Architektur und Umsetzung eines semantischen VR Asset Repositories

Schmidt, A.[1]; Pflüger, J.[1]; Schubotz, R.[2] ; Harth, A.[3] ;

[1] Audi AG, Ingolstadt
[2] DFKI, Saarbrücken
[3] KIT, Karlsruhe

Abstract

Wir beschreiben die Architektur und die prototypische Umsetzung eines Repositories für VR Assets.

Assets bilden die Grundbausteine modular aufgebauter VR-Szenen. Mittels eines Repositories lassen sich Assets nachhaltig verwalten und strukturiert bereitstellen.

Der Ansatz des semantischen Asset Repository bietet darüber hinaus Versionierung, Variantenmanagement, Modellierung von Kontexten, Verlinkung zu weiteren datenhaltenden Systemen und problemlose Erweiterbarkeit auf Basis einer cloudfähigen Containerarchitektur.

1 Hintergrund und Zielsetzung

Durch die Marktpräsenz preisgünstiger VR-Hardware und flexibler Autorenumgebungen wandeln sich die Einsatzszenarien von VR im industriellen Kontext grundlegend. Neben spezialisierten Anlagen, in denen oft Jahre an Forschung und Entwicklung stecken, gewinnen leichtgewichtige Arbeitsplatzlösungen auf Basis von Standardtechnologie zunehmend an Bedeutung.

In Ergänzung zu spezialisierten Visualisierungslösungen für spezifische Anwendungsszenarien, beispielsweise CAD-Visualisierung im Engineering oder fotorealistisches Rendering für Design und Vertrieb, etablieren sich verstärkt Game-Engines als flexible Autorensysteme für interaktive VR-Anwendungen.

Auf dieser Basis entstehen an vielen Stellen innovative Prototypen und „Proof of Concepts“ [18]. Um deren Übertragung in den produktiven Einsatz und einen dauerhaft wirtschaftlichen Betrieb zu ermöglichen, sind weitgehend automatisierbare Datenversorgungsprozesse und entsprechende Backendsysteme notwendig, die über eine reine Datenbereitstellung hinausgehen.

Während in den traditionellen Arbeitsfeldern strukturierte Datenversorgungsprozesse und -systeme die Regel sind, wie etwa PLM-Systeme im Engineering, fehlt dieser Ansatz in vielen der noch jungen Anwendungsfällen von Game-Engines. In der Regel werden hier noch manuell aufbereitete Daten im lokalen Filesystem verwaltet und in Form monolithischer Anwendungen verteilt. An verschiedenen Stellen redundant geleistete Arbeiten, mangelnde Quernutzung von Arbeitsergebnissen, statische Anwendungen und bei den jeweiligen Entwicklern gekapseltes Wissen sind die Folge. Das semantische AR Asset Repository adressiert diese Problemstellungen und bietet einen flexiblen Lösungsansatz auf mehreren Ebenen.

1.1 Wiederverwendbarkeit

Als Grundfunktion lassen sich Assets über das Netz im Repository ablegen und über eine Web-Oberfläche browsen und wieder herunterladen. Einmal erzeugte Assets werden so dem gesamten Entwicklerkreis im Unternehmen zugänglich gemacht. Weiterhin ermöglicht ein Versions- und Variantenmanagement die Weiterentwicklung und Anpassung einmal erstellter Assets. Zusätzlich lassen sich anhand semantischer Verknüpfungen Assets zu größeren Kontexten zusammenfassen.

1.2 Unterstützung der Entwickler

Die Verwaltung der Assets mittels eines Git-Repositories ermöglicht dem Entwickler die Handhabung von Assets mit bewährten Methoden moderner Softwareentwicklung: Ein- und Aus-checken von Assets, automatische Versionierung, Verwaltung von Entwicklungszweigen und Varianten. Die Möglichkeit, semantische Verknüpfungen bereits auf Code- oder Metadatenebene innerhalb der Assets anzulegen, erlaubt die automatisierte Modellierung und Pflege übergeordneter Kontexte. Diese Kontexte strukturieren das gesamte Asset Repository. Sie ermöglichen sowohl das Browsen des Repositories anhand semantischer Verknüpfungen als auch die Nutzung dieser Semantik innerhalb der Game-Engine, beispielsweise zu automatisierten Erstellung von Interaktionen zwischen Assets.

