HMI: UM MIDDLEWARE PARA OBJETOS DISTRIBUÍDOS SOBRE O PROTOCOLO HTTP

Aluno:
Abel Luiz Cechinel

Orientador:
Paulo Fernando da Silva

Sumário

- Introdução;
- Fundamentação Teórica;
- Desenvolvimento;
- Conclusão;

Introdução

- Nos sistemas computacionais há demanda permanente por compartilhamento de recursos
- A Internet popularizou o compartilhamento através das aplicações distribuídas
- A computação móvel criou novas demandas

Introdução

- Os ambientes de aplicações distribuídas são heterogêneos e precisam de *middlewares*
- Os *middlewares* que suportam o paradigma orientado a objetos apresentam limitações práticas:
  - Enfrenta bloqueios de *firewalls;*
  - Não oferecem uma solução adequada para os dispositivos móveis;

Introdução

- Para superar bloqueios dos firewalls os desenvolvedores tem substituídos os *middlewares* orientados a objetos por alternativas procedurais, como *web services e SOA*
  - Passa-se a ter misturas de paradigmas

Introdução
Objetivos do trabalho

- desenvolver um *middlewar* para objetos distribuídos sobre o HTTP(transparente aos *firewalls*) que seja adequadamente suportado por dispositivos móveis

Fundamentação Teórica
Conceitos Básicos – Sistemas Distribuídos

- Tanenbaum e Steen (2008, p. 1), apresenta Sistemas Distribuídos como “... um conjunto de computadores independentes que se apresenta a seus usuários com um sistema único e coerente”.

Fundamentação Teórica
Conceitos Básicos – Objetos Distribuídos

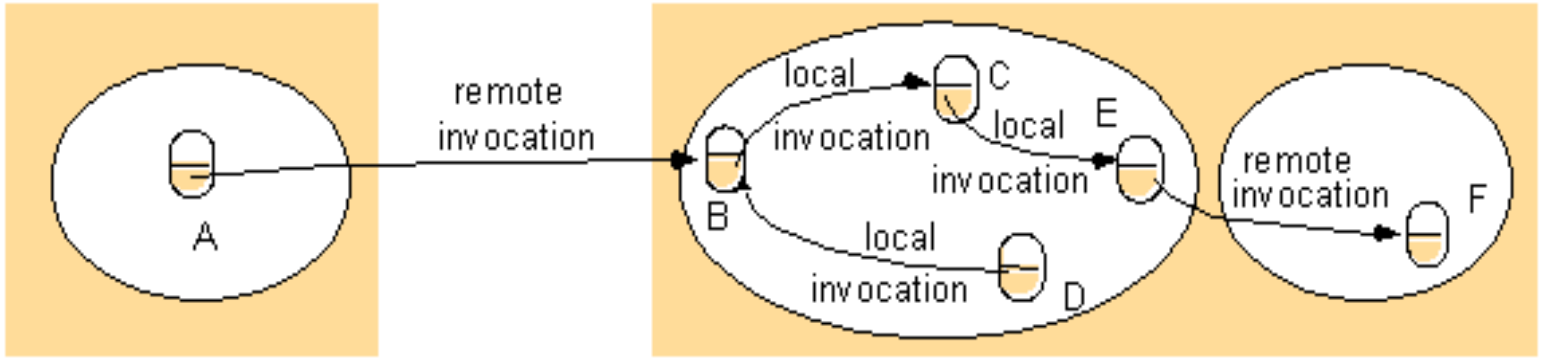
- Objetos distribuídos fornecem uma extensão ao modelo de objeto para torná-lo aplicável aos Sistemas Distribuídos (COULOURIS; DOLLIMORE; KINDBERG, 2007, p. 169).

Fundamentação Teórica
Conceitos Básicos – Interface de Objeto

- Lista de métodos públicos de um objeto. O único meio “legal” para um processo alterar o estado de um objeto é através da chamada de um método da sua interface

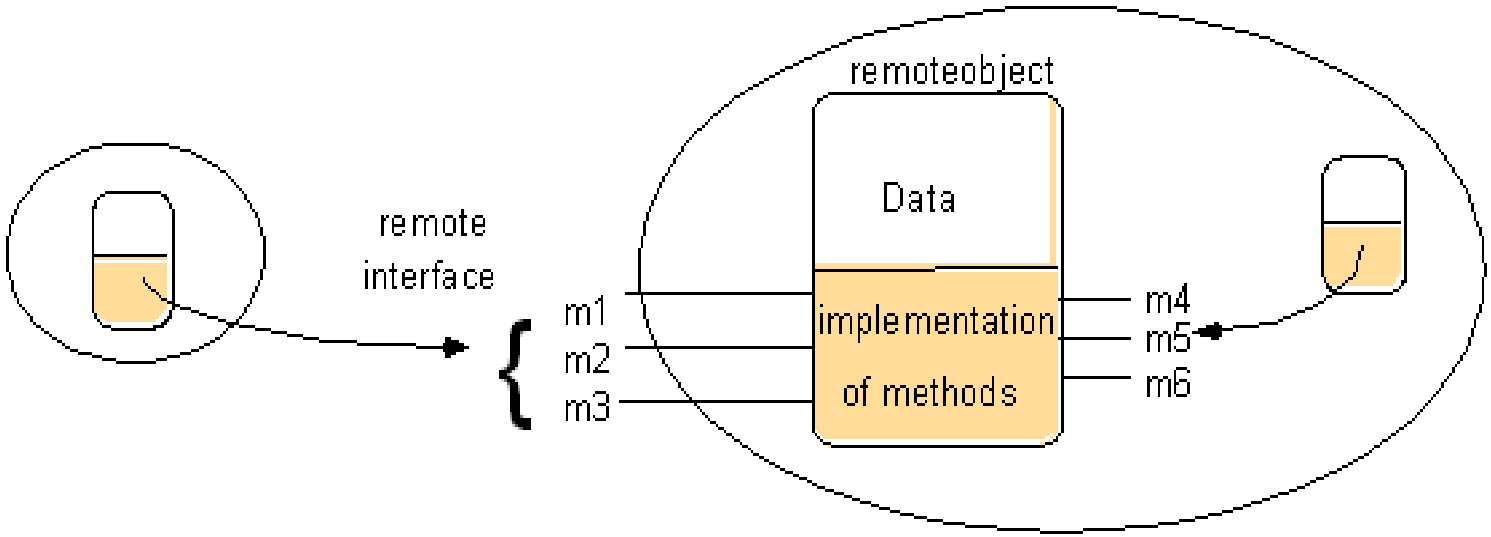
Fundamentação Teórica
Conceitos Básicos – Objeto Remoto

- Objetos que podem receber chamadas remotas a seus métodos



Fundamentação Teórica
Conceitos Básicos – Interface Remota

- Uma interface remota especifica os métodos de um objeto que estão disponíveis para invocação por parte dos objetos de outros processos,



Fundamentação Teórica
Conceitos Básicos – Referência Remota (Stub)

- Uma referência de objeto remoto, ou Stub, é uma instancia de uma classe que implementa a interface remota do objeto e que pode ser usada por todo um sistema distribuído para se referir a um objeto único em particular.
- é responsável por tornar a invocação a método remoto transparente, comportando-se como um objeto local; mas em vez de executar uma invocação local, ele a encaminha em uma mensagem para o um objeto remoto.

