Uma Proposta de Arquitetura de Software Genérica para Mobile Games Utilizando J2ME

Arthur Gonçalves de Carvalho

Universidade Federal de Pernambuco (UFPE), Centro de Informática, Recife - PE - Brasil

Resumo

Devido à crescente complexidade dos softwares e sistemas para dispositivos móveis, em especial a área de entretenimento digital e sua subárea jogos eletrônicos, se faz necessário o desenvolvimento de novas práticas, princípios e padrões voltados a esta plataforma. A arquitetura de software está emergindo como uma forte aliada para ajudar no gerenciamento desta complexidade, através de uma clara representação dos componentes do sistema, das comunicações intercomponentes e fornecendo meios para se averiguar a qualidade do sistema. Este artigo apresenta uma arquitetura de software genérica voltada para o desenvolvimento de jogos eletrônicos, baseados na plataforma Java 2 Micro Edition, para dispositivos móveis, preenchendo com isso um vácuo existente no cenário atual.

Palavras-Chave: arquitetura de software, jogos eletrônicos, entretenimento digital, engenharia de software.
Contato:
agc@cin.ufpe.br

1. Introdução

Apesar dos inúmeros avanços na Engenharia de Software, muito ainda é discutido acerca da baixa qualidade e produtividade da indústria mundial de software, refletindo-se na insatisfação dos seus usuários e em prejuízos financeiros de enormes proporções [Oliveira et al. 2005]. Dentre os problemas cruciais no desenvolvimento de softwares e sistemas, pode-se citar a crescente complexidade como um das causas que acarretam custos maiores e incrementam dificuldades em obter um sistema que atenda a necessidade do usuário com requisitos de complexidade cada vez maiores.

No desenvolvimento específico para plataformas móveis o cenário não é diferente. A necessidade de integração com outras plataformas, dispositivos com constante crescimento na capacidade de processamento e conseqüentemente um amplo desejo por aplicativos com requisitos mais complexos por parte dos usuários, são apenas alguns fatores que incrementam a complexidade no desenvolvimento. Tendo a área de entretenimento digital como a mais promissora e em grande voga atualmente, existe a necessita de novos paradigmas e padrões de desenvolvimento a fim de atender a já citada demanda de softwares mais complexos por parte dos usuários. Entretanto poucas propostas foram definidas focando as características específicas desta área e desta plataforma. O desenvolvimento de software, na grande maioria, ainda se faz por técnicas advindas de outras plataformas, muitas vezes inadequadas.

Devido a estes fatores, se faz necessário o desenvolvimento de novas práticas, princípios e padrões ou adaptações oriundas de outras plataformas e/ou áreas, para auxílio no desenvolvimento para a área de entretenimento digital, mais especificamente o campo dos jogos eletrônicos.

Uma disciplina pouco, ou nada, aproveitada no desenvolvimento para dispositivos móveis, no entanto bastante comum em outras plataformas, é a arquitetura de software. Esta é uma disciplina, em constante ascensão, que foca no design mais abstrato possível do software. Com ela é possível visualizar várias características dos componentes de um software, suas relações e diversas métricas de qualidade antes mesmo de seu desenvolvimento [Albin 2003].

Este artigo propõe uma definição genérica, para uma arquitetura de software, voltada exclusivamente para o desenvolvimento de jogos eletrônicos para dispositivos móveis. Além desta seção introdutória, o artigo apresenta outras cinco seções. A seção 2 fornece uma introdução geral da disciplina de arquitetura de software. A seção 3 aborda minuciosamente a arquitetura three-tier, a qual serviu como base para o desenvolvimento da arquitetura aqui proposta, por ser a mais conhecida e utilizada no desenvolvimento de sistemas de informação. A definição da arquitetura genérica é detalhada na seção 4. A seção 5 aborda um estudo de caso em que foi desenvolvido o jogo eletrônico Dómino Mobile utilizando a arquitetura aqui proposta. Finalmente, a seção 6 apresenta as considerações finais deste artigo.

