MODELAGEM DE UM *FRAMEWORK* PARA O DESENVOLVIMENTO DE JOGOS COM A LINGUAGEM DE PROGRAMAÇÃO JAVA USANDO CONCEITOS DA ORIENTAÇÃO A ASPECTOS

Jefferson Amaral da SILVA (1); Jackson Amaral da SILVA (2); Berto de Tacio Pereira GOMES (3)
(1) (2) (3) Centro Federal de Educação Tecnológica do Maranhão, Av. Getúlio Vargas, Nº 4 – Monte Castelo, São Luis – MA, (98) 32189000, (98) 32189067
(1) e-mail: jefferson.amarals@gmail.com
(2) e-mail: jackson.amarals@gmail.com
(3) e-mail: bertodetacio@gmail.com

RESUMO

A Programação Orientada a Aspectos (POA) objetiva a separação do código segundo a sua importância e interesses dentro da aplicação, permitindo que o programador encapsule o código secundário em módulos separados. Nesse contexto propõe-se a modelagem de um *framework* usando POA para o desenvolvimento de jogos, uma vez que esse tipo de *software* pode possuir diversas funcionalidades e conseqüentes necessidades que se tornam complexas na medida em que os interesses de seus módulos se entrelaçam. Usando a POA objetiva-se criar um modelo de desenvolvimento que permita fazer a separação dos módulos do jogo separando-os por interesses e dessa forma diminuir a quantidade de código gerado, facilitar a adição de novas funcionalidades ou a alteração das já existentes e culminar em um ganho de desempenho por parte do *software*. Nesta pesquisa de modelagem de *software*, inicialmente foram identificadas as funcionalidades cruciais de um jogo a fim de encapsulá-las, com isso foi possível fazer o levantamento de requisitos do *framework*. Usando a linguagem Java juntamente com POA criou-se um modelo de *framework* que torna o desenvolvimento de jogos mais ágil, flexível e extensível de forma desagregada e simples, adicionalmente, esse modelo permite delegar funções já implementadas a outros *frameworks* e tecnologias existentes.

Palavras-chave: Programação de Jogos em Java, Programação Orientada a Aspectos

1. INTRODUÇÃO

Jogos são, de uma forma geral, *softwares* complexos e conseqüentemente difíceis de serem implementados, além disso, há uma grande dificuldade em separar os módulos e os interesses desse tipo de *software.*

Outra dificuldade encontrada pelos desenvolvedores de jogos é o fato de que esses são muito híbridos entre si e necessitam de recursos e funcionalidades quase exclusivos. Existem jogos com gráficos 2D, gráficos 3D, *online*, que persistem dados, que usam tecnologias de inteligência artificial, etc. Adicionalmente há o problema de existirem diferentes Sistemas Operacionais nas máquinas em que o jogo desenvolvido será executado. Tudo isso torna complicado e às vezes inviável o reaproveitamento de código.

Hoje em dia já existem *frameworks* que tratam de sanar muitas dessas necessidades, como renderização e persistência de dados por exemplo, mas tais *frameworks* por vezes acabam misturando tarefas que deveriam ser modularizadas, fazendo com que os interesses da aplicação fiquem entrelaçados.

Nesse contexto propõe-se a modelagem de uma *framework* usando a Linguagem de Programação Java (Sun Microsystems, 2008) e conceitos do paradigma de programação Orientada a Aspectos para o desenvolvimento de jogo a fim de separar o código de forma que os interesses do mesmo fiquem bem definidos, e, além disso, permitir a integração com outros *frameworks* e tecnologias que realizam tarefas já bem implementadas para desenvolvimento de jogos atualmente. Por fim, se almeja que o *framework* seja independente de Sistema Operacional para ser executado, o que torna-se viável com uso da linguagem Java.

O *framework* modelado foi nomeado *Java Aspect Game Engine Framework* – JAGAF.

2. A LINGUAGEM DE PROGRAMAÇÃO JAVA

A linguagem de programação selecionada para a modelagem do JAGAF foi a linguagem Java, por ser uma linguagem robusta, amplamente divulgada e utilizada dentro da comunidade de desenvolvedores e devido à grande quantidade de recursos e vantagens que a mesma disponibiliza.

Dentre alguns destes recursos pode-se citar a portabilidade entre diferentes plataformas. Dessa maneira, o JAGAF poderá ser executado em qualquer Sistema Operacional que disponibilize uma implementação da Máquina virtual Java (*Java Runtime Environment*).