Fundamentação Teórica
Conceitos Básicos – Servidores de Objetos

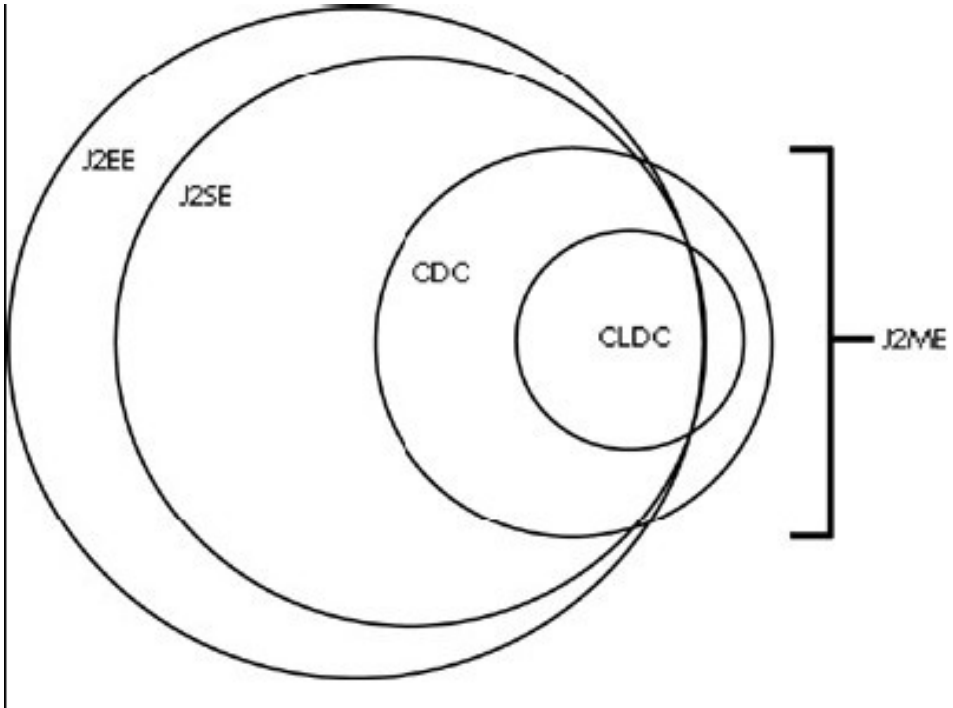
- Um servidor de objetos é um servidor configurado para suportar objetos distribuídos.
- Os serviços são implementados pelos objetos que residem no servidor.
- O servidor fornece apenas os meios de invocar objetos locais, com base em requisições de clientes remotos.
- Os serviços são configurados com adição ou remoção de objetos

Fundamentação Teórica
Conceitos Básicos – XDR

- Nem todos computadores armazenam as informações no mesmo formato
- Para que informações possam ser trocadas entre computadores que com diferentes formatos para os dados internos, é necessário que um padrão comum seja determinado
- Um padrão aceito para representação de estruturas de dados e valores é chamado de representação externa de dados, do inglês eXternal Data Representation (XDR)

Fundamentação Teórica
Conceitos Básicos – JAVA MICRO EDITION

- Edição da linguagem Java para dispositivos móveis PDA's, celulares, e eletrônicos em geral



Fundamentação Teórica

Contexto atual do tema - Middleware para Ambientes Pervasivos (MAP)

- MAP permite o desenvolvimento de aplicações distribuídas com arquitetura orientada à serviços em dispositivos móveis.
- MAP utiliza-se do protocolo SOAP para troca de mensagens, provendo acesso à procedimentos distribuído, não suportando a construção de sistemas baseados em objetos distribuídos.

Fundamentação Teórica

Contexto atual do tema - Chamada ReMota para j2mE (RME)

- RME fornece aos desenvolvedores JME com configuração CLDC, uma alternativa para a implementação da invocação remota de métodos
- opera sobre um range de portas para estabelecer as conexões, o que dificulta sua implantação devido aos bloqueios dos firewalls das empresas.

Desenvolvimento
Requisitos Principais

- permitir o registro de objetos distribuídos no servidor (Requisito Funcional – RF)
- permitir que o cliente localize os objetos distribuídos registrados no servidor (RF)
- retornar uma referencia remota quando a chamada para um método remoto retornar um objeto remoto (RF);
- permitir que o cliente libere do servidor os objetos que não serão mais utilizados (RF);

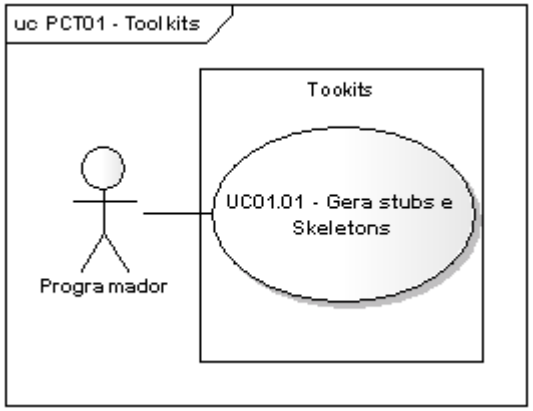
Desenvolvimento
Requisitos Principais

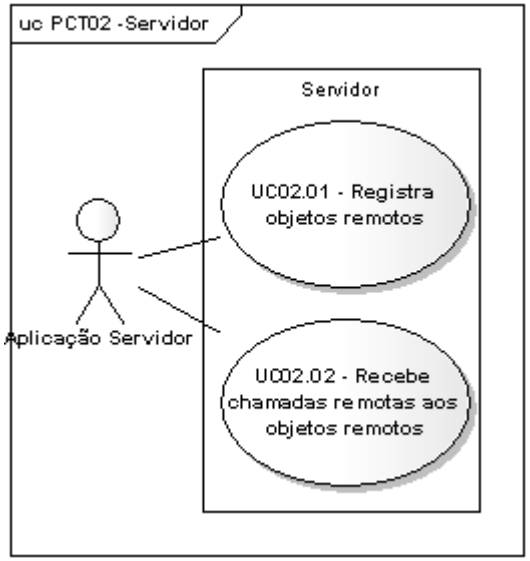
- liberar do servidor os objetos que não tenham mais referência em nenhum cliente (RF);
- ocupar menos de 40 Kbytes nos clients (Requisito Não Funcional – RNF);
- possuir uma especificação que permita a implementação do cliente em diferentes linguagens de programação (RNF);
- gerar stubs e skeletons para os objetos remotos (RF).

