Game Engine Architektur mit Hinblick auf Erweiterbarkeit

Scarlett Grenzemann

Institut Computervisualistik
Universität Koblenz

8. Oktober 2014

Inhalt

Quellen: Architekturstile

Prof. Dr. Jürgen Ebert, *Softwarearchitektur*:
`www.uni-koblenz-landau.de/koblenz/fb4/ist/rgebert/teaching/sose2013/softwarearchitektur/softwarearchitektur`
Wikipedia, *Listener Pattern*:
`de.wikipedia.org/wiki/Beobachter_(Entwurfsmuster)`

Quellen: Game Engines

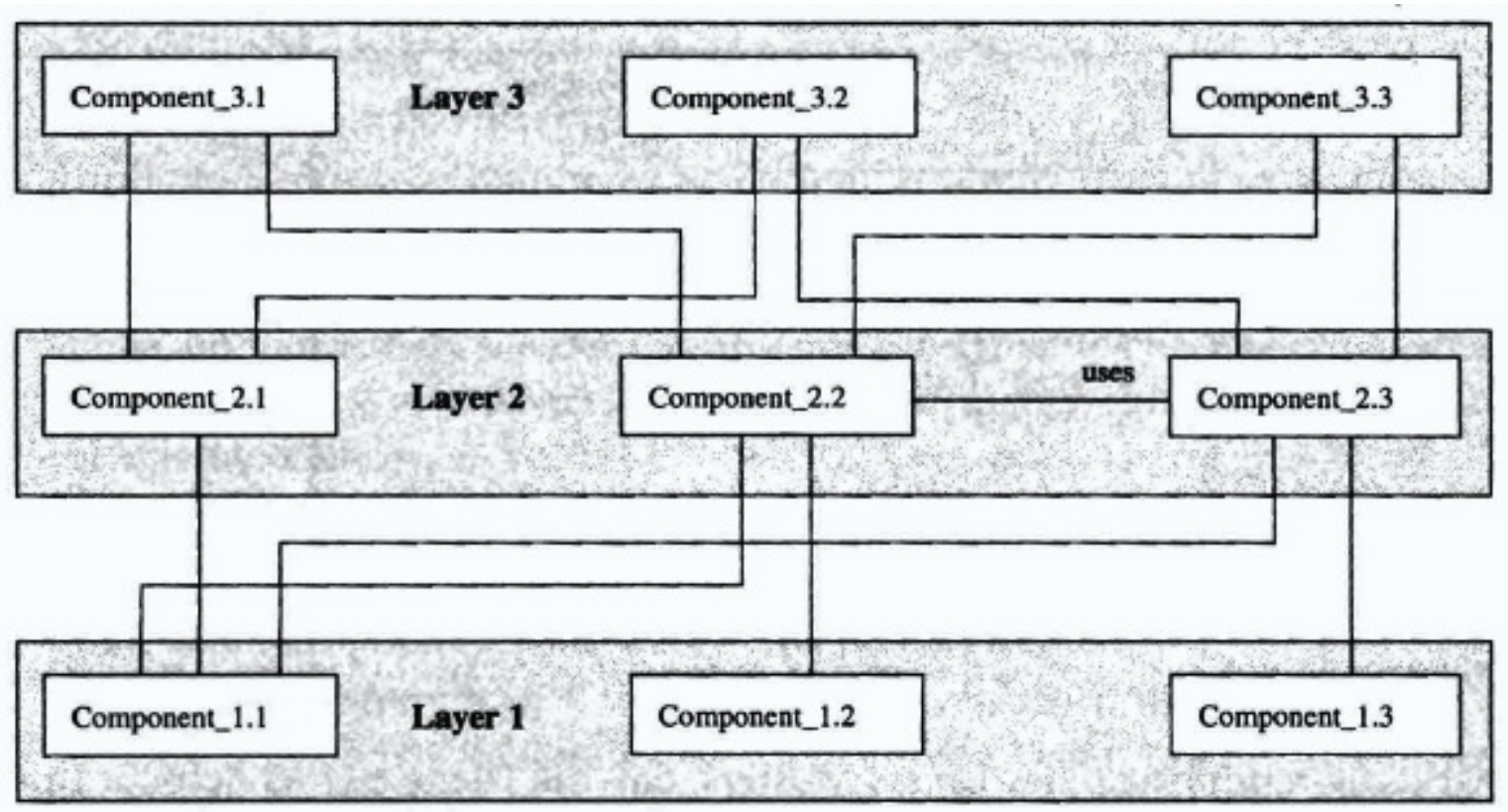
Wikipedia, *Unity*, *Unreal Engine*, *OGRE*:
de.wikipedia.org/wiki/Unity_(Spiel-Engine)
de.wikipedia.org/wiki/Unreal_Engine
de.wikipedia.org/wiki/OGRE
Wikipedia, *Spiel-Engine*:
de.wikipedia.org/wiki/Spiel-Engine

Architekturstile

- Layered
- Data Flow
- Data Centered/Repository
- Component Interaction
- User Interaction
- Adaption

Layered

- Aufgaben sind Abstraktionsniveaus zugeordnet und in Schichten zusammengefasst
- System in Schichten (abstrakte Maschinen) zerlegt und auf darunterliegender Maschine implementiert
- Enthalten Bausteine, die miteinander kooperieren
- Bausteine einer höheren Schicht benutzen Bausteine der darunter liegenden Schicht
- Schichten entkoppelt
- Horizontale Struktur

