Universidade Presbiteriana Mackenzie  
Faculdade de Computação e Informática  
Ciência da Computação  
Software Baseado em Componentes

Bruno Ikoma  
Diego Guachala  
Ricardo Bernardelli

**Sobre o sistema**

Um sistema de reserva de carros necessita permitir que sejam feitas reservas para qualquer tipo de carro e qualquer estado em que exista a franquia.

Atualmente cada loja conta com seu próprio sistema de reservas, porém existem confusões e inconsistências devido a clientes deixarem os carros em diferentes lojas (diferente da loja de retirada).

O principal ponto de desenvolver um novo sistema será a integração entre os tipos de carros idisponíveis e os pontos de retirada e devolução diversos. Caso o carro desejado não esteja disponível, deverá ofertar carros semelhantes.

Cada locadora conta com um administrador de reservas que é responsável por controllar as reservas, mas qualquer usuário autorizado faz a reserva.

Observação: Todas as figuras utilizadas neste documento, segue em anexo também junto ao zip.

**Use Case Model**

Requisitos: Cadastro de carros, cadastro de clientes, aluguel de carros

Caso de Uso:

Fazer reserva de veículo

Breve descrição

Este caso de uso permite a um cliente fazer reservas online de veículo dentre os disponíveis na locadora.

Ator(es)

Cliente

Condições prévias

Nenhuma

Fluxo básico

1) Cliente: Seleciona a opção "Fazer reserva";

2) Sistema: Pede os dados do cliente;

3) Cliente: Entra com os dados;

4) Cliente: Seleciona “Período da reserva”;

5) Sistema: Verifica que os dados estão corretos;

6) Sistema: Consulta e apresenta as cidades onde a locadora possui agências;

7) Cliente: Seleciona uma cidade;

8) Sistema: Consulta e apresenta as agências da locadora existentes na cidade

selecionada;

9) Cliente: Seleciona uma agência;

10) Sistema: Pede data e hora da retirada e devolução do veículo;

11) Cliente: Entra com data e hora da retirada e devolução do veículo;

12) Cliente: Seleciona “Veículo/Tarifa”

13) Sistema: Verifica que a data e hora da retirada e devolução do veículo estão

corretas;

14) Sistema: Verifica que o cliente não possui reserva para o período informado;

15) Sistema: Consulta e apresenta as categorias de veículos existentes;

16) Cliente: Seleciona uma categoria;

17) Sistema: Consulta e apresenta os veículos existentes para a categoria

selecionada;

18) Cliente: Seleciona um veículo;

19) Sistema: Consulta e apresenta as tarifas para o veículo selecionado;

20) Cliente: Seleciona uma tarifa;

21) Cliente: Seleciona a opção “Confirmar reserva”;

22) Sistema: Cadastra a reserva

Fluxos alternativos

1) Os dados obrigatórios do cliente não foram informados

a)Sistema apresenta mensagem de erro;

b)Cliente confirma o erro;

c)Caso de uso retorna ao passo 2 (dois) do fluxo básico

2) Data de devolução do veículo é anterior ou igual aquela de retirada do veículo

a)Sistema apresenta mensagem de erro;

b)Cliente confirma o erro;

c)Caso de uso retorna ao passo 10 (dez) do fluxo básico

3) Cliente já possui reserva para o período informado

a)Sistema apresenta mensagem em que informa o fato e solicita que o cliente

entre em contato com a central de reservas.

b)Caso de uso termina

Pós-condições

• A reserva foi cadastrada no sistema

Pontos de extensão

Nenhum

Requisitos especiais

Nenhum

Informações adicionais

Nenhuma

Caso de Uso

Cadastrar veículo

Breve descrição

Este caso de uso descreve a operação de cadastrar veículo, sendo chamado a partir do caso de uso “Manter Veículos”.