Desenvolvimento
Especificação – Técnicas e Ferramentas

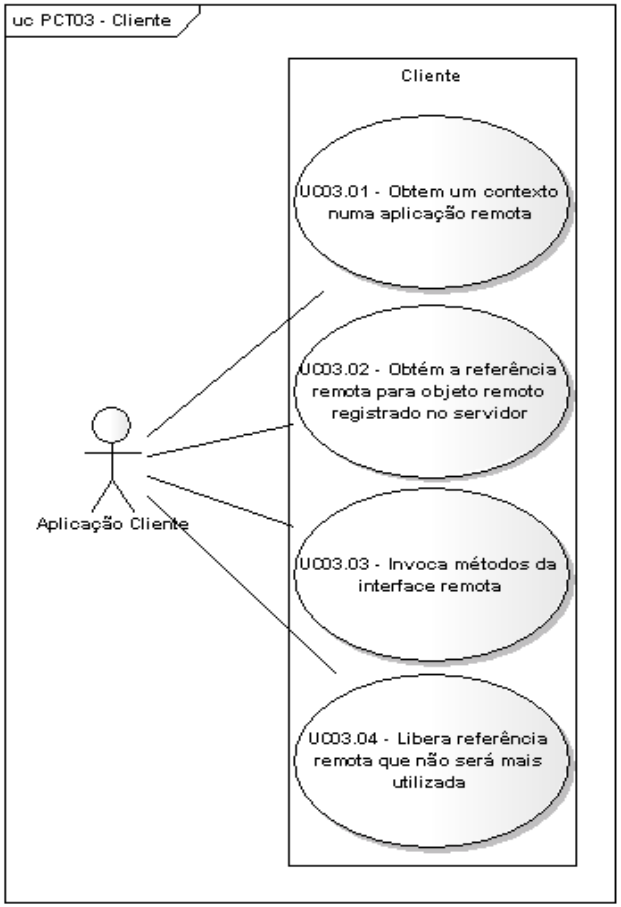
- Notação UML
- Apoio ferramenta Enterprise Architect (EA)
- São explanados diagramas de casos de uso, classes e seqüência.

Desenvolvimento
Especificação – Casos de Uso

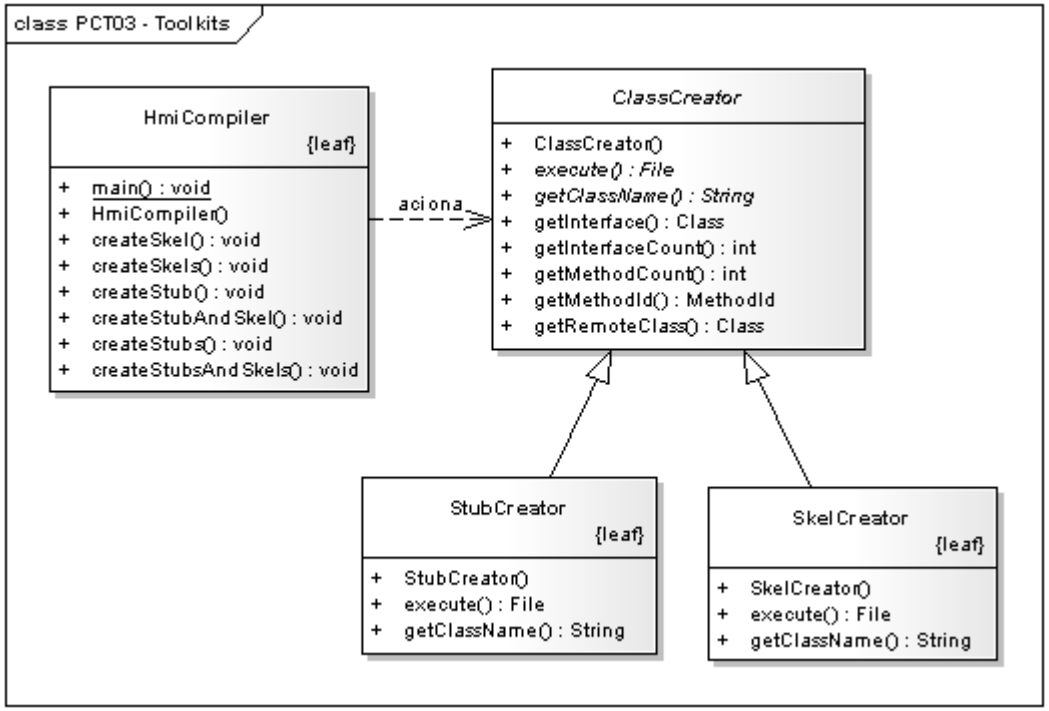




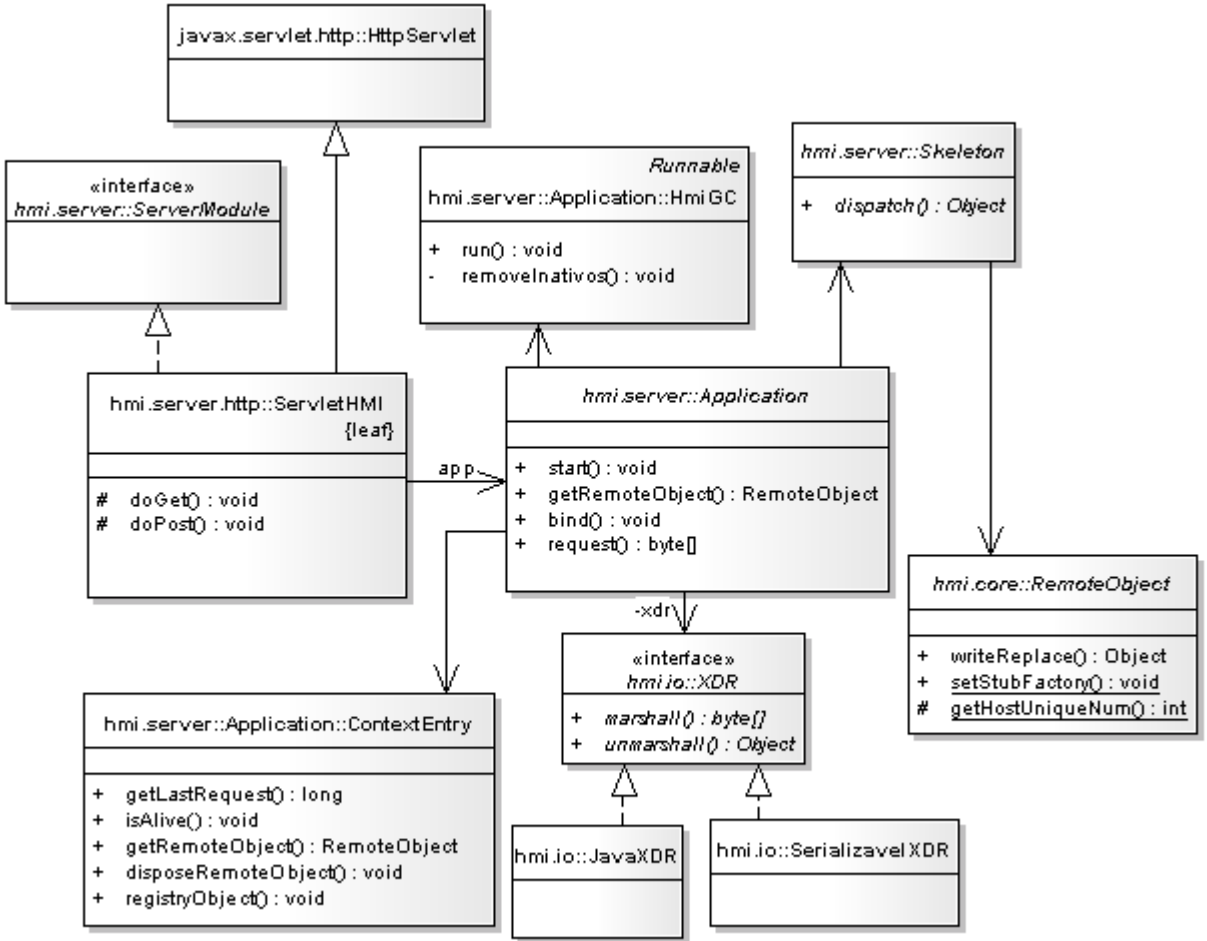
Desenvolvimento
Especificação – Casos de Uso



