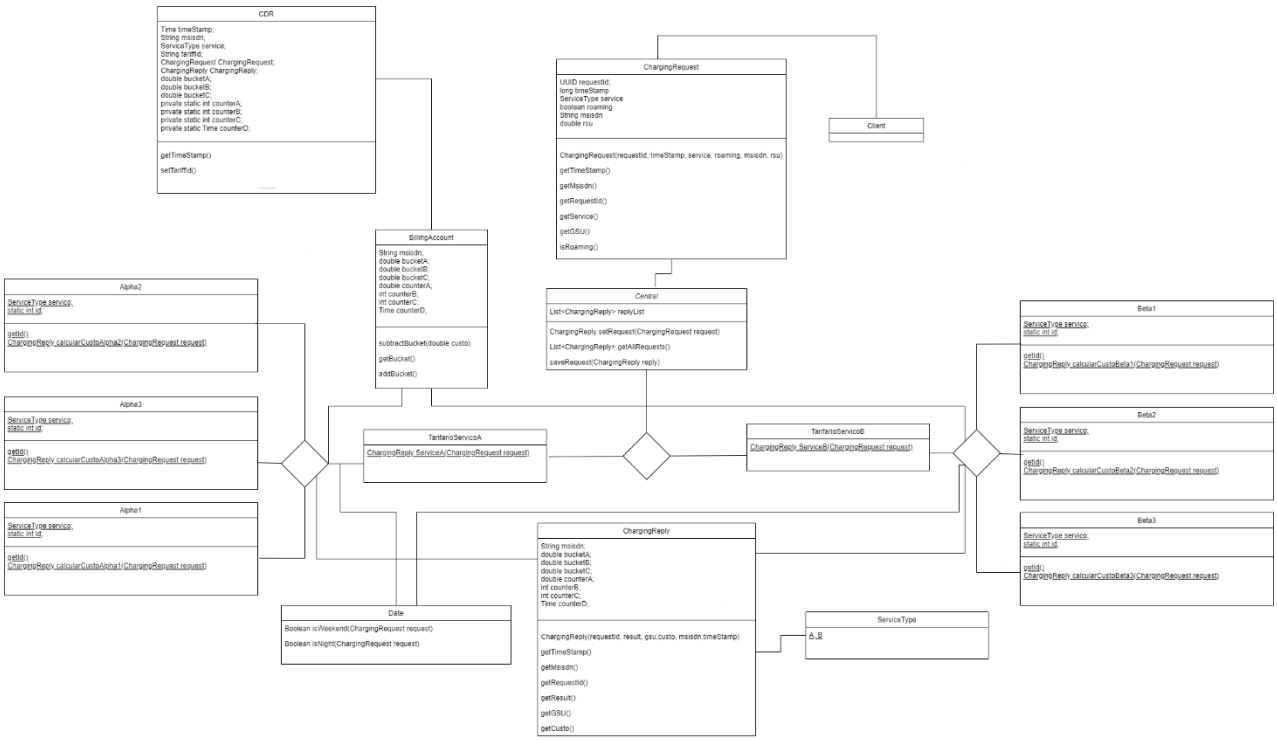
Sistema de Facturamento e Cobrança em Telecomunicações com Tarifários Dinâmicos

Contexto

Após o desafio proposto, tomei em consideração alguns pontos que suscitaram dúvidas. Uma das questões que surgiram foi se deveríamos considerar as subscrições nos tarifários (Beta1, Alpha2, etc.). Após assumir que não deveríamos fazê-lo, desenvolvi uma arquitetura que, com base no serviço (A ou B), toma a decisão sobre qual tarifário aplicar. Essa decisão é feita levando em conta as restrições estabelecidas para cada tarifário, conforme especificado no enunciado do exercício.

Arquitetura



Central: A classe "Central" desempenha um papel crucial no sistema. Esta classe é responsável por receber os pedidos (requests) enviados e determinar qual serviço está "subscrito". Se o serviço for do tipo A, o pedido é encaminhado para a classe "TarifárioA"; se for do tipo B, é direcionado para a classe "TarifárioB". Além disso, a classe "Central" mantém uma lista de todas as respostas (ChargingReply) e também

desempenha um papel importante na inserção de alguns dados do CDR (Client Data Record).