Ator(es)

Administrador

Condições prévias

O administrador deve ter efetuado login

Fluxo básico

1) O administrador seleciona “Novo veículo”;

2) O sistema pede os dados do veículo;

3) O administrador entra com os dados;

4) O sistema verifica que os dados estão corretos;

5) O sistema cadastra o veículo

Pós-condições

O veículo foi cadastrado no sistema

Pontos de extensão

Nenhum

Requisitos especiais

Nenhum

Informações adicionais

Nenhuma

Caso de uso

Manter veículo

Breve descrição

Este caso de uso descreve operações para consultar, editar e excluir veículos.

Ator(es)

Administrador do sistema

Condições prévias

O administrador deve ter efetuado login

Fluxo básico

1) O gerente seleciona a opção "Cadastro de veículos";

2) O sistema consulta os veículos existentes;

3) O sistema apresenta lista dos veículos existentes, juntamente com opções para consultar, editar e excluir cada veículo;

4) O sistema apresenta também opção “Novo veículo”, que chama o caso de uso “Cadastrar Veículo”.

Fluxos alternativos

1) Consultar veículo

1.1) Após o passo 3 (três) do fluxo básico, o gerente seleciona a opção “Consultar” junto a determinado veículo;

1.2) O sistema consulta os dados do veículo;

1.3) O sistema apresenta os dados do veículo;

2) Alterar veículo

2.1) Após o passo 3 (três), o gerente seleciona a opção “Alterar” junto a determinado veículo;

2.2) O sistema consulta os dados do veículo;

2.3) O sistema apresenta os dados do veículo;

2.4) O gerente atualiza os dados desejados do veículo;

2.5) O sistema verifica que os dados estão corretos;

2.6) O sistema atualiza os dados do veículo;

2.7) Retorna ao passo 3 (três) do fluxo básico.

3) Excluir veículo

3.1) Após o passo 3 (três), o gerente seleciona a opção “Excluir” junto a determinado veículo;

3.2) O sistema recupera os dados do veículo;

3.3) O sistema apresenta os dados do veículo; O sistema pede que o gerente confirme a exclusão do veículo;

3.4) O gerente confirma a exclusão;

3.5) O sistema exclui o veículo;

3.6) Retorna ao passo 3 (três) do fluxo básico.

Pós-condições

• Os veículos existentes foram listados;   
• Os dados do veículo selecionado foram exibidos;   
• Os dados do veículo selecionado foram atualizados;   
• O veículo selecionado foi excluído

Pontos de extensão

Nenhum

Requisitos especiais

Nenhum

Informações adicionais

Nenhum

Caso de uso

Receber locação de veículo

Breve descrição

Este caso de uso permite a um atendente receber o valor correspondente à locação de um veículo.

Ator(es)

Atendente

Condições prévias

Nenhuma

Fluxo básico

1. Atendente: Seleciona a opção "Receber locação";
2. Sistema: Consulta e apresenta os dados dos clientes que possuem contrato de locação de veículo;
3. Atendente: Seleciona um cliente e em seguida seleciona “Receber locação”;
4. Sistema: Consulta e apresenta os dados da locação;
5. Sistema: Calcula e apresenta o valor da locação;
6. Sistema: Trata pagamento (UC Tratar Pagamento)

Fluxos alternativos

1) Não existe cliente com contrato de locação “em ser”

a) Sistema apresenta mensagem informando o fato;

b) Atendente confirma;

c)Caso de uso termina

2) Atendente não seleciona um cliente

a)Sistema apresenta mensagem de erro;

b)Atendente confirma o erro;

c)Caso de uso retorna ao passo 2 (dois) do fluxo básico

Pós-condições

• O valor da locação foi contabilizado corretamente

Requisitos especiais

Nenhum

Informações adicionais

Nenhuma

Caso de uso

Tratar pagamento

Breve descrição

Este caso de uso permite a um atendente escolher a forma de pagamento e processar o pagamento da locação do veículo

Ator(es)

Atendente

Condições prévias

Nenhuma

Fluxo básico

1) Sistema: Apresenta as formas de pagamento aceitas;