Outra grande vantagem do Java é que, por ser uma linguagem Orientada a Objetos, permite que a aplicação possa ser dividida em camadas especializadas, através de um padrão de projeto chamado de MVC (*Model View Control* - Modelo Visualização Controle) permitindo que uma seja modificada sem danificar a outra. Sendo assim, o sistema fica extensível de forma que cada camada possa evoluir sem causar danos ao sistema como um todo.

Como o JAGAF visa reutilizar funcionalidades já bem implementadas por outras tecnologias e frameworks, também faz-se de destaque o grande leque de tecnologias e frameworks compatíveis com a linguagem Java que podem vir a ser utilizadas no desenvolvimento de jogos. A tabela 1 lista algumas dessas tecnologias e frameworks:

Tabela 1 – Exemplos de tecnologias e *Frameworks* compatíveis com a linguagem Java

Tecnologia/*Framework*	**Descrição**
Hibernate	*Framework* para persistência de dados do sistema.
Log4J	*Framework* para Log do sistema.
GTGE	*Framework* para criação de jogos com gráficos 2D.
JMonkeyEngine	*Framework* para criação de ambientes com gráficos 3D.
Java 2D	Tecnologia que permite a criação e manipulação de desenhos, cenas e animações 2D em Java.
Java 3D	Tecnologia que permite manipular ambientes tridimensionais em Java.
AspectJ Language	Linguagem de Programação Orientada a Aspectos suportada pela linguagem Java.
Jasperreport	*Framework* para criação de relatórios gerenciais.

JFreeChart	*Framework* para criação de gráficos em Java.

Por fim vale ressaltar que todas essas tecnologias e *frameworks* são *free*, assim como a própria linguagem Java.

3. O PARADIGMA DA PROGRAMAÇÃO ORIENTADA A ASPECTOS

O paradigma da Programação Orientada a Aspectos (POA) foi apresentado inicialmente há no ano de 1990, em Palo Alto, nos laboratórios da Xerox (Kiczales, 1997). Segundo Lopes (1997), "as aplicações estão ampliando os limites das técnicas de programação atuais, de modo que certas características de um sistema afetam seu comportamento de tal forma que as técnicas atuais não conseguem capturar essas propriedades de forma satisfatória.".

Diferentemente do que se possa pensar a POA não trabalha isoladamente dos outros paradigmas de programação, mas sim vem entender suas funcionalidades a fim prover recursos que venham a sanar o problema apontado por Lopes (1997).

Como apontam Becker e Geihs (1998):

> [...]existem problemas de programação que as técnicas de programação orientada a objetos ou de programação estruturada não são suficientes para separar claramente todas as decisões de projeto que o programa deve implementar. Portanto, interesses (*concerns*), não pertencem ao domínio de negócio modelado por Programação Orientada a Objetos ou estruturada, podendo se referir a requisitos não funcionais. Os interesses também podem ser definidos como aspectos.

Linguagem de Programação atuais, como Java, provêem recursos bem implementados para representação abstrata de elementos do mundo real usando funções, método, classes, etc. No entanto existem funcionalidades que não são inerentes à regra de negócio do sistema e que acabam ficando espalhados (entrelaçados). A essas funcionalidades denominam-se interesses ortogonais (*crosscut-concerns*). Ao tentar-se implantar estes interesses ortogonais pelo sistema, obtem-se código emaranhado, *Tangled code* (Kiczales, 1997).

O objetivo principal da Orientação a Aspectos é modularizar os interesses ortogonais da aplicação de forma a evitar o código emaranhado como exemplifica a figura 1:

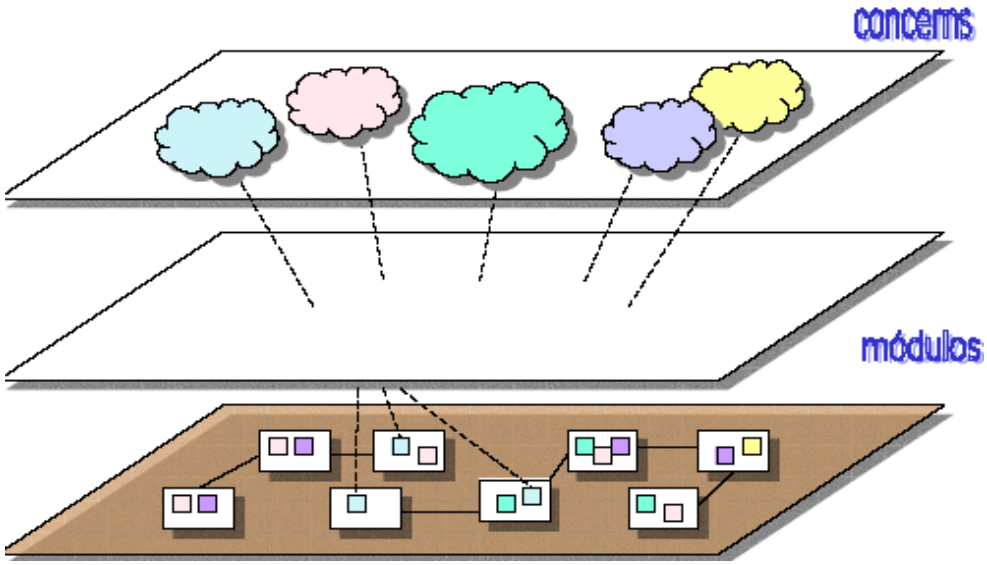

Figura 1 – Separação dos interesses ortogonais no sistema (Souza Junior et al)

3.1. Composição de um Sistema Orientado a Aspectos

Os seguintes componentes integram um Sistema Orientado a Aspectos:

- Linguagem de componentes. Segundo Irwin (1997), “a linguagem de componentes deve permitir ao programador escrever programas que implementem as funcionalidades básicas do sistema, ao mesmo tempo em que não prevêem nada a respeito do que deve ser implementado na linguagem de aspectos.”
- Linguagem de aspectos. A linguagem de aspectos deve suportar a implementação das propriedades desejadas de forma clara e concisa, fornecendo construções necessárias para que o programador crie estruturas que descrevam o comportamento dos aspectos e definam em que situações eles ocorrem (IRWIN et al.,1997).
- Combinador de aspectos. A tarefa do combinador de aspectos (*aspect weaver*) é combinar os programas escritos em linguagem de componentes com os escritos em linguagem de aspectos (Souza Junior et al).
- Programas escritos em linguagem de componentes.
- Um ou mais programas escritos em linguagem de aspectos.

A figura 2 demonstra a composição de um sistema utilizando o paradigma da programação orientada a aspectos.

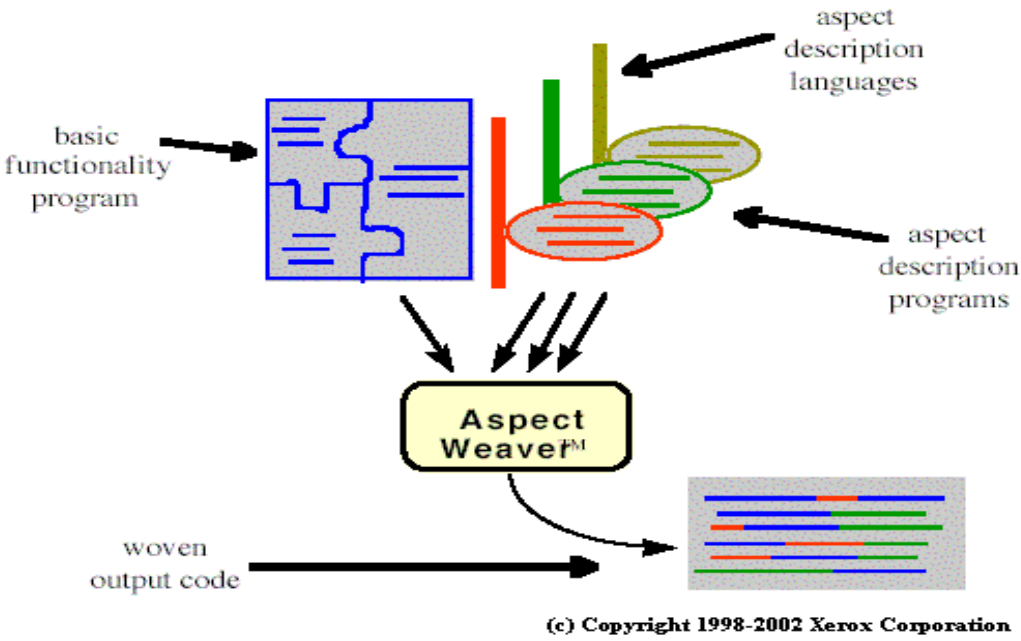

Figura 2 – Composição dos sistemas utilizando programação orientada a aspectos

3.2. Conceitos Fundamentais da Orientação a Aspectos

A POA define alguns conceitos que tornam possível modularizar os interesses ortogonais da aplicação. São estes os conceitos:

- Pontos de Junção (*Join Points*): São locais bem definidos em um programa, como a chamada de um método, uma instanciação, alteração em um atributo, a ocorrência de uma exceção, etc. Todos os pontos de junção tem um contexto associado, como por exemplo a chamada de um método contém de qual classe ele pertence e os argumentos passados, tudo isso ficará disponível para você.
- Pontos de Corte (*Pointcuts*): Os pontos de Atuação são como “regras” que definirão quais serão os pontos de junção que irão ser executados. Eles têm por objetivo criar regras genéricas onde vários pontos de junção serão identificados pelo mesmo ponto de atuação.
- Adendo (*Advice*): O adendo é todo o código que você vai “disparar” quando um ponto de junção definido pelo ponto de atuação for executado.

- *Inner-Type Definitions*: São definições feitas em tempo de execução que afetam determinadas classes. Essas definições podem ser a adição de um método ou atributo, adição de uma herança ou implementação de uma interface.
- Aspecto (*Aspect*): O Aspecto é a unidade que agrupa os adendos e *Inner-Type Definitions*. Assemelha-se a uma Classe em linguagens Orientadas a Objetos.

4. ESTRUTURA BÁSICA DE UM JOGO EM JAVA

Todos os jogos podem possuir um esqueleto em comum que funciona como uma raiz e que vai ficando cada vez mais específico com o passar do desenvolvimento até o jogo estar concluído. Mediante pesquisa bibliográfica, foi verificado que inicialmente um jogo pode se resumir nas seguintes etapas discriminadas na figura 3:

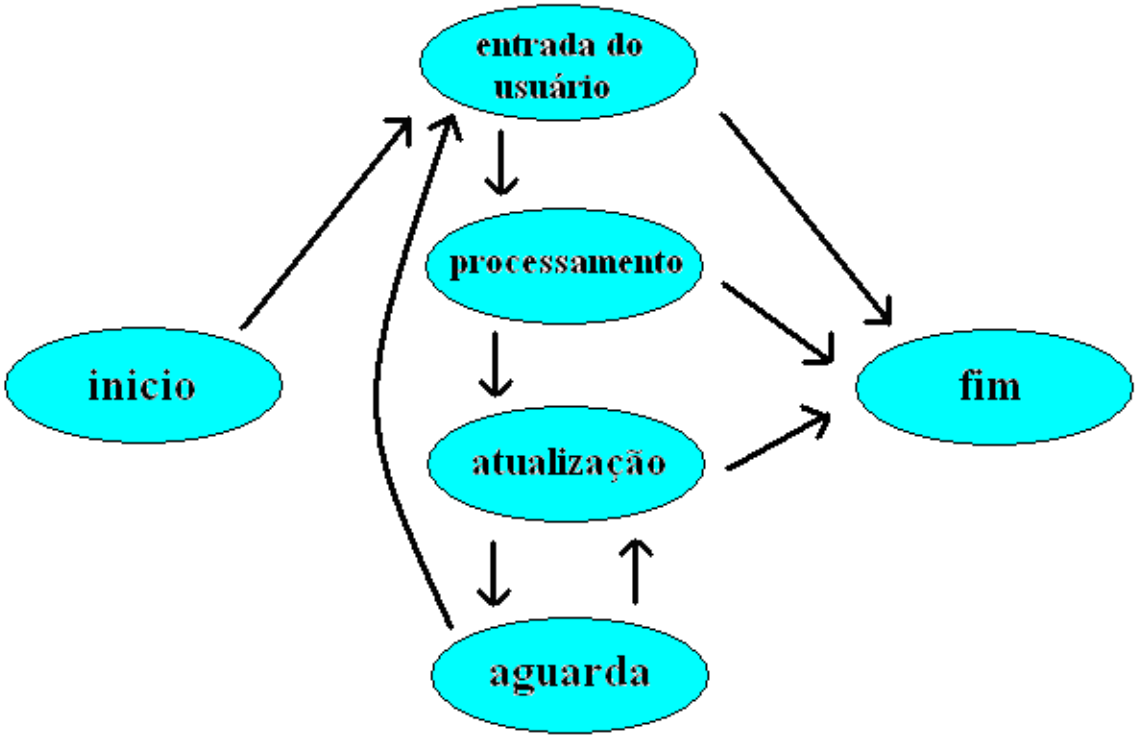

Figura 3 – Etapas de execução de um jogo

Seguindo esse diagrama é possível implementar um jogo qualquer.

