Sistema de Proteção para Servidores de Jogos Online Contra Softwares Clientes Não Oficiais

Thiago Alexandre Gesser

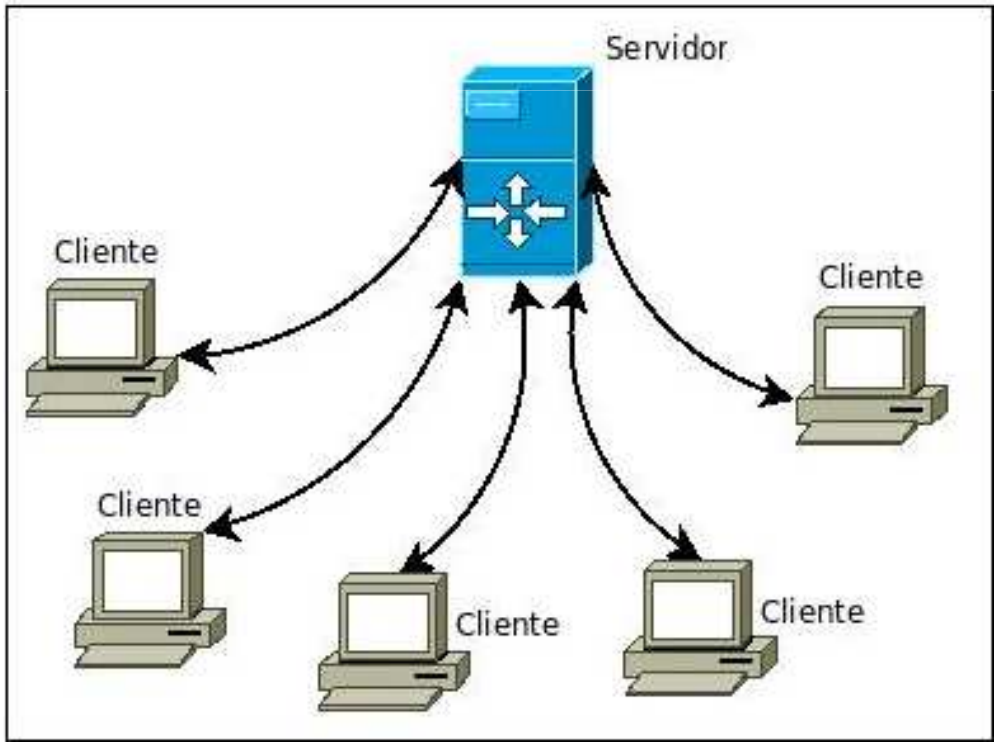
Orientador
Paulo Fernando da Silva

Roteiro

- Introdução
- Fundamentação Teórica
- Desenvolvimento
- Conclusão
- Extensões
- Demonstração do sistema

Introdução

- Jogos online
  - Comunicação



Introdução

- Jogos online
  - Problemas de segurança
    - Modificações no software cliente
    - Engenharia reversa
    - Resulta em trapaças
  - Problemas se aplicam a qualquer sistema cliente/servidor.
- Sistemas de segurança atuais
  - Comerciais
  - Burláveis
- Sistema de segurança desenvolvido
  - Novas técnicas de segurança

Introdução

- Objetivos
  - Desenvolver um sistema de proteção para servidores de jogos online
    - Impeça a utilização de softwares clientes não oficiais
    - Controle das conexões recebidas pelo software servidor
    - Monitoramento do software cliente.

Fundamentação Teórica - Roteiro

- Ataques contra a criptografia
- Criptografia de caixa branca
- Algoritmos de hashing gerados aleatoriamente
- Proteção de código Java
- Atualização dinâmica de código Java
- Confiabilidade de software remoto

- Trabalhos Correlatos

Fundamentação Teórica

- Ataques contra a criptografia
  - Caixa preta
    - Invasor apenas observa a comunicação
    - Duas entidades mutuamente confiáveis
  - Caixa cinza
    - Tenta buscar informações sobre comportamento interno das entidades
      - Algoritmos de criptografia
  - Caixa branca:
    - Entidades de comunicação não mutuamente confiáveis
    - Uma das entidades pode ser um invasor
    - Possui total acesso algoritmo de criptografia utilizado

Fundamentação Teórica

- Criptografia de caixa branca
  - Técnica de ofuscação de código
  - Impossibilitar extração de informações sobre a chave
    - Chave deve ser espalhadas sobre toda a implementação
    - Compor o algoritmo de maneira aleatória
  - Autenticação de entidades de comunicação não confiáveis

Fundamentação Teórica

- Criptografia de caixa branca
  - Gerador de AES de caixa branca
    - Advanced Encryption Standard (AES)
      - Algoritmo de criptografia de bloco padrão dos Estados Unidos.
      - Operações de criptografia:
        - SubBytes, ShiftRows, MixColumns e AddRoundKey
    - Recebe como entrada uma chave normal de AES
    - Saída é um AES customizado para a chave de entrada
    - Operações de criptografia do AES podem ser convertidas em tabelas de consulta
      - Rede de tabelas de consulta
      - Chave escondida nas tabelas

Fundamentação Teórica

- Algoritmos de hashing gerados aleatoriamente
  - Mesmas funcionalidades de algoritmos de hashing comuns
  - Cada versão gerada produz resultados diferentes
  - Autenticação de entidades de comunicação não confiáveis

Fundamentação Teórica

- Proteção de código Java
  - Classes Java compiladas são fáceis de fazer engenharia reversa
  - Encriptação de classes
    - Não garante proteção
    - JVM não disponibiliza uma maneira nativa de carregar classes encriptadas
  - Ofuscação de código
    - Código compilado menos compreensível
    - Dificulta a engenharia reversa ou até impede
    - Produz o mesmo resultado
    - Menos performance

Fundamentação Teórica

- Atualização dinâmica de código Java
  - Técnica para alterar as instruções de uma aplicação enquanto ela está executando
  - Máquinas virtuais torna mais fácil.
  - Java - carregadores de classes alternativos
    - Atualização dinâmica de código vindo de qualquer lugar

Fundamentação Teórica

- Confiabilidade de software remoto
  - Como confiar em um software executado em um ambiente remoto não confiável?
    - Recursos utilizados pelo software sob controle de usuários mal-intencionados
      - memória, processador, dispositivos de entra/saída
  - Verificação da integridade do código
    - Antes de ser carregado
    - Depois de executado
  - Verificação do ambiente de execução
    - Softwares maliciosos, recursos do sistema