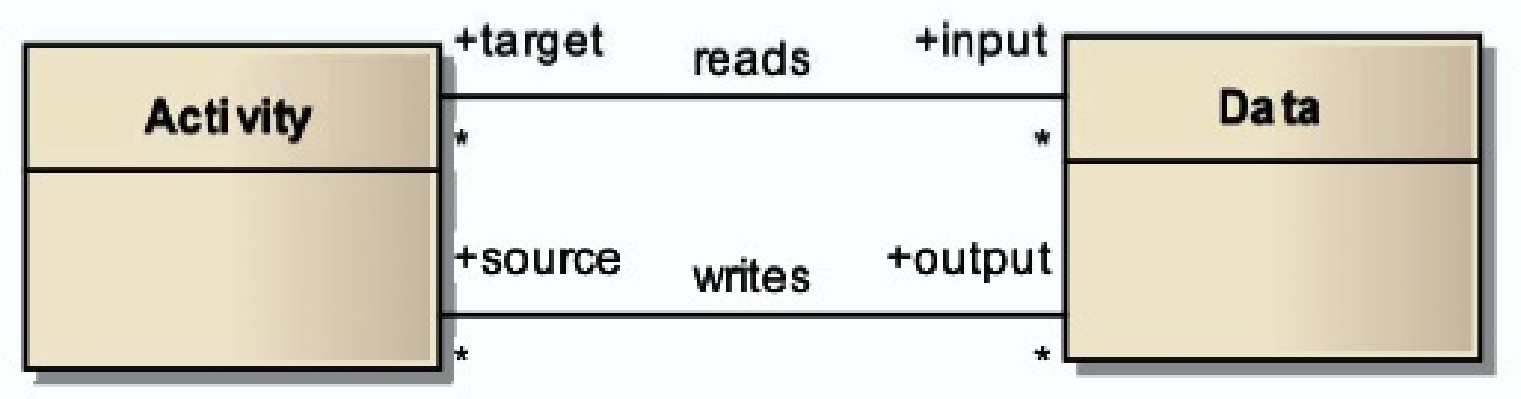

Component_3.1
Layer 3
Component_3.2
Component_3.3
Component_2.1
Layer 2
Component_2.2
uses
Component_2.3
Component_1.1
Layer 1
Component_1.2
Component_1.3

Layered: Vor- und Nachteile

+ Abstraktion vereinfacht Entwurf und Verstehen
+ Änderungen lokal, solange sich Schnittstellen nicht ändern
+ Wiederverwendbarkeit unter Schichten leicht möglich
– Performanzverluste bei zu starker Entkopplung
– Objekte müssen durchgereicht werden
– Schnittstellen müssen stabil bleiben
– Richtige Ebenen zu definieren ist schwer

Data Flow

- System besteht aus Menge von unabhängigen Aktivitäten (Prozessen), die Daten produzieren und verarbeiten
- Fokus liegt auf Aktivitäten, Daten spielen sekundäre Rolle
- *Batch Sequential*: Unabhängige transformationale Prozesse; jeder Prozess vor dem nächsten vollständig ausgeführt
- *Pipe Filter*: Aktivitäten (Filter) lesen und schreiben Datenströme; Datenverbindungen (Pipes) transportieren Daten an Nachfolger