4.1. Interesses Ortogonais em um Jogo em Java

Para implementação do JAGAF é necessário identificar em que etapas podemos encontrar interesses ortogonais. É possível verificar a presença de interesses ortogonais em pelo menos duas das etapas de execução de um jogo, sendo elas:

- Processamento: O processamento diz respeito a como o jogo vai reagir a entradas de dados feitas pelo usuário ou pelo próprio *engine* do jogo, e essa entrada gera uma saída ainda na etapa de processamento, como a alteração da pontuação de um jogador por exemplo na base de dados, ou na etapa de atualização. O problema em questão é que a lógica da etapa de processamento, tanto no que diz respeito a entrada quanto a saída de dados, fica espalhada no código, uma vez que é necessário o tratamento de eventos como entradas feitas pelo teclado e movimentação ou clique do mouse e esses eventos podem ser capturados em diferentes trechos do código e ter sentidos diferentes dependendo de fatores internos do jogo. Assim o tratamento desses eventos carrega consigo trechos lógicos e os espalha pela aplicação. Na saída de dados podem haver chamadas a métodos de controle do *status* do jogo seqüencialmente ao tratamento das entradas de dados e assim acaba por se espalhar também.

- Atualização: Geralmente a atualização é feita mudando-se algum elemento do jogo de lugar, realizando uma animação, movimentando o cenário, ou qualquer tarefa que implique no redesenho da tela ou tratamento de eventos, como colisões por exemplo. No entanto atualizações na tela são feitas em diversas classes diferentes do código. Assim sendo o acesso a rotinas que por algum motivo alteram o estado da tela ficam espalhados por várias classes carregando consigo uma quantidade considerável de lógica.

Vale ressaltar que muitas vezes essas duas etapas acabam por misturar-se entre si, por conta de códigos mal implementados ou limitação de alguns *frameworks* já utilizados para construção de jogos.

5. USANDO POA PARA INTERFERIR NO FLUXO DE EXECUÇÃO DO JOGO

A proposta do JAGAF é criar um modelo que permita o desenvolvimento de jogos em Java usando POA para separar e centralizar os interesses ortogonais presentes nessa categoria de *software*.

Depois de identificados os pontos cruciais onde ocorre espalhamento de código foi proposto um modelo que interfere no fluxo normal de execução de um jogo. Essa interferência é feita de forma bem simples.

Utilizou-se os conceitos de POA para "cortar" o código e separar ao máximo os interesses encapsulando-os em Aspectos. Os Aspectos ficam encarregados por capturar os momentos em que, normalmente, o código ficaria entrelaçado, possibilitando uma melhor modularização. É possível visualizar esses "cortes" na figura 4:

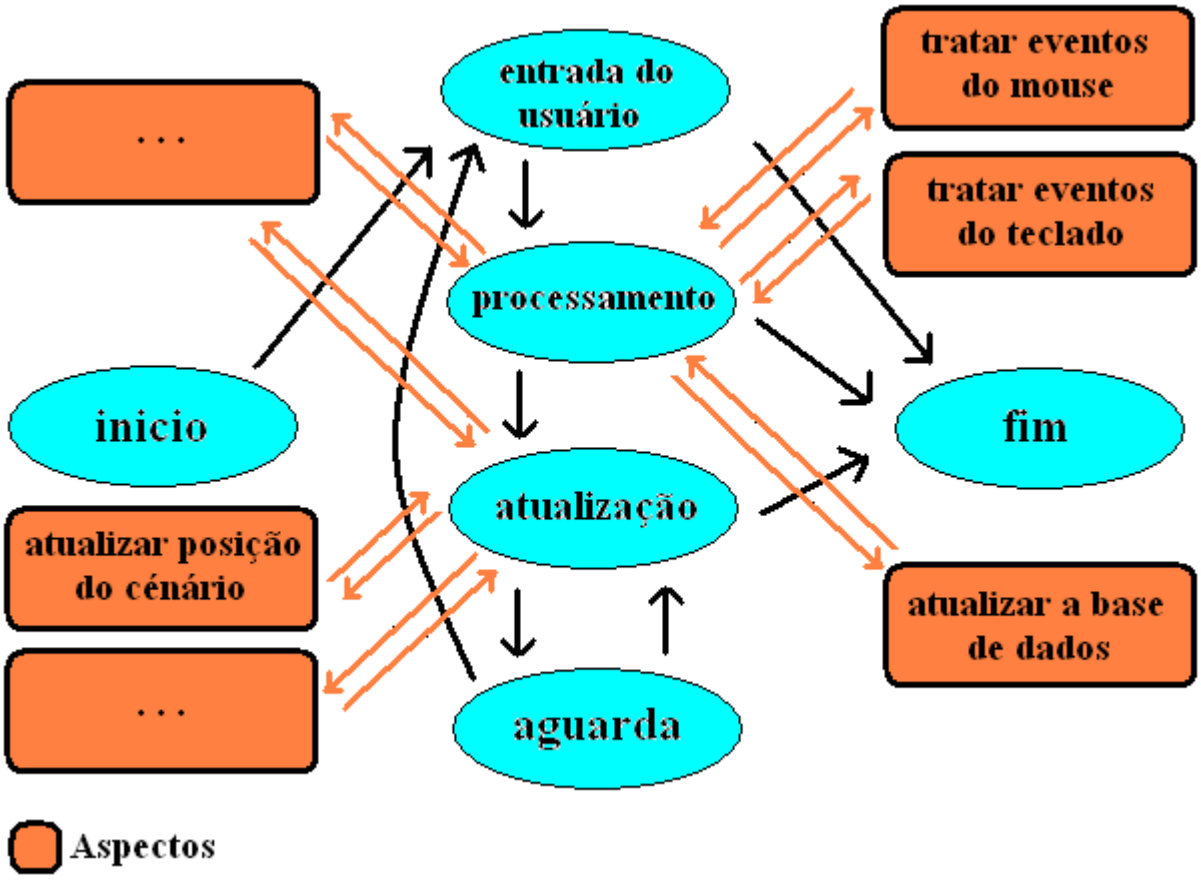

Figura 4 – Execução do jogo adicionando Aspectos

Inicialmente o diagrama parece confuso, essa sensação de confusão se dá pelo fato de agora o diagrama ter mais itens e mais setas indicando fluxo, as setas adicionais na verdade indicam a interferência dos aspectos sobre o código por meio de *pointcuts* e *advices*. Analisando o diagrama podemos concluir que:

- Um determinado trecho de código pode ser afetado por um ou mais aspectos.
- Um aspecto pode afetar um ou mais trecho de código.

- A relação entre aspectos e trechos de código é bidirecional, pois o aspecto corta a aplicação, executa e depois retorna ao trecho que foi cortado.

As funções dos aspectos podem ser mais generalizadas, como “tratar eventos de entrada”, ou mais específicos, como “tratar eventos do mouse”, ou ainda fazer herança de aspectos.

6. MODELAGEM DO JAGAF

Com base na bibliografia, tecnologias estudadas e diagramas construídos criou-se então um modelo para o JAGAF, mostrado na Figura 5:

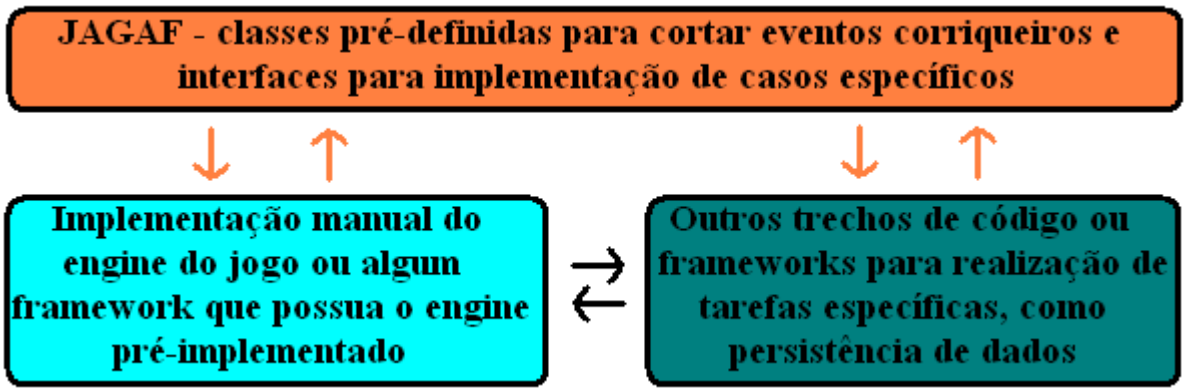

Figura 5 – Modelo de execução do JAGAF

A modelagem baseia-se na criação de aspectos que cortam pontos corriqueiros em um jogo, para casos mais específicos são criados aspectos com chamadas a métodos abstratos que ao serem implementados por terceiros exercerão a funcionalidade desejada. Por padrão cada aspecto deve ser carregado na memória da máquina virtual para ser ativado, porém pode-se optar pela opção de carregar todos automaticamente. Percebe-se que o JAGAF pode ser utilizado para controlar algumas funcionalidades do jogo, e deixar outras para outros *frameworks* compatíveis com a linguagem Java, como era almejado.

7. DISCUSSÃO E CONSIDERAÇÕES FINAIS

O JAGAF foi modelado com base nas necessidades que de um jogo possui e não são atendidas pelo paradigma Orientado a Objetos de forma satisfatória no que diz respeito à estruturação do código e facilidade de programação. Para suprir as necessidades citadas, a modelagem do framework foi feita lançando mão dos recursos do paradigma Orientado a Aspectos pois concluiu-se que tais recursos, como definição de pointcuts e advices, podem resolver ou pelo menos amenizar os problemas de interesses ortogonais e código espalhado.

Como os aspectos trabalham monitorando as classes isoladamente não se tem forte nível de acoplação, o que permite grande flexibilidade do código e compatibilidade com códigos de terceiros, permitindo que jogos dos mais diferentes tipos o utilizem.

O próximo passo é a implementação do modelo criado e sua aplicação no desenvolvimento de jogos. Para a implementação do modelo faz-se necessária a escolha de uma linguagem Orientada a Aspectos compatível com a linguagem Java, no caso, a linguagem escolhida foi o AspectJ (Eclipse Foundation, 2008) por ser uma linguagem já madura com ampla documentação e por ter um desempenho notável, que é algo crucial em um jogo, pois trabalha com o conceito de Modificação de ByteCode. Para a manipulação da linguagem AspectJ juntamente com a linguagem Java foi escolhida a ferramenta Eclipse IDE (Eclipse Foundation, 2008), que é amplamente utilizada por desenvolvedores Java pelo seu magnífico ambiente de desenvolvimento.

Espera-se implementar um *framework* que realmente facilite e organize o desenvolvimento de um jogo a ponto de ser utilizado em projetos futuros e assim contribuir com a comunidade acadêmica.

REFERÊNCIAS

KICZALES, G., LAMPING, J., MENDHEKAR, A., MAEDA, C., Lopes, C., LOINGTIER, J.-M., and IRWIN, J. (1997). **Aspect-Oriented Programming**. Proceedings Europe an Conference on Object-Oriented Programming. Disponível em:
<http://citeseer.nj.nec.com/63210.html > Acesso em: 10 ago 2008.

LOPES, Cristina Isabel Videira. **D: a language framework for distributed programming** . Ph. D. Thesis, Northeast Universi ty, 1997.

BECKER, Christian, GEIHS, Kurt. **Quality of Service - Aspects of Distributed Programs**. Proceedings of the Aspect-Oriented Programming Workshop at ICSE'98. Kyoto (Japão), 1998.

SOUZA JUNIOR, Vicente Goetten de, WINCK, Diogo Vinícius, MACHADO, Caio de Andrade Penteado e. **Programação Orientada a Aspectos Abordando Java e AspectJ**. Disponível em:
<http://inf.unisul.br/~ines/workcomp/cd/pdfs/2337.pdf> Acesso em 10 ago 2008.

DEITEL, P. J. e DEITEL, H. M. **Java como Programar**. Editora Bookman. 2002.

HORSTMANN, Cay S. e CORNEL, Gary. **Core Java volume I (Conceitos Básicos) e volume II (Recursos Avançados)**. Editora Pearson Makron Books. 2003.

Java Oficial Site, Disponível em:
<http://www.sun.com/java/> Acesso em 10 ago 2008.

Eclipse.org Web Site, Disponível em:
<http://eclipse.org> Acesso em 10 ago 2008.

AGRADECIMENTOS

Agradecemos ao Departamento Acadêmico de Informática do CEFET-MA pela disponibilização do Laboratório de Projetos para o desenvolvimento de nossa pesquisa.