Fundamentação Teórica

- Confiabilidade de software remoto
  - Monitoramento do software remoto
    - Através de um software monitor confiável
    - Autenticidade do monitor
      - Criptografia de caixa branca
      - Algoritmos de hashing gerados aleatoriamente
    - Coleta informações
      - Software remoto e ambiente de execução
    - Envia para o servidor validar

Fundamentação Teórica

- Trabalhos correlatos
  - GameGuard, NProtect, Coréia
  - HackShield, AhnLab, Coréia
  - GameFort, GameFort, Brasil

  - Ferramentas comerciais
    - Não foi possível fazer um estudo aprofundado

  - GameGuard e GameFort já tiveram a proteção burlada
    - Comunidade do cliente não oficial OpenKore
    - Jogo online Ragnarök

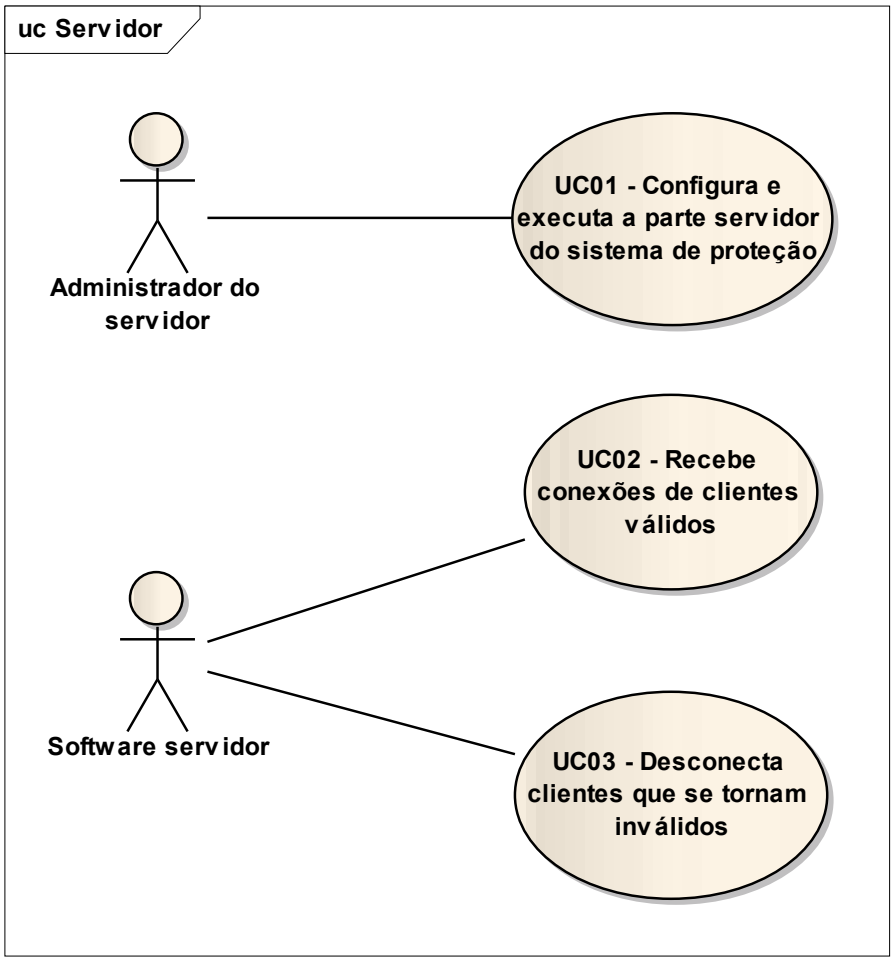
Desenvolvimento - Roteiro

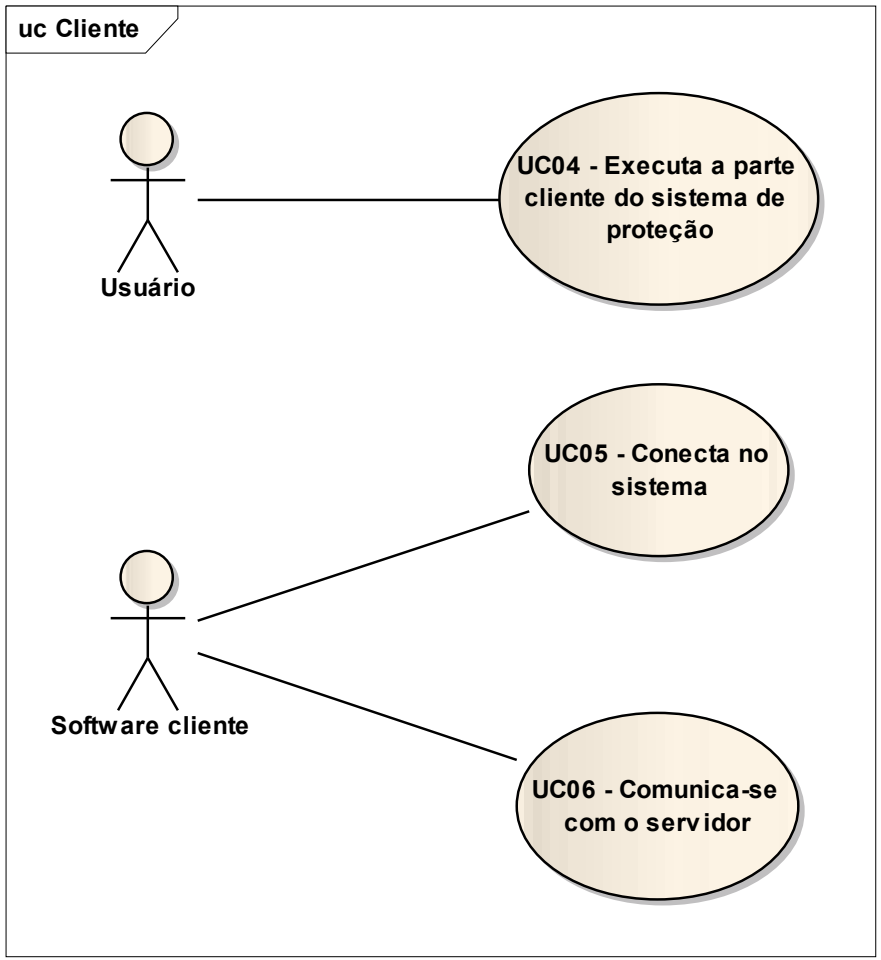
- Requisitos
- Especificação
- Implementação
- Operacionalidade do sistema
- Resultado e discussão

