UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

ARQUITETURA PARA DESENVOLVIMENTO DE JOGOS COM O USO DE COMPONENTES REAPROVEITÁVEIS

DISSERTAÇÃO DE MESTRADO

Carlos Edmilson da Silva Maia

Santa Maria, RS, Brasil

2010

ARQUITETURA PARA DESENVOLVIMENTO DE JOGOS COM O USO DE COMPONENTES REAPROVEITÁVEIS

por

Carlos Edmilson da Silva Maia

Dissertação apresentada ao Programa de Pós-Graduação em Informática da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para a obtenção do grau de

Mestre em Computação

Orientador: Prof. Dr. Marcos Cordeiro d'Ornellas (UFSM)

Dissertação de Mestrado Nº 12
Santa Maria, RS, Brasil

2010

Universidade Federal de Santa Maria
Centro de Tecnologia
Programa de Pós-Graduação em Informática

A Comissão Examinadora, abaixo assinada,
aprova a Dissertação de Mestrado

ARQUITETURA PARA DESENVOLVIMENTO DE JOGOS COM O USO DE COMPONENTES REAPROVEITÁVEIS

elaborada por
Carlos Edmilson da Silva Maia

como requisito parcial para obtenção do grau de
Mestre em Computação

COMISSÃO EXAMINADORA:

Prof. Dr. Marcos Cordeiro d'Ornellas (UFSM)
(Presidente/Orientador)

Prof. Dr. Cesar Tadeu Pozzer (UFSM)

Prof. Dr. Esteban Walter Gonzalez Clua (UFF)

Santa Maria, 26 de março de 2010.

AGRADECIMENTOS

Em primeiro lugar gostaria de agradecer à minha família e à minha namorada - que é parte da mesma - pelo apoio, compreensão e suporte ofertados a mim ao longo de minha vida, por fazerem de mim o que sou hoje e por propiciar as condições que me permitiram desenvolver este trabalho.

Em seguida gostaria de agradecer aos meus colegas da Decadium, por serem excelentes companheiros de trabalho, pela amizade, pelo apoio e pelas contribuições relativas à minha graduação e ao meu mestrado.

Também agradeço aos meus amigos e conhecidos que não estão enquadrados nos grupos acima pelas conversas, pela diversão e por estarem presentes no meu convívio.

Por fim, gostaria de agradecer ao meu orientador no mestrado, o professor Marcos Cordeiro d'Ornellas, e ao professor Cesar Tadeu Pozzer, por todas as discussões relacionadas ao desenvolvimento de jogos e computação gráfica. Agradeço também a todos que trabalharam para fornecer a estrutura disponibilizada pelo LaCA.

“Lógica te levará do ponto A ao ponto B. Imaginação te levará a qualquer lugar.”
— ALBERT EINSTEIN

RESUMO

Dissertação de Mestrado
Programa de Pós-Graduação em Informática
Universidade Federal de Santa Maria

ARQUITETURA PARA DESENVOLVIMENTO DE JOGOS COM O USO DE COMPONENTES REAPROVEITÁVEIS

Autor: Carlos Edmilson da Silva Maia
Orientador: Prof. Dr. Marcos Cordeiro d'Ornellas (UFSM)
Local e data da defesa: Santa Maria, 26 de março de 2010.

O desenvolvimento de jogos apresenta mais desafios à medida que softwares desse tipo tornam-se mais complexos e detalhados. Assim como no desenvolvimento de softwares tradicionais, esse crescimento do escopo incorre em custos maiores e prazos de produção mais longos, aumentando os riscos enfrentados por empresas do ramo que buscam situar-se ou manter-se no mercado de jogos eletrônicos. Com isso, torna-se cada vez mais desejável buscar e implementar formas de desenvolvimento que possibilitem a redução de recursos necessários para a produção desse tipo de software. Também se observa um aumento da necessidade de disponibilizar às equipes de desenvolvimento novas ferramentas que possibilitem a criação mais rápida de protótipos, permitindo testar e evoluir novas ideias de jogos de forma mais ágil e menos onerosa.

Este trabalho trata do projeto e da implementação de uma plataforma modular que objetiva a resolução ou amenização dos problemas citados, oferecendo uma ferramenta que possibilite o desenvolvimento de jogos com custo reduzido, além de simplificar o processo de prototipação rápida. Através do uso de uma abordagem de programação baseada no uso de componentes de software reaproveitáveis escritos em diferentes linguagens de programação, este trabalho busca tornar possível metodologias que sejam capazes de incentivar o reuso de código, facilitar o trabalho em equipe, reduzir o tempo de desenvolvimento e melhorar a qualidade do produto final.

Palavras-chave: Software baseado em componentes, desenvolvimento de jogos, engenharia de software, padrões de projeto.

ABSTRACT

Master's Dissertation
Programa de Pós-Graduação em Informática
Universidade Federal de Santa Maria

CREATION OF A PLATFORM FOR COMPONENT BASED GAME DEVELOPMENT
Author: Carlos Edmilson da Silva Maia
Advisor: Prof. Dr. Marcos Cordeiro d'Ornellas (UFSM)

Game development presents more challenges as this type of software becomes more complex and detailed. As in the development of traditional software, this growth in scope incurs in a development process that costs more and has a longer production time, increasing the risks faced by companies that wish to enter or maintain themselves in the electronic game market. With this, it becomes increasingly desirable to search and implement tools that allow for the reduction of required resources to develop games. It is also noticeable that there is an increasing need to proved development teams with new tools that allow faster creation of prototypes, enabling them to test and evolve new game ideas in a nimbler and less expensive way.

This work presents the project and implementation of a modular platform that aims to solve or lessen the aforementioned problems, offering a tool that allows for the development of games with reduced costs, in addition to simplifying the quick prototyping process. Through the use of a programming approach based on reusable software components written in different programming languages, this work aims to enable methodologies that are able to foster code reuse, to facilitate team work, to reduce development time and to improve the quality of the final product.

Keywords: component-based software development, game development, software engineering, design patterns.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

API	Application Programming Interface
JAR	Java Archive
JOGL	Java OpenGL
JNI	Java Native Interface
JSR	Java Specification Request

SUMÁRIO

1 INTRODUÇÃO

1.1 Contexto e Motivação

A indústria de desenvolvimento de jogos é uma das que mais cresce no mundo, sendo que, nos Estados Unidos, ela já gera receita maior do que a indústria de filmes (ARAKJI; LANG, 2007). Esta indústria de jogos é um dos setores de mais rápida expansão na economia mundia, desenvolvendo tecnologia que está sendo aplicada até mesmo em outras áreas como a militar e a de treinamento médico (TULIP; BEKKEMA; NESBITT, 2006). Ela também apresenta a característica de permitir a entrada de pequenas empresas e empreendedores no mercado, mesmo contando com equipes reduzidas em relação às grandes empresas. Tendo isto em vista, mostra-se desejável, tal como em qualquer indústria, encontrar formas de reduzir os custos de produção e desenvolver produtos de melhor qualidade.

O processo de desenvolvimento de jogos caracteriza-se pela sua complexidade. A disponibilização de novos recursos e tecnologias na área amplia e evolui as possibilidades na produção deste tipo de software, permitindo jogos com mais funcionalidades. Cada vez mais se busca atrair a atenção do público para os jogos sendo desenvolvidos na indústria, o que aumenta as expectativas do mesmo em relação aos produtos desenvolvidos, aumentando, assim, o custo relacionado à sofisticação tecnológica e criativa envolvida no processo de desenvolvimento de jogos (CUTUMISU et al., 2007). Jogos com maior complexidade, assim como qualquer tipo de software nessa mesma situação, tornam o desenvolvimento desses sistemas mais oneroso, demorado e sujeito a erros e falhas. É importante, portanto, buscar novas técnicas e formas de desenvolvimento de jogos que amenizem o impacto da crescente complexidade presente nos jogos.

As várias técnicas e métodos de desenvolvimento e gerenciamento do processo de desenvolvimento usados com softwares também podem ser aplicados a jogos, uma vez

que os mesmos são essencialmente um caso de software específico. Um desses métodos envolve a utilização de componentes de software reaproveitáveis (BASILI et al., 1997), que se referem ao uso de módulos que foram desenvolvidos em um projeto anterior como parte de um novo projeto de desenvolvimento de software (YAO; ETZKORN, 2004). O uso de tais componentes traz diversas vantagens ao processo, como redução do tempo inicial de desenvolvimento, redução do custo de componentes individuais, maior qualidade e mais fácil manutenção (VITHARANA, 2003), além de reduzir o custo geral do software desenvolvido e aumentar a produtividade dos programadores (YAO; ETZKORN, 2004).

1.2 Objetivos, Hipóteses e Justificativas

Este trabalho trata da criação de uma solução baseada na utilização de componentes de software reaproveitáveis para a criação de jogos de forma mais rápida e menos onerosa, permitindo também o desenvolvimento de componentes que poderão ser reaproveitados em projetos futuros. Também é objetivo deste trabalho permitir o uso de código pré-existente em novos projetos desenvolvidos sobre a arquitetura proposta. Com isso, busca-se incentivar o reuso de código com o objetivo de aumentar a produtividade no desenvolvimento de jogos.

Outro foco deste trabalho é permitir que os componentes desenvolvidos para a plataforma delineada pelo mesmo sejam escritos em linguagens de programações diferentes, de forma a permitir que programadores com conhecimentos ou especialidades diferentes possam colaborar em um mesmo projeto, utilizando, possivelmente, as mesmas ferramentas que já utilizam. Isso tem a capacidade de não só aumentar a gama de conhecimentos e experiências que são úteis a um projeto de jogo como também permite que seja obtido dos programadores código na linguagem com a qual eles estão mais familiarizados.

Em resumo, podemos destacar os seguintes objetivos da arquitetura desenvolvida neste trabalho:

- Permitir o desenvolvimento de jogos utilizando módulos: componentes isolados reaproveitáveis que fornecem uma funcionalidade especializada;
- Oferecer moldes simples que promovam uma maior facilidade para a criação de novos módulos, buscando manter descomplicadas a sintaxe e estrutura necessárias para a criação de um novo componente;

- Permitir que o programador escolha qual a linguagem mais adequada a ser usada para um determinado módulo ou a linguagem com a qual está mais familiarizado;
- Possibilitar a participação em um mesmo projeto de programadores com conhecimento e experiência em diferentes linguagens de programação;
- Desenvolver a comunicação entre os componentes de tal forma que a mesma seja feita de forma transparente para o programador;
- Testar e provar a hipótese de que é possível desenvolver jogos e efetuar a prototipação rápida dos mesmos com uma plataforma baseada na arquitetura em questão.

Este trabalho, então, propõe-se a atender os objetivos citados com uma solução baseada em componentes de software reaproveitáveis, ao passo que pretende validar se tal abordagem é usável e adequada para o desenvolvimento de jogos. A arquitetura definida como resultado deste processo é, por sua vez, implementada em uma plataforma, criada com o intuito de validar a arquitetura. O uso bem-sucedido da plataforma é, portanto, capaz de validar as propostas e hipóteses nas quais este trabalho se baseia. É importante salientar que, do ponto de vista da engenharia de software, a plataforma proposta neste trabalho é essencialmente um *framework* de desenvolvimento de jogos.

Em relação à plataforma almejada por este trabalho, definem-se os seguintes requisitos técnicos:

- A plataforma deve ser modular, permitindo a criação e uso de componentes independentes que possam se comunicar;
- Deve ser utilizada a linguagem de programação Java, unida às funcionalidades oferecidas por linguagens de script, de forma a facilitar e simplificar o processo de desenvolvimento de jogos e de comunicação entre os módulos;
- A comunicação entre os módulos deve seguir formas pré-determinada pelas especificações da plataforma.

Como consequência destes requisitos, podemos delinear a necessidade de:

- Criar um sistema que permita a comunicação entre os módulos;
- Decidir a forma ideal de realizar a comunicação entre Java e linguagens de script;

- Definir se é possível oferecer suporte adequado a diversas linguagens de script ou se é necessário oferecer suporte a somente uma em especifico;
- Descobrir como realizar a comunicação entre os módulos de forma a permitir o reuso de componentes e a expansão do sistema sendo desenvolvido;
- Especificar como oferecer suporte a diferentes linguagens de programação de script e como conectar a plataforma e módulos escritos em Java com os módulos escritos nessas linguagens.

1.3 Limitações

O suporte a funcionalidades como suporte a módulos de software, a comunicação entre esses componentes e o suporte a diferentes linguagens de script resulta em uma plataforma que irá apresentar um fluxo de execução complexo, especialmente no tocante ao tráfego de informações entre os módulos. O próprio uso de linguagens de script, que são normalmente interpretadas, inclui um *overhead* à plataforma que se acrescenta à essa complexidade. Devido a isso, a plataforma resultante deste trabalho apresenta uma redução no desempenho em relação a um jogo implementado sem o uso da mesma ou sobre uma outra plataforma que tem o desempenho como um dos fatores prioritários.

Apesar do desempenho ser uma prioridade comum no desenvolvimento de jogos (CLAYPOOL; CLAYPOOL, 2009), a criação de um sistema baseado em módulos intercambiáveis, como apresentado neste trabalho, já precede a necessidade de um *overhead* adicional para o próprio gerenciamento de módulos. É inviável para este trabalho sacrificar as funcionalidades propostas em nome de desempenho. Devido a isso, o mesmo não é uma prioridade para a plataforma proposta.

Portanto, este trabalho almeja apresentar uma plataforma que ofereça o poder e flexibilidade necessários para atender os objetivos propostos da melhor forma possível, mesmo que isto sacrifique a possibilidade de criação de aplicações de alto desempenho sobre a mesma. Aceita-se, neste trabalho, a premissa de que alto desempenho não precisa ser um fator prioritário para o desenvolvimento de jogos de pequeno porte e para a prototipação rápida de jogos.

1.4 Organização do Texto

Este trabalho está organizado da seguinte forma: no capítulo 2 apresenta-se uma revisão da literatura no tocante ao uso de Java e linguagens de script no desenvolvimento de jogos, o uso de prototipação rápida nessa área e o uso e desenvolvimento de sistemas modulares. Também se discute plataformas existentes com abordagem similar.

Já no capítulo 3 discute-se o desenvolvimento da arquitetura e da implementação de uma plataforma que valide a estrutura proposta, bem como os desafios encontrados no processo.

O capítulo 4 discute a verificação e validação dos resultados, mostrando as estruturas resultantes do desenvolvimento. Também são mostrados exemplos implementados sobre a plataforma resultante que objetivam por validar a mesma.

Por fim, o capítulo 5 oferece conclusões e sugestões de trabalhos futuros acerca do trabalho aqui apresentado, discutindo usos, expansões e resultados relativos à plataforma.