2) Sistema: Solicita uma forma de pagamento;

3) Atendente: Seleciona uma forma de pagamento

Fluxos alternativos

1) Pagamento com cartão de crédito

a) Atendente: Seleciona “Cartão de crédito” no passo 3 (três);

b) Sistema: Solicita os dados do cartão de crédito

c) Atendente: Entra com os dados do cartão de crédito;

d) Sistema: Solicita autorização de pagamento ao Banco;

e) Sistema: Verifica que foi autorizado o pagamento pelo Banco;

f) Sistema: Envia dados do pagamento para o Contas a Receber;

g) Sistema: Apresenta mensagem em que informa que foi autorizado o

pagamento pelo Banco;

h) Atendente: Confirma a mensagem;

i) Sistema: Imprime comprovante de autorização de pagamento

j)Caso de uso termina

Pontos de extensão

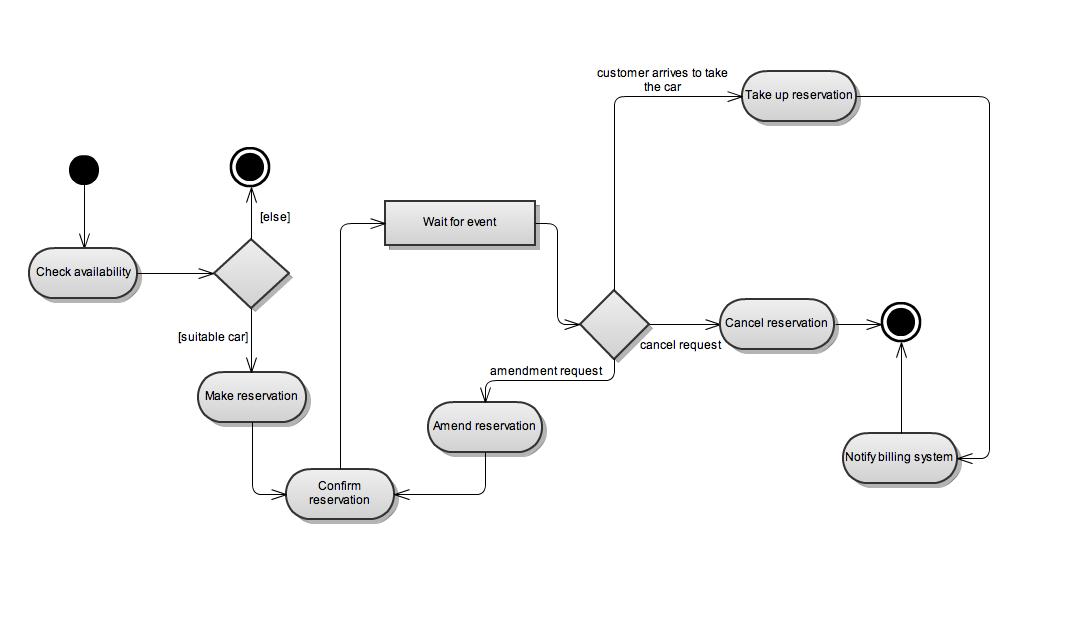
Nenhum

Requisitos especiais

Nenhum

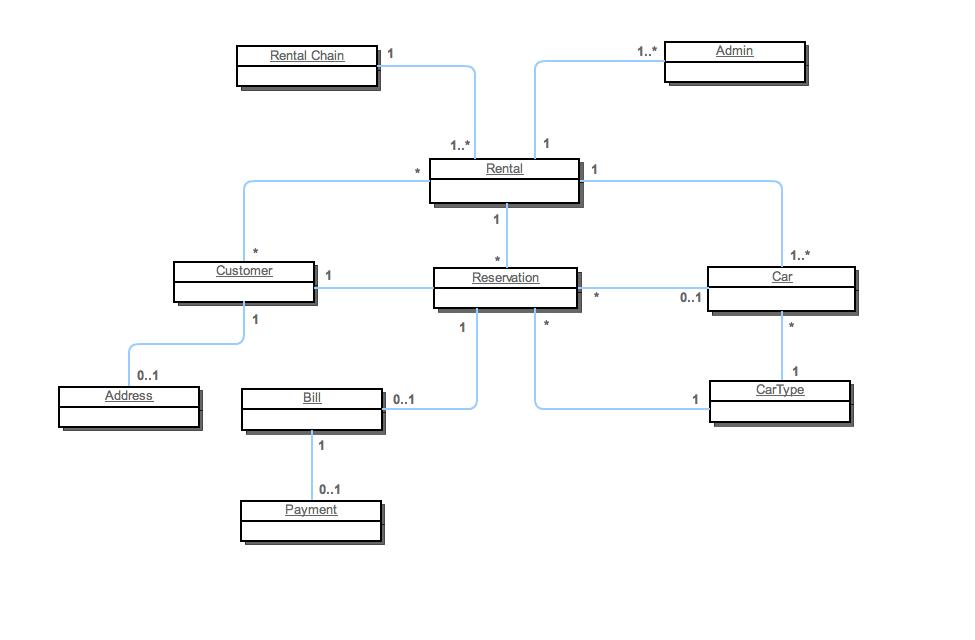
Informações adicionais

Nenhuma

**Business Process**

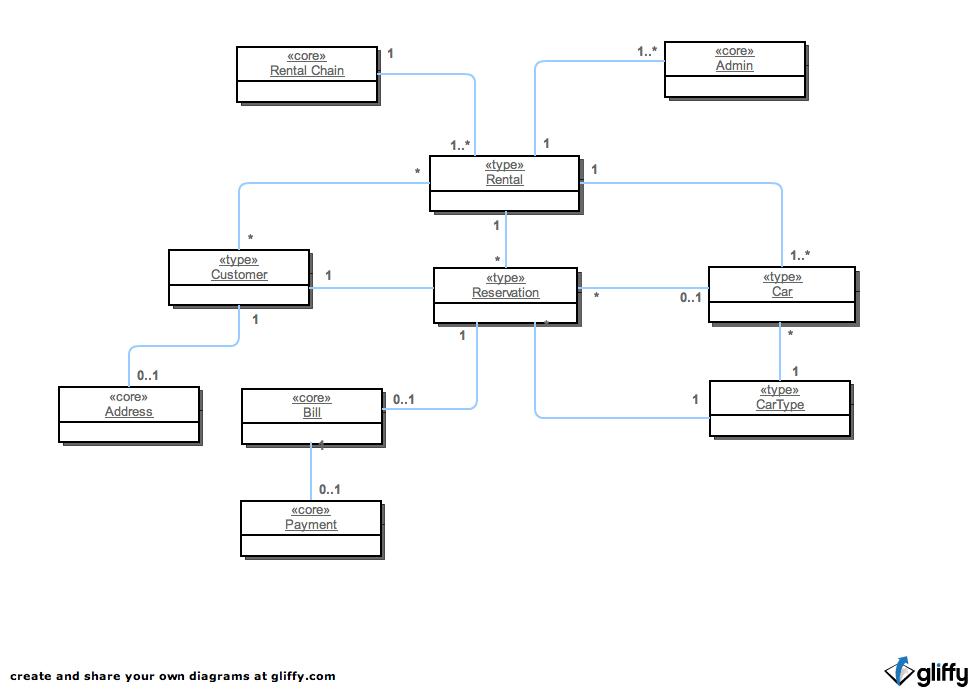
**Business Concept Model**

O Business Concept Model define a relação entre as classes do sistema proposto.

