dem Regelinterpreter per Nachricht mitgeteilt (siehe Listing 2). Dieser benutzt einen hier nicht dargestellten Mechanismus zum Registrieren der gefallenen Dosen und überprüft deren Anzahl in der `GameEndRule` aus Listing 3. Das Zusammenspiel von Regelinterpreter und Simulationszustand wird in Listing 4 demonstriert: Unter Verwendung eines Eventhandlers wird die in Zustandsvariable `state` überwacht. Die Physik-Engine wird im Falle des Spielendes von diesem benachrichtigt, so dass sie den Startzustand des Spiels wiederherstellen kann.

## 6 Fazit und Ausblick

In diesem Beitrag wurde ein World-Interface vorgestellt, welches die Entkopplung der Simulationskomponenten interaktiver Echtzeitsysteme unterstützt. Seine Verwendung ermöglicht die flexible Anpassung der Anwendungskonfiguration an verschiedene Simulationskontexte. Dies wird einerseits durch die Minimierung der Kopplung bis auf die Referenzierung auf Basis symbolischer Bezeichner erreicht, auf der anderen Seite erlaubt die Verwendung von Eventhandlern die dynamische Konfiguration von Ereignissen und Aktionen und somit die Anpassung der Anwendung an gegebene Anforderungen. Um die eindeutige Bezeichnung der Anwendungselemente zu garantieren, wurde die Verwendung von Objektsemantik reflektierenden Bezeichnern vorgeschlagen.

Der angebundene Regelinterpreter profitiert von der auf diese Weise im World-Interface eingebetteten semantischen Repräsentationsschicht, die aufgrund der frei wählbaren Bezeichner ebenfalls dem Anwendungskontext anpassbar ist. Die Implementierung eines einfachen

Spiels diente zur Demonstration der Tragfähigkeit des Ansatzes. In diesem wurde der Regelinterpreter zur Kontrolle der Simulationslogik eingesetzt.

In der aktuellen Implementierung kann es im Zuge der Wahl von Symbolen vorkommen, dass ein zweiter Entwickler ein Synonym einer semantischen Beschreibung verwendet. In diesem Fall würde die Abfrage einer Zustandsvariablen fehlschlagen, da diese unter einem anderen Symbol registriert wurde. Um dieses Problem zu vermeiden soll in Zukunft eine Ontologie zur Verfügung gestellt werden, welche den Entwickler im Symbolbenennungsprozess unterstützt.

Der hier vorgestellte Regelinterpreter stellt ein ausreichendes Maß an Funktionalität bereit, um aus dem aktuellen Weltzustand Informationen zu extrahieren und gegebenenfalls adäquate Aktionen zu veranlassen. Um eine intelligent erscheinende interaktive Umgebung zu generieren sind oft jedoch auch komplexe Anfragen an die KI-Komponente notwendig. Diese beschränken sich momentan auf das Durchsuchen der Wissensbasis nach existierenden Fakten. Anfragen wie „Gib mir eine Liste aller Dosen“ sind noch nicht möglich. Aus diesem Grund ist die Einbindung einer weiteren KI-Komponente geplant, welche diese Art von Anfragen zulässt.

## Literatur


[AGL+04] Allard, J., V. Gouranton, L. Lecointre, S. Limet, E. Melin, B. Raffin und S. Robert: *FlowVR: a Middleware for Large Scale Virtual Reality Applications.* In: *Proceedings of Euro-par 2004*, Pisa, Italia, August 2004.

[AL00] Aylett, Ruth und Michael Luck: *Applying Artificial Intelligence to Virtual Reality: Intelligent Virtual Environments.* Applied Artificial Intelligence, 14(1):3–32, 2000.

[BLRS98] Blach, Roland, Jürgen Landauer, Angela Rösch und Andreas Simon: *A Highly Flexible Virtual Reality System.* In: *Future Generation Computer Systems Special Issue on Virtual Environments*, Band 14, Seiten 167–178. Elsevier Amsterdam, 1998.

[Hag96] Hagsand, O.: *Interactive MultiUser VEs in the DIVE System.* IEEE Multimedia Magazine, 3(1), 1996.

[KCM06] Kalogerakis, E., S. Christodoulakis und N. Moumoutzis: *Coupling Ontologies with Graphics Content for Knowledge Driven Visualization.* In: *Proceedings of the IEEE VR2006*, Seiten 43–50, 2006.

[LC07] Lugrin, Jean-Luc und Marc Cavazza: *Making Sense of Virtual Environments: Action Representation, Grounding and Common Sense.* In: *Proceedings of the Intelligent User Interfaces IUI'07*, 2007.

[LS03] Latoschik, Marc Erich und Malte Schilling: *Incorporating VR Databases into AI Knowledge Representations: A Framework for Intelligent Graphics Applications.* In: *Proceedings of the Sixth International Conference on Computer Graphics and Imaging*, Seiten 79–84. IASTED, ACTA Press, 2003.

[MF09] Millington, Ian und John Funge: *Artificial Intelligence for Games.* Morgan Kaufmann Publishers, 2nd Auflage, 2009.

[PS03] Peters, Stephen und Howie Shrobe: *Using Semantic Networks for Knowledge Representation in an Intelligent Environment.* In: *PerCom '03: 1st Annual IEEE International Conference on Pervasive Computing and Communications*, Ft. Worth, TX, USA, March 2003. IEEE.

[SA02] Soto, Michel und Sébastien Allongue: *Modeling Methods for Reusable and Interoperable Virtual Entities in Multimedia Virtual Worlds.* Multimedia Tools Appl., 16(1-2):161–177, 2002.

[SF04] Steed, Anthony und Emmanuel Frécon: *Construction of Collaborative Virtual Environments.* In: Segura, Maria-Isabel Sanchez (Herausgeber): *Developing Future Interactive Systems*, Nummer ISBN 1591404126, Seiten 235–268. Idea Group, November 2004.

[Sot97] Soto, Michael: *Semantic approach of virtual worlds interoperability.* In: Capps, Michael (Herausgeber): *Proceedings of IEEE WET-ICE '97, Cambridge, MA.* IEEE Press, june 1997.

[Tra99] Tramberend, Henrik: *Avocado: A Distributed Virtual Reality Framework.* In: *IEEE Virtual Reality Conference*, Seiten 14–21, 1999.