2 DESENVOLVIMENTO DE JOGOS E PROTOTIPAÇÃO RÁPIDA

2.1 Usando Java e Linguagens de Script no Desenvolvimento de Jogos

O uso de linguagens de script, naturalmente mais flexíveis e dinâmicas do que linguagens de programação tradicionais, tem se tornado mais comum aos programadores devido às diversas vantagens que elas oferecem, especialmente levando-se em consideração a crescente velocidade de processamento encontrada nos computadores modernos (O'CONNER, 2006). Essas linguagens fornecem usualmente um nível de abstração maior que linguagens estáticas como C++ e Java (CUTUMISU et al., 2007), podendo ser usadas de forma mais simples, facilitando, para o programador, o uso de funcionalidades e componentes pré-existentes (OUSTERHOUT, 1998). Dentre as vantagens do uso desse tipo de linguagem podemos citar uma maior facilidade para o desenvolvimento de inteligência artificial (SPRONCK et al., 2006), automação de sequências de animação (ZHANG; MCLAUGHLIN; KATCHABAW, 2007), lógica de jogo (RUSSELL; DONALDSON; SHEPPARD, 2008), script de história (CUTUMISU et al., 2007) e comportamento de ambientação[1] (CUTUMISU et al., 2006).

Linguagens de script e linguagens de programação tradicionais são complementares, sendo frequentemente utilizados juntos em *frameworks* de componentes (OUSTERHOUT, 1998). É importante usar uma linguagem de programação tradicional em união a linguagens de script, pois linguagens de script são geralmente interpretadas, o que leva a um desempenho menor do que o alcançado por linguagens de programação estáticas, nas quais são mais comumente implementados componentes especializados (CUTUMISU et al., 2007).

[1] Ambient Behaviours, inteligência artificial aplicada especificamente ao ambiente de jogo

Java foi escolhida por se tratar de uma linguagem que possui uma boa flexibilidade e consistência na tarefa de lidar com linguagens de script e carregamento dinâmico de classes (O'CONNER, 2006). Além disso, a sinergia entre a plataforma Java e linguagens de script produz um ambiente no qual desenvolvedores e usuários podem colaborar para criar aplicações mais dinâmicas e úteis (O'CONNER, 2006). Vale também citar que Java é uma linguagem de paradigma orientado a objetos - projetada desde o início com esse fim - e o uso de componentes reaproveitáveis em software é considerado uma "extensão natural" do paradigma de orientação a objetos (HOPKINS, 2000). No âmbito deste trabalho, é importante ressaltar que poucos domínios na computação mapeiam-se tão diretamente a um paradigma de programação como o desenvolvimento de jogos à orientação a objetos (PASSOS et al., 2009).

Java oferece diversas funcionalidades como independência de plataforma para portabilidade, o uso de um modelo de orientação a objetos, suporte a *multithreading*, suporte a programação distribuída e coleta de lixo automática (KAZI et al., 2000). Outras características de Java incluem tipagem estática, sintaxe derivada de C/C++ e compilação para um código intermediário que é executável independente da plataforma contanto que a máquina virtual Java esteja instalada. Vale citar que a independência de plataforma provê uma maior utilidade para este projeto, visto que os produtos desenvolvidos com base nesta abordagem oferecerão uma maior abrangência devido à sua portabilidade.

2.2 Sistemas Modulares e suas Vantagens

O desenvolvimento de jogos utilizando técnicas baseadas no uso de componentes reaproveitáveis é capaz de prover significante redução de custos e de tempo de desenvolvimento (FOLMER, 2007). Por esse motivo, a maior parte das *engines* de jogos é baseada na componentização de objetos e comportamentos, pois essa abordagem permite uma visualização clara da arquitetura do sistema, boa reusabilidade de código e prototipação rápida (PASSOS et al., 2009).

O uso de plataformas de desenvolvimento de jogos ou *engines* de jogo é frequente na indústria de jogos (ANDERSON et al., 2008). Ferramentas desse tipo são geralmente orientadas a componentes, pois a tarefa de decomposição funcional dos jogos é normalmente feita com o uso de modularização (TULIP; BEKKEMA; NESBITT, 2006).

As *engines* de jogo, de modo geral, têm buscado cada vez oferecer maior desempe-

nho para os jogos (TULIP; BEKKEMA; NESBITT, 2006), além de oferecer mais ferramentas voltadas a designers. Usualmente quando se referencia a ideia de plataformas modulares para desenvolvimento de jogos, o que se encontra são componentes, plugáveis em objetos de jogos, que são responsáveis por funcionalidades como inteligência artificial, física, renderização e outros aspectos de um determinado objeto de jogo. Esse objeto de jogo, por sua vez, faz as partes de um determinado elemento lógico no sistema, como um personagem, uma parte de um cenário, uma câmera, uma fonte de som ou a própria interface de usuário.

Uma das *engines* modulares proeminentes no mercado atual é a Unity. A Unity é uma ferramenta de autoria de jogos 3D para Mac e PC que permite o desenvolvimento de jogos para estas plataformas, além de jogos web, para o Nintendo Wii e para o Apple iPhone, com uma abordagem modular direcionada a designers de jogos (GOLDSTONE, 2009). Esta *engine* possibilita o desenvolvimento de jogos utilizando-se as linguagens C++, C # , Boo (derivada de Python) e JavaScript, e a modularidade da mesma está diretamente relacionada ao uso de *GameObjects*, objetos de jogos construídos a partir de diversos componentes que ofertam funcionalidades relacionadas à renderização, física, som e outras funcionalidades (GOLDSTONE, 2009). A figura 2.1 mostra a parte da interface da Unity relacionada a um objeto de jogo. Nesta imagem pode ser visto que um objeto de jogo equivalente a uma câmera tem conectado a si componentes relacionados a funcionalidades como áudio e interface de usuário. Os diversos componentes possuem atributos configuráveis que definem o seu comportamento, podendo também possuir código atrelado aos mesmos na forma de scripts.

Outra *engine* modular para desenvolvimento de jogos proeminente é a Unreal Engine. Este *framework* de desenvolvimento provê uma vasta gama de tecnologias de base, ferramentas de criação de conteúdo, concentrando no uso feito por artistas e designers, sendo usado por empresas como a Microsoft e a 2K Games, além de também ser utilizada pela televisão e pelo cinema (LACORT, 2009). É uma *engine* altamente expansível e modular, suportando animação, física, áudio e efeitos de partículas, entre outras funcionalidades (LACORT, 2009). Para programação, ela fornece a sua própria linguagem de script, a UnrealScript.

Assim como a Unity, a Unreal Engine utiliza a abordagem usual para sistemas modulares de desenvolvimento de jogos, adotando objetos de jogos denominados Atores

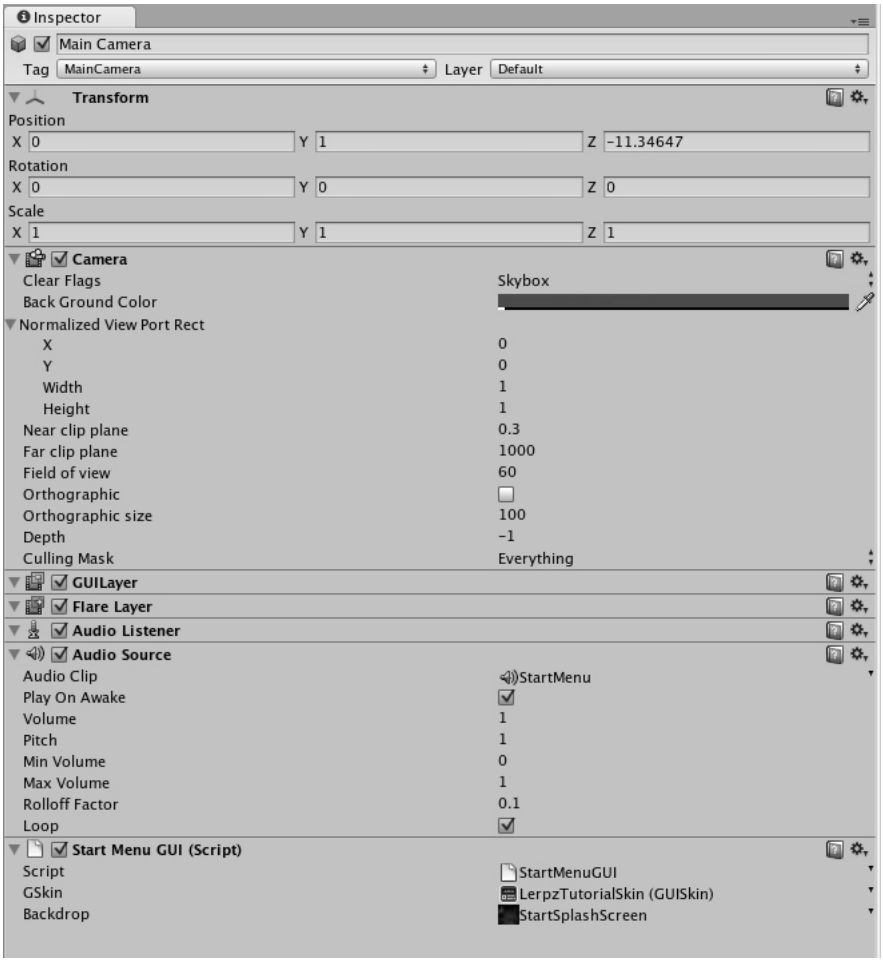

Figura 2.1: Interface da Unity

que possuem uma série de propriedades e componentes conectados ao mesmo. A figura 2.2 mostra uma lista de atores disponíveis em uma determinada cena na Unreal Engine. Assim como os *GameObjects* presentes na Unity, cada ator possui suas propriedades e componentes modulares atrelados aos mesmos.

A abordagem de uso de componentes reaproveitáveis proposta para a plataforma da qual trata este trabalho, contudo, é diferente da abordagem habitual encontrada nas *engines* de jogos citadas até o momento. Ao invés de definir objetos de jogo onde determinados módulos são plugados, a plataforma define somente que cada módulo é uma unidade independente, especializada e intercambiável que provê uma funcionalidade específica para os outros módulos no sistema. Essa abordagem busca, além de ser voltada especificamente para programadores, fornecer uma plataforma mais simples para a tarefa de desenvolvimento de código, facilitando o desenvolvimento de jogos com foco no código e não em ambientes específicos de desenvolvimento. Isto também oferece uma maior liberdade ao programador, que pode definir como funcionam as relações entre os módulos

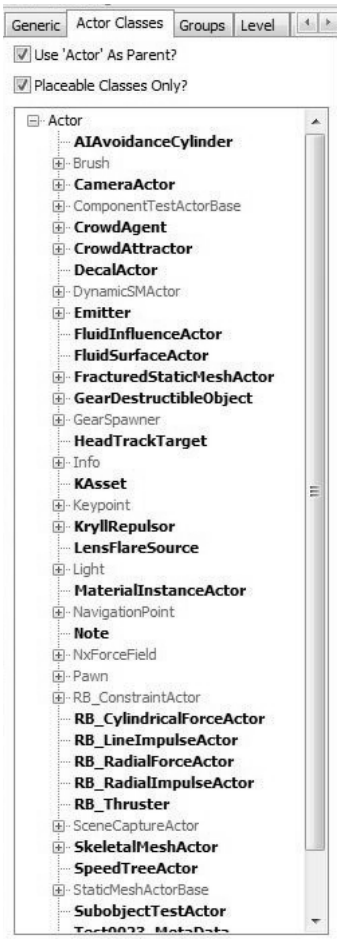

Figura 2.2: Lista de atores disponíveis em uma cena na Unreal Engine

do sistema da forma que for mais adequada para o seu problema.

Para o suporte a componentes reaproveitáveis, uma das formas mais proeminentes de uso de módulos em Java aparece na forma da plataforma OSGi, que é uma plataforma de serviços para a criação de programas modulares orientados a serviço em Java. É uma plataforma composta por quatro camadas - camada de segurança, camada de modularização, camada de ciclo de vida e camada de serviço - que foi criada para oferecer uma solução ao problema de modularização em Java (CHAPPELL; KAND, 2009). O problema com OSGi, do ponto de vista deste projeto, é justamente a abundância de serviços, interfaces e camadas disponibilizados pelo mesmo que, embora sejam muito úteis para projetos como o Eclipse, um dos principais projetos que utilizam essa tecnologia, acabariam adicionando complexidade desnecessária à plataforma proposta; a mesma se beneficiaria muito mais de uma abordagem mais simplificada.

Até a versão 1.6 da plataforma Java, contudo, OSGi ainda era a melhor opção. Já com essa versão, tornou-se muito mais viável e simples utilizar a funcionalidade da Ap-

plication Programming Interface (API) ServiceLoader de Java, uma vez que essa API, existentes desde a versão 1.2, foi revista e aprimorada na nova versão. É muito mais simples suportar plataformas baseadas em componentes em Java usando esta API (HUGHES, 2007), que permite um gerenciamento simples de módulos que podem ser descobertos, carregados e usados dinamicamente em tempo de execução. A partir desta análise, é possível afirmar que esta API apresenta funcionalidade suficiente para atender às necessidades da plataforma proposta por este trabalho. Isso possibilita evitar a complexidade inerente à plataforma OSGi.

De posse das informações obtidas, o gerenciador de módulos, com ajuda do ServiceLoader, é o responsável pela localização de serviços conforme a necessidade, o que permite que seja usada a implementação mais adequada de um determinado módulo (FOWLER, 2004). Dessa forma, é possível criar a plataforma de tal forma que os módulos utilizem as funcionalidades de outros componentes do sistema delegando para o gerenciador de módulos a tarefa de encontrar o componente que irá prover essas funcionalidades. Com isso, torna-se possível um maior nível de abstração para o programador.

2.3 A Prototipação Rápida no Contexto de Desenvolvimento de Jogos

Na indústria atual de jogos, a competição está intensificando e, com isso, encorajando programadores a atender as novas demandas em prazos mais curtos; projetistas estão, portanto, buscando validar seus produtos o mais cedo possível durante o desenvolvimento. O uso de protótipos oferece respostas práticas a esses dois problemas, especialmente quando unido ao uso de sistemas baseados em componentes com ênfase no reuso (TESSIER et al., 2003).

Devido a isso, desenvolvedores de software constantemente dedicam atenção ao projeto de ferramentas de prototipação de software, com a esperança de que o uso desse tipo de ferramenta possa aumentar a fluidez de desenvolvimentos de natureza iterativa (SCHRAGE, 1996). Isso é bastante válido também no âmbito de desenvolvimento de jogos, pois o uso de protótipos reduz o risco de investimento em um novo jogo. Um dos valores da prototipação na fase inicial de um projeto desse tipo é que ela pode revelar logo no início se o jogo irá ou não funcionar e ser divertido (FEDEROFF, 2006).

Permitir a prototipação rápida de novas funcionalidades e elementos de jogo é algo muito importante para se permitir explorar novas ideias bem como encontrar falhas pre-

sentes no planejamento do jogo (KOIVISTO; ELADHARI, 2006) (OLLILA; SUOMELA; HOLOPAINEN, 2008). O processo de desenvolvimento e evolução de sistemas de software complexos é intrinsecamente exploratório por natureza; por isso, atividades de prototipação são inevitáveis em cada estágio do processo (GOLDMAN; NARAYANASWAMY, 1992). É, portanto, desejável a disponibilidade de ferramentas que possibilitem e suporte um processo simplificado de criação de protótipos.