Desenvolvimento

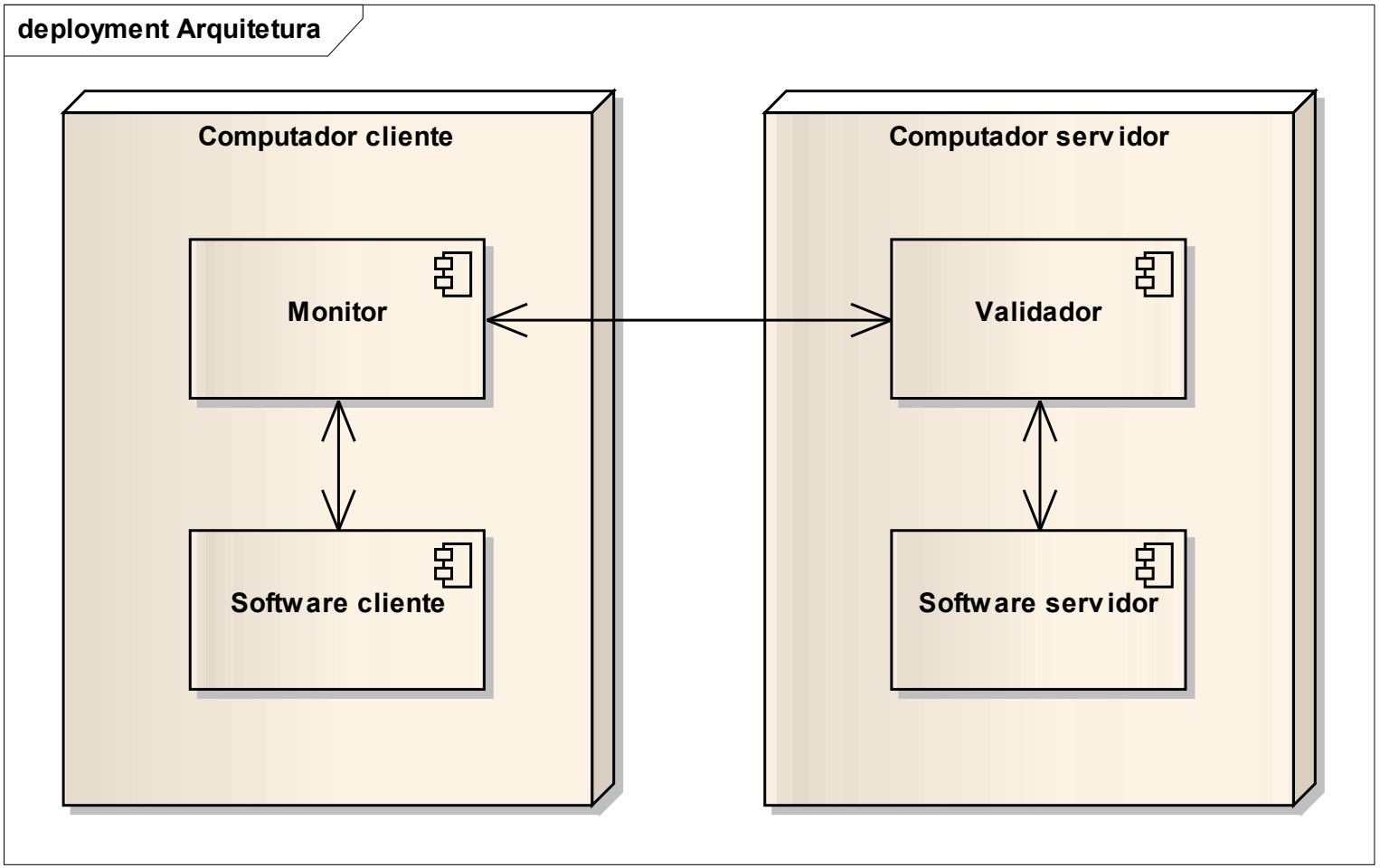
- Requisitos
  - O software cliente só pode ser executado depois que o conteúdo de seu arquivo executável e o estado do ambiente de execução forem validados (RF);
  - O software servidor deve receber apenas conexões de softwares clientes validados (RF);
  - Permitir que sejam feitas validações do estado do ambiente de execução e do software cliente durante sua execução (RF);
  - As validações devem ser feitas automaticamente, sem a necessidade de interação com o usuário do software cliente (RF).