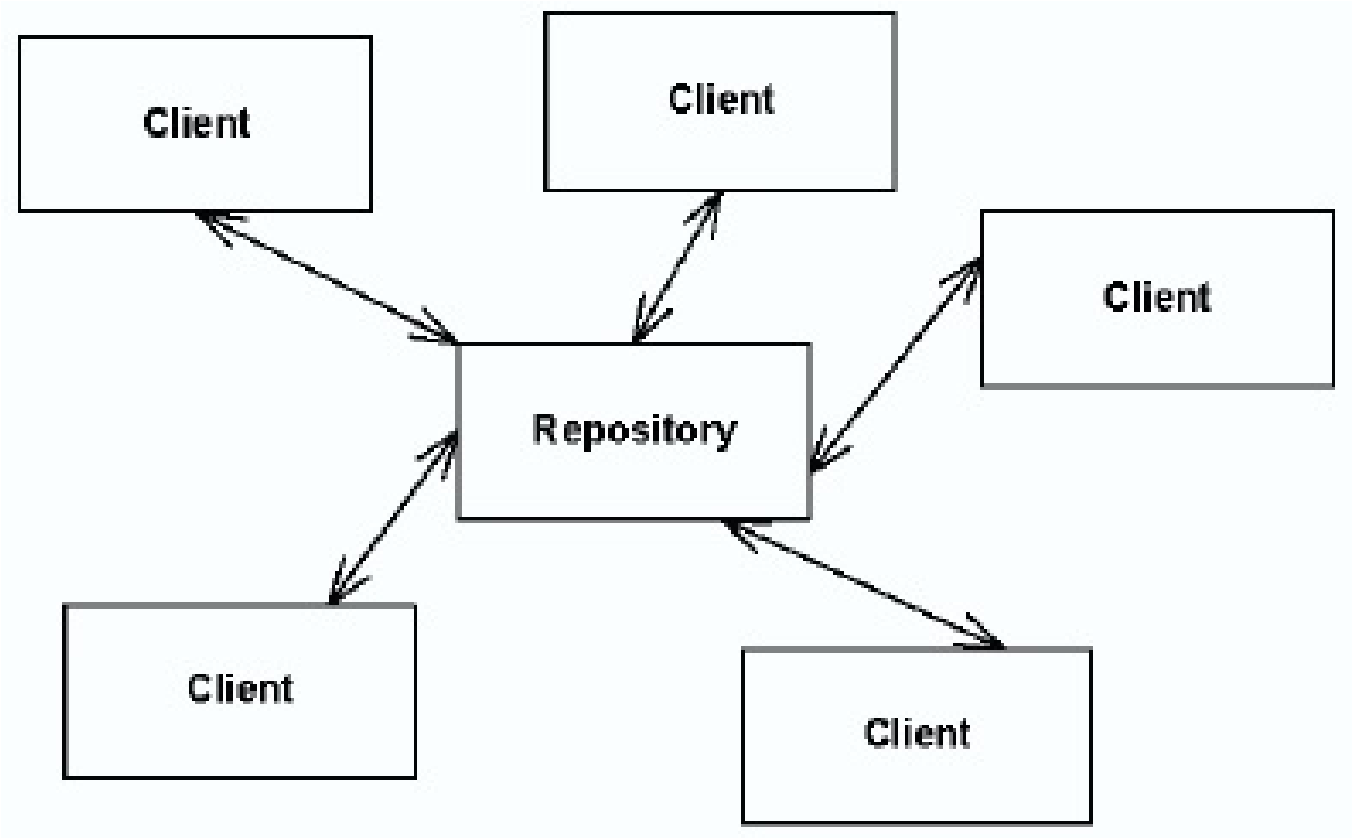

Activity
+target
reads
+input
Data
*
*
+source
writes
+output
*
*

Data Flow: Vor- und Nachteile

+ Leicht verständlich
+ Unterstützung der Wiederverwendung
+ Saubere Trennung der Prozesse
+ Leicht zu **erweitern**/evolvieren/analysieren
– Nicht für interaktive und zustandsbasierte Systeme

Data Centered/Repository

- Mehrere verschiedene unabhängige Bausteine (ohne direkte Kommunikation) entnehmen, verarbeiten und verändern eine gemeinsame Menge von Daten oder fügen Daten hinzu
- Diese werden mit zugehörigen Diensten zentral in einem Repository zur Verfügung gestellt
- Repository ist Komponente, die die Daten hält (Datenbank)


Client
Client
Client
Repository
Client
Client

Data Centered/Repository

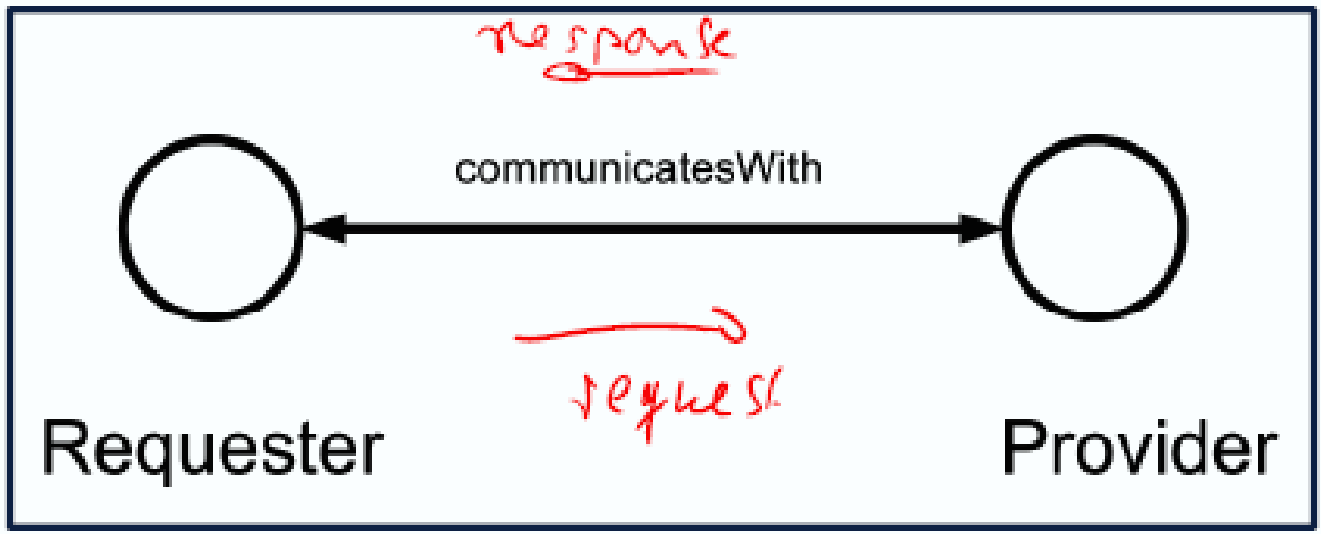
- *Shared Repository*: Komponenten aktiv und selbstständig, verwenden Services; gemeinsame Datenbank durch verschiedene Programme; Zugriff über Schnittstelle
- *Active Repository*: Repository informiert die Komponenten aktiv über Änderungen
- *Blackboard*: Blackboard ist eine gut organisierte Datenstruktur; Knowledge Sources sind aktive Komponenten, die die Information im Blackboard einstellen; Control stößt die Verarbeitung an

Data Centered/Repository: Vor- und Nachteile

+ Effiziente, gemeinsame Speicherung
+ Synchronisation, Transaktionen, Redundanzarmut etc. möglich
+ Entkopplung von Anwendungen
+ Leichte **Erweiterbarkeit**
– Flaschenhals

Component Interaction

- System als Ansammlung interagierender Komponenten
- Komponenten unabhängig und kommunizieren miteinander
- Sie kontrollieren einander nicht unbedingt, können verteilt sein


response
communicatesWith
request
Requester
Provider

Component Interaction

- *Client Server*: Clients beziehen Service und Servers stellen Service zur Verfügung; Client optimiert für Interaktion, Server optimiert für Bedienen mehrerer Clients; jeder Request vom Server unabhängig bearbeitet; hohe Sicherheitsanforderungen
- *Peer to Peer*: Ähnelt Client Server; Requester und Provider werden nicht unterschieden, alle Beteiligten spielen beide Rollen; dynamische Anzahl Komponenten
- *Explicit Invocation*: Requester kennen den Ort des Providers; eventuell werden Broker verwendet, um Provider zu finden; Requester initiiert die Kommunikation explizit