3 DESENVOLVIMENTO E IMPLEMENTAÇÃO

3.1 Projeto Inicial

O primeiro passo para o desenvolvimento da proposta deste trabalho é o delineamento da estrutura geral da plataforma objetivada pelo mesmo. Para tal, foi considerada a possibilidade de implementar os módulos em Java, usando uma linguagem de script para realizar o controle dos mesmos e a comunicação entre eles. Dessa forma, os módulos, que teriam a implementação das partes mais importantes e que demandam mais recursos do sistema, seriam escritos em uma linguagem compilada e com desempenho superior a linguagens de script. Além disso, o desenvolvimento de um software sobre essa plataforma seria feito com uma linguagem de script, mais dinâmica e acessível, já que seria a mesma a responsável pelo controle e uso das funcionalidades providas pelos módulos do sistema. Uma visão geral de como esse sistema foi proposto é mostrado na figura 3.1. Na mesma, podemos ver que a linguagem de script faz o papel de criar, acessar e lidar com a comunicação entre os módulos, que são escritos em Java. A lógica do jogo em si é implementada somente na linguagem de script, e os módulos não são capazes de se comunicar diretamente entre si. Esta é uma abordagem habitual nesse tipo de sistema, pois linguagens de script normalmente são usadas para unir funcionalidades providas por outros componentes (OUSTERHOUT, 1998).

Apesar das vantagens oferecidas por este método, existem limitações no mesmo, afetando negativamente o uso da plataforma, que se tornaria bem mais restritiva ao desenvolvedor que fosse utilizar a mesma. Algumas das limitações são:

- Não fica claro de quem é a responsabilidade de gerenciar a comunicação entre os módulos. Cada módulos teria que se comunicar com outros módulos diretamente, já que o centro de comunicação, representado pela linguagem de script, está no controle do desenvolvedor que usa a plataforma;

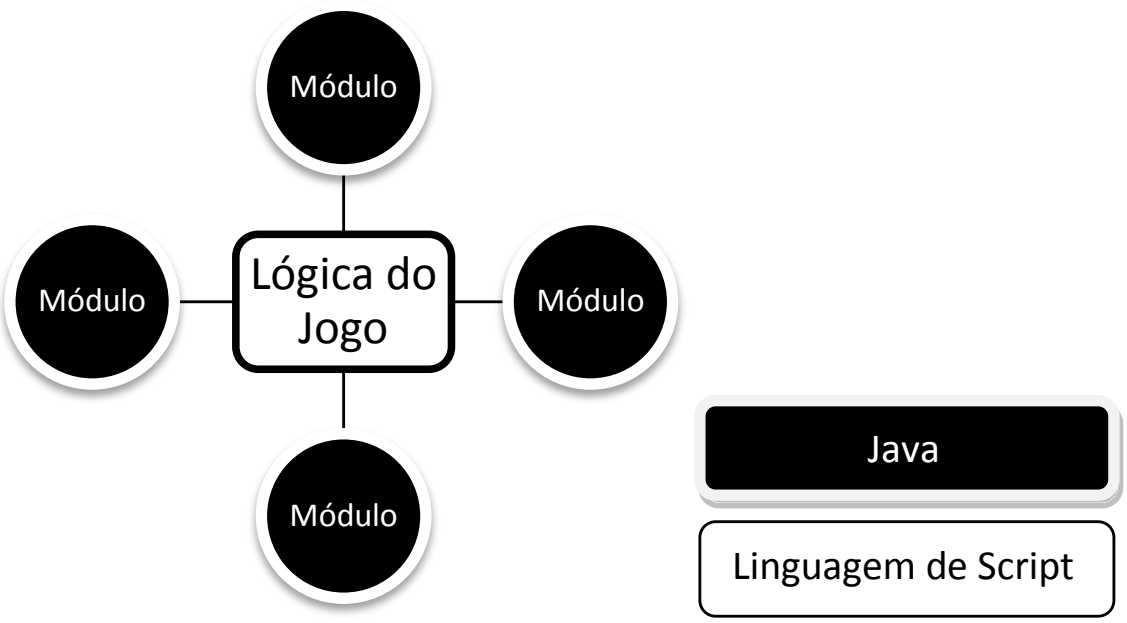

Figura 3.1: Estrutura inicialmente idealizada para a plataforma

- É usada somente uma linguagem de script para a qual as interfaces dos módulos em Java são disponibilizadas. Isso cria o problema de decidir qual linguagem de script a ser utilizada, além de limitar a escolha do programador e os potenciais usuários da plataforma;

- Os módulos são obrigatoriamente implementados em Java, mesmo que haja uma linguagem de script mais adequada para a funcionalidade deles. Em alguns casos, isso poderia inclusive passar às mãos do programador a responsabilidade de implementar certas funcionalidades, que deveriam estar isoladas em módulos, no seu próprio software. Deseja-se que o programador crie módulos utilizando linguagens de script;

- O software construído em cima da plataforma deve obrigatoriamente ser desenvolvido usando a linguagem de script suportada. Deve ser possível não só escolher a linguagem de script mais adequada para o problema como deveria também ser possível escrever o sistema usando somente Java.

Para resolver esses problemas e limitações, foi delineada uma nova estrutura para o desenvolvimento da plataforma. Primeiro, decidiu-se criar um elemento central de controle da plataforma, o gerenciador de módulos. Esse componente, escrito em Java, é o núcleo da plataforma, sendo responsável por carregar os módulos e gerenciar as comunicações

entre eles.

Em seguida, o software escrito pelo programador sobre a plataforma é considerado como simplesmente mais um módulo. Ele é carregado pelo gerenciador de módulos e se comunica com os módulos que precisa usar da mesma forma que os outros componentes se comunicam. Essa comunicação é feita através do gerenciador de módulos que provê as ferramentas necessárias para tal.

Por fim, os módulos são escritos em Java ou em uma linguagem de script suportada pela plataforma, não limitando os mesmos a uma única linguagem específica. Dessa forma, é permitido ao programador possuir diferentes módulos escritos em diferentes linguagens interagindo entre si. Como o programa escrito sobre essa plataforma seria somente mais um módulo, isso também permite que ele seja escrito tanto em Java como em qualquer uma das linguagens de script suportadas pela plataforma.

Uma visualização desta estrutura pode ser vista na figura 3.2.

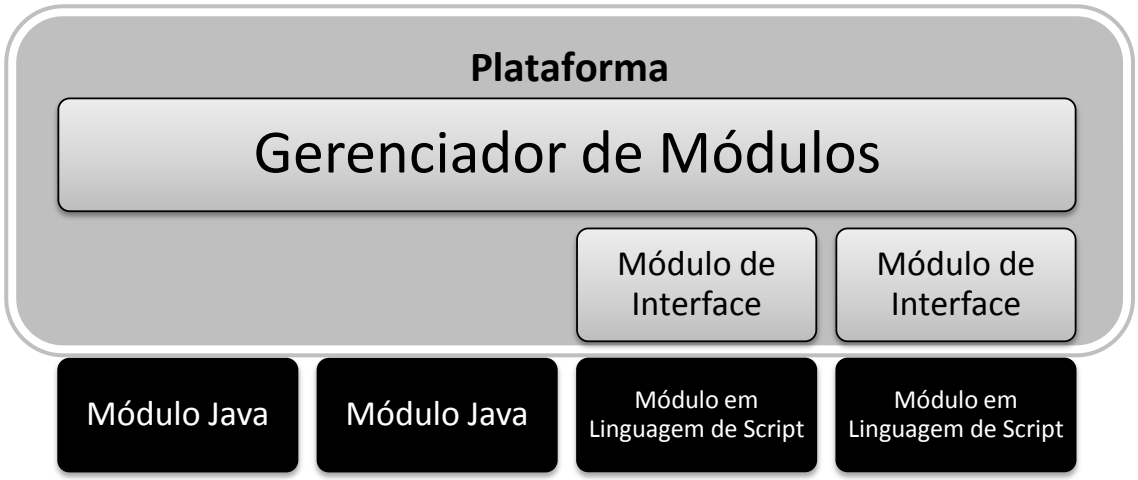

Figura 3.2: Estrutura atual da plataforma

Como pode ser visto na figura que mostra a estrutura atual da plataforma, são adicionados pela plataforma, em tempo de execução, módulos de interface, que realizam a tarefa de conectar os componentes de script ao resto do sistema. Tais módulos são parte integrante da plataforma, sendo instanciados a partir de uma só implementação capaz de lidar com as diferentes linguagens de script. Isto é feito para permitir que os módulos escritos nessas linguagens sejam vistos pelo sistema da mesma forma que os módulos escritos em Java. Esses componentes de interface, além de oferecer o suporte direto aos componentes de script, também realizam as tarefas necessárias para comunicação com estes módulos, incluindo gerenciamento de chamadas e eventos e tradução de argumentos.

A seção seguinte mostra como as especificações discutidas até o momento são implementadas e as tecnologias aplicadas com essa finalidade.

3.2 Implementação

Acerca da implementação das especificações da plataforma proposta, as duas questões principais são relativas à comunicação entre os módulos presentes e o suporte a diferente linguagens de script, além da comunicação dos módulos escritos nestas linguagens com os módulos escritos em Java.

3.2.1 Comunicação Entre os Módulos

A comunicação entre os módulos da plataforma é feita utilizando-se dois tipos básicos de troca de mensagens. O primeiro é a chamada de métodos, onde um componente efetua uma chamada de função que é atendida por outro módulo, podendo enviar argumentos com esta chamada e receber o retorno da execução do método chamado. O segundo tipo é o despacho de eventos, onde um módulo no sistema pode gerar um evento, opcionalmente acompanhado de argumentos, que é enviado, pela plataforma, a todos os componentes do sistema que registraram-se com o gerenciador de módulos como ouvintes do evento em questão.

A primeira solução considerada no projeto para implementar as funcionalidades relativas à comunicação entre componentes foi a definição de interfaces de módulos específicas que explicitassem os métodos a serem ofertados pelos componentes que implementassem as mesmas. Com isso, é possível criar diferentes interfaces, como, por exemplo, uma interface para métodos relacionados a renderização 3D, uma interface relacionada a input e uma interface relacionada a som. Para implementar um novo módulo no sistema, seria necessário que o mesmo implementasse uma dessas interfaces pré-existentes na plataforma. Com isso, é possível acessar funcionalidades providas por outros componentes através do uso dos métodos definidos nas interfaces. A implementação dessa abordagem pode ser feita com o uso da funcionalidade de Interfaces de Java, que é imediatamente suportada pela API ServiceLoader, tornando esta uma solução de efetivação simples.

Essa abordagem, porém, é muito limitada e apresenta pouca possibilidade de expansão para a plataforma sendo desenvolvida. Com essa metodologia, só é possível implementar módulos referentes a funcionalidades já previstas na plataforma. Para criar um

tipo de módulo inteiramente novo, é necessário alterar a plataforma para adicionar uma nova interface. Isso gera uma nova versão da plataforma e, com isso, passam a ser possíveis problemas de compatibilidade dos módulos com as diferentes versões da plataforma. Esse tipo de problema se agrava mais ainda no caso de surgir a necessidade de alterar uma interface para adicionar, remover ou modificar métodos existentes, pois uma tarefa dessas torna incompatíveis módulos implementados seguindo as especificações anteriores da interface. Esses problemas, em adição à complexidade deste processo, vão contra as propostas da plataforma que buscam soluções de uso simples e que ofereçam o mínimo de restrições possíveis ao programador.

Tendo em vista os problemas encontrados com essa primeira abordagem, foi planejada e desenvolvida uma nova solução para a comunicação entre os módulos que é capaz de prover ao programador a simplicidade e liberdade almejadas pela plataforma. A nova solução reduz a quantidade de restrições associada com a especificação de interfaces ou tipos de módulos da plataforma, possibilitando que sejam desenvolvidos componentes com os mais variados tipos de utilidades mesmo que não sejam previstos originalmente pela mesma.

A solução implementada consiste na resolução de chamadas de métodos sendo feita internamente pela plataforma, ao invés de ser delegada aos mecanismos habituais de Java. Isso oferece à plataforma o controle sobre as funcionalidades oferecidas pelos módulos, permitindo à mesma repassar as chamadas de função para o componente correto.

Esta nova solução faz com que um determinado módulo, durante a sua inicialização, registre com o gerenciador de módulos os nomes dos métodos que deseja oferecer ao sistema. Quando um componente deseja chamar um método oferecido por outro componente, ele envia a chamada de função ao gerenciador de módulos acompanhada dos argumentos do método. A plataforma, por sua vez, descobre o módulo que responde pelo método chamado. Nesse momento, é feita uma tradução automática dos métodos para os tipos de dados corretos, se necessário, e a chamada é encaminhada ao módulo provedor. Por fim, o retorno do método é encaminhado pelo gerenciador de módulos de volta para o componente que realizou a chamada.

Essa solução, muito embora incorra na adição de *overhead*, não só soluciona o problema proposto evitando as limitações da solução anterior como também oferece várias novas ferramentas para a plataforma desempenhar suas funções. Como a comunicação

entre os módulos e feita através da plataforma, é possível para a mesma oferecer novas funcionalidades como rastreamento de chamadas de funções, gerenciamento de erros, tradução de argumentos e seleção de provedores de métodos. Isso também permite uma integração mais adequada com linguagens de script. Como a plataforma é escrita em Java, é necessário somente que Java seja capaz de se comunicar com componentes escritos em outras linguagens, já que a comunicação entre os módulos é transparente para os componentes em si. Para estes, a comunicação é feita diretamente com a plataforma. Como ela decide por si só como resolver as chamadas de métodos, os detalhes de comunicação entre módulos escritos em diferentes linguagens são abstraídos para os mesmos.

Outra vantagem oferecida por esta solução é que ela permite ao programador realizar chamadas de métodos que não estão ainda disponíveis na plataforma, visto que essas chamadas não geram erros fatais. Isso torna fácil ao programador remover funcionalidades inteiras do sistema simplesmente removendo o módulo provedor, caso deseje. Essa funcionalidade também auxilia o trabalho em equipe, uma vez que é possível escrever código que usa funcionalidades presentes em módulos que estão sendo desenvolvidos simultaneamente por outro programador antes mesmo deste terminar a sua produção de código.

A figura 3.3 mostra uma visão geral do fluxo resultante de uma chamada de método na plataforma.