2. Arquitetura de Software

Em 1968, Edsger Dijkstra publicou um artigo sobre o design de um sistema chamado "THE" [Dijkstra 1968]. Este foi um dos primeiros artigos a documentar o design de um sistema de software. Dijkstra o organizou em camadas com o intuito de reduzir sua complexidade geral. Embora o termo arquitetura não tivesse sido usado ainda para descrever o design de um sistema, este certamente foi um dos primeiros vislumbres de arquitetura de software. Apenas em meados de 1980s foi que este termo apareceu na literatura.

A arquitetura de software surgiu devido à ineficiência de outros modelos em mostrar vários atributos de qualidade requeridos nos sistemas atuais. Com o seu uso e possível raciocinar sobre os cinco atributos básicos em um sistema de software (confiabilidade, funcionalidade, modularização, performa e segurança) antes mesmo de seu desenvolvimento, baseando-se em modelos de design e descrições de arquitetura.

O design de uma arquitetura foca na decomposição do sistema em componentes e em interações entre estes a fim de satisfazer requerimentos funcionais e não funcionais.

2.1 Design de Arquitetura

O processo de design de uma arquitetura de software pode ser descrita pelos seguintes passos:

Entender o problema a ser resolvido com o software e definir o contexto do sistema: O contexto do sistema ajuda a visualizar a aplicação de uma perspectiva externa, facilitando a descrição da proposta do sistema assim como a identificação de interfaces externas.

Identificar os componentes: Este passo envolve a aplicação de operações a fim de dividir o sistema em componentes. Estes podem existir tanto na forma binária como em código fonte. Na forma binária, um componente é integrado dinamicamente ao sistema, sem uma prévia necessidade de recompilação. Na forma de código fonte, um componente representa um conjunto de arquivos de código fonte que exibem alta coesão interna e baixa dependência externa. Este último termo se refere às interações ou conexões entre componentes. O termo coesão se refere à interação ou conexão entre elementos dentro de um módulo. Para se alcançar uma baixa dependência externa, a interface entre dois componentes precisa ser estável e estática.

Descrever os componentes e conectores. Este passo envolve a criar descrições de exemplos de instâncias de componentes em uma típica configuração de execução.

Avaliação do design da arquitetura: Este passo envolve avaliar o design da arquitetura a fim de determinar se ela satisfaz os requerimentos iniciais. O design é avaliado recolhendo-se medidas quantitativas e/ou qualitativas e comparando-as com as exigências dos atributos de qualidade. Para se avaliar um projeto da arquitetura, deve-se ter claramente articulado as exigências dos atributos de qualidade que serão afetadas pela arquitetura.

Transformar o design da arquitetura: Se o design da arquitetura não completa, satisfatoriamente, os atributos de qualidade requeridos, então novas reformulações se farão necessárias até que os satisfaça.

3. Trabalhos Relacionados

Diversas arquiteturas de software já foram propostas para o desenvolvimento de softwares e sistemas, para as mais diversas plataformas. Apesar de uma grande quantidade permanecer privada e pertencer a grandes organizações, diversas arquiteturas estão publicamente disponíveis. A que mais se destaca na atualidade é a clássica three-tier (também conhecida como three-layers ou arquitetura em camadas). A sua principal característica é a modularidade. A Figura 1 demonstra a arquitetura.

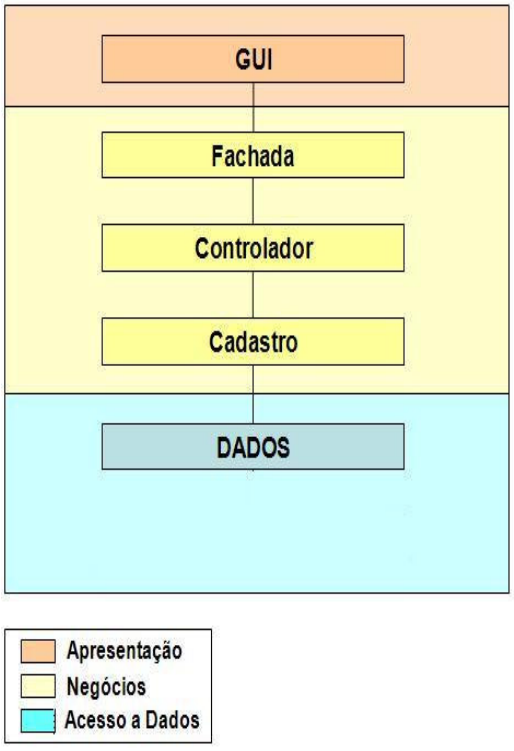

Figura 1: Arquitetura Three-Tier

A seguir serão descritas as camadas da arquitetura bem como os elementos que compõe cada camada.