Component Interaction

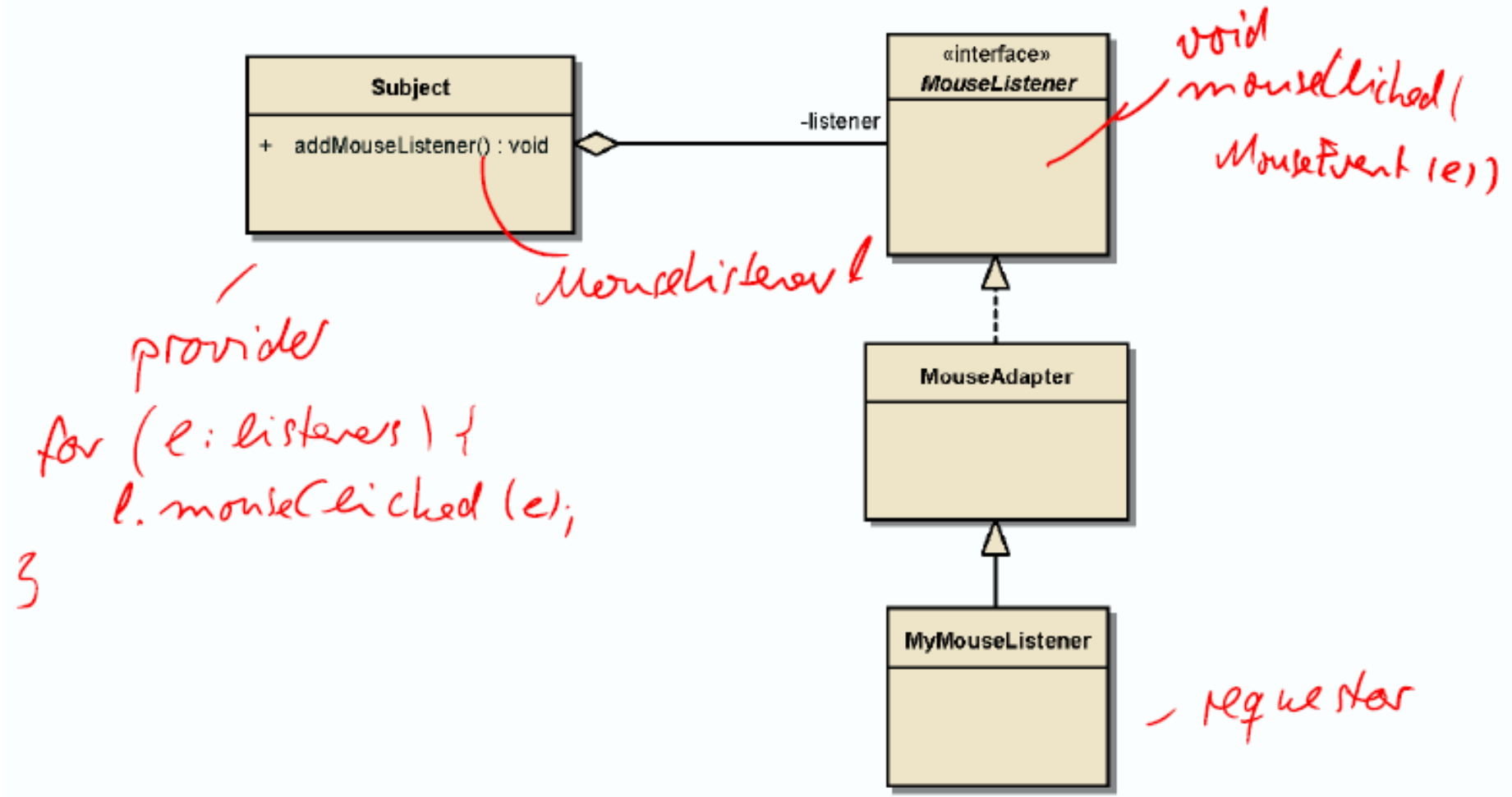
- *Implicit Invocation*: Kein direkter Aufruf, sondern Nachrichtenaustausch (z.B. Publish Subscribe)
- *Publish Subscribe*: Komponenten automatisch über Ereignisse informiert; Producer und Consumer entkoppelt; Consumer registrieren sich für ein spezielles Ereignis und Producer publizieren Ereignisse; **Variante: Listener Pattern**

Component Interaction: Vor- und Nachteile

+ Wiederverwendbarkeit von Komponenten
+ Hinzufügbarkeit zur Laufzeit
+ Erleichterung der Evolution
– Keine Kontrolle über das System
– Analyse des Systems erschwert
– Kein Wissen darüber, ob Event verarbeitet wird

Listener Pattern

- Komponenten stellen den Zustand eines Objekts grafisch dar und kennen die gesamte Schnittstelle des Objekts
- Ändert sich der Zustand, müssen Komponenten darüber informiert werden
- Objekt unabhängig von Komponenten, kennt also ihre Schnittstelle nicht
- Beobachtetes Objekt bietet Mechanismus, um Beobachter an-/abzumelden und diese über Änderungen zu informieren
- Meldung unspezifisch an jeden Beobachter, das Objekt kennt also nicht deren Struktur
- Beobachter implementieren Methode, um auf Änderungen zu reagieren und fragen relevante Teile des Zustands ab