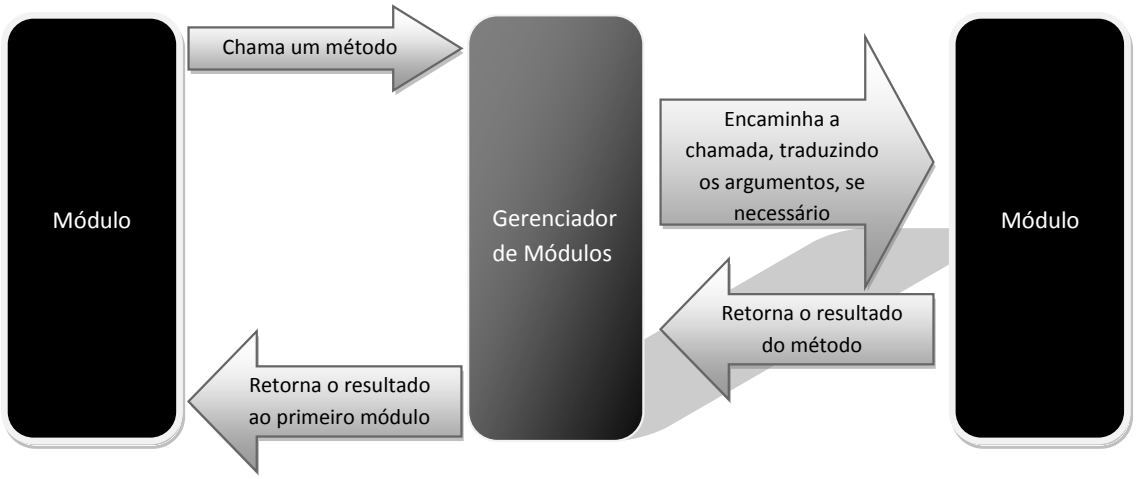

Figura 3.3: Fluxo de mensagens em uma chamada de método

Uma outra funcionalidade bastante importante e útil, especialmente no desenvolvimento de jogos, é o suporte a geração e despacho de eventos. Adicionar esta funcionalidade à plataforma mostrou-se bastante simples uma vez que já havia o suporte a chamadas

de métodos, pois boa parte do que precisou ser implementado foi reaproveitado dessa funcionalidade.

Os eventos, assim como os métodos, são identificados unicamente no sistema pelos seus nomes. Cada componente pode registrar, junto ao gerenciador de módulos, o nomes dos eventos dos quais deseja receber notificações caso os mesmos sejam despachados no sistema. Quanto um módulo (ou a própria plataforma) gera um novo evento, a plataforma faz o trabalho de buscar todos os componentes que pediram para ser notificados do mesmo e encaminha a notificação a todos.

Como a implementação da comunicação pelo uso de chamadas de métodos já havia criado suporte ao transporte de argumentos de funções entre os componentes, essa funcionalidade foi também reaproveitada para o sistema de eventos. Cada evento gerado no sistema pode ter agregado a si uma lista de parâmetros. Esses argumentos são repassados aos módulos que recebem a notificação do evento. Assim como na chamada de funções, a plataforma executa quaisquer traduções de argumentos que possam vir a ser necessárias.

A figura 3.4 mostra uma visão geral do fluxo resultante de um despacho de evento na plataforma.

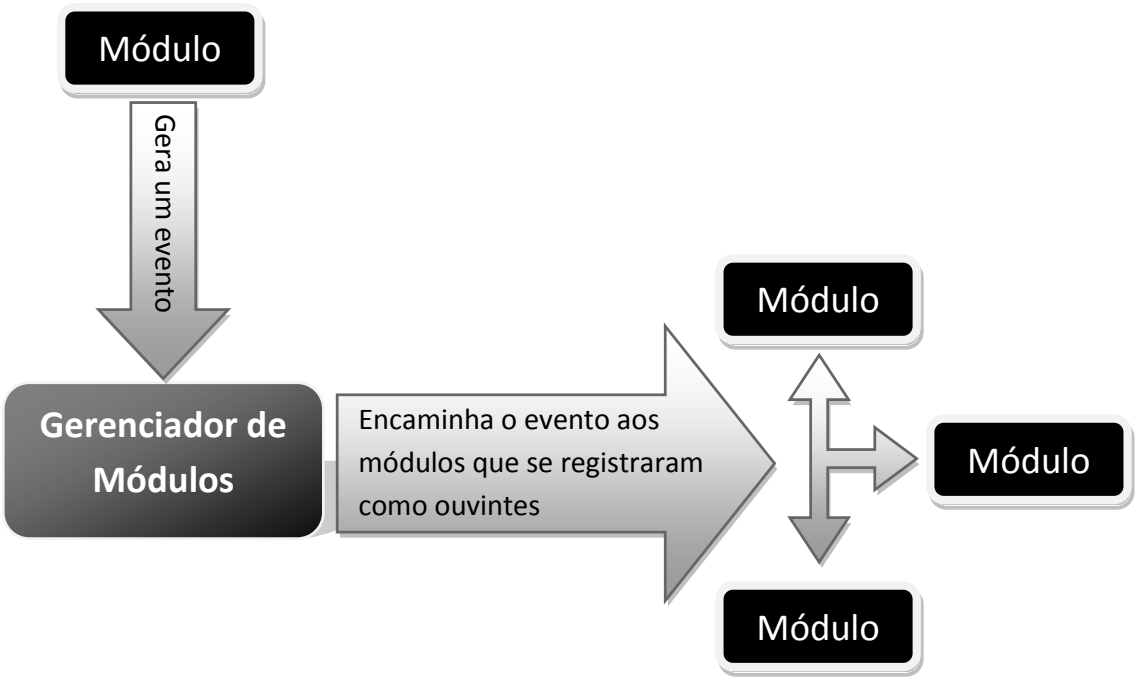

Figura 3.4: Fluxo de mensagens em um despacho de evento

3.2.2 Suporte a Linguagens de Script

As soluções definidas e implementadas no tocante à comunicação entre os módulos simplificam várias questões relacionadas ao suporte da plataforma a diferentes linguagens de script. Uma vez que é responsabilidade da mesma gerenciar, processar, encaminhar e traduzir a comunicação entre os componentes do sistema, e visto que a plataforma é escrita em Java, o desafio do suporte a outras linguagens se resume na questão de como integrar Java com linguagens de script.

O uso de linguagens dinâmicas e de script em Java é uma funcionalidade disponível nativamente na versão 1.6 da plataforma Java e é descrito pela Java Specification Request (JSR) 223. O JSR 223 fornece várias novas capacidades no uso de linguagens dinâmicas, permitindo o carregamento e execução de scripts, acesso de objetos Java a partir desses scripts, passar argumentos do Java para os scripts e acessar objetos do script a partir do Java (O'CONNER, 2006). A tabela 3.1 mostra uma lista das diversas linguagens de script atualmente suportadas por esta tecnologia (GROGAN; SUNDARARAJAN; VENKATARAMANAPPA, 2010).

Com a integração desta tecnologia à plataforma, ela passou a ser capaz de reconhecer, em tempo de execução, os módulos disponíveis e as linguagens nos quais os mesmos foram escritos, sem a necessidade do uso de metadados adicionais sobre os módulos. A partir disso, foi desenvolvido um sistema de abstração, representado pela existência dos módulos de interface mostrados na figura 3.2, que faz com que os módulos escritos em linguagens de script funcionem como os já existentes módulos em Java. Esse sistema passou a ser responsável não só por encaminhar a comunicação destinada aos novos componentes, realizando quaisquer traduções necessárias, como também por fornecer ao módulo uma interface simples que permitisse a ele comunicar-se com o gerenciador de módulos e, por consequência, com os outros componentes do sistema.

A seguir são descritos dois cenários que mostram o processo que ocorre na plataforma quando são feitas certas chamadas de métodos.

O primeiro cenário, mostrado na figura 3.5, é referente a quando um módulo em Java chama um método que é disponibilizado por um módulo escrito em linguagem de script. Quando isso ocorre, o processo executado pela plataforma é o seguinte:

- Um módulo escrito em Java chama um método disponibilizado por um módulo escrito em linguagem de script. Esta chamada é enviada ao gerenciador de módulos;

Linguagem	Implementação	Suporte oficial
ABCL	Armed Bear Common Lisp	Não
AppleScript	Java Applescript Connector	Não
AWK	Jawk 0.14	Sim
BeanShell	BeanShell 2.0b5	Sim
CajuScript	CajuScript	Não
ejs	Arquivo JavaScript único	Sim
Bex Script	Bex Script	Não
FreeMarker	Freemarker 2.3.11	Sim
Groovy	Groovy 1.5.6	Sim
Jaskell	Jaskell 1.0	Sim
Java	Java Compiler API (JSR199)	Sim
JavaFX	OpenJFX Compiler	Não
JavaScript	Rhino 1.6R7	Sim
JavaScript (Web Browser)	Implementação do navegador	Sim
Jelly	Jelly1.0	Sim
JEP	JEP 2.4.0	Sim
Jexl	Jexl 1.0	Sim
jst	TrimPath JavaScript Templates 1.0.38	Sim
JudoScript	JudoScript 0.9	Sim
JUEL	JUEL 2.1.0-rc2	Sim
MathEclipse	MathEclipse	Não
OCaml	OCaml Scripting Project	Não
OGNL	OGNL 2.6.9	Sim
PHP (Java)	Quercus	Não
PHP (Nativo)	PHP Java Bridge	Não
Pnuts	Pnuts 1.1	Sim
Python (Java)	Jython 2.2.1	Sim
Python (Nativo)	Java Embedded Python	Não
Ruby	JRuby 1.1	Sim
Scheme	SISC1.16.6	Sim
Sleep	Incluso com JSR223	Sim
Smalltalk	Athena	Não
Tcl	Jacl 1.3.3	Sim
Velocity	Velocity 1.5	Sim
XPath	javax.xml.xpath	Sim
XSLT	javax.xml.transform	Sim

Tabela 3.1: Linguagens atualmente suportadas pela JSR 223

- O gerenciador de módulos descobre qual o componente que fornece o método chamado e encaminha a chamada ao mesmo;
- O módulo que recebe a chamada é, na realidade, um módulo de interface, que realiza a tarefa de encaminhar a mesma ao script gerenciado por ele;
- O script executa o método sobre os argumentos recebidos e retorna o valor, se houver, ao módulo de interface;
- O módulo de interface retorna o valor devolvido pelo script ao gerenciador de módulos;
- O gerenciador de módulos encaminha o valor devolvido ao módulo que fez a chamada.

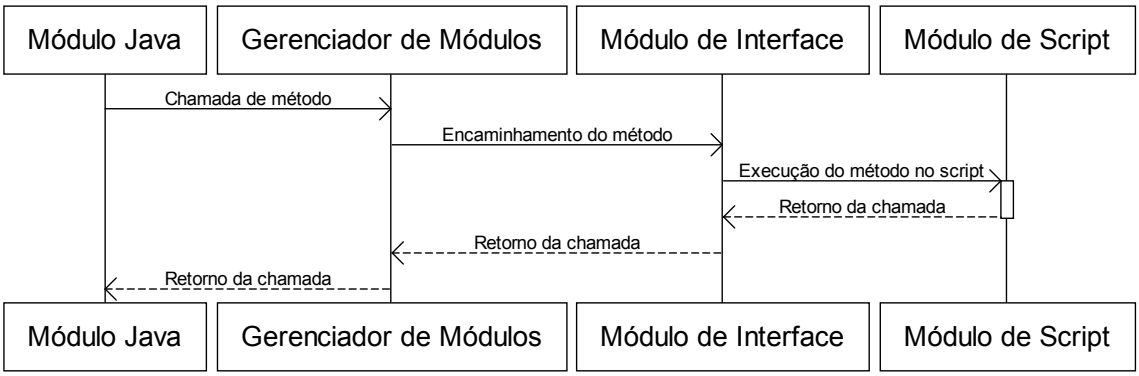

Figura 3.5: Chamada de método feita por um módulo Java para um módulo de script

O segundo cenário, similar ao primeiro, é mostrado na figura 3.6. Ele descreve o processo que ocorre quando um módulo escrito em linguagem de script chama um método fornecido por um módulo Java. Nesta situação, os passos de execução são os seguintes:

- Um módulo escrito em linguagem de script chama um método disponibilizado por um componente escrito em Java. Esta chamada é interceptada pelo módulo de interface;
- O módulo de interface verifica se é necessário realizar a tradução dos argumentos do método, realiza estas se necessário e encaminha a chamada ao gerenciador de módulos;

- O gerenciador de módulos descobre qual o componente que fornece o método chamado e encaminha a chamada ao mesmo;
- O componente Java executa o método e retorna o resultado ao gerenciador de módulos;
- O gerenciador de módulos encaminha o resultado ao módulo de interface;
- O módulo de interface devolve o resultado da chamada ao script, que retoma a execução de onde parou.

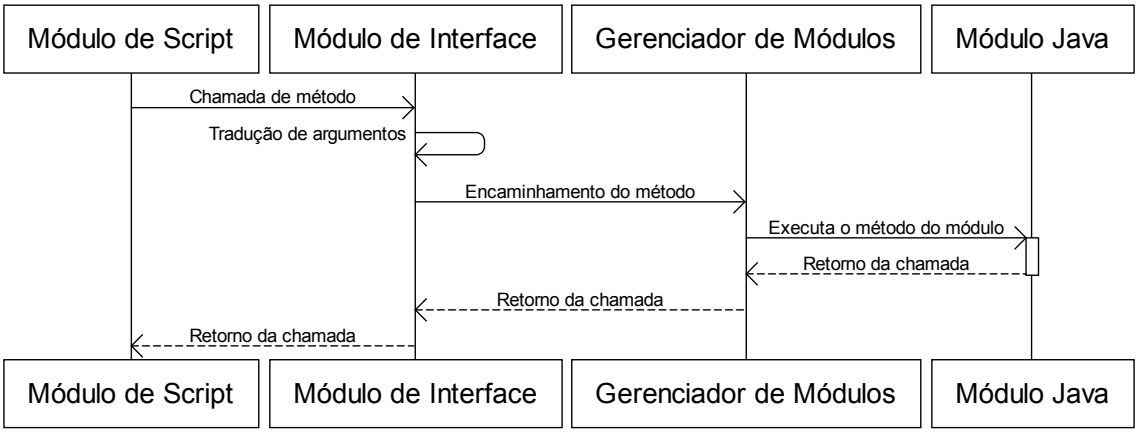

Figura 3.6: Chamada de método feita por um módulo de script para um módulo Java

A especificação oferecida pela JSR 223 é parte da plataforma Java e possibilita que seja implementado suporte às mais variadas linguagens dinâmicas e de script, bastando seguir as especificações da mesma. As partes da especificação que são opcionais para o suporte a uma nova linguagem não foram utilizadas pela plataforma. Com isso, tornou-se possível o suporte pela mesma a qualquer linguagem de programação que siga a especificação mínima da JSR 223, sem a necessidade de nenhuma funcionalidade ou característica específica para a plataforma. Isso permitiu que a plataforma automaticamente passasse a suportar as mais de 25 linguagens que já são parte do projeto relacionado à JSR 223, dentre as quais podemos citar Javascript, Ruby, Python, Groovy e PHP.

3.2.3 Tradução de Argumentos

Durante o desenvolvimento da integração das linguagens de script com Java, foi constatado que, em algumas situações, ocorre um problema relacionado à incompatibilidade de argumentos. Testes mostraram que o erro ocorre somente em alguns casos quando

módulos escritos em linguagens de script realizam chamadas a métodos disponibilizados por módulos escritos em Java.