Desenvolvimento – Casos de uso





Desenvolvimento - Arquitetura

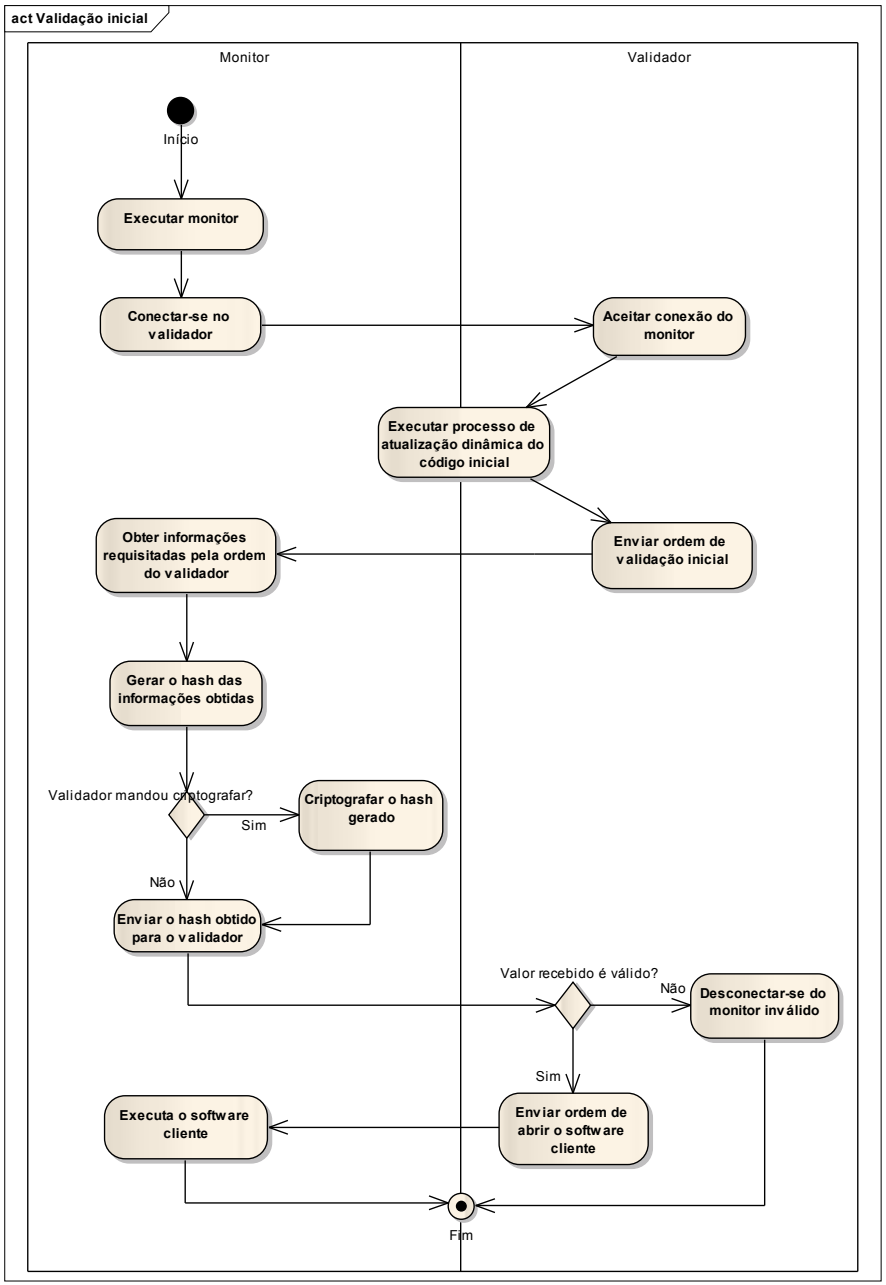


Desenvolvimento

- Especificação
  - Arquitetura
    - Comunicação
      - Um canal de comunicação de controle
      - N canais de comunicação para a transmissão de dados entre o software cliente e o software servidor
    - Proteção do código do monitor
      - Ofuscado utilizando o JBCO
      - Atualização dinâmica de código
        - Inicialmente, para contextualizar o monitor (ordens do validador)
        - Periodicamente, em intervalos de tempos aleatórios

Desenvolvimento

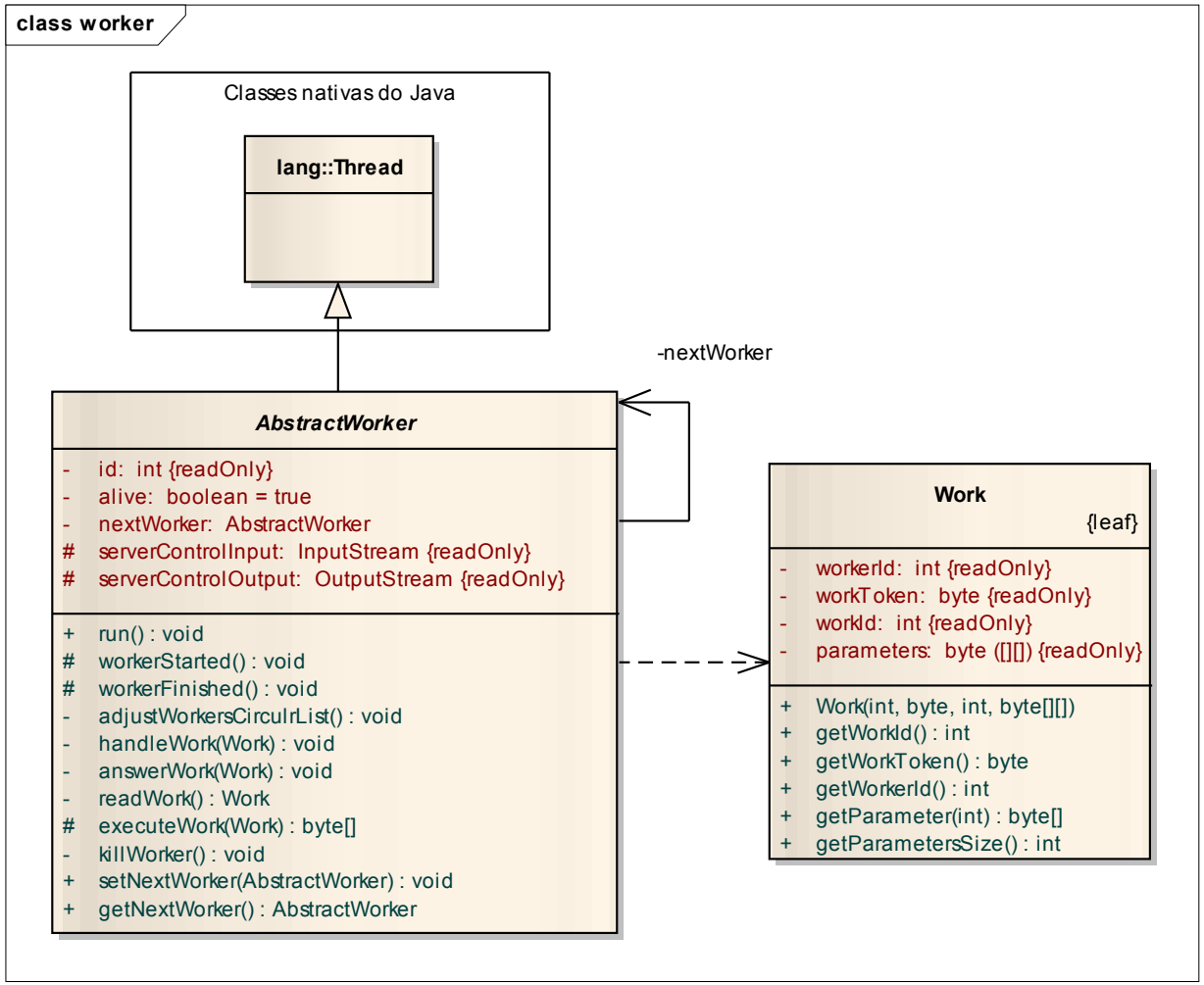
- Especificação
  - Arquitetura
    - Validações
      - Utilizam algoritmos de hashing e de AES de caixa branca, ambos gerados aleatoriamente
      - Validação inicial
        - Hash do conteúdo do software cliente
        - Ambiente de execução
        - Autenticidade do monitor
      - Validação periódica


act Validação inicial
Monitor
Validador
Início
Executar monitor
Conectar-se no validador
Aceitar conexão do monitor
Executar processo de atualização dinâmica do código inicial
Enviar ordem de validação inicial
Obter informações requisitadas pela ordem do validador
Gerar o hash das informações obtidas
Validador mandou criptografar?
Sim
Criptografar o hash gerado
Não
Enviar o hash obtido para o validador
Valor recebido é válido?
Não
Desconectar-se do monitor inválido
Sim
Enviar ordem de abrir o software cliente
Executa o software cliente
Fim