1.3 Dynamische VR-Szenen und Anbindung von Drittsystemen

In vielen Fällen können VR-Szenen nicht vollumfänglich in Form eines einzigen Executables abgespeichert und verteilt werden. Ein Beispiel ist die Visualisierung eine Fahrsimulationsumgebung, bei der eine ganze Region durchfahrbar sein soll. Hier müssen Daten dynamisch nachgeladen werden können. Die bewährte Methode besteht darin, diese Daten an persistenten http-URIs vorzuhalten, so dass sie seitens der Applikation bei Bedarf nachgeladen werden können. Diese Methode wird durch das semantische Asset Repository erweitert: Applikationen können durch Anfragen an das Semantik-Layer des Repositories auch neu hinzugekommene Assets identifizieren und einbinden. Zusätzliche datenhaltende Drittsysteme können ebenfalls über semantische Verknüpfungen angebunden werden. Diese Verknüpfungen sind hierbei nicht durch das Datenmodell festgelegt, sondern lassen sich dynamisch erzeugen. So kann die Visualisierung technischer Bauteile angereichert werden durch die Ergebnisse von Festigkeitsberechnungen aus einer separaten Datenbank. Liefert die Abfrage nach weiteren Verknüpfungen zu diesem Bauteil nun eine neue, zusätzliche Datenquelle, beispielsweise Bauteiltoleranzen aus einer Messdatenbank, können diese direkt angezeigt werden ohne das Visualisierungsprogramm erweitern und neu kompilieren zu müssen.

1.4 Speicherung von Kontextwissen

Die Modellierung und Pflege semantischer Kontexte sichert über die Erleichterung des Browsens von Assets und die Nutzung der Verknüpfungen innerhalb von Programmen hinaus das Wissen über die Kontexte der durch die Assets repräsentierten Entitäten. Diese Wissenskontexte lassen sich auch über die Grenzen der VR-Welt hinaus nutzen. Vor allem die Abbildungen der Verknüpfungen mit Drittsystemen beschreiben übergeordnete Informationszusammenhänge, die auch in anderen Systemen genutzt werden können. Bei dem vorher genannten Beispiel der Bauteilvisualisierung können so durch die Vernetzung mit den Daten der Festigkeitssimulation und den Toleranzmaßen automatisierte Reports generiert werden. So wird die manuelle Zusammenstellung von Daten zu Präsentationsfolien überflüssig.

2 Systematik

Im Folgenden gehen wir zuerst in 2.1 auf die atomaren Bausteine einer Applikation ein: sog. Assets, die einzelne Teile der Applikation kapseln und somit wiederver-

wendbar machen. Des Weiteren präsentieren wir in 2.2 ein Szenario im Bereich Logistiktraining (siehe Bild 1). In 2.3 beschreiben wir, wie einzelne Assets zu einer VR-Szene zusammengestellt werden können. Optional (in 2.4) skizzieren wir eine mögliche Erweiterung, um Arbeitsabläufe generisch zu beschreiben. Wir möchten die Arbeitsabläufe nicht in Code programmieren, da es pro Werk leicht unterschiedliche Varianten eines Trainings gibt. Deshalb erleichtert eine abstraktere Repräsentation der Applikationslogik die Anpassung an unterschiedliche Varianten der Arbeitsabläufe.

Bild 1: Screenshot Logistiktraining

2.1 Repräsentation von Assets

Assets beinhalten:

- die Geometrie und Materialien/Texturen
- den Code, der den internen Zustand des Assets hält sowie die Kommunikation mit der Außenwelt (Methodenaufrufe bzw. ausgehende Events) abwickelt
- semantische Beschreibung der Geometrien und Materialien/Texturen, sowie

• die semantische Beschreibung der Schnittstelle (Methoden, die aufgerufen werden können und Events, die vom Asset emittiert werden).

In Unity gibt es das Konzept eines Prefabs, welches unserem Asset nahekommt. Prefabs erlauben es, einzelne Komponenten einer VR-Szene zu kapseln und in unterschiedlichen VR-Szenen wiederzuverwenden.

Mittels Code-Annotation können wir aus dem Quellcode die semantischen Beschreibungen in RDF generieren. D.h. wir können automatisch ein Objekt-Modell ableiten, das Grundlage für die Kombination von Assets ist. Diese Metadaten erlauben es, Assets im Repository so abzulegen, dass Suchanfragen (z.B. nach dem Typ eines Assets oder der verwendeten Events eines Assets) möglich werden.

Längerfristig streben wir eine Standardisierung bzw. Homogenisierung der Objektmodelle der verschiedenen Assets an, um die Wiederverwendung der Assets in unterschiedlichen VR-Szenen zu erleichtern.