A pesquisa feita sobre esse problema levou à conclusão de que as implementações em Java do suporte às variadas linguagens de script lidam de forma diferente com o envio de variáveis numéricas. Ao se efetuar uma chamada de método para o Java passando como argumento um número, cada linguagem converte o valor para um tipo diferente de acordo com sua própria implementação. Em alguns casos são enviados tipos primitivos como *int* ou *float*, enquanto que em outros casos são enviados objetos de classes como *Long* ou *Double*. Isso gera conflito quando os métodos em Java são programados para receber variáveis, por exemplo, do tipo *int* e o módulo em linguagem de script envia valores numéricos encapsulados em objetos do tipo *Double*.

As primeira soluções consideradas para esse problema envolvem a criação da regra de sempre se declarar os métodos em Java como recebendo variáveis do tipo *Double* ao invés de qualquer outro tipo numérico. Isso, contudo, além de criar uma nova restrição à plataforma e possivelmente criar incompatibilidade com código já existente, não resolve todos os caso, pois, dependendo da linguagem de script usada pelo módulo, essas variáveis podem ser enviadas encapsuladas em formatos diferentes, isto é, algumas linguagens utilizam *Long* enquanto outras linguagens utilizam *Double* mesmo para números inteiros.

A solução encontrada para esse problema foi a criação de um método de tradução de argumentos que captura as chamadas de métodos que partem de módulos escritos em linguagens de script e que são destinadas a módulos escritos em Java. Toda vez que uma chamada desse tipo é encontrada, a plataforma verifica os tipos dos argumentos de origem e de destino antes de encaminhar a chamada ao componente Java. Nos casos onde é necessária uma conversão de argumentos, esta é feita automaticamente. Por fim, a chamada é encaminhada com os novos argumentos.

3.2.4 Coleta de Lixo

Enquanto eram feitos testes envolvendo módulos escritos em linguagens de script controlando a lógica do jogo, foi percebido que haviam problemas sérios de desempenho durante a execução do programa. O sintoma era uma pausa intermitente que durava uma fração de segundo e que ocorria com uma frequência que variava de acordo com a linguagem sendo usada. Em um dos testes, utilizando Ruby, o programa sofria esse tipo de

problema com uma frequência de aproximadamente 50 vezes por segundo, o que afetava negativamente a taxa de quadros por segundo.

O problema foi investigado e o diagnóstico a que se chegou é de que a causa das pausas é a coleta de lixo efetuada pelo Java. Com uma frequência relativamente alta de comunicação sendo efetuada com o componente em script, a máquina virtual realiza uma alocação de memória frequente durante a execução do programa, gerando lixo a ser coletado constantemente durante a execução. De tempos em tempos a máquina virtual precisa, então, efetuar a coleta, pausando a execução do programa enquanto realiza esta tarefa.

Esse problema foi resolvido reconfigurando-se a máquina virtual Java para usar o coletor de lixo concorrente de baixa interrupção (Concurrent Mark Sweep Garbage Collector) e o coletor paralelo de geração jovem (Parallel Young Generation Collector) para a coleta de lixo de gerações antigas e jovem, respectivamente. Como pode ser visto na figura 3.7, esses dois métodos de coleta operam de forma diferente dos coletores padrão de Java, sendo capazes de executar corretamente sem a necessidade de interromper o fluxo de execução do programa.

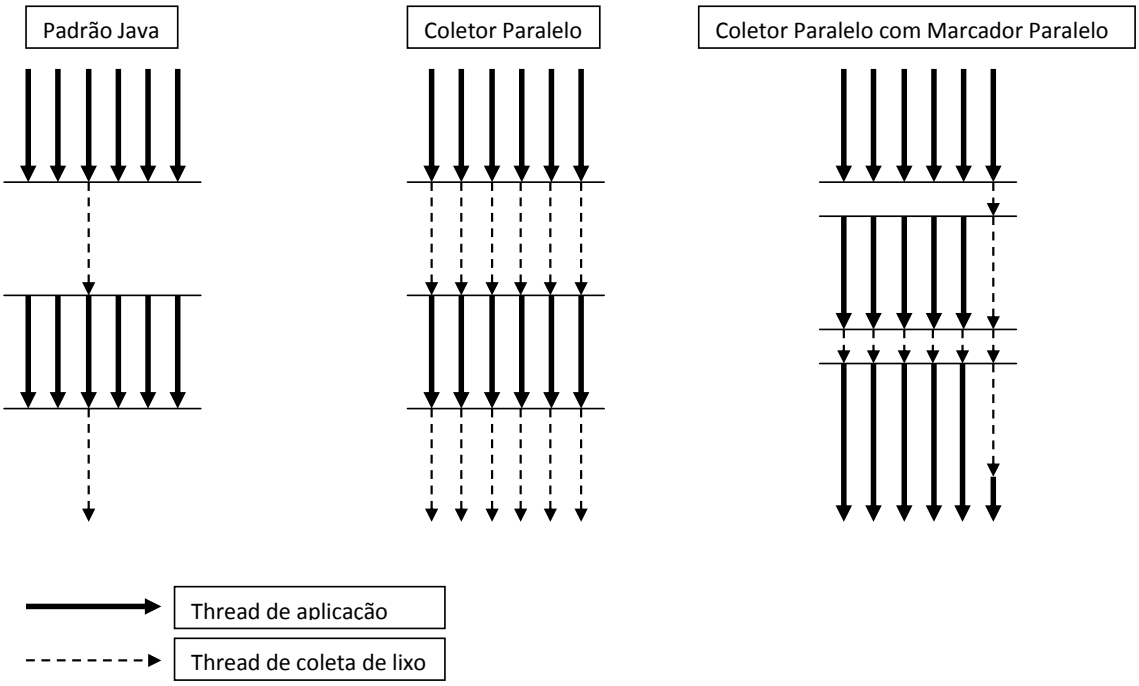

Figura 3.7: Coletores de lixo em Java

3.2.5 Suporte a Bibliotecas Java e Bibliotecas Nativas

Como último item essencial para a plataforma, foi planejado e implementado na plataforma o suporte a acesso de bibliotecas por parte dos módulos. Essas bibliotecas precisam ser descobertas e carregadas em tempo de execução pela plataforma para que suas funcionalidades estejam disponíveis para os componentes que precisem das mesmas.

O primeiro tipo de biblioteca a ser suportado foi as bibliotecas em Java. Esses elementos são tradicionalmente distribuídos em arquivos do tipo Java Archive (JAR) que contêm classes Java compiladas. Para possibilitar o suporte a este tipo de biblioteca, foi escrito um *Classloader* customizado para a plataforma que busca essas bibliotecas em locais específicos dentro dos diretórios dos módulos do sistema. Ao encontrar arquivos JAR, essa ferramenta carrega automaticamente as classes disponibilizadas pelos mesmos para que suas funcionalidades estejam disponíveis para os módulos antes destes serem inicializados.

O segundo tipo de biblioteca a ser suportado se trata das bibliotecas disponíveis em código nativo escritas em linguagens como C, C++ e assembly, que podem ser encontrado em biblitotecas Java como JOGL e JInput. Esse suporte foi adicionado com o uso de uma funcionalidade da plataforma Java chamada Java Native Interface (JNI), que possibilita a incorporação de código nativo em programas escritos em Java (LIANG, 1999). O *Classloader* implementado para o carregamento de bibliotecas Java foi expandido para ser capaz de também descobrir e carregar este novo tipo de biblioteca, encontrando arquivos com o código nativo em locais específicos nos diretórios dos módulos. Como bibliotecas nativas dependem do sistema operacional no qual a plataforma é executada, há suporte para código destinado a diferentes sistemas que é carregado de acordo com a extensão do arquivo da biblioteca, que pode ser *.dll* para Windows, *.so* para Linux ou *.jnilib* para Mac OS X.

4 VERIFICAÇÃO E VALIDAÇÃO DOS RESULTADOS

Este trabalho trata do desenvolvimento de uma plataforma que atenda aos objetivo principal de permitir o desenvolvimento de jogos usando componentes reaproveitáveis. A plataforma deve ser dirigida principalmente a programadores, para os quais deve oferecer uma forma simples e rápida de se desenvolver jogos e protótipos. Os módulos devem funcionar de forma independente e devem ser intercambiáveis.

Avaliando o quesito da simplicidade da plataforma, podemos considerar o trabalho necessário para a criação de um novo jogo. Primeiro, o programador precisa copiar para o diretório “Modules” os diretórios dos componentes que ele deseja usar. Em seguida, ele irá criar um novo diretório com o nome do jogo, pois o jogo em si é também um módulo. Depois disso, ele irá criar, dentro desse novo diretório, um arquivo cujo nome é o mesmo nome do módulo e cuja extensão reflete a linguagem de programação que ele deseja usar. Por fim, ele irá criar uma função denominada “onLoad” nesse arquivo. Essa função é executada quando o módulo é carregado, e é o local ideal para o programador registrar os eventos dos quais deseja ser notificado e os métodos que deseja oferecer para o resto do sistema, se houver. Esta função representa o único método obrigatório que deve estar presente nos módulos a serem carregados e executados pela plataforma. A figura 4.1 mostra esta estrutura. Para executar o novo jogo criado, basta executar o arquivo JAR referente à plataforma.

A criação de um jogo escrito em Java, ao invés de uma linguagem de script, é possível e semelhante ao processo descrito no parágrafo anterior. A diferença é que o módulo deve implementar a interface “Module” da plataforma, ser compilado, ter metadados relativos ao ServiceLoader adicionados ao mesmo e, por fim, deve ser empacotado em seu próprio arquivo JAR. Este arquivo funciona da mesma forma que o arquivo em linguagem de script do exemplo anterior, sendo colocado em sua pasta. Módulos em Java possuem a

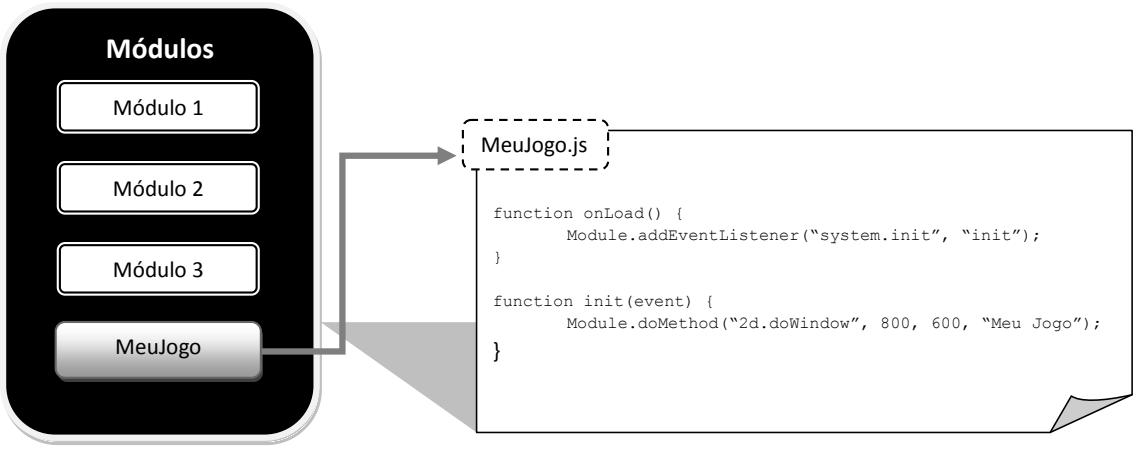

Figura 4.1: Exemplo mínimo de um jogo na plataforma

funcionalidade adicional de poder registrar seus métodos ofertados e os eventos dos quais desejam ser notificados através do uso de Java Annotations.

O processo para criação de módulos que não são módulos de controle de jogo é o mesmo. Para o sistema, os módulos são todos iguais; não ha nada de especial com um módulo de jogo em relação aos outros módulos do sistema.

Essa estrutura de diretórios simplifica o processo de instalação e remoção de módulos, pois é somente necessário ao programador realizar operações com os diretórios dos mesmos para adicionar ou deletar módulos da aplicação. Ao iniciar, a plataforma descobre, em tempo de execução, todos os módulos disponibilizados no diretório "Modules". A seguir, ela carrega cada um deles individualmente, executando o método "onLoad" encontrado na implementação dos mesmos. A partir deste ponto, a plataforma despacha eventos de inicialização que permitem que o jogo execute o seu fluxo normal, no qual os módulos realizam comunicações entre si.

Com isso, é possível concluir que a plataforma apresenta a simplicidade almejada para o programador. Criar módulos, distribuí-los, incluí-los e removê-los da plataforma é um processo sucinto e sem complicações.

4.1 Exemplos de Uso

O quesito seguinte a ser avaliado é a capacidade da plataforma de ser utilizada para o desenvolvimento de jogos. Para isso, foram desenvolvidos pequenos protótipos. Um exemplo da estrutura de um jogo implementado na plataforma pode ser visto na figura 4.2.

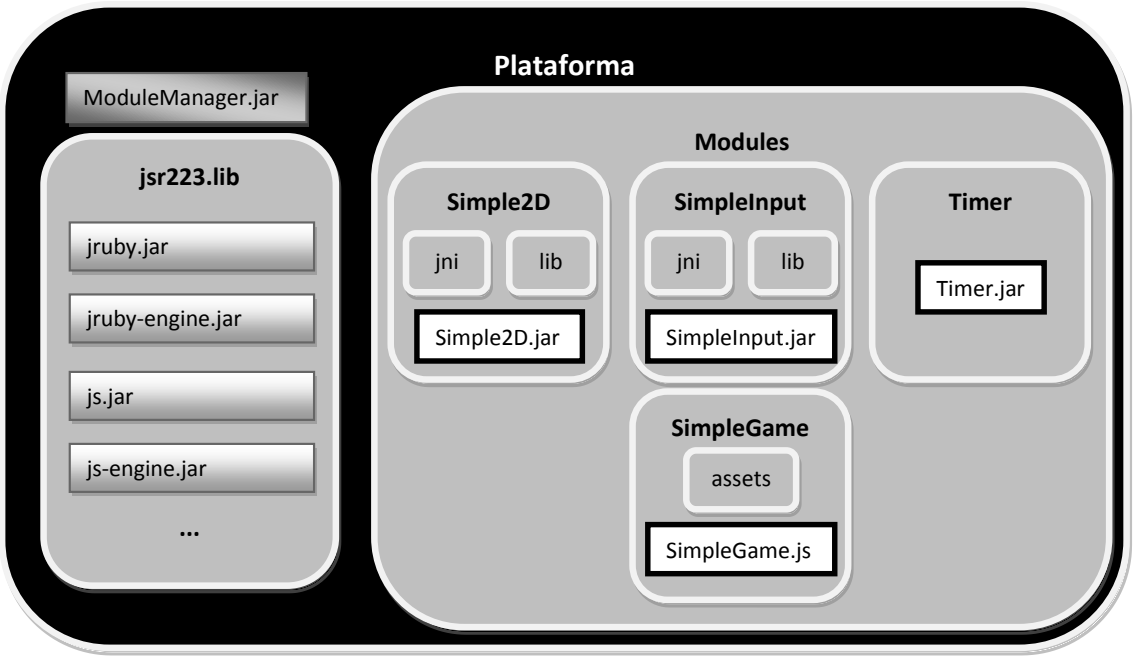

Figura 4.2: Estrutura de um jogo implementado na plataforma

A figura 4.2 mostra primeiramente a presença do arquivo "ModuleManager.jar". Este arquivo é o gerenciador de módulos e funciona como ponto de entrada da aplicação. Ao ser executado, ele realiza a tarefa de encontrar e carregar os módulos. O gerenciador de módulos é o núcleo da plataforma em questão e é o único item obrigatório para uso da mesma.