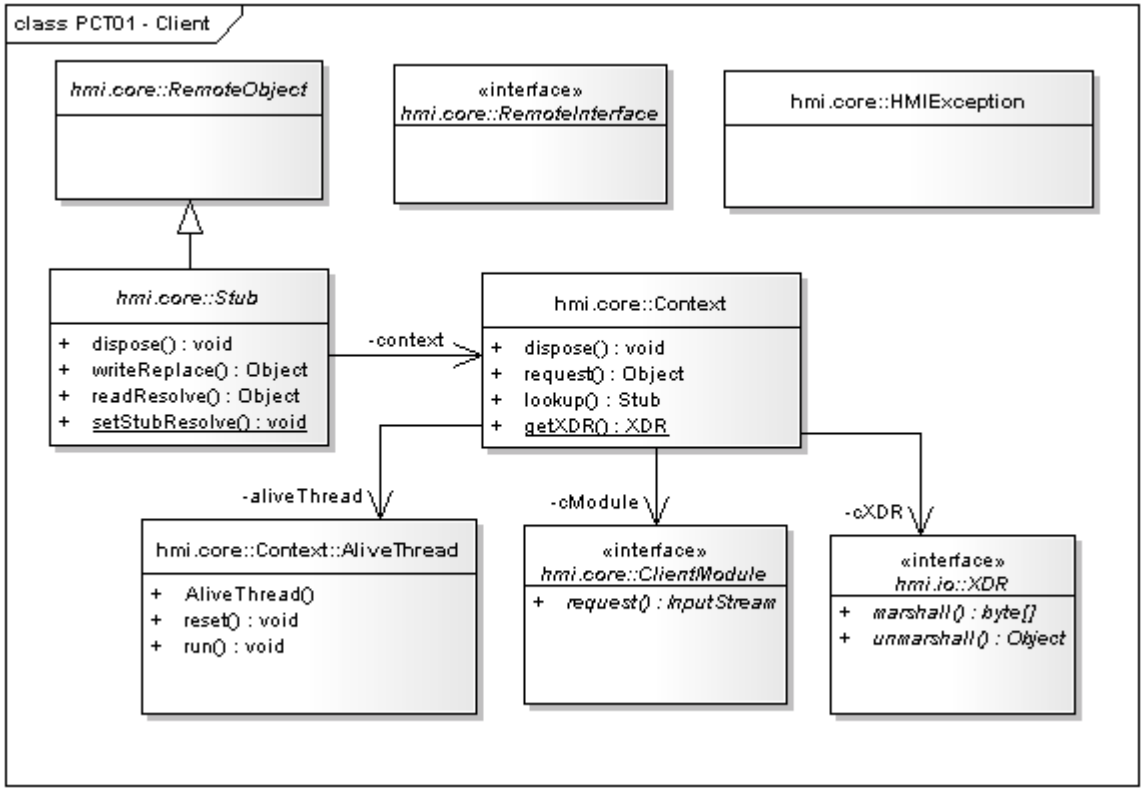
Desenvolvimento
Especificação – Diagramas de Classes



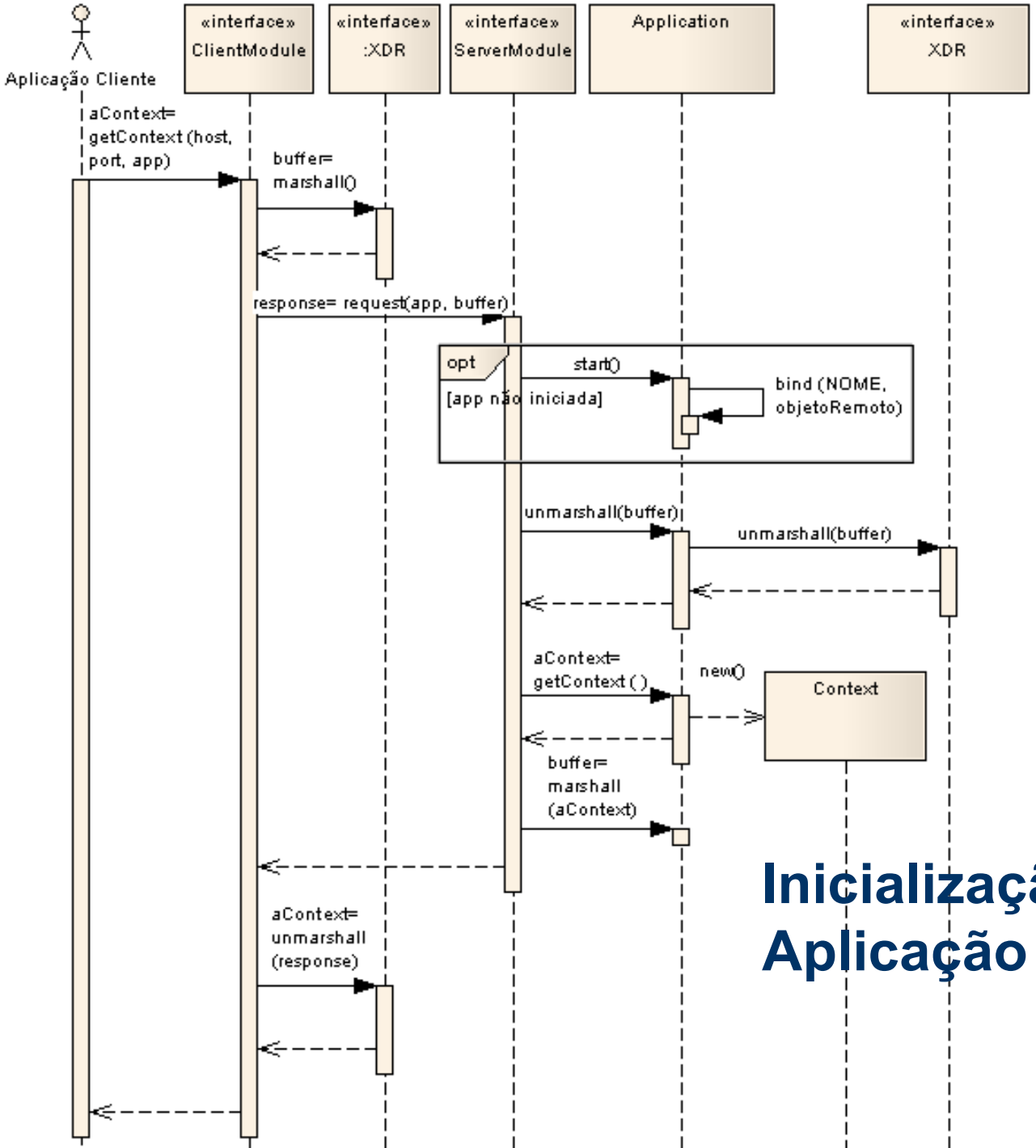
Desenvolvimento
Especificação – Diagramas de Classes