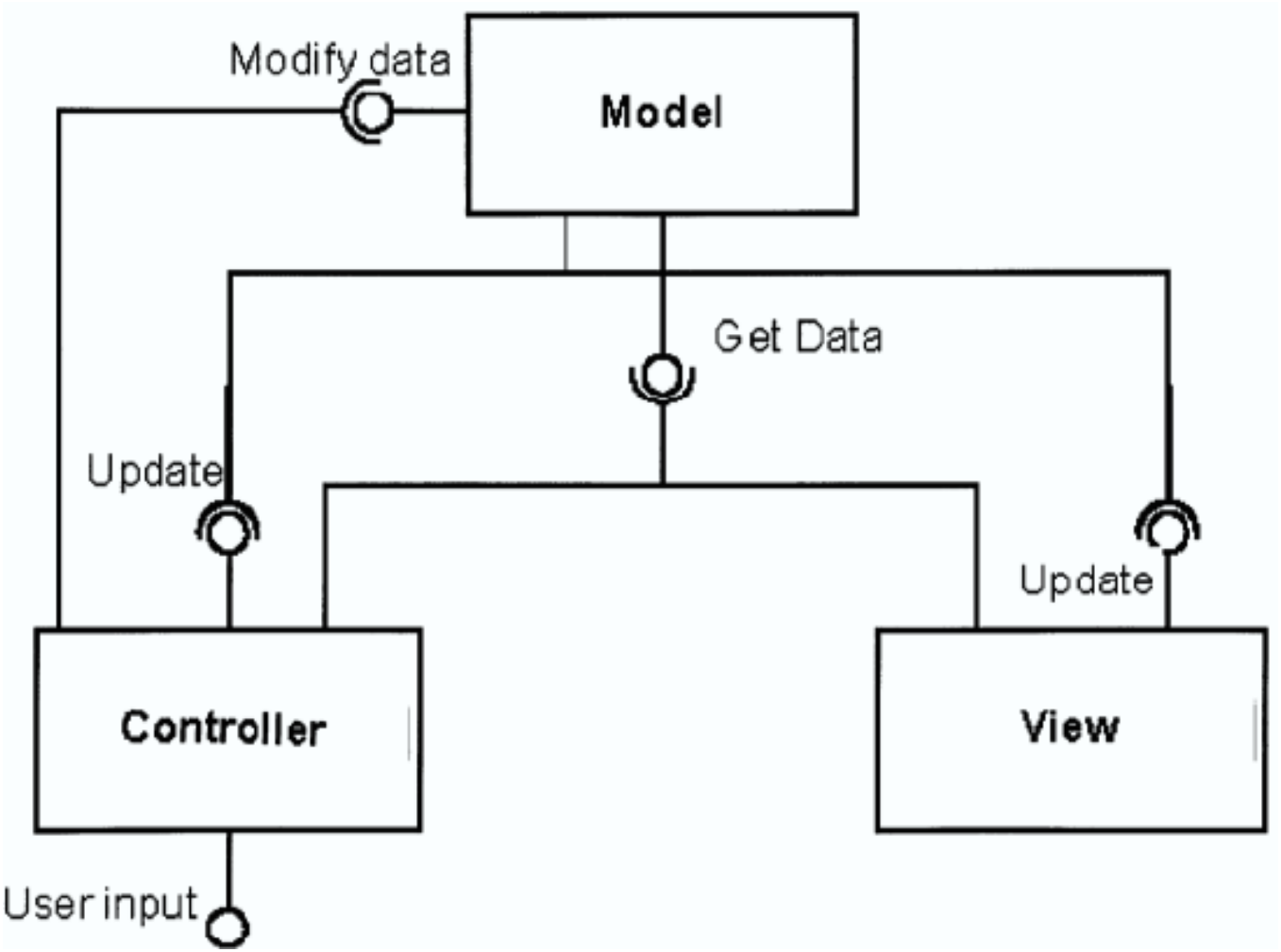

Subject
+ addMouseListener() : void
-listener
«interface»
MouseListener
MouseAdapter
MyMouseListener
void mouseClicked(MouseEvent e))
MouseListener l
provider
for (e : listeners) {
l.mouseClicked(e);
}
- requestor

Listener Pattern: Vor- und Nachteile

+ Objekte/Beobachter unabhängig variierbar und auf abstrakte, minimale Art lose gekoppelt
+ Objekt braucht keine Kenntnis über Struktur der Beobachter
+ Ein abhängiges Objekt erhält Änderungen automatisch
– Hohe Änderungskosten bei großer Beobachterzahl
– Rufen Beobachter während der Bearbeitung einer Änderung Änderungsmethoden des Objekts auf, kann es zu Endlosschleifen kommen

User Interaction

- Modell (Verarbeitung) enthält Kernfunktionalität/Daten
- Ansichten (Ausgabe) präsentieren die Information, die sie sich selbst besorgen
- Kontrollbausteine (Eingabe, Mechanismus) verarbeiten Bedieneingaben
- Ansichten und Kontrollbausteine zusammen bilden User Interface