Camada de Apresentação: É a camada responsável por promover a interação entre o usuário e a aplicação. Capturando as entradas fornecidas e apresentado respostas.

GUI (*Graphic User Interface*): é a porta de entrada do software. Implementa as telas da aplicação e é responsável por exibir as informações, ou seja, o resultado do processamento dos dados pelo sistema. Resumidamente e a responsável por capturar as ações realizadas pelo usuário e exibir as respostas advindas do sistema. Entre as possibilidades de interfaces estão: comandos de texto e janelas de programas.

Camada de Negócios: Tem por objetivo encapsular a manipulação de dados e o funcionamento da aplicação, separando assim, as regras de negócios da interface gráfica e da persistência dos dados. Nesta camada estão armazenadas todas as regras relacionadas ao domínio da aplicação, necessárias para manter o sistema consistente do ponto de vista do negócio. É responsável por validar os dados fornecidos pelos usuários. Geralmente é onde estão modeladas as entidades (classes).

Fachada: o sistema possui apenas um elemento desse tipo (padrão de projeto Singleton). Tem a responsabilidade de fornecer uma entrada única do sistema para acesso às regras de negócio, tornando o sistema independente de interface gráfica.

Controlador: responsável por mapear e controlar as ações, servindo como uma camada intermediária entre a camada de apresentação e a camada lógica. Têm a finalidade de agrupar funcionalidades, fluxos de execução semelhantes, provendo uma melhor organização lógica ao software. É responsável pela implementação das regras de negócio necessárias a execução dos fluxos de atividades;

Cadastro: faz a comunicação entre a camada de negócios e a camada responsável pela persistência dos dados, tornando o sistema independente da implementação do acesso a dados.

Camada de Acesso a Dados: Nesta camada estão definidos os mecanismos para tornar persistentes os dados em processamento pelo sistema e a recuperação destes. Aqui se toma a decisão dos mecanismos de persistência que serão usados, ou seja, é responsável pelo armazenamento e recuperação dos dados.

Com o advento da popularização da plataforma JAVA, a camada three-tier é hoje extremamente popualar com o uso do trio Java servlets (JavaServer Pages - JSP), JavaBeans e algum SGBD open-source (geralmente MySql).

Devido à arquitetura three-tier não levar em consideração as características específicas dos dispositivos móveis, da área de entretenimento digital e da plataforma J2ME, ela está fadada à inutilidade no desenvolvimento de softwares complexos que possuam estas características.

4. Arquitetura Genérica

A arquitetura aqui definida utilizará a three-tier como base, devido à sua enorme popularidade. A sua principal meta será modularizar o software de tal forma que possa oferecer uma grande facilidade de visualização, reuso e facilidade na implementação dos componentes, ao mesmo em que busca um alto grau de satisfatoriedade nos atributos de confiabilidade, funcionalidade, performa e segurança. A figura 2 demonstra a arquitetura utilizando diagrama com pacotes e a figura 3 utilizando matrix estruturada.

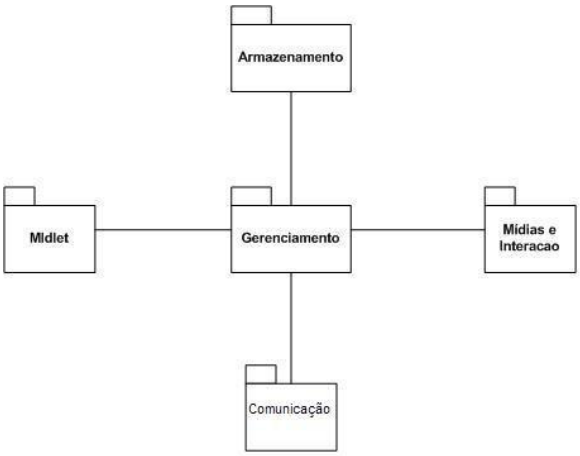

Figura 2: Arquitetura Proposta

	Midlet	Arma.	Mídias interação	Comu.	Geren
	0				X
Armazenamento		0			X
Mídias e interação			0		X
Comunicação				0	X
Gerenciamento	X	X	X	X	0

Figura 3. As dependência de um elemento são representadas com um X.