Desenvolvimento
Especificação – Diagramas de Classes

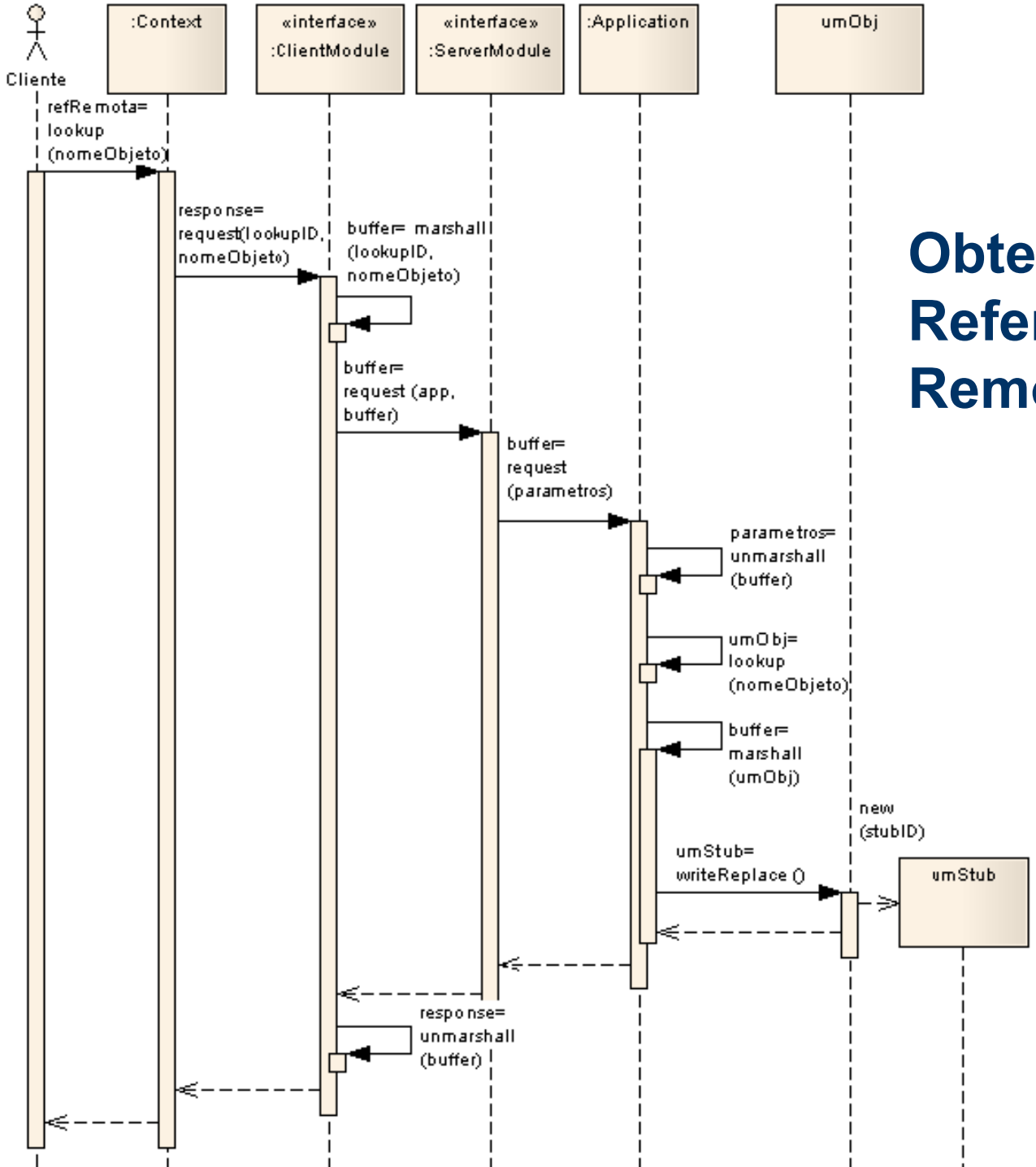


Desenvolvimento
Especificação – Diagramas de Sequência

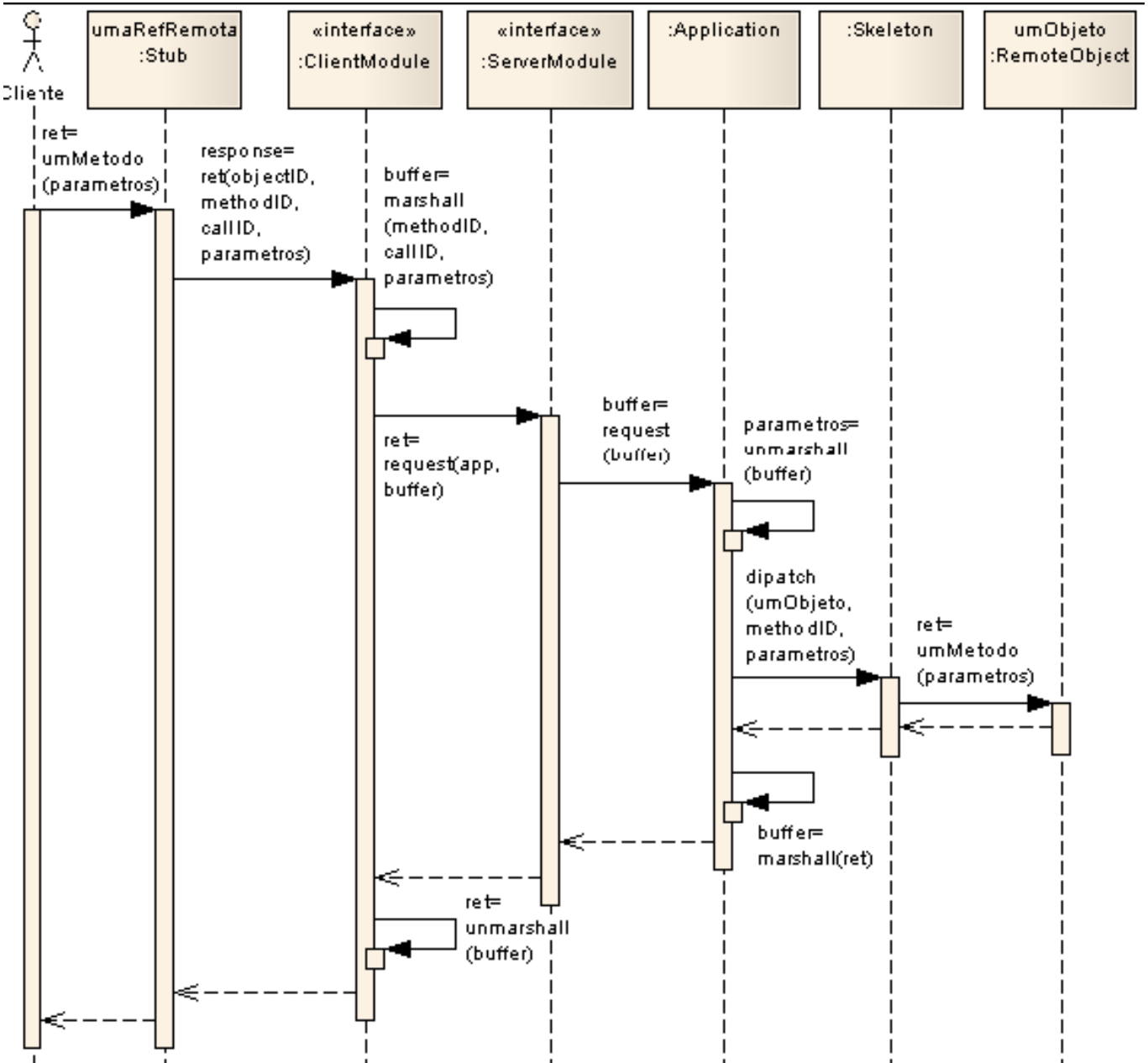


Inicialização da Aplicação

Obtenção de Referência Remota



Chamada de Método Remoto



Desenvolvimento
Implementação – Técnicas e Ferramentas

- Linguagem JAVA
- Ambiente de desenvolvimento integrado Eclipse 3.3