2.2 Modellierung der Szene am Beispiel Logistiktraining

Ausgangspunkt ist eine bestehende Unity-Applikation im Bereich Logistiktraining. In der bestehenden monolithischen Applikation sind die einzelnen Komponenten prinzipiell identifizierbar. Um Teile der Applikation über das Repository weiterverwenden zu können, wird separater Zugriff auf die einzelnen Assets benötigt. Deshalb werden in einem ersten Schritt die einzelnen Assets (Label, Handscanner…) aus einer bestehenden VR-Szene (siehe Bild 1) herausgelöst, um diese in das Asset-Repository laden und verwalten zu können.

Zur semantischen Beschreibung der Assets werden die Begriffe eines Vokabulars genutzt, welches teilweise in Bild 2 abgebildet ist. Mit diesem Vokabular können Assets, wie Bauteile, Behälter, Handscanner oder verschiedene Labeltypen, beschrieben werden.

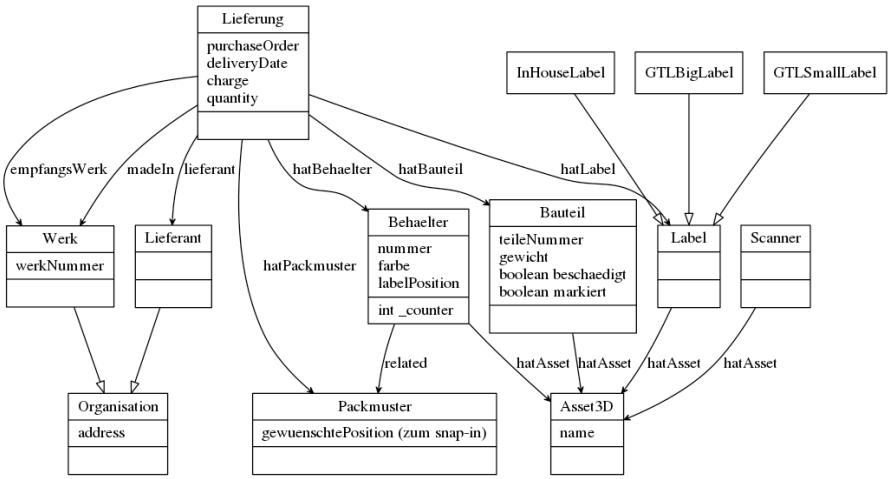

Bild 2: RDF Modellierung Logistiktraining

2.3 Zusammenspiel Asset-Logik und VR-Szene

Um Assets in einer VR-Szene zu verbinden, muss sowohl der Zustand der einzelnen Komponente an sich (interner Zustand) als auch der Zustand der VR-Szene (globaler Zustand) repräsentiert werden. Des Weiteren müssen einzelne Assets Nachrichten untereinander austauschen (z.B. der Handscanner und das Label mit Daten zur Lieferung). Dazu wird ein Event-Broker verwendet, der so z.B. in Unity verfügbar ist.

2.4 Szenensteuerung und Prozessmodellierung

Um einzelne Assets flexibel zu einer Unity-Applikation kombinieren zu können, müssen 1) die einzelnen Assets Nachrichten/Events austauschen sowie 2) der globale Zustand der Applikation dargestellt und überwacht werden. Geplant ist, den globalen Zustand der Applikation basierend auf einem Zustandsautomaten darzustellen und zu überwachen. Obwohl langfristig eine Applikationserstellung ohne das dedizierte Programmieren in C# angestrebt ist, werden in ersten Ausbaustufen vermutlich noch manuelle Programmierarbeiten zur Applikationserstellung nötig sein.

3 Umsetzung

3.1 Erzeugung von Linked Data Beschreibungen

Das World Wide Web Consortium (W3C) definiert einen Standard zur semantischen Beschreibung von Data-Provenance-Informationen auf Basis von Linked-Data-Ansätzen, PROV [1]. Dieser Standard definiert eine Ontologie zur semantischen Beschreibung von unter anderem Urheber, Datum, und Inhalt einer auf Daten durchgeführten Änderung. Auf Basis von PROV entwickelten De Nies et al. das System Git2Prov, das es erlaubt, diese Informationen aus der Commit-History eines Git-Repositories zu gewinnen [2]. Eine auf Node.js basierende Implementierung von Git2Prov ist sowohl als Node.js-Package, als auch quelloffen auf GitHub verfügbar [3].

Aktuelle Lösungen zur Verwaltung von Git Repositories erlauben die Erstellung so genannter Webhooks, HTTP Callbacks, deren URL über einen POST Request aufgerufen werden. Wir stellen einen einfachen auf Python basierenden Webservice zur Verfügung, auf den im Geometrie-Repository ein Webhook registriert wird. Wird dieser Webhook ausgelöst, beispielsweise durch Hinzufügen neuer Geometrie, sowie Modifizierung oder Entfernen bestehender Geometrie, stößt der Dienst eine erneute Erstellung der semantischen Provenance-Daten auf Basis der neuen Informationen mittels Git2Prov an.