A seguir serão descritas as camadas da arquitetura, observando, na medida do possível, suas funções e os componentes que a compõe.

Camada MIdlet: Camada que possui a classe que inicia o aplicativo, geralmente única. Contem apenas uma referencia às demais camadas do projeto: gerenciamento.

Camada Gerenciamento: Camada projetada visando tornar o sistema modular e com componentes independentes. Ela é responsável por fazer um link entre todas as demais camadas, que estão ligadas

diretamente a ela, seja através de suas classes ou através de interfaces de acesso.

Camada de Armazenamento: Devido à peculiaridade de J2ME com relação ao armazenamento persistente, com uma forma padronizada e única até o presente momento, a camada equivalente do three-tier é desvinculada da camada de negócios e posta diretamente em contato com a de gerenciamento. Pode possuir várias entidade (classe) ou apenas uma única com vários registros, dependendo da necessidade do aplicativo.

Camada de Comunicação: Camada responsável por realizar e gerenciar a comunicação e a segurança entre dispositivos.

Camada de mídias e Interação: Camada que será responsável por atualizar a tela indiretamente, tratar eventos de entrada e saída com o usuário, tratar da sonorização específica para cada tela além de contar com várias entidades relativas ao game. O uso de hierarquias pode ser bastante útil para o reuso de imagens, sons e manipulações de entrada e saída.

5. Estudo de Caso

Para introduzir a teoria desta arquitetura em uma implementação foi desenvolvido o game Dominó Mobile, cujo diagrama de classes simplificado é demonstrado na figura 4.

O game trata de um jogo de dominó, single player, onde existe a posibilidade do usuário alterar nome dos adversários NPCs (Non-Player Characters), visualizar e alterar recordes, conta com recursos multimídias como sonorização e uma interface fácil e intuitiva. A inteligência artificial do game foi negligenciada por não ser o foco inicial deste software.

O game, tal qual proposto na arquitetura, com exceção da camada de comunicação, está estruturado em cinco pacotes cujas classes são descritas a seguir: Informações sobre o projeto, código fonte e diagrama de classe, podem ser encontradas em http://www.cin.ufpe.br/~agc/es.

Pacote MIdlet: Pacote possui uma única classe, DominoMobile, responsável por iniciar o jogo e interagir com o pacote de Gerenciamento para imprimir imagens na tela.

Pacote Gerenciamento: Pacote que contém uma única entidade, Container, responsável por gerenciar chamadas e transações entre as demais classes dos diversos pacotes (componentes) do software. É responsável também por atualizar as telas, indiretamente, através de chamadas a métodos da classe DominoMobile.