****

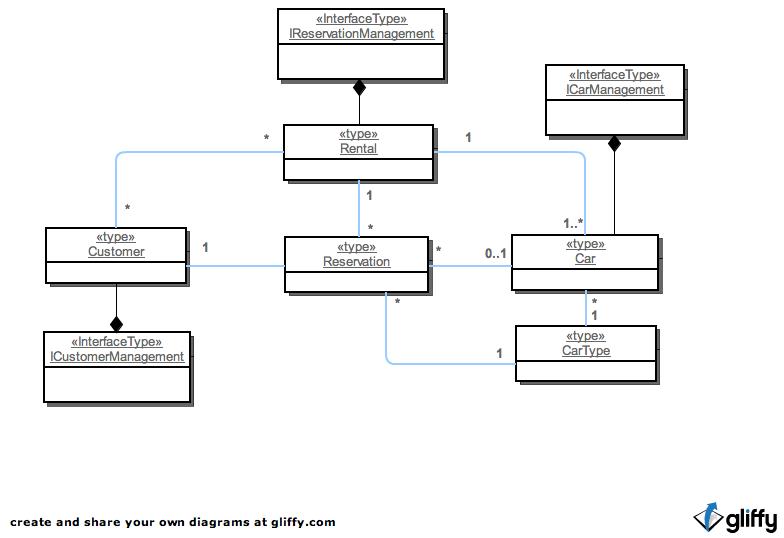
**Business Type Model**

O Business Type Model define o tipo de dados em cada classe. Os dados do tipo CORE, são dados que não necessitam de informações de outras classes para serem criadas, diferente do TYPE que necessita.



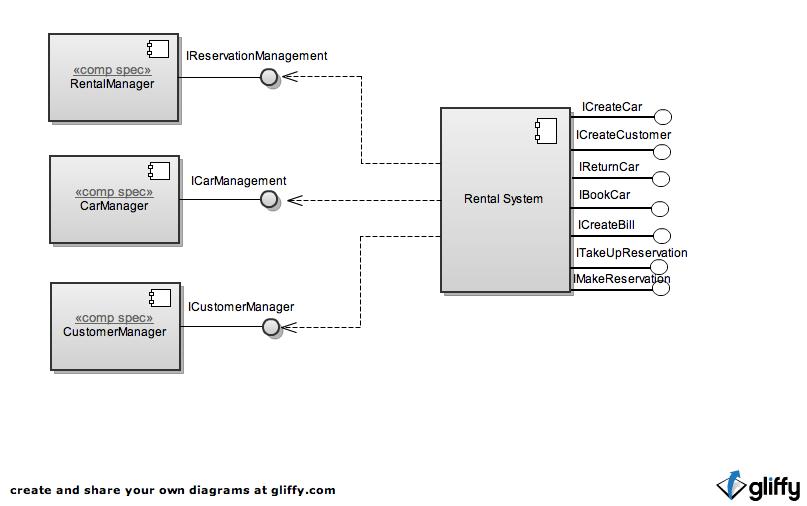
**Business Interfaces**

Através do Business Type Model, o diagrama Business Interfaces é criado. A funcionalidade dele é demonstrar quais as classes que necessitam de um management. Exemplificando: no caso da classe Customer, é necessário tratar os dados, e para isso, é necessário uma interface do cliente com o sistema.