Modify data
Model
Get Data
Update
Update
Controller
View
User input

User Interaction

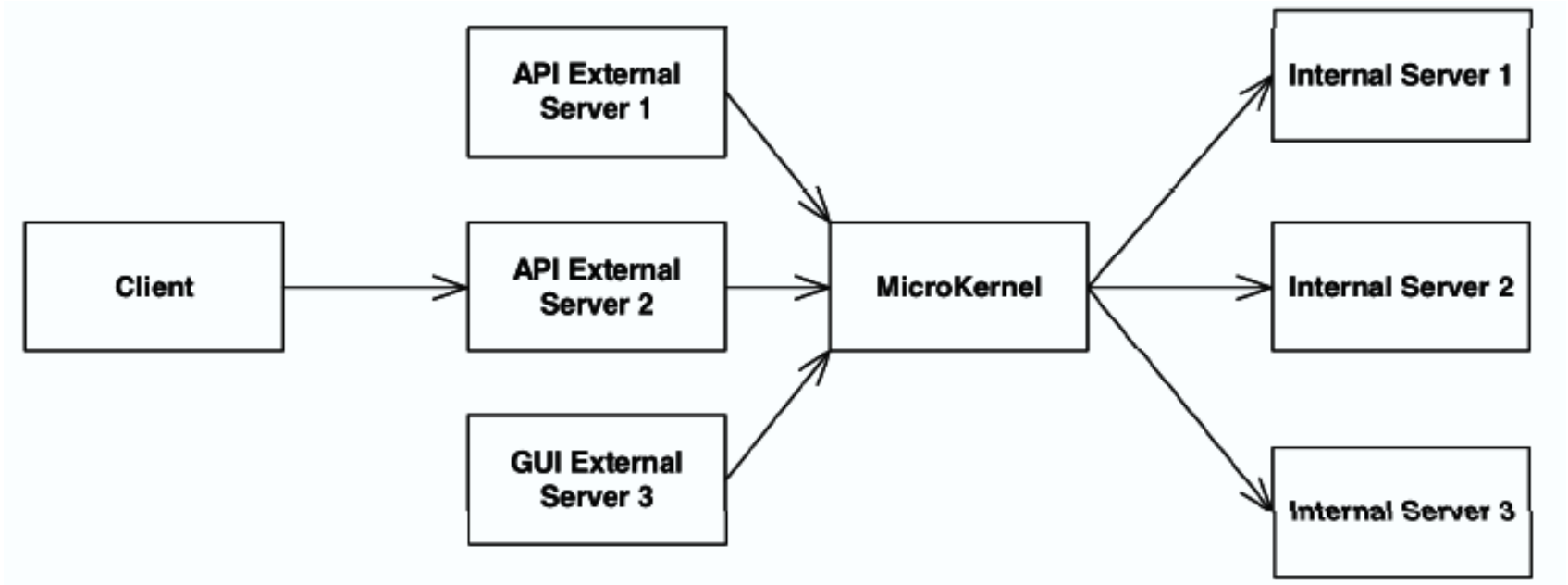
- *Model View Controller*: Behandlung mehrfacher Benutzungsschnittstellen bei gleicher Anwendungslogik; Änderungen automatisch in der GUI sichtbar; Ändern einer Darstellung in der GUI ohne Anwendungslogik anzufassen
- *Presentation Abstraction Control*: Koordination vieler Views (Ansichten); hierarchische Kombination verschiedener Agenten (Services); Präsentationsteil bestimmt visuelles Verhalten; Abstraktionsteil enthält Daten/Funktionalität; Kontrollteil verbindet Präsentation/Abstraktion und kommuniziert mit anderen Agenten

User Interaction: Vor- und Nachteile

+ Gute Trennung der Belange
+ Leicht anpassbar und **erweiterbar**
+ Gut portierbar an neue Basistechnologie/anderes Look-and-Feel
+ Verschiedene Interaktionsmöglichkeiten gleichzeitig handhabbar

Adaption

- *Microkernel*: Familie von Systemen, deren Mitglieder unterschiedliche Funktionalitäten liefern; Kern, der auf variantenspezifische interne Dienste zugreift, wird ergänzt durch externe Dienste, die dann eine API nach außen bereitstellen
- *Reflection*: Struktur-/Verhaltensaspekte in Meta-Objekten gehalten, die von Anwendungslogik getrennt sind; Anwendungslogik kann Anfragen an Meta-Objekte stellen
- *Interception*: Services werden beim Aufruf durch weitere Bausteine modifiziert


API External Server 1
Client
API External Server 2
GUI External Server 3
MicroKernel
Internal Server 1
Internal Server 2
Internal Server 3