Na mesma figura também é mostrado o diretório "jsr223.lib". Esse diretório contém as bibliotecas Java necessárias para o suporte às linguagens de script, suportadas através da JSR 223. A plataforma realiza automaticamente a tarefa de descobrir as linguagens a serem suportadas, carregando as bibliotecas relativas às mesmas e passando a ser capaz de carregar componentes escritos nas linguagens descobertas. A presença deste diretório é opcional, sendo necessária somente se houver módulos no sistema escritos em linguagens de script.

Por fim, a figura mostra a pasta de módulos com alguns módulos de exemplo. No sistema mostrado, tem-se um módulo denominado "Simple2D", que oferece funções de renderização 2D, um módulo chamado "SimpleInput", que disponibiliza funcionalidade relacionada a controle por teclado, mouse e joysticks, e um módulo "Timer" que implementa mecanismos de temporização que podem ser utilizados por outros módulos. Como último componente, está presente um módulo de jogo com o nome de "SimpleGame" escrito em JavaScript.

A seguir é apresentado o código presente no arquivo SimpleGame.js. Como pode ser visto no mesmo, o método “onLoad”, que é executado pela plataforma quando o módulo é carregado, é usado pelo componente para fazer o registro no sistema de que deseja ser notificado dos eventos “system.init” e “timer.fixed”, informando quais funções devem ser chamadas quando esses eventos são gerados. Na inicialização do sistema, o componente utiliza o módulo de renderização 2D para criar uma janela. Toda vez que “SimpleGame” é notificado do evento de temporização fixa, o mesmo efetua a atualização das coordenadas baseando-se no retorno de métodos fornecidos pelo componente de entrada e usa o módulo “Simple2D” para desenhar uma textura na tela.

```
var texture = Module.getDirectory() + "/assets/character.png";
var x = 0; // Posição da textura no eixo X
var y = 0; // Posição da textura no eixo Y
var speed = 1.5; // Pixels a serem movidos por centésimo de segundo

function onLoad() {
  // O evento "system.init" é gerado quando o sistema foi carregado
  // e está pronto para iniciar. Quando isso ocorrer, o método
  // "init" deste módulo deve ser chamado.
  Module.addEventListener("system.init", "init");
  // O evento "timer.fixed" é gerado 100 vezes por segundo. Quando
  // isso ocorrer, o método "update" deste módulo deve ser chamado.
  Module.addEventListener("timer.fixed", "update");
}

function init(event) {
  // Na inicialização, cria uma janela de 800 por 600
  Module.doMethod("2d.doWindow", 800, 600, "My Simple Game");
}

function update(event) {
  // Atualiza as posições X e Y da textura com base nos dados
  // informados pelo módulo de Input
  if(Module.doMethod("input.isPressed", "left") == true) x -= speed;
  if(Module.doMethod("input.isPressed", "right") == true) x += speed;
  if(Module.doMethod("input.isPressed", "up") == true) y -= speed;
  if(Module.doMethod("input.isPressed", "down") == true) y += speed;

  if(x < 0) x = 0;
  if(x > 800) x = 800;
  if(y < 0) y = 0;
  if(y > 600) y = 600;

  // Utiliza o módulo de desenho 2D para desenhar a textura na
  // nova posição
  Module.doMethod("2d.begin");
  Module.doMethod("2d.draw", texture, x, y);
  Module.doMethod("2d.end");
}
```

Código 1

Este exemplo oferece uma comprovação inicial de que a plataforma oferece funcionalidade suficiente para o desenvolvimento de jogos de pequeno porte, isto é, com poucos elementos de jogo. Também pode ser observada no exemplo a simplicidade oferecida pela mesma.

Para exemplificar as capacidades da plataforma em relação à independência e isolamento dos módulos, pode-se citar o que ocorreria no exemplo acima caso os módulos fossem removidos um a um. Se fosse removido o módulo de renderização, o jogo funcionaria normalmente, mas não seria criada nem atualizada uma janela onde a textura seria renderizada. Se fosse removido o módulo "SimpleInput", a posição da textura nunca seria modificada. Se fosse removido o módulo de temporização, a lógica do jogo nunca seria processada, não sendo desenhado nada na tela. Por fim, se fosse removido o módulo "SimpleGame" o sistema seria todo inicializado corretamente mas entraria diretamente em um estado ocioso por não ter nenhum comando de resultado relevante sendo executado.

É possível substituir as funcionalidades disponibilizadas por estes módulos por outros componentes a qualquer momento, sendo até mesmo possível fornecer toda essa funcionalidade em um único módulo, embora isso não seja recomendado, pois invalidaria algumas das vantagens de sistemas modulares. Além disso, a plataforma propositalmente não gera erros fatais no caso de chamadas de métodos inexistentes, apenas realizando a tarefa de notificar o programador quando isso ocorre. Essa funcionalidade é provida para que seja mais fácil desenvolver um jogo sobre a plataforma, permitindo inclusive que se preveja o uso de funções que serão providas por módulos que ainda não foram implementados. Com isso, é possível concluir que a plataforma oferece a modularidade, independência e intercambialidade de módulos almejadas por este projeto.

Muito embora este exemplo seja adequado para mostrar o funcionamento básico da plataforma, o mesmo foi considerado demasiado simples para validar a plataforma que, por sua vez, valida a arquitetura proposta neste trabalho. Com o intuito de testar o funcionamento da plataforma de forma mais extensa, decidiu-se criar um novo exemplo com mais funcionalidades do que o primeiro, reescrevendo, para isso, o módulo "SimpleGame".

Construído em cima dos mesmos módulos anteriores (SimpleInput, Simple2D e Timer), o novo exemplo é um jogo de tiro de navegação horizontal onde o jogador controla

uma nave no espaço utilizando as teclas direcionais. Neste exemplo, naves inimigas surgem a partir do lado direito da tela; essas naves podem ser destruídas com tiros disparados pela nave quando o jogador aperta a tecla de espaço. O jogo também conta com um fundo de estrelas que movimenta-se na horizontal para dar a ideia de movimento contínuo para a direita. Este novo exemplo, escrito também em JavaScript, é apresentado a seguir.

```
// Texturas utilizadas pelo jogo
var playerShipTex = Module.getDirectory() + "/assets/playerShip.png";
var enemyShipTex = Module.getDirectory() + "/assets/enemyShip.png";
var backgroundTex = Module.getDirectory() + "/assets/background.png";
var bulletTex = Module.getDirectory() + "/assets/bullet.png";
// Velocidades das entidades
var playerSpeed = 1.65;
var enemySpeed = 1.4;
var bulletSpeed = 1.8;
var backgroundSpeed = 1.75;

var playerShip = new Object(); // Nave do jogador
var enemyShips = new Array(); // Lista de naves inimigas
var bullets = new Array(); // Lista de balas do jogador na tela

function onLoad() {
  // Quando o evento "system.init" for gerado, "init" é executado
  Module.addEventListener("system.init", "init");
  // Quando o evento "timer.fixed" for gerado, "update" é executado
  Module.addEventListener("timer.fixed", "update");
  // O evento "input.keyPressed" é gerado quando uma tecla é
  // pressionada. Quando isso ocorrer, o método "keyPressed" deve
  // ser chamado.
  Module.addEventListener("input.keyPressed", "keyPressed");
}

function init(event) {
  // Inicializa a posição da nave do jogador
  playerShip.x = 50; playerShip.y = 300;
  // Cria a janela de jogo
  Module.doMethod("2d.doWindow", 800, 600, "My Simple Game");
}

function update(event) {
  // Obtém-se a contagem de tempo do jogo, em centésimo de segundo
  var updateCount = event.getParameters().get("updatecount");

  // Inicia o processo de desenho
  Module.doMethod("2d.begin");

  // Desenha-se o fundo do jogo (duas vezes para fazer tiling),
  // movimentando-o para a esquerda
  Module.doMethod("2d.draw", backgroundTex,
          400 - (updateCount*backgroundSpeed)%800, 300);
  Module.doMethod("2d.draw", backgroundTex,
          1200 - (updateCount*backgroundSpeed)%800, 300);

  // Movimenta a nave do jogador de acordo com os dados do módulo
  // de input, limitando a posição da mesma na tela
```

```
if(Module.doMethod("input.isPressed", "left") == true)
  playerShip.x -= playerSpeed;
if(Module.doMethod("input.isPressed", "right") == true)
  playerShip.x += playerSpeed;
if(Module.doMethod("input.isPressed", "up") == true)
  playerShip.y -= playerSpeed;
if(Module.doMethod("input.isPressed", "down") == true)
  playerShip.y += playerSpeed;
if(playerShip.x < 0) playerShip.x = 0;
if(playerShip.x > 800) playerShip.x = 800;
if(playerShip.y < 0) playerShip.y = 0;
if(playerShip.y > 600) playerShip.y = 600;

// A cada 2 segundos cria-se uma nave inimiga nova além da borda
// direita da tela
if(updateCount % 200 == 0) {
  var enemyShip = new Object();
  enemyShip.x = 900;
  enemyShip.y = Math.random() * 500 + 50;
  enemyShips.push(enemyShip);
}

// Movimenta-se as naves inimigas para a esquerda e as desenha na
// tela, deletando as que saírem pela esquerda
for(var i in enemyShips) {
  var enemyShip = enemyShips[i];
  enemyShip.x -= enemySpeed;
  Module.doMethod("2d.draw", enemyShipTex, enemyShip.x, enemyShip.y);
}
while(enemyShips.length > 0 && enemyShips[0].x < -100)
  enemyShips.shift();

// Movimenta-se as balas geradas pelo jogador para a direita,
// deletando as naves inimigas que entrarem em contato com elas,
// juntamente com as balas em si
for(var i = 0; i < bullets.length; i++) {
  var bullet = bullets[i];
  bullet.x += bulletSpeed;
  var hit = false;
  for(var j in enemyShips) {
    var d = Math.sqrt(Math.pow(enemyShips[j].x - bullet.x, 2) +
              Math.pow(enemyShips[j].y - bullet.y, 2));
    if(d < 25) {
      enemyShips.splice(j, 1);
      hit = true;
      break;
    }
  }
  if(hit == true) {
    bullets.splice(i--, 1);
    continue;
  }
  // Desenha a bala
  Module.doMethod("2d.draw", bulletTex, bullet.x, bullet.y);
}

// Apaga as balas que saírem da tela
while(bullets.length > 0 && bullets[0].x > 850) bullets.shift();
```

```
  // Desenha a nave do jogador
  Module.doMethod("2d.draw", playerShipTex, playerShip.x, playerShip.y)
      ;

  // Finaliza o desenho
  Module.doMethod("2d.end");
}

// Este método é chamado quando uma tecla é pressionada
function keyPressed(event) {
  var key = event.getParameters().get("key");
  // Se for pressionada a tecla de espaço, é criada uma nova bala
  if(key == "space") {
    var bullet = new Object();
    bullet.x = playerShip.x; bullet.y = playerShip.y + 10;
    bullets.push(bullet);
  // Se for pressionada a tecla ESC, o jogo é finalizadi
  } else if(key == "escape") {
    Module.doEvent("system.quit");
  }
}
```

Código 2

Muito embora este novo exemplo não seja completo - as naves inimigas não oferecem ainda nenhuma ameaça ao jogador, por exemplo - ele mostra mais funcionalidades da plataforma e como a mesma pode ser utilizada para o desenvolvimento rápido de jogos simples ou protótipos rápidos. Com isso, considera-se satisfatória a validação da plataforma no atendimento dos objetivos propostos pela arquitetura. A figura 4.3 mostra uma captura de tela deste novo jogo durante a sua execução.

4.2 Desempenho

Muito embora já tenha sido estabelecido que este trabalho não tem como prioridade o alto desempenho, isto é normalmente um item importante no desenvolvimento de jogos e habitualmente um item prioritário em ferramentas destinadas a esta finalidade (CLAYPOOL; CLAYPOOL, 2009) (TULIP; BEKKEMA; NESBITT, 2006), portanto é necessário avaliar como a plataforma afeta o desempenho de jogos desenvolvidos sobre a mesma. Busca-se, com isso, validar a hipótese de que a comunicação entre módulos e o gerenciamento dos mesmos tem um efeito negativo no desempenho geral do software construído sobre a plataforma. Também espera-se avaliar se esse efeito é impactante o suficiente a ponto de comprometer a usabilidade da mesma.

Como primeiro teste neste sentido foi utilizado o exemplo mostrado na figura 4.2

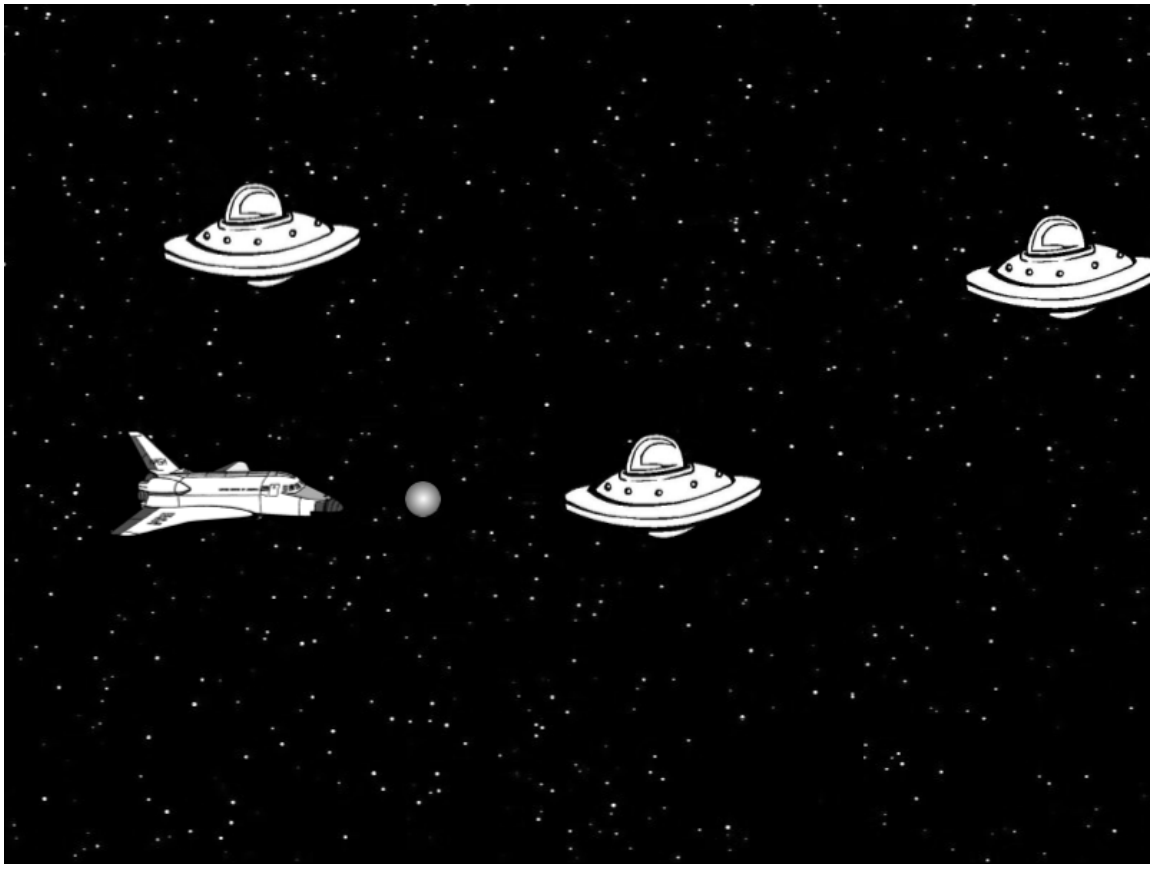

Figura 4.3: Exemplo de jogo implementado sobre a plataforma

como base de código para um jogo. O módulo “Simple2D” utiliza Java OpenGL (JOGL), uma biblioteca Java para uso de OpenGL, uma API de gráficos 2D e 3D para a renderização de gráficos 2D. O método utilizado para avaliar o desempenho do jogo foi realizar quantidades diferentes de operações de desenho neste módulo quando ordenado por um outro componente do sistema e avaliar a taxa de quadros por segundo obtida. Os resultados são comparados com um programa equivalente escrito sem o uso da plataforma em uma linguagem de programação estática, no caso C++, que utiliza a API OpenGL diretamente para realizar as mesmas operações de desenho. Os resultados obtidos nesses experimentos são mostrados na figura 4.4.