****

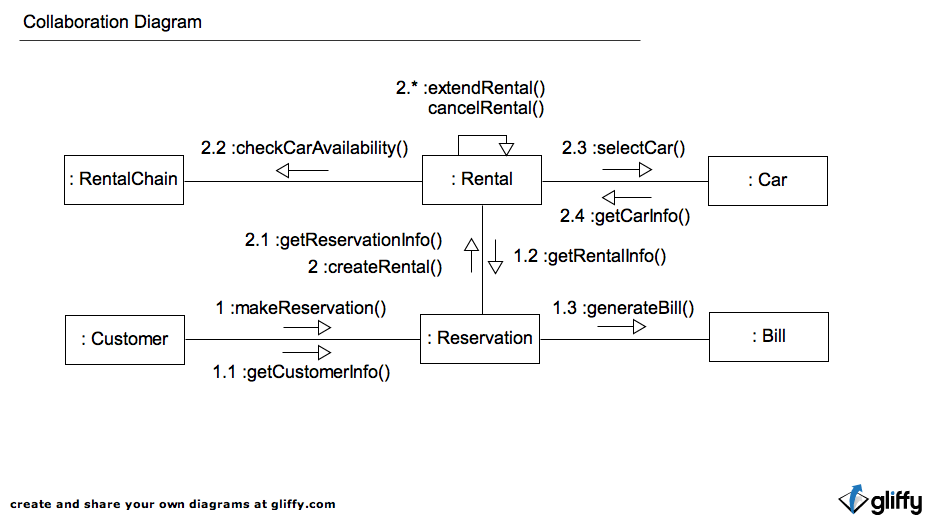
**Architecture**

Uma vez definidas as interfaces, o diagrama da arquitetura do sistema é criado, relacionando módulos do sistema e interfaces.

****

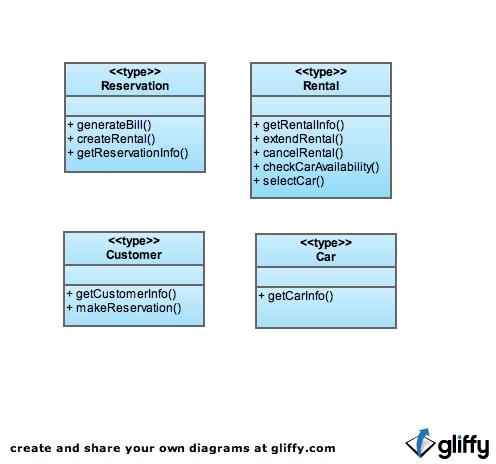
**Collaboration Diagrams**

O diagrama de colaboração modela as interações entre objetos ou partes, em termos de mensagens em sequência.

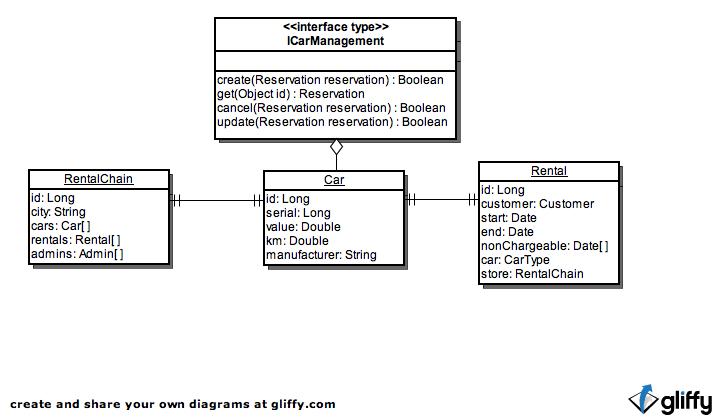
****

**Business Operations**

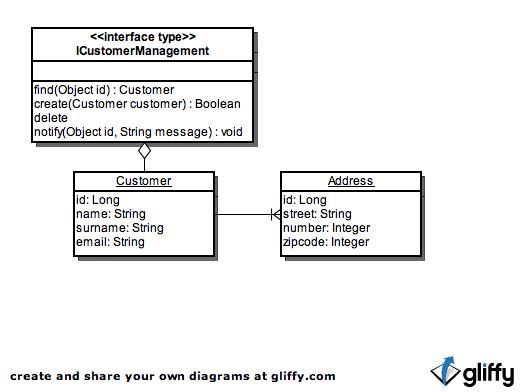
Uma vez com as interfaces definidas, suas operações principais surgem com maior clareza e é possível montar o business operations.