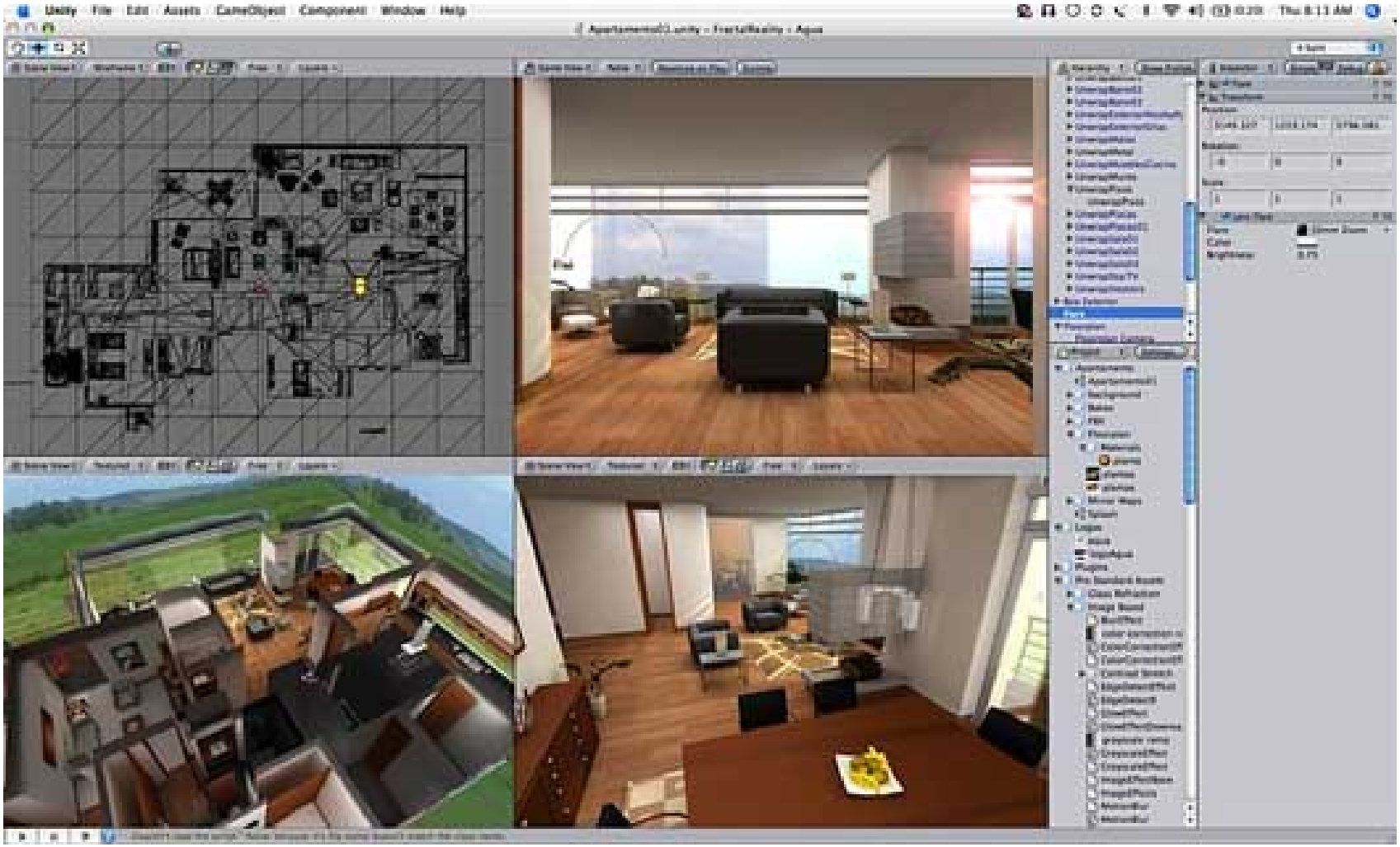
Game Engines

- Unity
- Unreal Engine
- OGRE

Unity

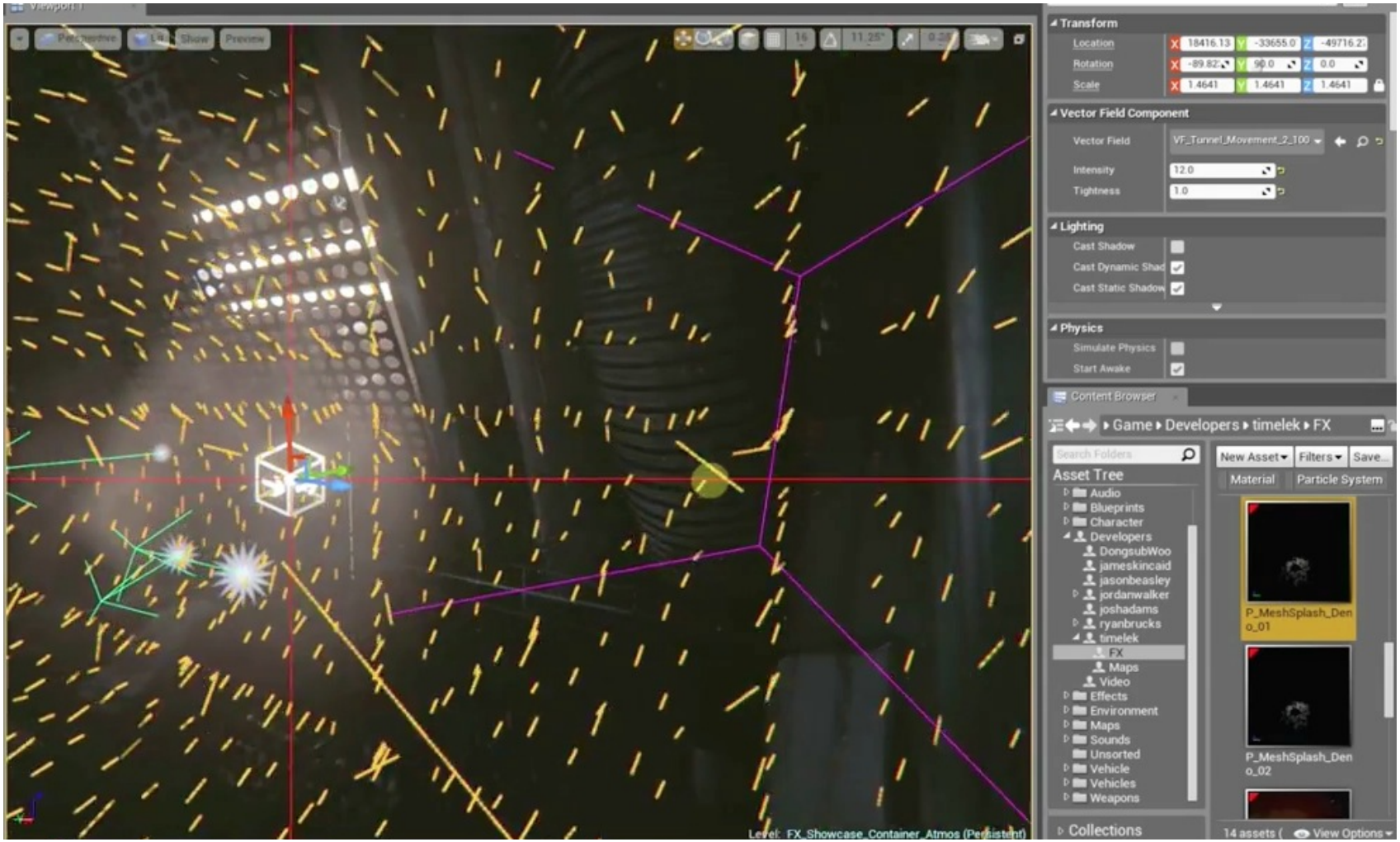
- Szenengraph aus „Game-Objekten“; diesen Objekten werden Komponenten zugeordnet
- Einfache Objekte direkt im Editor erzeugt; komplexe Komponenten, die in anderen Programmen erstellt wurden, per Drag'n'Drop importiert
- Automatische Aktualisierung bei Änderungen
- „Game-View“ simuliert Darstellung/Verhalten des Spiels