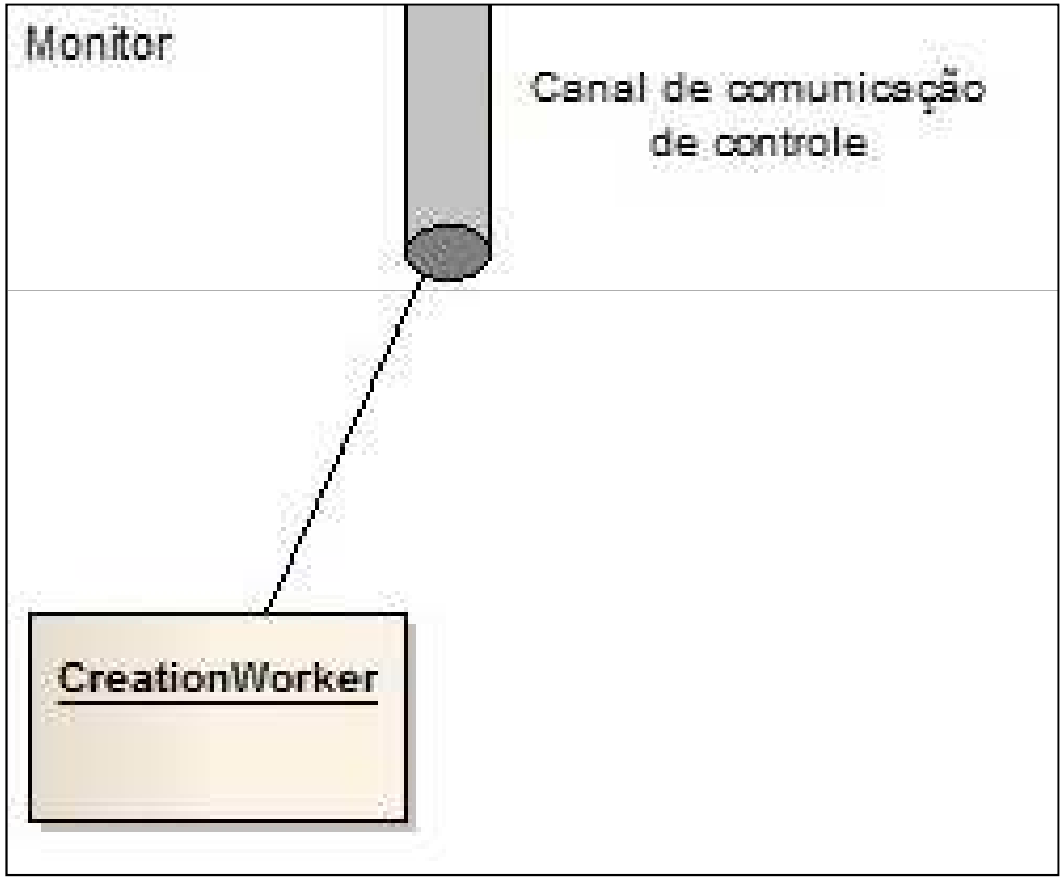
Desenvolvimento

- Especificação
  - Trabalhadores
    - Mão de obra que o validador envia para o monitor
      - Mínimo de lógica mantida no monitor
        - Trabalhador de Criação
    - Validador gerencia os trabalhadores
      - Envia trabalhos
      - Valida as respostas
    - Validador
      - Gera e compila
    - Monitor
      - Recebe, instancia e executa
    - Controle total do código do monitor