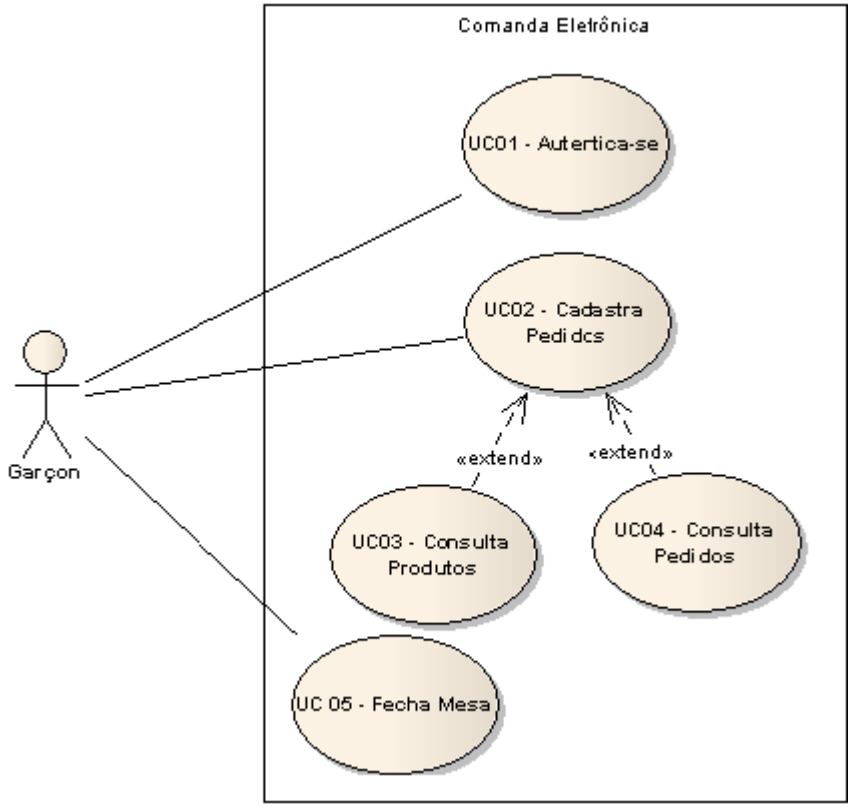
Desenvolvimento

Operacionalidade da Implementação - Comanda Eletrônica (CE)

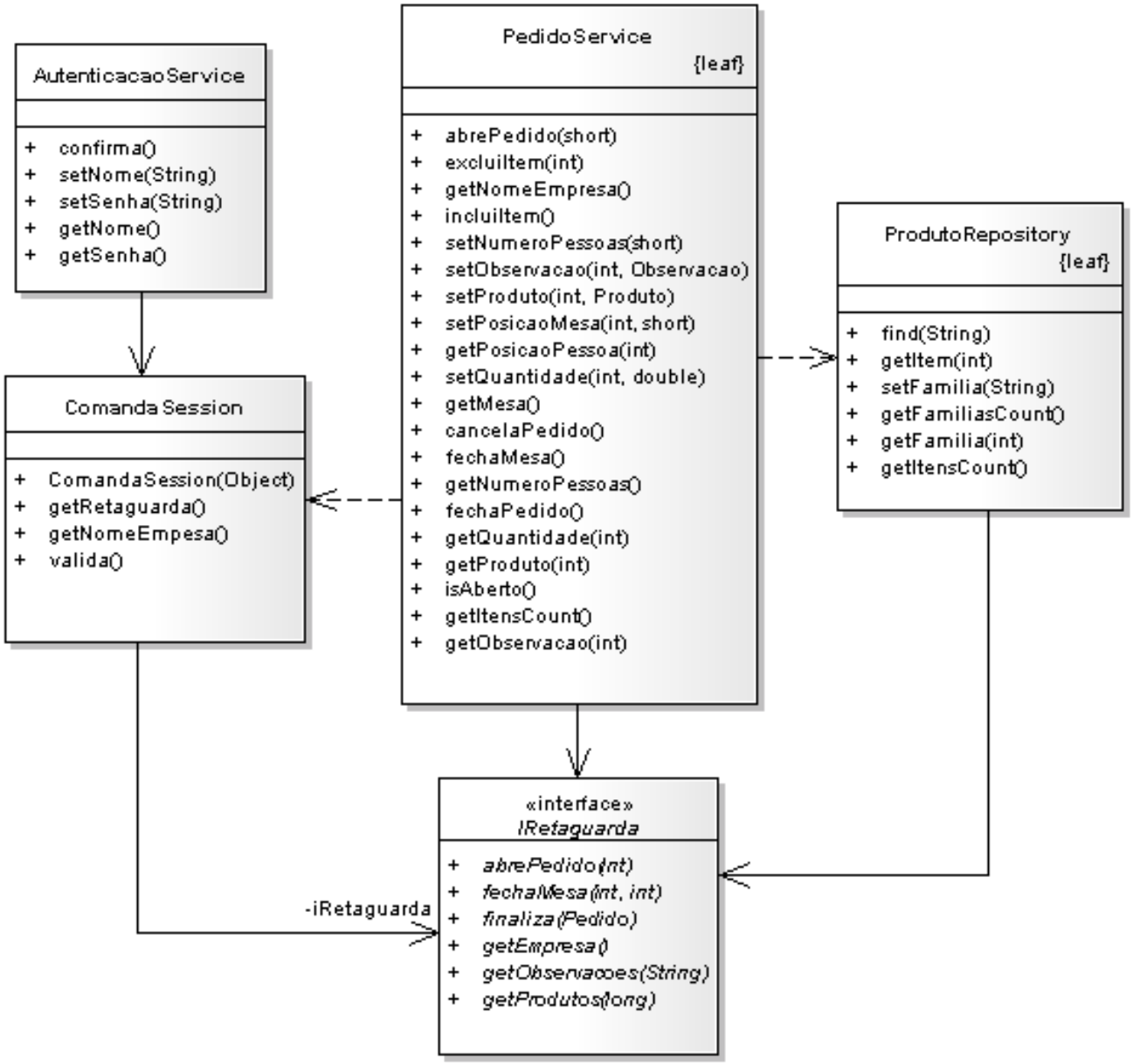
- A CE deve rodar em dispositivos móveis e seus usuários são os garçons do restaurante.
- Quando um garçom realiza um atendimento, informa o número da mesa e os itens da cardápio que os clientes desejam com suas respectivas quantidades.