Game Engines



Unreal Engine

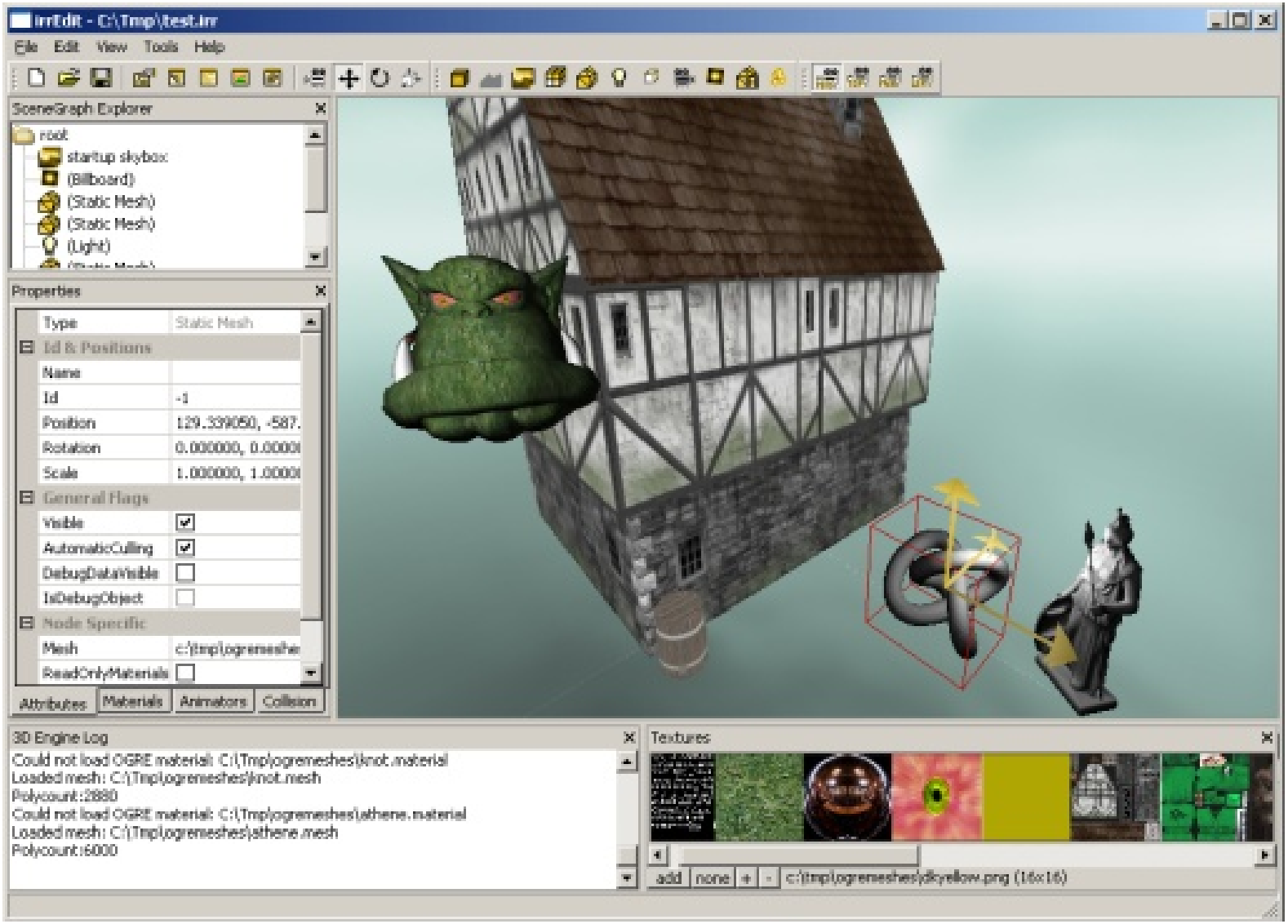
- Modular aufgebaut
- Modul: Führt eine Reihe von Verarbeitungsschritten durch, liefert bei der Rückkehr an das aufrufende Programm Daten als Ergebnis zurück
- Kapselung: Schnittstelle enthält Daten, Implementierung enthält Programmcode
- Teile der Engine werden neu geschrieben, aber es bleibt dieselbe Engine
- Keine Versionsnummern, sondern nummerierte Builds welche bestimmte Funktionen (nicht) enthalten


Transform
Location
Rotation
Scale
Vector Field Component
Vector Field
Intensity
Tightness
Lighting
Cast Shadow
Physics
Simulate Physics
Start Awake
Content Browser
Game ▸ Developers ▸ timelek ▸ FX
Asset Tree
Audio
Blueprints
Character
Developers
DongsubWoo
jameskincaid
jasonbeasley
jordanwalker
joshadams
ryanbrucks
timelek
FX
Maps
Video
Effects
Environment
Maps
Sounds
Unsorted
Vehicle
Vehicles
Weapons
Collections
New Asset
Filters
Save...
Material
Particle System
P_MeshSplash_Den o_01
P_MeshSplash_Den o_02
View Options

OGRE

- Programmbibliothek bietet Szenengraphen und Unterstützung für gängige Grafikschnittstellen
- Sämtliche Details der Systembibliotheken werden in einer Klassenstruktur abstrahiert
- Plugin-Struktur, um Modifikationen zu integrieren

Game Engines



Anforderungen

- Spiele-Engines benötigen *Grafik-Engine*, *Physiksystem*, *Soundsystem*, *Zustandsspeicherung*, *Steuerung* und *Datenverwaltung*
- Zusätzlich *Erweiterbarkeit*, z.B. durch Plugins
- Je schneller Spieleentwickler das Ergebnis der Änderungen im Spiel sehen, desto produktiver arbeiten sie

Erweiterbarkeit

- Data Flow leicht zu erweitern, aber nicht für interaktive/zustandsbasierte Systeme geeignet
- Data Centered/Repository bietet leichte Erweiterbarkeit, Flaschenhals ist aber ein Problem
- User Interaction bietet Erweiterbarkeit, Portierbarkeit und schnelle Sichtbarkeit von Änderungen in der GUI

Konzept

- User Interaction-Stil
- Grafik-Engine
- Steuerung
- Datenverwaltung
- Eventuell Physiksystem

Evaluation

- Erfüllt Anforderungen bezüglich Erweiterbarkeit für zukünftige Projekte und schnelle Sichtbarkeit von Änderungen für den Spieleentwickler
- Portierbarkeit als zusätzlicher Vorteil
- Besitzt Grundausstattung einer gängigen Spiele-Engine