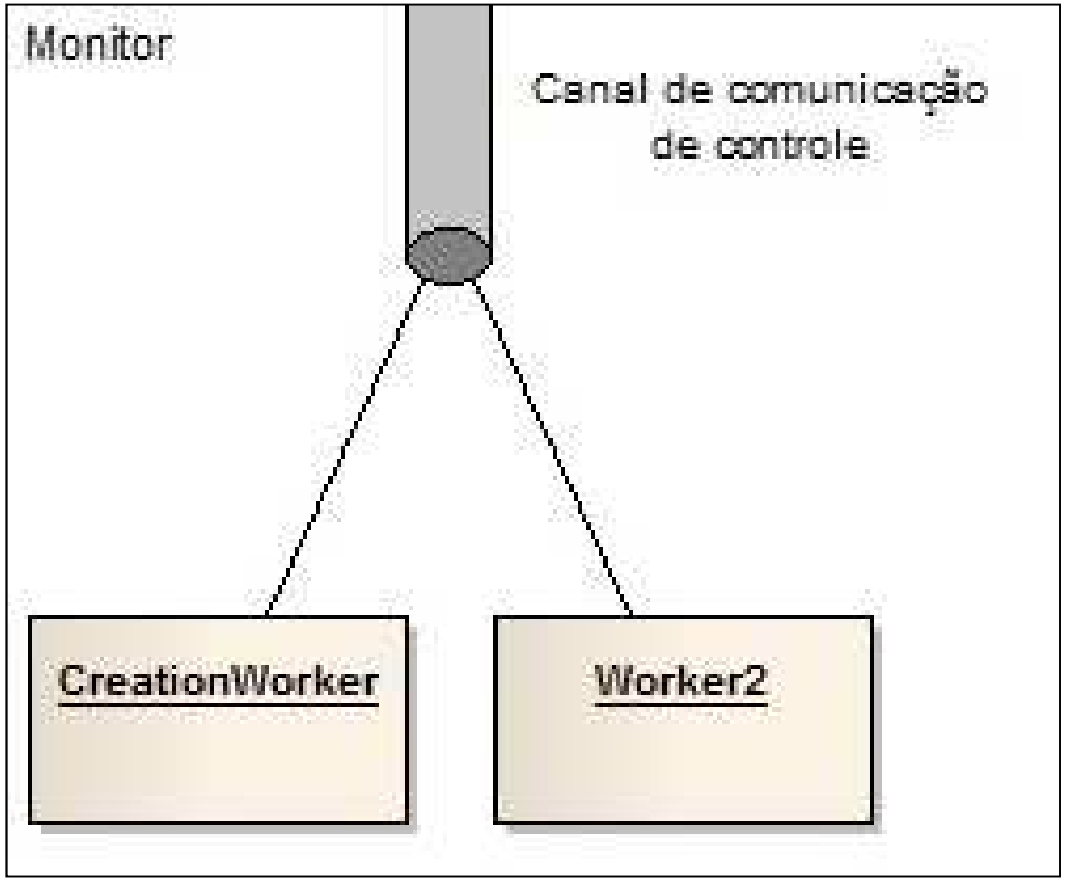
Desenvolvimento - Trabalhadores



Desenvolvimento - Trabalhadores



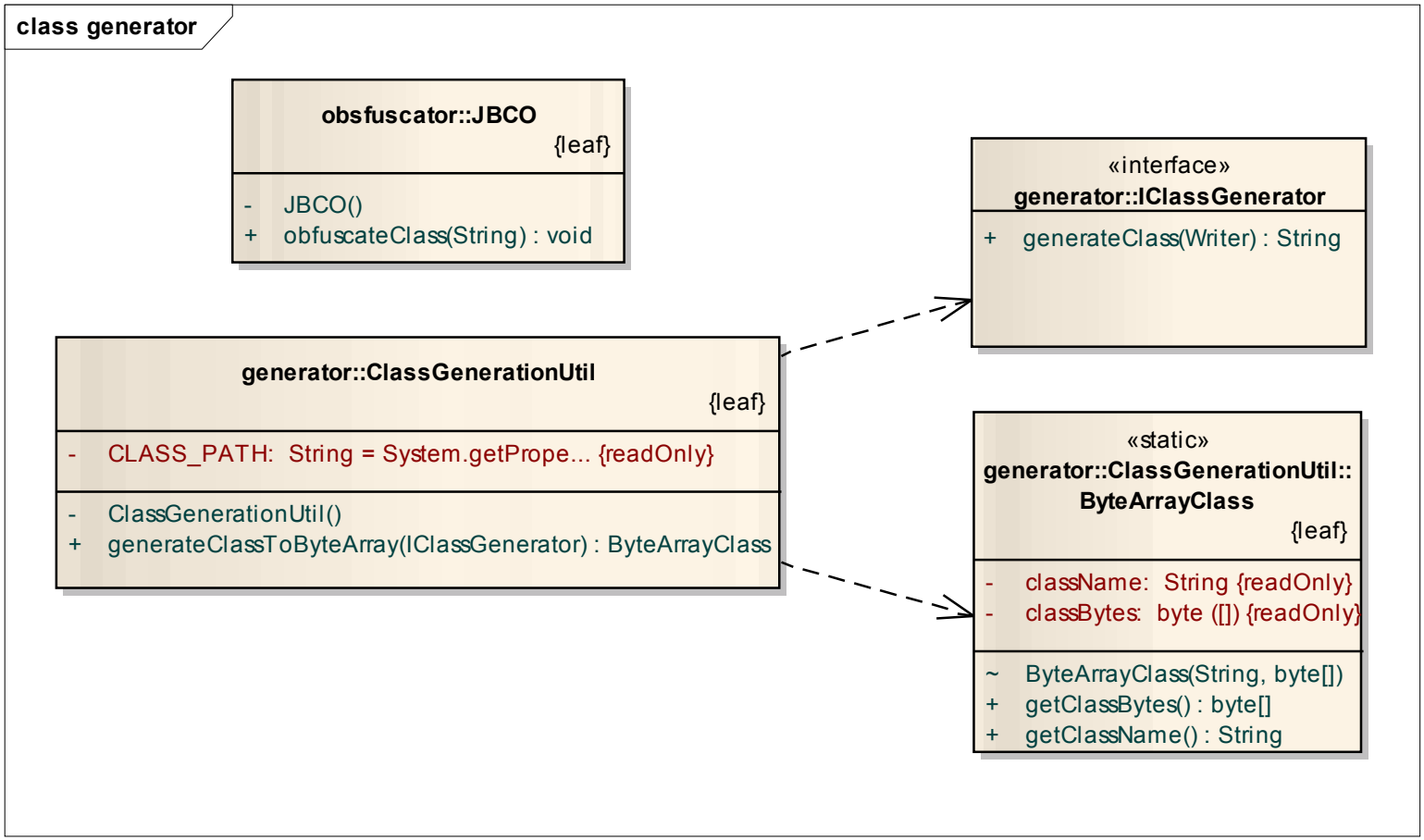
Desenvolvimento - Trabalhadores



Desenvolvimento

- Especificação
  - Geradores de classes
    - Todo código enviado ao monitor é gerado através de um gerador de classes
    - Trabalhadores
    - Algoritmos de AES de caixa branca gerados aleatoriamente
    - Algoritmos de hashing gerados aleatoriamente

Desenvolvimento



Desenvolvimento

- Implementação
  - Técnicas e ferramentas utilizadas
    - Linguagem de programação: Java 1.6
    - Ambiente de desenvolvimento: Eclipse 3.6
    - Geração de código: Apache Velocity 1.6.4
    - Compilação de código: biblioteca tools, nativa do JDK
    - Algoritmo de compressão de dados: LZMA
    - Ofuscador de código Java: JBCO
    - Testes automatizados: Junit 4.3.1

Desenvolvimento

- Implementação
  - Gerador aleatório de algoritmos de AES de caixa branca
    - Implementado de acordo com o estudo realizado
    - Testes automatizados
      - Garantir que criptografava como o AES durante o desenvolvimento
    - Classes geradas passam de 2 mb
      - Com o LZMA, passaram a pouco mais de 200 kb
    - Implementação da decriptografia do AES de caixa branca
      - Não estava especificado nos estudos

Desenvolvimento

- Implementação
  - Gerador aleatório de algoritmos de hashing
    - Criada estrutura de decoradores de filtros de entrada e saída
      - Número de filtros de cada algoritmo gerado é aleatório
      - Filtros gerados aleatoriamente
    - Algoritmo de hashing principal é da própria API Java
      - Escolhido aleatoriamente

Desenvolvimento

- Implementação
  - Controle de sessão
    - Todo monitor que se conecta no validador recebe uma sessão
    - Cliente invalidado
      - Sessão invalidada
    - Invalidação da sessão
      - Todos os canais de comunicação com o cliente inválido são fechados.