O que pode ser visto no gráfico representado na figura 4.4 é que não há diferença na taxa de quadros por segundo em um jogo escrito em C++ e um jogo escrito sobre a plataforma quando o único quesito sendo avaliado é o desempenho em operações de desenho utilizando-se OpenGL. Com esse resultado, é possível afirmar que a plataforma apresenta desempenho compatível, na questão de renderização 2D, com um jogo escrito em linguagem estática, não apresentando redução perceptível no desempenho.

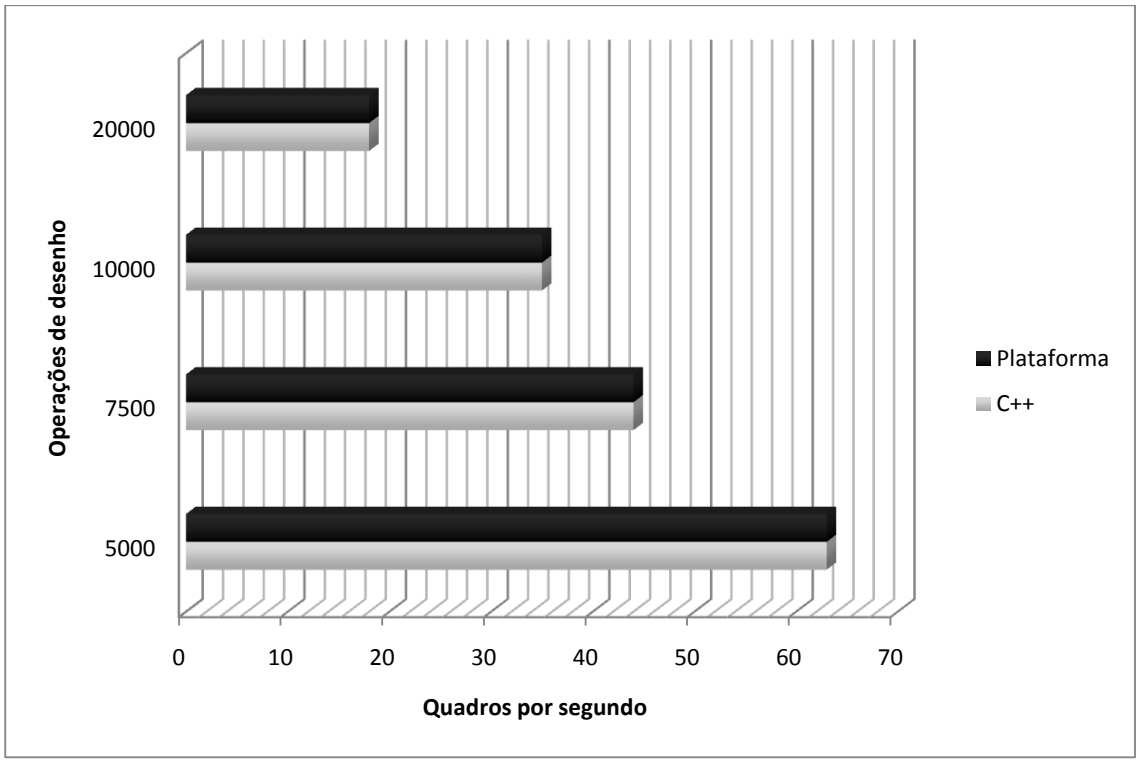

Figura 4.4: Comparação de desempenho de um jogo escrito na plataforma e o mesmo jogo escrito em C++

Este primeiro resultado dos testes, contudo, contradiz uma das hipóteses originais deste trabalho que assume que o *overhead* de processamento introduzido pelo gerenciamento dos módulos e a comunicação entre eles afetaria negativamente o desempenho da plataforma. O fato de isso não ter sido percebido nos testes realizados indica que os mesmos não geraram um fluxo de comunicação entre módulos grande o suficiente para afetar o desempenho do jogo, indicando que este teste inicial não é suficiente para responder as questões propostas.

Com o intuito de confirmar essa hipótese, foram realizados novos testes. Nestes, é transferida parte do código de desenho para outro módulo, escrito em linguagem de script, forçando assim uma maior frequência na troca de mensagens entre os componentes e a necessidade da tradução dos argumentos dessas mensagens. O objetivo dessa transferência de código é gradativamente aumentar o processo necessário pela plataforma para mantar o jogo executando. O método utilizado para esses testes foi o aumento gradativo da quantidade de comunicações feita entre os módulos envolvidos na tarefa de renderização 2D e a medição da taxa de quadros por segundo ao longo desse aumento, enquanto que a quantidade de operações de desenho se mantém constante em 10000 operações por

laço de renderização, o que, no teste anterior, resultava em uma taxa de 35 quadros por segundo. Os resultados desses novos testes são mostrados na figura 4.5.

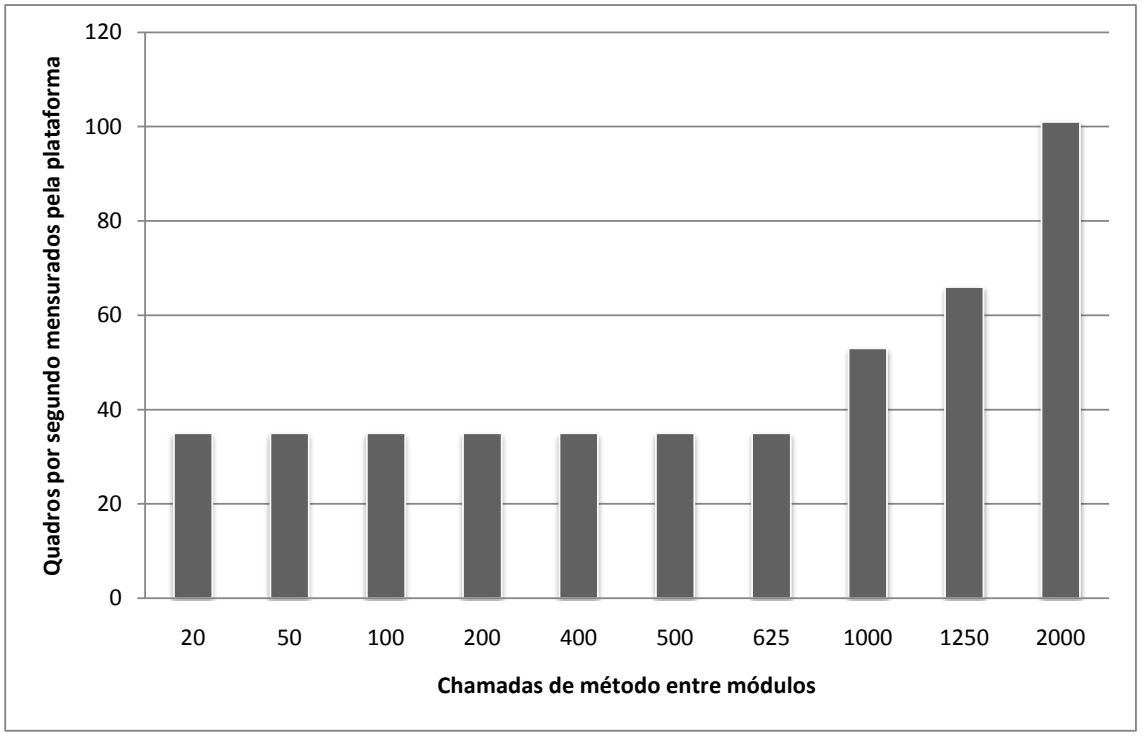

Figura 4.5: Efeito da comunicação entre módulos sobre a taxa de quadros por segundo

O gráfico da figura 4.5 apresenta um resultado que indica que a taxa de quadros por segundo se mantém constante até um certo ponto enquanto se aumenta a quantidade de chamadas de método utilizadas para cada laço de renderização. A partir desse ponto, que no caso se encontra entre 625 e 1000 chamadas por laço de desenho, a taxa de quadros por segundo aumenta.

Inicialmente este resultado é inesperado, uma vez que o aumento na quantidade de chamadas de métodos entre os módulos implica em uma maior carga de processamento sobre o gerenciador de módulos. Contudo, o resultado pode ser explicado pela forma como a taxa de quadros por segundo é medida. No caso, a plataforma utiliza as funções de temporização providas pelo módulo "Timer" e mede quantos quadros de imagem são renderizados a cada segundo reportado por este módulo. O que ocorre é que a quantidade de quadros gerada se mantém inalterada, visto que não é alterada a quantidade de operações de desenho executado, mas ocorre um atraso na execução do módulo "Timer", aumentando, por consequência, o tempo entre as medições de quadros renderizados devido ao atraso na entrega de mensagens geradas pelo módulo. Isto condiz com a hipótese

de que a plataforma fica sobrecarregada com a comunicação entre os componentes.

Para validar essa nova hipótese e para descobrir qual o limite de carga da plataforma, foi realizada uma nova série de testes. O método aplicado para estes foi concentrar não no processo de renderização 2D e sim no processo de envio de mensagens na plataforma. Foram geradas quantidades diferentes de mensagens entre módulos escritos em diferentes linguagens em períodos de um segundo, e medido o tempo entre os eventos de temporização fixa relativas a um segundo gerados pelo módulo "Timer". Também se mediu o uso do processador por parte da plataforma durante esses testes. Os resultados são mostrados na figura 4.6.

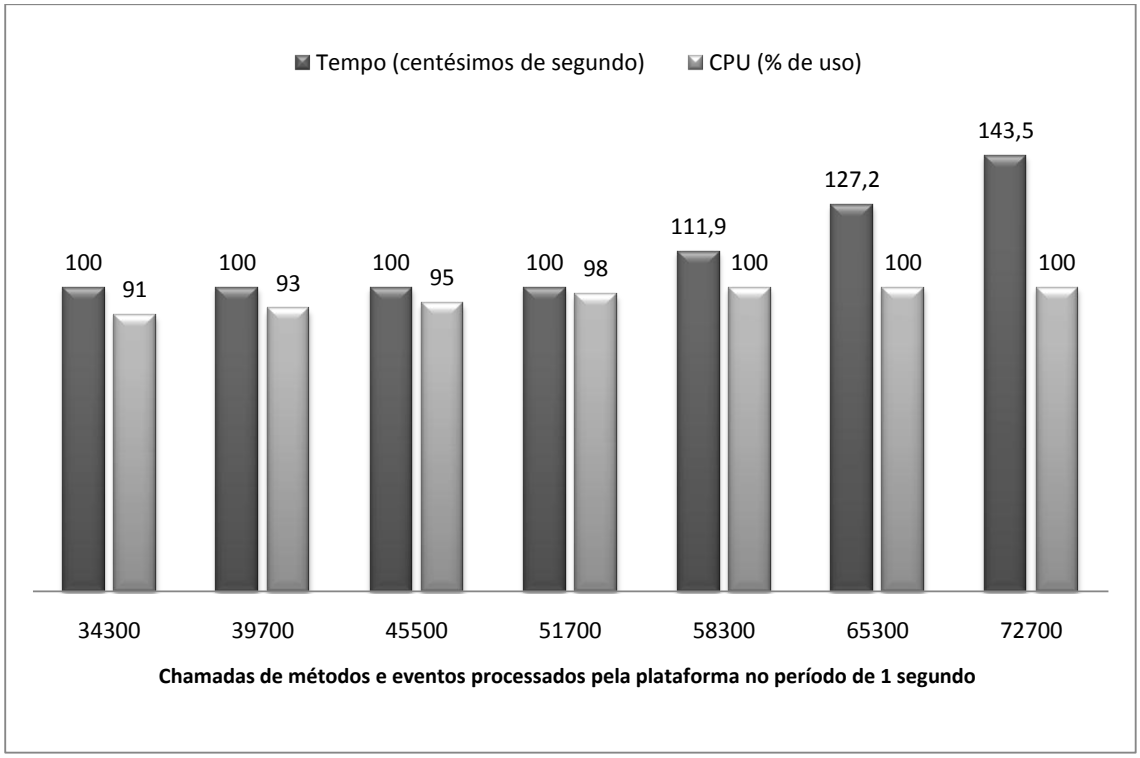

Figura 4.6: Desempenho da plataforma relativo ao volume de comunicação entre os módulos

O que pode ser visto no gráfico que representa esses novos testes é que existe um limite prático, dependente do processador, de quantas mensagens podem ser manejadas pela plataforma antes de o desempenho da mesma ser comprometido. A partir de um certo ponto, que nos testes se encontra entre 51700 e 58300 atos de comunicação entre módulos por segundo, o processador é utilizado em sua totalidade e ocorre um atraso no processamento das mensagens. Isso é confirmado com o fato de que o evento relativo à passagem de um segundo, gerado pelo módulo "Timer", passar a levar mais de um segundo para ser entregue aos componentes que devem ser notificados do mesmo.