Tarifário A/B: As classes "TarifárioA" e "TarifárioB" desempenham um papel crucial na determinação do tarifário a ser aplicado com base no serviço especificado (A ou B) no ChargingRequest. A decisão varia de acordo com o serviço, e existem diferentes tarifários possíveis para cada um.

No caso do serviço A, a classe "TarifárioA" é responsável por escolher entre os tarifários Alpha1, Alpha2 e Alpha3, dependendo das restrições e condições aplicadas a cada um deles. Se nenhuma das opções de tarifário for aplicável a classe retornará um ChargingReply com o resultado "Não Elegível".

Da mesma forma, no caso do serviço B, a classe "TarifárioB" toma uma decisão entre Beta1, Beta2 e Beta3, com base nas condições e restrições específicas de cada tarifário. Se nenhuma das opções de tarifário for aplicável, a resposta será "Não Elegível".

Alpha1, Alpha2, Alpha3: Cada uma destas classes é responsável por aplicar as regras e fórmulas específicas do tarifário que representam. São avaliadas as informações fornecidas no ChargingRequest, como o tempo da chamada, localização, saldos nos buckets e outros fatores relevantes, para determinar o custo da chamada e fornecem uma resposta do tipo ChargingReply.

Beta1, Beta2, Beta3: As classes "Beta1," "Beta2," e "Beta3" desempenham um papel semelhante às classes "Alpha1," "Alpha2," e "Alpha3," mas para o serviço B.

BillingAccount: Esta classe desempenha um papel crucial no armazenamento e gestão dos valores dos Buckets e Counts, que são essenciais para determinar o custo de uma chamada ou qual tarifário será “ativado”. Além disso, o BillingAccount é responsável por manter a classe CDR (Cliente Data Record) atualizada em relação ao Buckets e Counts.

CDR: A CDR (Client Data Record) é um componente fundamental do sistema, o seu papel é fornecer informações atualizadas e acessíveis pelos utilizadores em relação ao request em andamento.

ChargingRequest: O ChargingRequest é o modelo de dados que representa uma solicitação de serviço. Cada ChargingRequest inclui um ID exclusivo, gerado automaticamente para identificação, a data e hora em que a solicitação foi criada, o tipo de serviço a ser utilizado (A ou B), informações sobre o uso de roaming, o número MSISDN e a quantidade de pedidos, denominada RSU (Requested Service Units). No contexto desta aplicação, o RSU é interpretado como a duração da chamada telefônica em minutos.

ChargingReply: O ChargingReply é o modelo de dados que representa a resposta a uma solicitação de serviço, ou seja, a resposta a um ChargingRequest. Este modelo é fundamental para fornecer feedback após a execução de uma solicitação e registar informações relevantes. Cada ChargingReply inclui o ID exclusivo da solicitação à qual está a responder, a data e hora em que foi feita a solicitação, o resultado da solicitação,

o custo associado à solicitação e o valor de GSU (Granted Service Units). No contexto desta aplicação, o GSU é interpretado como a quantidade de minutos garantidos para a chamada, mesmo que o cliente tenha solicitado mais minutos do que seu saldo monetário permitiria.

Date: A classe "Date" foi projetada para centralizar e simplificar a execução de ações que são realizadas em várias partes da aplicação. Esta classe têm duas tarefas, verificar se a data atual corresponde a um fim de semana e informar se o momento atual está dentro do período noturno.

Relatório de testes

Foram realizados alguns testes de unidade para garantir a qualidade e funcionalidade da aplicação. É importante ressaltar que a abordagem de testes foi ampla, abrangendo uma variedade de cenários e componentes da aplicação, sem contração especifica em um tipo específico de teste, como testes Black-box, White-box e testes de integração. Os testes realizados tinham como objetivo identificar possíveis problemas, verificar a funcionalidade geral da aplicação e desvendar qualquer comportamento inesperado.

Os testes abrangeram diversos cenários, incluindo solicitações de serviço A e B com e sem roaming, cálculos de custos, seleção de tarifários, atualizações de registos (CDR) e limites de crédito. Estes testes servem como um ponto de partida para avaliar o funcionamento geral da aplicação, mas devem ser complementados por testes mais detalhados e especializados, dependendo dos requisitos e necessidades da aplicação.