Die aktualisierten Provenance-Informationen werden dann über eine Graph Store HTTP Protocol [4] Schnittstelle in einem Triplestore (Apache Fuseki [5], Stardog [6]) hinterlegt. Auf diesem können die im Repository hinterlegten Geometriedaten anhand ihrer hinzugefügten Informationen über Autor, Änderungsdatum, und Inhalt, mittels standardisierter Anfragesprachen (z.B. SPARQL [7]) abgefragt und gefiltert werden.

Dadurch steht einem Benutzer stets eine aktuelle und mit den im Repository hinterlegten Daten konsistente Linked-Data-Beschreibung der verwalteten Geometriedaten zur Verfügung.

Zusätzlich können weitere Informationen aus den Dateien extrahiert werden. Mit Hilfe von file [8] Unix-Tools kann Dateiformat identifiziert werden. Im Falle eines Bildformats wie z.B. PNG oder JPEG können weitere Informationen wie Bildgröße extrahiert werden. Darüber hinaus können Quellcode-Dateien wie z.B. Unity C# Skripte mit Annotationen versehen werden. Diese lassen sich mit passender Compiler-Technologie, wie z.B. dem Roslyn-Compiler für C# [9], aus Quellcode extrahieren und in den Triplestore speichern.

3.2 Architektur: Microservices, Docker und Kubernetes

Seit dem Aufstieg der Docker-Software [10] wird Container-Technologie [11] zur immer beliebteren Methode für die Ausführung von Diensten in der Cloud. Das Prinzip der Container-Technologie ähnelt dem der virtuellen Maschinen (VirtualBox [12], VMware Workstation [13]). Allerdings werden bei Containern weder die Hardware noch das Betriebssystem virtualisiert. Damit sind die Container nicht so stark von dem Hostbetriebssystem isoliert, verbrauchen jedoch viel weniger Arbeitsspeicher als virtuelle Maschinen. So können auf einem physikalischen Rechner mehr Containerinstanzen als virtuelle Maschinen ausgeführt werden.

Zusätzliche Vorteile bietet das Docker-Container Format als aktueller offener Quasi-Standard für das Bereitstellen der Virtual Appliances als Container-Images sowie Docker Hub als öffentlichen Dienst zur Speicherung und Verwaltung von Container-Images. Dadurch vereinfacht die Bereitstellung von Softwarekomponenten als Docker-Container die Installation und Wartung von Software.

Der verwendete Software-Stack enthält multiple Dienste, die nach dem Microservice-Paradigma [14] entwickelt werden. Statt einer komplexen monolithischen Anwendung sind es viele einfache Komponenten, die miteinander verbunden sind: Git-Repository Server mit Webhooks, Git2Prov Converter, Triplestore etc. Jede dieser Komponenten bietet einen Dienst, der im Vergleich mit einer komplexen Anwendung einfach zu definieren, zu entwickeln und zu testen ist. Jeder der separaten Dienste ist verfügbar als ein Docker-Container, entweder im Docker-Hub [15] oder unserer internen Docker-Registry. Da sie als Container eigenständige Anwendungen sind, müssen sie verbunden, verwaltet und überwacht werden. Diese Aufgabe übernimmt Kubernetes [16] als System zur Container-Orchestrierung. Es erlaubt sowohl die Verteilung der Container auf die virtuellen Maschinen bzw. auf die physikalischen Rechner, als auch ihre Verbindung und Verwaltung. Allein die Verbindungen zwischen den einzelnen Komponenten müssen vom User konfiguriert werden. Kubernetes stellt sicher, dass alle Komponenten funktionieren und bei Ausfall von einzelnen Rechner-Knoten auf die anderen Knoten verteilt werden.

4 Fazit

Das semantische Asset Repository in der hier beschriebenen Ausprägung stellt einen ersten nutzbaren Arbeitsstand dar. Neben den im Abschnitt 1 beschriebenen konkreten Funktionen soll auf Basis des aktuellen Standes die Bearbeitung der im Abschnitt 2 vorgestellten Themen ermöglicht werden. Parallel zur Entwicklung des Repositories entsteht eine Musterapplikation, die die funktionalen Zusammenhänge zwischen Asset, Semantik und Unity-Szene exemplarisch darstellt. Sie soll als

Blaupause für neue Anwendung dienen soll. Erste Anwendungen, welche die Funktionalitäten des Repositories nutzen, sind aktuell in Vorbereitung.