Como um último experimento relacionado ao desempenho da plataforma, foi testado o tempo necessário para execução de diversas chamadas de métodos entre módulos. Estas chamadas foram executadas entre módulos Java e Javascript em todas as combinações possíveis. Para fins de comparação, foi também medido o tempo necessário para execução de chamadas de métodos internas aos módulos. Os resultados destes testes são mostrados na figura 4.7. Como pode ser visto na mesma, o pior desempenho é encontrado em chamadas que partem de módulos Javascript destinados a módulos Java, onde a tradução de argumentos pode ser necessária.

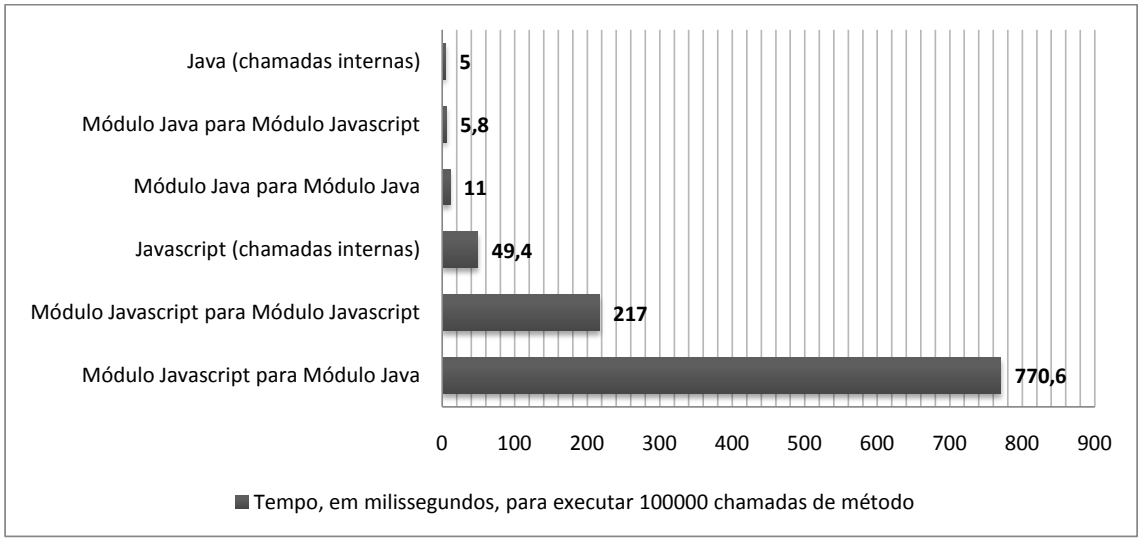

Figura 4.7: Desempenho da plataforma relativo às diferentes formas de chamadas de método

Com base nos testes de desempenho realizados, é possível, enfim, afirmar que, em conformidade com a hipótese inicial relativa a este assunto, o fator determinante na perda de desempenho com o uso da plataforma é a gerência e comunicação entre os módulos, tarefas realizadas pelo gerenciador de módulos que adicionam *overhead* a programas desenvolvidos sobre a mesma. Considerando a alta taxa de comunicação entre módulos permitida antes do comprometimento do desempenho ocorrer, o fato dessa limitação ser dependente do processador da máquina onde a plataforma está rodando e os resultados positivos dos protótipos desenvolvidos sobre a mesma, é sensato afirmar que essa limitação da plataforma não inviabiliza o uso da mesma para as tarefas às quais a mesma se propõe.

4.3 Uso Geral

Durante a pesquisa e desenvolvimento da plataforma, cogitou-se a possibilidade de desenvolver uma plataforma que fosse mais adequada para um tipo específico de jogo ou para uma categoria mais abrangente como jogos 2D, jogos de plataforma ou alguma especificação similar. Essa consideração, contudo, foi descartada na fase de planejamento inicial do projeto com o intuito de permitir o desenvolvimento de uma plataforma de uso mais abrangente, já que as funcionalidades oferecidas pela mesma oferecem vantagens na criação de qualquer tipo de jogo. Com base nessa decisão, foi investido trabalho na busca de uma plataforma o mais genérica possível para o desenvolvimento de jogos, evitando ao máximo a possibilidade de restrições adicionais no uso da mesma. Até mesmo funcionalidades comuns a praticamente todos os jogos, como *input* e temporização, foram mantidos fora da plataforma em prol de mantê-las em módulos isolados, permitindo assim que o programador utilize a implementação mais adequada às suas necessidades ou que implemente as suas próprias soluções.

Um dos resultados da decisão de manter a plataforma genérica é o fato da plataforma resultante deste projeto ficar caracterizada por não ter nenhuma característica exclusiva ao desenvolvimento de jogos. O cerne da mesma é o suporte a módulos, a comunicação entre os mesmos, ao suporte a diferentes linguagens de programação e gerência de despacho de eventos. Estes itens não são vantajosos somente para o desenvolvimento de jogos. Portanto, é sensato afirmar que esta plataforma é também útil no desenvolvimento de outros tipos de aplicações onde é desejável ou necessário o suporte essas funcionalidades.

Uma outra funcionalidade que é incidentalmente oferecida pela plataforma é a de ponto de partida para execução de programas escritos em linguagens de script. Utilizando a mesma, é possível executar esses programas sem a necessidade de instalar um interpretador ou um ambiente de execução próprio para essas linguagens, bastando apenas ter a máquina virtual Java instalada. Por exemplo, é possível executar um programa escrito em Ruby ou Python criando um único módulo que sirva de ponto de entrada ao mesmo e executando o gerenciador de módulos, sem a necessidade de ter um ambiente de execução dedicado a Ruby ou Python na máquina. Isso tem a capacidade inclusive de simplificar a tarefa de testar, experimentar ou estudar diferentes linguagens de programação, pois reduz o *overhead* inicial encontrado para trabalhar com as mesmas.

5 CONCLUSÃO E SUGESTÕES PARA TRABALHOS FUTUROS

Este trabalho buscou o desenvolvimento de uma plataforma que servisse de ferramenta a programadores que buscam a possibilidade de utilizar várias linguagens de programação diferentes em um ambiente modular para o desenvolvimento de jogos. A plataforma desenvolvida atende a estes requisitos, diferenciando-se de outros sistemas similares por ser voltado à simplicidade, modularidade e versatilidade, especialmente no tocante ao uso de linguagens de script, ao invés de buscar desempenho. Muito embora o *overhead* inserido devido à comunicação entre os módulos aumente o processamento necessário para o funcionamento dos programas, esse é um custo que se considerou aceitável em vista das vantagens oferecidas pela plataforma. Uma das propostas de trabalhos futuros é justamente investigar novas formas de comunicação entre os módulos que reduzam este impacto no processamento, tomando como base os resultados dos testes realizados neste trabalho relativos ao desempenho da plataforma.

A busca de uma solução genérica também abre novas possibilidades para a ferramenta desenvolvida, que se mostra apta a ser usada para o desenvolvimento de softwares que não sejam necessariamente jogos. O desenvolvimento de protótipos nessa direção para validar a possibilidade deste uso para a plataforma é outra possível proposta de trabalho sobre a ferramenta.

A plataforma desenvolvida torna possível oferecer todo tipo de funcionalidade que o programador desejar, desde gerenciamento de arquivos à renderização 3D, incluindo funcionalidades não relacionadas a jogos. Como a plataforma torna essas funcionalidades acessíveis para programas escritos nas mais diversas linguagens de script, surge a possibilidade de utilizar a plataforma como base para ensino de diversos tópicos relacionados à programação. Como torna-se simplificada a oferta de funcionalidades, é possível para

um professor desenvolver ou obter módulos relacionados ao assunto que deseja ensinar, passando aos alunos tarefas relacionadas ao uso dessas funcionalidades. Caso os alunos já tenham conhecimento ou experiência de alguma das linguagens de script suportadas, a linguagem de programação a ser usada torna-se uma barreira a menos para a realização das tarefas, permitindo aos mesmos concentrarem-se no conteúdo das mesmas.

Por fim, é importante afirmar que foi provado que a abordagem proposta por este trabalho é suficientemente adequada para o desenvolvimento de jogos e de protótipos de jogos, oferecendo, através do suporte a diferentes linguagens de script, ferramentas adicionais que podem auxiliar neste processo.

REFERÊNCIAS

ANDERSON, E. F. et al. The case for research in game engine architecture. In: *Future Play '08: Proceedings of the 2008 Conference on Future Play*. New York, NY, USA: ACM, 2008. p. 228–231. ISBN 978-1-60558-218-4.

ARAKJI, R.; LANG, K. Digital consumer networks and producer-consumer collaboration: Innovation and product development in the video game industry. *J. Manage. Inf. Syst.*, M. E. Sharpe, Inc., Armonk, NY, USA, v. 24, n. 2, p. 195–219, 2007. ISSN 0742-1222.

BASILI, V. R. et al. Characterizing and modeling the cost of rework in a library of reusable software components. In: *ICSE '97: Proceedings of the 19th international conference on Software engineering*. New York, NY, USA: ACM, 1997. p. 282–291. ISBN 0-89791-914-9.

CHAPPELL, D.; KAND, K. Universal middleware: What's happening with osgi and why you should care. 2009. Disponível em: <http://soa.sys-con.com/node/492519>. Acesso em: 16 fev. 2010.

CLAYPOOL, M.; CLAYPOOL, K. Perspectives, frame rates and resolutions: it's all in the game. In: *FDG '09: Proceedings of the 4th International Conference on Foundations of Digital Games*. New York, NY, USA: ACM, 2009. p. 42–49. ISBN 978-1-60558-437-9.

CUTUMISU, M. et al. Scriptease: A generative/adaptive programming paradigm for game scripting. *Sci. Comput. Program.*, Elsevier North-Holland, Inc., Amsterdam, The Netherlands, The Netherlands, v. 67, n. 1, p. 32–58, 2007. ISSN 0167-6423.

CUTUMISU, M. et al. Generating ambient behaviors in computer role-playing games. *IEEE Intelligent Systems*, IEEE Educational Activities Department, Piscataway, NJ, USA, v. 21, n. 5, p. 19–27, 2006. ISSN 1541-1672.

FEDEROFF, M. A. *Heuristics and Usability Guidelines for the Creation and Evaluation of Fun in Video Games*. [S.l.], 2006.

FOLMER, E. Component based game development - a solution to escalating costs and expanding deadlines? In: *CBSE*. [S.l.: s.n.], 2007. p. 66–73.

FOWLER, M. Inversion of control containers and the dependency injection pattern. 2004. Disponível em: <http://www.martinfowler.com/articles/injection.html>. Acesso em: 16 fev. 2010.

GOLDMAN, N.; NARAYANASWAMY, K. Software evolution through iterative prototyping. In: *ICSE '92: Proceedings of the 14th international conference on Software engineering*. New York, NY, USA: ACM, 1992. p. 158–172. ISBN 0-89791-504-6.

GOLDSTONE, W. *Unity Game Development Essentials*. Birmingham, UK: Packt Publishing, 2009. ISBN 897-1-847198-18-1.

GROGAN, M.; SUNDARARAJAN, A.; VENKATARAMANAPPA, S. Scripting project home. 2010. Disponível em: <https://scripting.dev.java.net/>.

HOPKINS, J. Component primer. *Commun. ACM*, ACM, New York, NY, USA, v. 43, n. 10, p. 27–30, 2000. ISSN 0001-0782.

HUGHES, A. A framework for mobile java applications. In: *PPPJ '07: Proceedings of the 5th international symposium on Principles and practice of programming in Java*. New York, NY, USA: ACM, 2007. p. 243–248. ISBN 978-1-59593-672-1.

KAZI, I. H. et al. Techniques for obtaining high performance in java programs. *ACM Comput. Surv.*, ACM, New York, NY, USA, v. 32, n. 3, p. 213–240, 2000. ISSN 0360-0300.

KOIVISTO, E. M. I.; ELADHARI, M. Paper prototyping a pervasive game. In: *ACE '06: Proceedings of the 2006 ACM SIGCHI international conference on Advances in computer entertainment technology*. New York, NY, USA: ACM, 2006. p. 97. ISBN 1-59593-380-8.

LACORT, A. M. *Integration of a 3D rendering engine with a physics simulator*. June 2009.

LIANG, S. *Java Native Interface: Programmer's Guide and Reference*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1999. ISBN 0201325772.

O'CONNER, J. Scripting for the java platform. 2006. Disponível em: <http://java.sun.com/developer/technicalArticles/J2SE/Desktop/scripting/>. Acesso em: 16 fev. 2010.

OLLILA, E. M. I.; SUOMELA, R.; HOLOPAINEN, J. Using prototypes in early pervasive game development. *Comput. Entertain.*, ACM, New York, NY, USA, v. 6, n. 2, p. 1–17, 2008. ISSN 1544-3574.

OUSTERHOUT, J. K. Scripting: Higher-level programming for the 21st century. *Computer*, IEEE Computer Society, Los Alamitos, CA, USA, v. 31, p. 23–30, 1998. ISSN 0018-9162.

PASSOS, E. B. et al. Smart composition of game objects using dependency injection. *Comput. Entertain.*, ACM, New York, NY, USA, v. 7, n. 4, p. 1–15, 2009. ISSN 1544-3574.

RUSSELL, G.; DONALDSON, A. F.; SHEPPARD, P. Tackling online game development problems with a novel network scripting language. In: *NetGames '08: Proceedings of the 7th ACM SIGCOMM Workshop on Network and System Support for Games*. New York, NY, USA: ACM, 2008. p. 85–90. ISBN 978-1-60558-132-3.

SCHRAGE, M. Cultures of prototyping. ACM, New York, NY, USA, p. 191–213, 1996.

SPRONCK, P. et al. Adaptive game ai with dynamic scripting. *Mach. Learn.*, Kluwer Academic Publishers, Hingham, MA, USA, v. 63, n. 3, p. 217–248, 2006. ISSN 0885-6125.

TESSIER, P. et al. A component-based methodology for embedded system prototyping. In: *RSP '03: Proceedings of the 14th IEEE International Workshop on Rapid System Prototyping (RSP'03)*. Washington, DC, USA: IEEE Computer Society, 2003. p. 9. ISBN 0-7695-1743-1.

TULIP, J.; BEKKEMA, J.; NESBITT, K. Multi-threaded game engine design. In: *IE '06: Proceedings of the 3rd Australasian conference on Interactive entertainment*. Murdoch University, Australia, Australia: Murdoch University, 2006. p. 9–14. ISBN 86905-902-5.

VITHARANA, P. Risks and challenges of component-based software development. *Commun. ACM*, ACM, New York, NY, USA, v. 46, n. 8, p. 67–72, 2003. ISSN 0001-0782.

YAO, H.; ETZKORN, L. Towards a semantic-based approach for software reusable component classification and retrieval. In: *ACM-SE 42: Proceedings of the 42nd annual Southeast regional conference*. New York, NY, USA: ACM, 2004. p. 110–115. ISBN 1-58113-870-9.

ZHANG, W.; MCLAUGHLIN, M.; KATCHABAW, M. Story scripting for automating cinematics and cut-scenes in video games. In: *Future Play '07: Proceedings of the 2007 conference on Future Play*. New York, NY, USA: ACM, 2007. p. 152–159. ISBN 978-1-59593-943-2.