Além dos testes executados, foi também desenvolvida a Matriz de Rastreabilidade, uma ferramenta essencial no processo de garantia de qualidade. Esta matriz estabelece uma conexão direta entre os casos de teste e os requisitos ou funcionalidades da aplicação. A Matriz de Rastreabilidade de Testes proporciona uma visão clara e das relações entre os casos de teste e os requisitos do sistema. Cada entrada na matriz indica um requisito ou funcionalidade que foi testada. Desta forma, podemos identificar rapidamente todas as funcionalidades que foram devidamente testadas.

Testes

Teste #	Função	Descrição	Teste #	Dados de Teste	Resultado Expectável	
1	requestServiceARoaming	pedido 5 minutos com roaming, este teste irá percorrer a aplicação na sua totalidadeaté retornar uma resposta	Pedido para o service A	service = A roaming =true	reply.getResult()=OK	
	requestServiceBRoaming		Pedido para o service B	service = B roaming =true	reply.getResult()=OK	
2	requestServiceBRoamingFalse	pedido 5 minutos sem roaming,	Pedido para o service A	service = A roaming =false	reply.getResult()=OK	
	requestServiceARoamingFalse		Pedido para o service B	service = B roaming =false	reply.getResult()=OK	
3	requestServiceARoamingFalseBucketA	Pedido 5 minutos para o serviço A, espera se que seja retirado o valor de 5euros do bucket A	Testar o calculo do custo totla de uma chamada do serviço A	BucketA=10 Service=A Roaming=false	bucketA=5	
4	requestServiceBRoamingFalseBucketA	Pedido 5 minutos para o serviço B, espera se que seja retirado o valor de 0,50euros do bucket A	Testar o calculo do custo totla de uma chamada do serviço B	BucketA=20 Service=B Roaming=false	bucketA=9,95	
5	requestServiceARoamingTrueBucketC	Pedido 5 minutos para o serviço A, espera se que seja retirado o valor de 10euros do bucket A	Testar o calculo do custo totla de uma chamada do serviço A com raming	BucketC=15 Service=A Roaming=true	bucketC=5	
6	CDRHasUpdated	será feito um requst com um determindo msisdn, apos o rquest será comparado se o msisdn que está do CDR é igual ao do request.	Testar se a CDR está atualizada	ChargingRequest request=(...msisdn="99999"....)	CDR.getMsisdn()=99999	
7	requestServiceARoamingTrueBucketACreditLimitReached	Fazer um pedido de 15 minutos com roaming oq levará a 30euros de total, no entanto vai ser testado apenas com 10euros no bucekt A	teste CreditLimitReached	BucketA=10 Service=A Roaming=true	reply.getResult()=CreditLimitReached	
8	consultarRequests	Fazer dois requsts e depois vizualiar a lista de requsts feita tem o tamnha de 2	testar se os requsts estão a ser guardados	reply1 = Central.setRequest(request1); reply2= Central.setRequest(request2);	replyList.size()=2	
9	RequestServiceAIsWeekendRoamingFalse	requst para o serviço A com um Timestamp de domingo neste caso tem de ativar o Alpha2	testar o serviço A ao fim de semana se ativar o tarifario certo	timestamp= day.Sunday request=(...Timstamp=timestamp....) roaming=false	reply.getTariff()=Alpha2	
10	RequestServiceAIsWeekendRoamingTrue	requst para o serviço A com um Timestamp de domingo neste caso tem de ativar o Alpha3	testar o serviço A ao fim de semana	timestamp= day.Sunday request=(...Timstamp=timestamp....) roaming=true	reply.getTariff()=Alpha3	
11	RequestServiceBIsWeekendRoamingFalseIsDay	Testar o serviço B para domingo ao meio dia suposto ativar o Beta2	testar o serviço B ao fim de semana ao meio dia	timestamp= day.Sunday timestamp= hour.12 request=(...Timstamp=timestamp....) roaming=false	reply.getTariff()=Beta2	
12	RequestServiceBIsWeekendRoamingFalseIsNight	Testar o serviço B para domingo á noite suposto ativar o Beta1	testar o serviço B ao fim de semana á noite	timestamp= day.Sunday timestamp= hour.22 request=(...Timstamp=timestamp....) roaming=false	reply.getTariff()=Beta1	
13	RequestServiceBIsWeekendRoamingTrue	Testar o serviço B para domingo á noite suposto ativar o Beta3 , neste caso com roaming	testar o serviço B ao fim de semana com roaming	timestamp= day.Sunday timestamp= hour.12 request=(...Timstamp=timestamp....) roaming=true	reply.getTariff()=Beta3	