Desenvolvimento

Operacionalidade da Implementação - Comanda Eletrônica (CE)

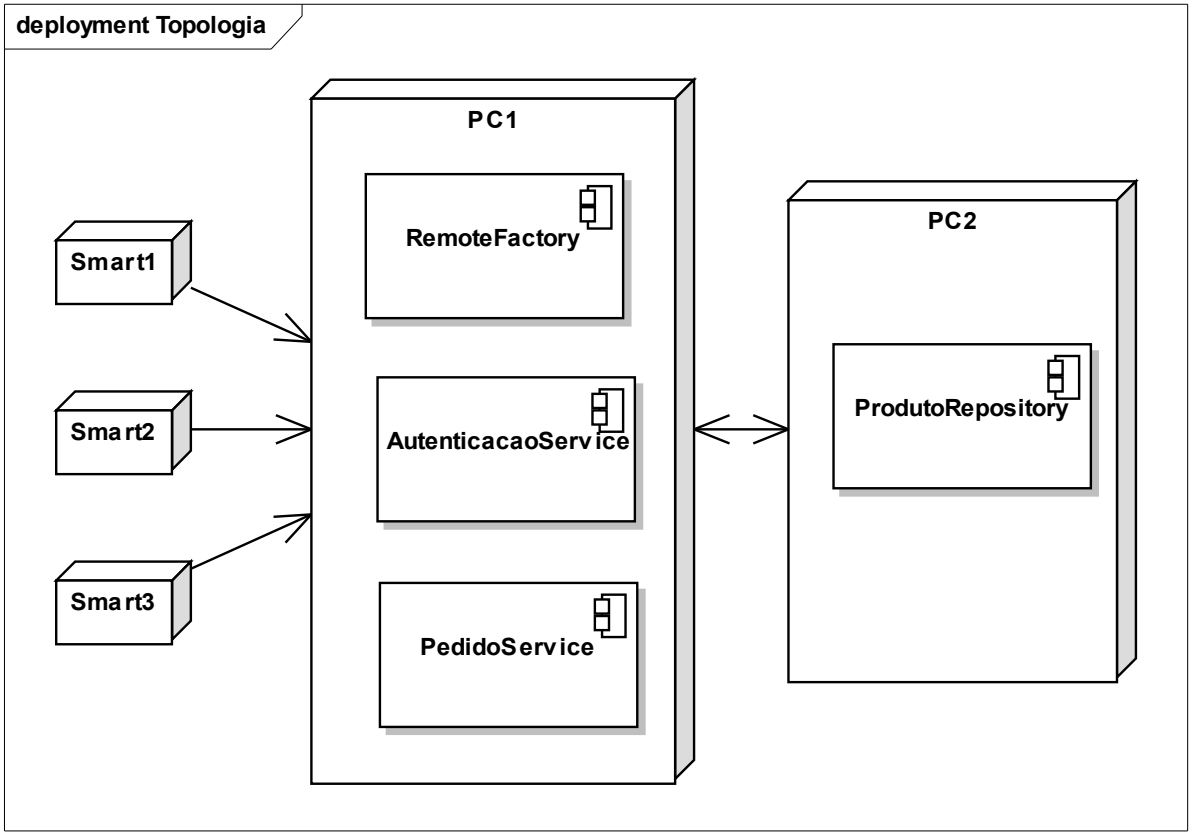


Serviços da Comanda Eletrônica (CE)