5 Ausblick

Wie einleitend erwähnt nimmt die Verfügbarkeit von AR&VR Hardware ebenso zu wie die Integration der Technologien in den (Arbeits-) Alltag. Handheld Devices wie zum Beispiel Smartphones ermöglichen den einfachen Eintritt in die AR Welt, wie es Anwendungen wie Pokémon Go in letzter Zeit eindrucksvoll bewiesen.

Zahlreiche Applikation werden sich mit der Vereinfachung des technologischen Zugangs und der weiteren Verbreitung durch Player wie Apple (ARKit) und Google (AR Core) erschließen. Dennoch bleibt den meisten Anwendungen ihre Limitierung auf das spezielle Nutzungsumfeld und deren dafür dediziert produzierten Informationen und Assets: Informationen sind meist nur innerhalb ihres eigenen Systems nutzbar.

Die Herausforderung liegt in der systemübergreifenden Bereitstellung von Informationen in übergeordneten Kontexten. Das semantische Asset Repository ist ein möglicher Baustein, um die bisherigen Grenzen zwischen Informationsart und Informationsbereichen aufzulösen (siehe 1.4 Speicherung von Kontextwissen).

Dieser Ansatz kann somit als Basis für ein breites Anwendungsspektrum dienen. Im Zusammenspiel mit weiteren Technologien wird dadurch der Mehrwert von AR&VR deutlich gesteigert. Die breite Tauglichkeit des semantischen Modells gilt es in praxisnahen Anwendungen weiter zu prüfen. Das Zusammenspiel zwischen Kontexterstellung und Informationslieferung sollte dabei ebenso erforscht werden wie die generelle Übertragbarkeit auf beliebige Informationen und Assets.

Literaturangaben

[1] W3C, „PROV-Overview," [Online]. Available: https://www.w3.org/TR/prov-overview/.

[2] T. De Nies, S. Magliacane, R. Verborgh, S. Coppens, P. Groth, E. Mannens und R. Van De Walle, „Git2PROV: exposing version control system content as W3C PROV," in Proceedings of the 12th International Semantic Web Conference (Posters & Demonstrations Track), Aachen, Germany, 2013.

[3] T. De Nies, S. Magliacane, R. Verborgh, S. Coppens, P. Groth, E. Mannens und R. Van de Walle, „Git2PROV: Unleash the potential of Git in the new

W3C standard for provenance.,“ [Online]. Available: https://github.com/IDLabResearch/Git2PROV.

[4] W3C, „SPARQL 1.1 Graph Store HTTP Protocol,“ [Online]. Available: https://www.w3.org/TR/sparql11-http-rdf-update/.

[5] The Apache Software Foundation, „Apache Jena - Apache Jena Fuseki,“ [Online]. Available: https://jena.apache.org/documentation/fuseki2/.

[6] Stardog Union, „Stardog: the Enterprise Knowledge Graph,“ [Online]. Available: http://www.stardog.com/.

[7] W3C, „SPARQL 1.1 Overview,“ [Online]. Available: https://www.w3.org/TR/sparql11-overview/.

[8] „file(1): determine file type - Linux man page,“ [Online]. Available: https://linux.die.net/man/1/file. [Zugriff am 17 09 2017].

[9] .NET Foundation, „.NET Compiler Platform ("Roslyn"),“ [Online]. Available: https://dotnetfoundation.org/net-compiler-platform-roslyn. [Zugriff am 17 09 2017].

[10] „Docker - Build, Ship, and Run Any App, Anywhere,“ [Online]. Available: https://www.docker.com/.

[11] „Docker: Was sind eigentlich Container?,“ [Online]. Available: http://t3n.de/news/was-sind-container-756373/.

[12] Oracle, „Oracle VM VirtualBox,“ [Online]. Available: https://www.virtualbox.org/.

[13] VMware, „VMware Workstation,“ [Online]. Available: http://www.vmware.com/products/workstation.html.

[14] „An introduction to microservices | Opensource.com,“ [Online]. Available: https://opensource.com/resources/what-are-microservices.

[15] Docker Inc., „Docker Hub,“ [Online]. Available: https://hub.docker.com/.

[16] „Kubernetes - Production-Grade Container Orchestration,“ [Online]. Available: https://kubernetes.io/.

[17] Docker Inc, „Docker Swarm,“ [Online]. Available: https://www.docker.com/products/docker-swarm.

[18] Werner Schreiber; Konrad Zürl; Peter Zimmermann. „Web-basierte Anwendungen Virtueller Techniken“. Springer, 2017.