Desenvolvimento – Conexão controle

```
@Override
protected void newConnection(Socket socket) {
      //Cria a sessão.
      ClientSession session = new ClientSession();

    //Cria e inicia o manager deste client.
    ClientSessionManager sessionManager = new ClientSessionManager(socket,
session);
    sessionManager.start();

    //Se a sessão está valida, coloca no mapa de sessões válidas.
    synchronized (session) {
      if (session.isValid()) {
            validSessions.put(session.getId(), session);
      }
    }
}
```

Desenvolvimento – Conexão cliente

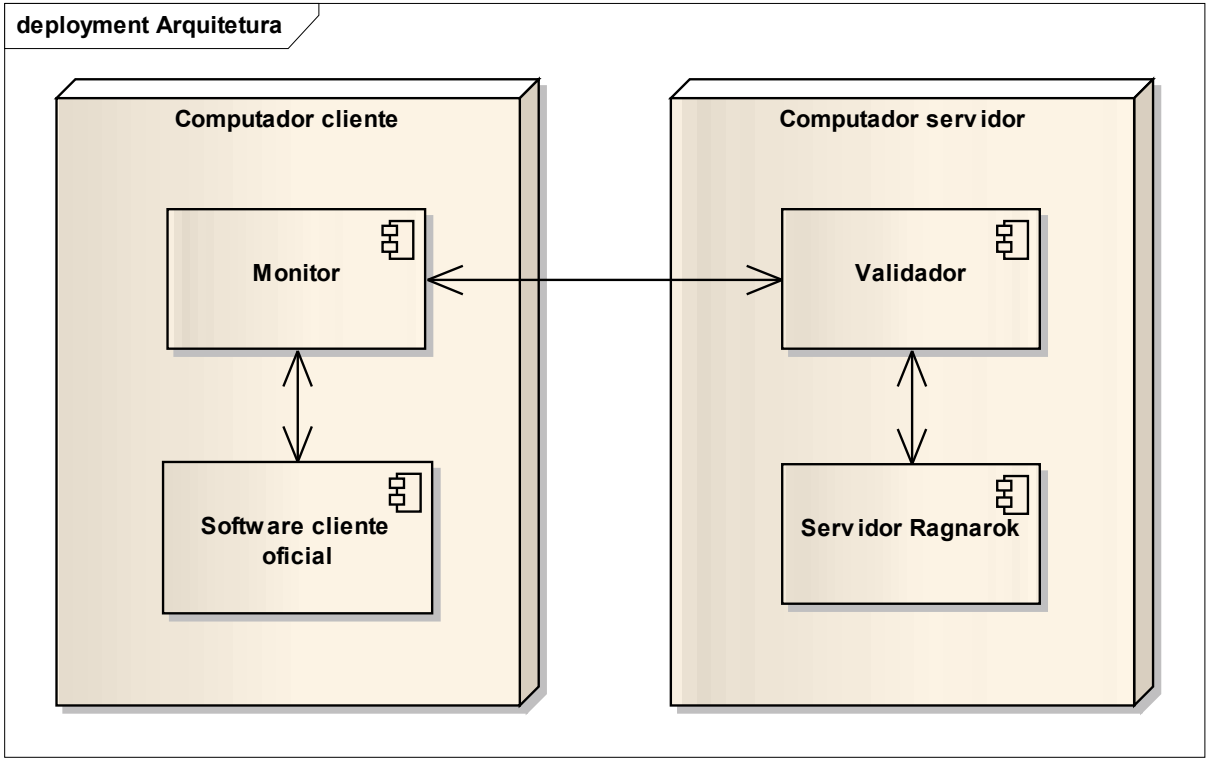
```
@Override
protected void newConnection(Socket socket) {
	//Lê o id do cliente, que deve ser o primeiro dado enviada por um
cliente.
	UID clientId = UID.read(new
DataInputStream(socket.getInputStream()));

	ClientSession session = control.getValidSession(clientId);
	//Se não for válido, retorna.
	if (session == null) {
		return;
	}

	//Se adiciona como listener da sessão, para que quando ela for
invalidada, desconecte o servidor do client inválido.
	session.addClientSessionListener(this);

	//Cria as threads de transmissão de dados.
	ProxyThread clientServerThread = new
ProxyThread(socket.getInputStream(), server.getOutputStream());
	clientServerThread.start();
	ProxyThread serverClientThread = new
ProxyThread(server.getInputStream(), socket.getOutputStream());
	serverClientThread.start();

	//Coloca no mapa de conexões válidas.
	validConnections.put(clientId, socket);
}
```

Desenvolvimento

- Operacionalidade do sistema
  - Demonstração do funcionamento do jogo online Ragnarök
    - Cliente oficial
    - Cliente não oficial (OpenKore)
  - Demonstração da instalação do sistema de proteção
    - Computador servidor
    - Computador cliente

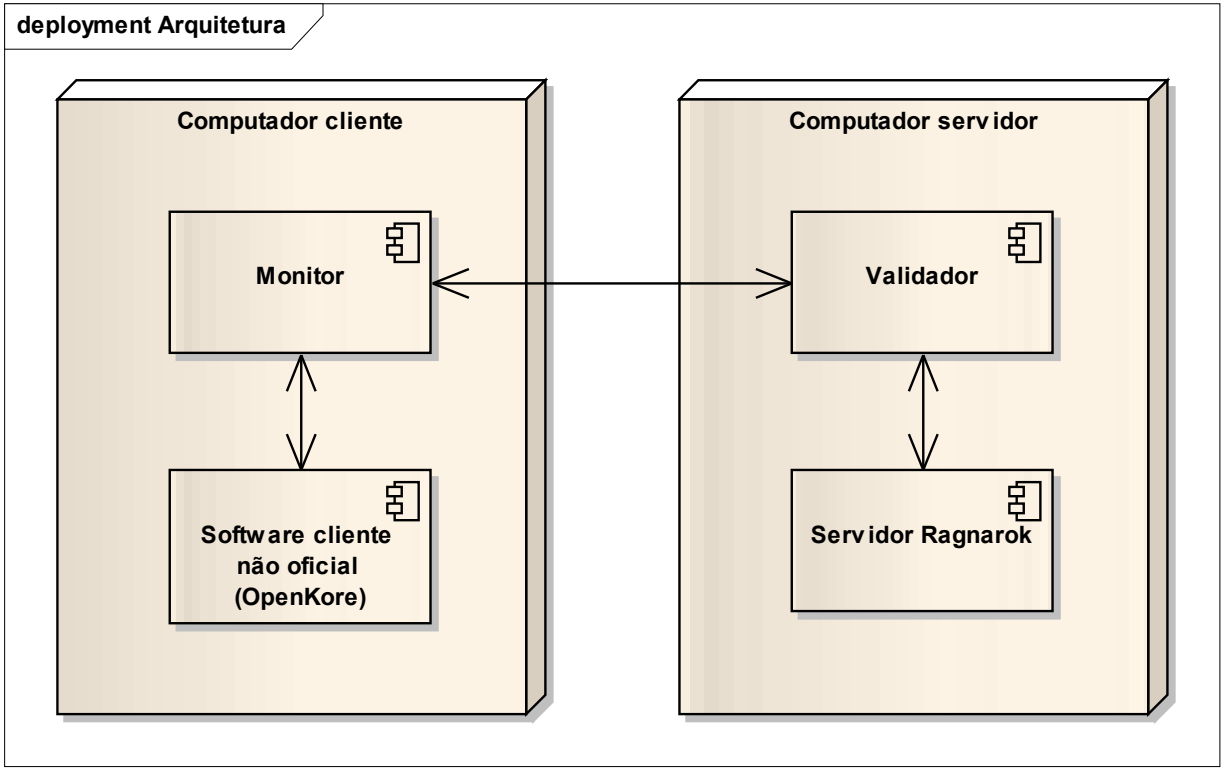
Desenvolvimento

- Operacionalidade do sistema
  - Caso 1: funcionamento através do cliente oficial