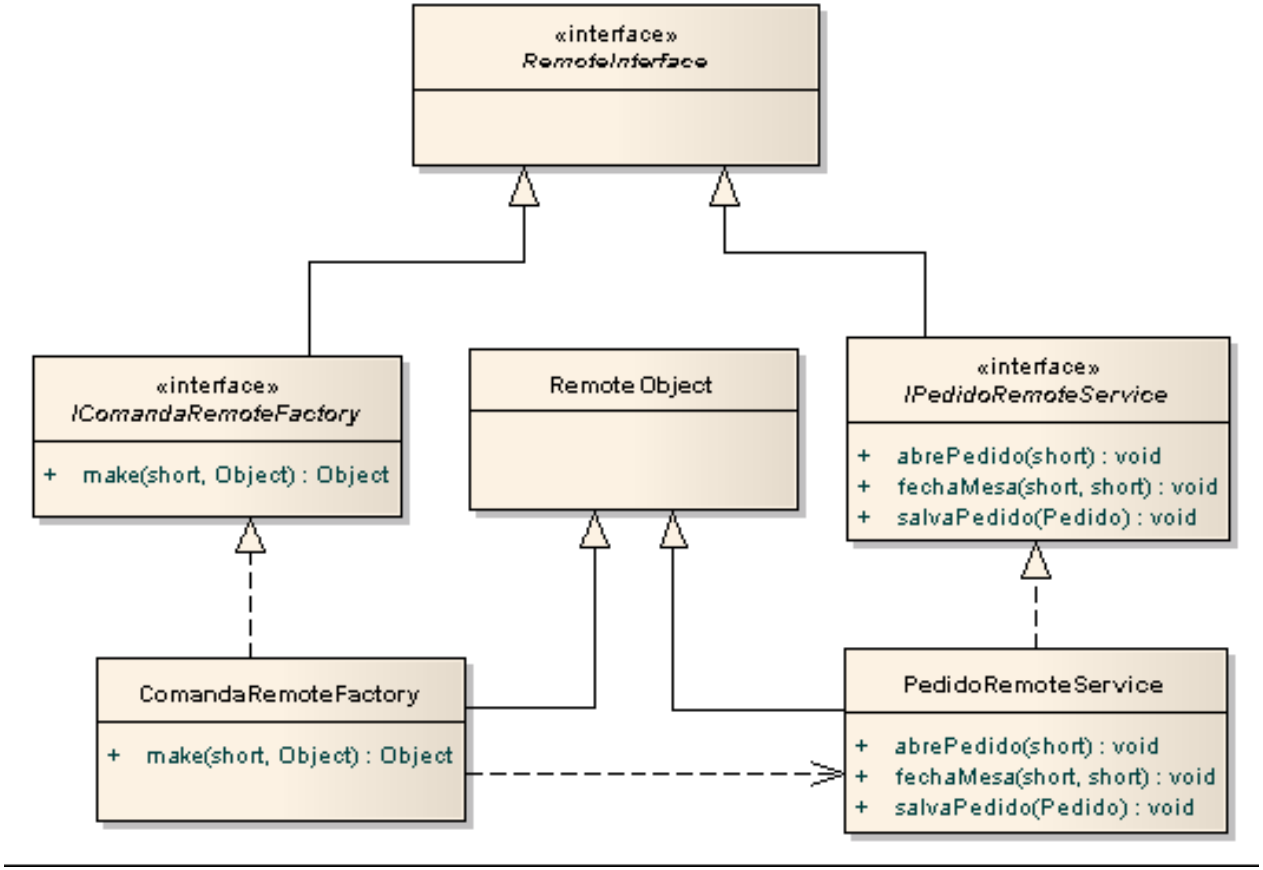
Desenvolvimento

Operacionalidade da Implementação - Comanda Eletrônica (CE)



Desenvolvimento

Operacionalidade da Implementação - Comanda Eletrônica (CE)



Desenvolvimento

Operacionalidade da Implementação - Comanda Eletrônica (CE)

```
public class ComandaRemoteFactory extends RemoteObject implements
IComandaRemoteFactory {
   public Object make(short service,Object parametro) throws HMIException,
CTxException {
       switch (service) {
              case ConstantesComanda.LOGIN:
                 return new ComandaSession(parametro);
              case ConstantesComanda.PEDIDO:
                 return new PedidoRemoteService((ComandaSession)parametro);
              case ConstantesComanda.REPOSITORIO_OBSERVACAO:
                 return new ObservacaoRemoteRepository((ComandaSession)parametro);
              case ConstantesComanda.REPOSITORIO_PRODUTO:
                 return new ProdutoRemoteRepository((ComandaSession)parametro);
              default:
                    throw new CTxException ("Classe desconhecida");
       }
   }

}
```

Desenvolvimento

Operacionalidade da Implementação - Comanda Eletrônica (CE)

```
Java -Dout=dirSaida br.com.ctx.hmi.compiler.HmiCompiler classe1 [classe2, classe3..]
```

Exemplo:

```
Java -Dout=C:/temp br.com.ctx.hmi.compiler.HmiCompiler  ComandaRemoteService
PedidoRemoteService
```

Desenvolvimento

Operacionalidade da Implementação - Comanda Eletrônica (CE)

```
public final class ComandaRemoteFactory_HmiStub extends Stub implements
IComandaRemoteFactory {

   // Stubs podem ser serializados via "writeObject"
   private static final long serialVersionUID = 1L;

   // Construtor para serialização
   public ComandaRemoteFactory_HmiStub() {}

   // Construtor parametrizado
   public ComandaRemoteFactory_HmiStub(br.com.ctx.hmi.RemoteRef remoteRef) {
       super(remoteRef);
   }
   // ID -1301179154 "make"
   public Object make(short arg0, Object arg1) throws HMIException, CTxException {
       try{
              return this.call( -1301179154, new Object[]{new Short(arg0), arg1});
       }
       catch (Throwable t){
              if (t instanceof HMIException){
                     throw (HMIException)t;
              }
              if (t instanceof CTxException){
                     throw (CTxException)t;
              }
              throw new HMIException(t);
       }
   }
}
```

Desenvolvimento

Operacionalidade da Implementação - Comanda Eletrônica (CE)

```
public final class ComandaRemoteFactory_HmiSkel extends Skeleton {

	// ID -1301179154 "make"
	private Object make(ComandaRemoteFactory ro,short arg0, Object arg1) throws
		HMIException, CTxException {
		return ro.make(arg0, arg1);
	}

	// implementação do método "dispatch"
	public Object dispatch(RemoteObject ro, int methodId, Object arg) throws
		HMIException {
		Object result = null;
		Object[] args = null;
		try{
			switch (methodId) {
				case -1301179154 : // ID -1301179154 "make"
					args = (java.lang.Object[]) arg;
					result = this.make((ComandaRemoteFactory)ro,
((Short)                                                    args[0]).shortValue(),
args[1]);
					break;
			}
		}
		catch(Throwable t){
			t.printStackTrace();
			if (t instanceof br.com.ctx.hmi.HMIException  ) {
				throw (br.com.ctx.hmi.HMIException )t;
			}
			throw new br.com.ctx.hmi.HMIException (t);
		}
		return result;
	}
}
```

Desenvolvimento

Operacionalidade da Implementação - Comanda Eletrônica (CE)

```
public class Pedido extends Domain implements Serializavel {

                        . . .

      private static final long serialVersionUID = 1L;
      private int mesa;
      private List itens;

                        . . .

      public void restaura(DataInputStream in) throws IOException {
            mesa = in.readInt();
            itens = (List) Serializador.getInstance().restaura(in);

      }

      public void serializa(DataOutputStream out) throws IOException {
            out.writeInt(mesa);
            out.write(Serializador.getInstance().serializa(itens));

      }
```

Desenvolvimento
RESULTADOS E DISCUSSÃO

- Este trabalho partiu da dificuldade prática de se construir sistemas distribuídos na web, dentro do paradigma da POO, particularmente para dispositivos móveis.
- Middlwares orientados a objetos como Java RMI, utilizam um range de portas de comunicação, tornando-se incompatíveis com a configuração da maioria dos firewalls

Desenvolvimento
RESULTADOS E DISCUSSÃO

- Alternativas como os Web Services, são adaptados à web, mas não são alinhados com a POO.
- O middleware objeto deste trabalho une as facilidades do Java RMI e dos Web Services e é compatível com JME,

Desenvolvimento
RESULTADOS E DISCUSSÃO

	Orientado a Objetos	Transparente Firewalls	Compatível JME	Linguagens	Callback
Java RMI	SIM	NÃO	NÃO	Java	SIM
Web Service	NÃO	SIM	Com adaptações	Diversas	NÃO
CORBA	SIM	NÃO	NÃO	Diversas	SIM
HMI	SIM	SIM	SIM	Java	NÃO

- HMI foi implementado apenas em Java. Mas sua especificação é compatível com qualquer linguagem orientada a objeto.
- Não foi implementado o recurso de callback.
- HMI é utilizado com resultados satisfatórios em sistemas comerciais.

Conclusão

- O middleware resultado deste trabalho, tem apresentado bons resultados, frente aos objetivos a que se propôs. Como demonstrado na operacionalidade da implementação, há uma abstração das questões de rede envolvidas na distribuição dos objetos e como todas as requisições ocorrem em HTTP, não há riscos de bloqueios por firewalls. Atingiu-se também o objetivo de se ter um middleware compatível com dispositivos móveis.