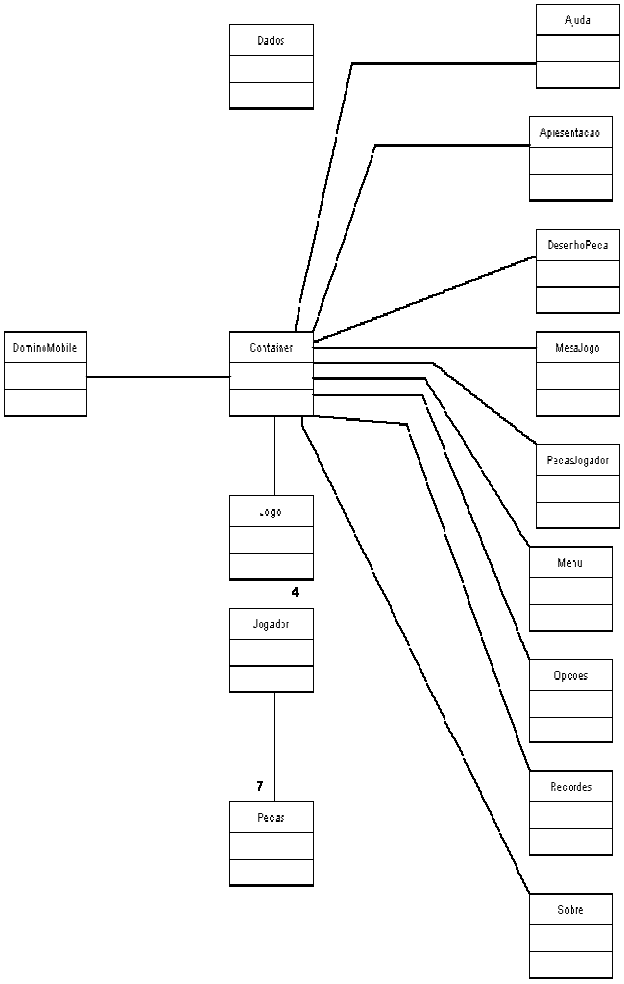

Figura 4: Diagrama de classes do jogo Dominó Mobile

Pacote Dados: Este pacote contém uma única classe chamada Dados que armazena e fornece o acesso a informações, como o nome do personagem que o usuário controla, nome dos NPCs e pontuação acumulada de todos os jogadores. Um único registro foi criado para estas tarefas.

Pacote Telas: Pacote composto por varias classes, cada qual responsável por informar o que vai ser impresso na tela, gerenciar a sonorização, além de ser responsável por tratar a entrada e saída feita pelo usuário. Este deverá ser o módulo no qual o desenvolvedor tenha menos oportunidades de reuso, devido a ser neste componente que residem as peculiaridades do jogo.

Apesar de negligenciado na definição da arquitetura, existe um outro pacote chamado básicas, que contêm as entidades do jogo. Aqui são definidas todas as classes necessárias para representação fidedigna do jogo de dominó. As entidades aqui

definidas incluem a classe jogo, estado onde o aplicativo estará no momento do "jogo em si", a classe jogador, que representa os integrantes da mesa, e a classe peça, que representa as peças do jogo. Como a classe jogo é instanciada apenas uma vez no aplicativo, o container fica com esta responsabilidade além de ficar incubido de repassar esta classe para várias referências que a utilizam no jogo.

Este exemplo foi extremamente útil, pois proporcionou a aplicabilidade de uma pesquisa na prática, demonstrando muito potencial a ser utilizado no presente e explorado no futuro.

6. Conclusão

Com a crescente complexidade exigida nos softwares para dispositivos móveis, em particular nos jogos eletrônicos, a disciplina de arquitetura de software propõe uma abordagem que ajuda a gerenciar tal complexidade. Criar abordagens que considerem a plataforma de desenvolvimento, que possibilite visualizar o sistema em módulos e observar as relações entre estes, torna o passo da implementação extremamente mais eficaz e eficiente.

Este artigo propôs uma arquitetura de software genérica para o desenvolvimento de jogos eletrônicos, utilizando a plataforma J2ME, em dispositivos móveis. A sua potencialidade ainda não foi plenamente averiguada, mas através do caso de uso proposto pode-se observar o potencial que ela possui.

Os resultados, empíricos, dos testes com está arquitetura são descritos a seguir com base nas principais métricas de testes para arquiteturas de software:

Desempenho: Devido à grande modularização no software, o desempenho da arquitetura proposta aparenta sofrer um grande impacto com chamadas extras a métodos. Apesar de haver esse acréscimo de chamadas a modularização favorece o desempenho no seguinte aspecto: Ao se desenvolver um jogo em grandes blocos monolíticos, toda esta classe ficará armazenada na memória do celular, consumindo recursos desnecessariamente, enquanto com a modularização aqui proposta, apenas pequenas entidades, que estão em uso atualmente, consumirão recursos. Um exemplo concreto seria um jogo de ação com várias fases (levels). Com um grande bloco monolítico, todas as fases estariam na memória mesmo não sendo utilizadas. Com a arquitetura aqui proposta, apenas uma fase seria carregada na memória, uma classe responsável por esta fase, e com seu término existiria um gasto extra de uma chamada para que o gerenciador mude de fase e chame o Garbage Collector de JAVA para limpar aquela classe da memória. Além disso, dispositivos móveis atualmente (como por exemplo, celulares e Palms) crescem constantemente na capacidade de armazenamento e processamento.