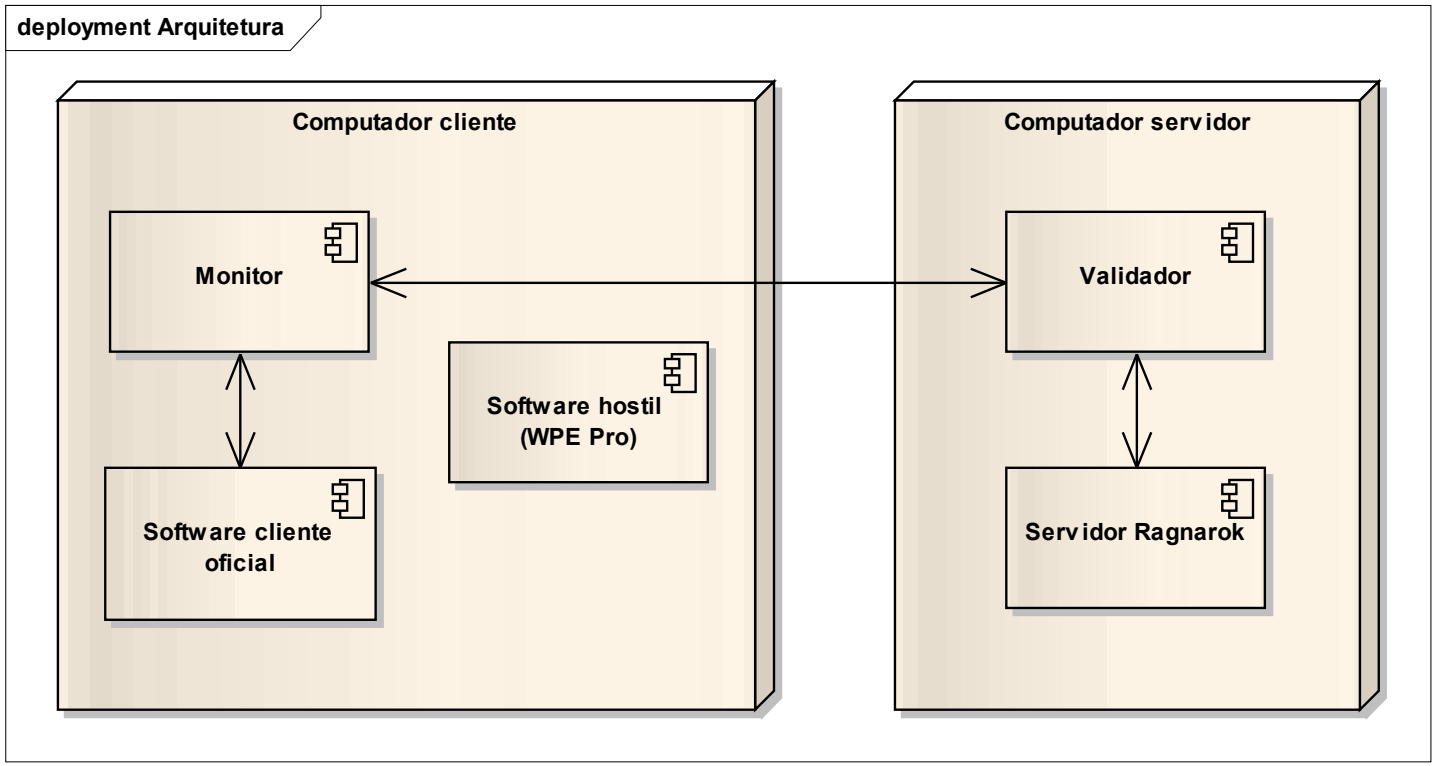
Desenvolvimento

- Operacionalidade do sistema
  - Caso 2: bloqueio de cliente não oficial



Desenvolvimento

- Operacionalidade do sistema
  - Caso 3: identificação de ambiente de execução hostil



Desenvolvimento

- Resultado e discussão
  - Protegeu o software servidor de clientes não oficiais
  - Possível problema de segurança no monitor
    - Como quase todo seu código é enviado pelo validador, um invasor pode tentar absorver todo esse código e ter seu próprio monitor.
    - A ofuscação de código reforça, mas não garante.
  - O algoritmo de AES de caixa branca poderia ser reforçado ainda mais com bijeções aleatórias.

Desenvolvimento

- Resultado e discussão
  - Estrutura de trabalhadores
    - Fácil extensão
    - Implementação de outras validações
      - Validações adicionais do ambiente de execução e do software cliente.
    - Implementação de novas ações para o monitor
  - Não foram feitas comparações aprofundadas com os trabalhos correlatos

Conclusão

- Os objetivos foram atingidos
- Técnicas tradicionais de segurança não atenderiam
  - Certificado digital
  - Assinatura de código
- Proteção de qualquer sistema cliente/servidor
  - Não apenas de jogos online
- Criada uma base para sistemas que visam garantir a confiabilidade em softwares remotos
  - Facilmente estendido
- Geradores de AES de caixa branca e de algoritmos de hashing aleatórios podem ser utilizados em outras aplicações

Extensões

- Integrar as bijeções aleatórias no gerador de AES de caixa branca;
- Aplicar o sistema de proteção em outros softwares, verificando se há impacta no tempo de resposta do software cliente com o software servidor;
- Implementar novos trabalhadores para a execução de novas validações no cliente;
- Colocar o sistema de proteção a prova, utilizando qualquer técnica de invasão conhecida;
  - Identificar possíveis falhas de segurança no sistema e propor soluções para elas;
- Fazer análises mais profundas em outras ferramentas de segurança
  - Comparar os pontos positivos e negativos com o sistema desenvolvido.

Demonstração do sistema