Manutenção: A clara modularização e divisão de tarefas ajudam na prevenção, correção e manutenção do software.

Modularidade e Reuso: O grande foco no desenvolvimento desta arquitetura. O maior ganho na sua implementação está contida nesta característica. A divisão, feita através de técnicas de design de arquiteturas, tenta prover o máximo de modularidade ao software e independência entre os módulos. Devido a características intrínsecas da área de jogos eletrônicos, a total independência é completamente impossível de se alcançar. Devido a esta característica, o reuso de componentes, ou parte dele, poderá ser utilizado sem nenhuma, ou pouca, modificação na sua implementação. Exemplos podem ser tirados do estudo de caso Dominó Mobile. Partes do componente mídias e interação, como as classes Sobre, Menu, ou Apresentação poderão ser complemente reusadas. Além disso, módulos inteiros, como o de comunicação e o de acesso a dados, sofrerão poucas, ou nenhuma, mudanças em um possível reuso.

Complexidade: A facilidade de se visualizar o software como um todo, seus componentes e suas interações, facilitam de sobremaneira o desenvolvimento e gerenciamento do software, além de ajudar na divisão de tarefas para especialistas. Com essa arquitetura um perito em comunicação e segurança poderá realizar sua tarefa em paralelo com o desenvolvimento do restante do jogo e na integração o componente de gerenciamento facilitará a junção dos componentes.

Devido à positividade destes fatores, Acredita-se no potencial desta arquitetura como provedora de meios de se obter maior eficiência e eficácia no desenvolvimento de jogos eletrônicos para dispositivos móveis.

6.1 Trabalhos Futuros

Como futuros trabalhos para a conclusão desta pesquisa, existem a necessidade de uma formalização da arquitetura com ADLs (Architecture description languages) que ainda se encontram em estado da arte. Apesar de muito poderosas, elas são pouco utilizadas na prática, devido à complexidade existente no seu aprendizado e à falta de padrões.

Uma outra necessidade seria a de aplicar testes com jogos pertencentes às mais variadas categorias. Com isso, uma confirmação da corretude da arquitetura e sua generalização poderiam ser obtidas.

Referências

ALBIN, Stephen T. The Art of Software Architecture : Design Methods and Techniques, Wiley Press 2003 ISBN 0-471-22886-9.

BASS, LEN, CLEMENTS, PAUL and KAZMAN, RICK. Software Architecture in Practice, Second Edition, Addison-Wesley 2003 ISBN 0-321-15495-9.

BALDWIN, C.,CLARK, K. 2000. *Design Rules: Volume 1. The Power of Modularity*. Cambridge, MA: The MIT Press.

BOOCH, G. & RUMBAUGH, J. & JACOBSON, I.: The Unified Modeling Language User Guide. Addison-Wesley, 1999.

BOSCH, J. 2000. Design and Use of Software Architectures: Adopting and Evolving a Product-Line Approach. Harlow, England: Addison-Wesley.

DIJKSTRA, E. W. 1968. "The Structure of the 'THE' Multiprogramming System." *Communications of the ACM* 11, no. 5: 341-346.

IEEE 1990. "IEEE Standard Glossary of Software Engineering Terminology." *IEEE Std 610.12-1990*. New York, NY: Institute of Electrical and Electronics Engineers.

IEEE 2000. "IEEE Recommended Practice for Architectural Descriptions of Software Intensive Systems." *IEEE Std 1471-2000*. New York, NY: Institute of Electrical and Electronics Engineers.

OLIVEIRA, S. R. B., VASCONCELOS, A. M. L., ROUILLER, A. C. Uma Proposta de um Ambiente de Implementação de Processo de Software, Revista InfoComp – Revista de Ciência da Computação da UFLA – vol. 4, n. 1, Lavras-MG, 2005.

SEWELL, M., SEWELL, L. 2002. *The Software Architect's Profession: An Introduction*. Upper Saddle River, NJ: Prentice Hall PTR.

SOMMERVILLE, Ian. Engenharia de Software: Addison Wesley, 2003.