****

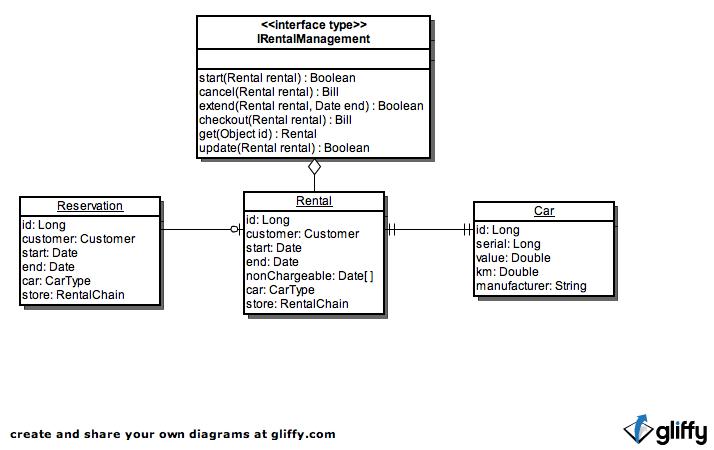
**Interface Information Model: Car**

****

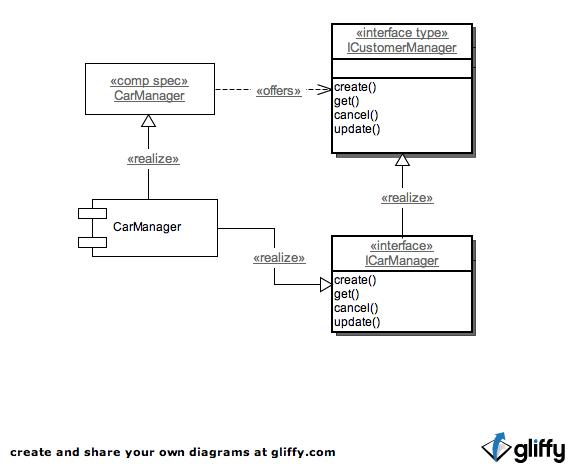
**Interface Information Model: Customer**

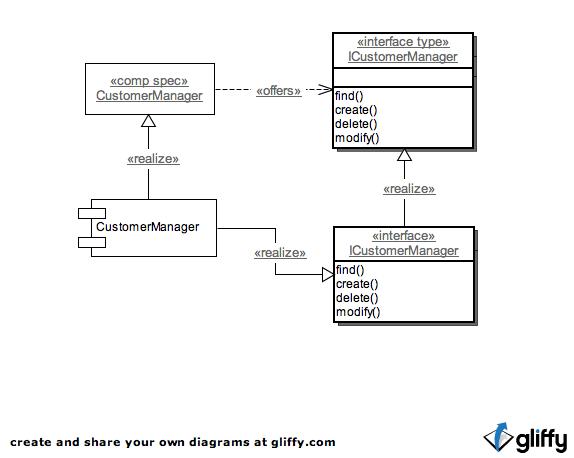
****

**Interface Information Model: Rental**

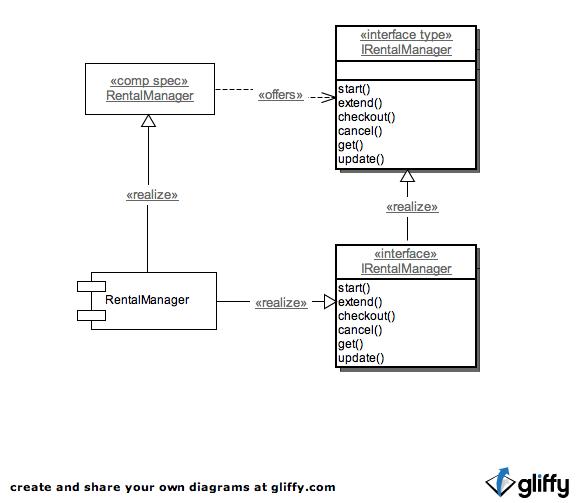
****

**Mapping Class Diagrams: Car**

****

**Mapping Class Diagrams: Customer**

**Mapping Class Diagrams: Rental**

****

**Interface Source Code: IMakeCarReservation**

public interface IMakeCarReservation{

Boolean createReservation(Car car, Date start, Date end, Client client);

List<Car> getAvailableCars(RentalChain rentalChain, Customer customer);

List<RentalChain> getRentalChains(String city);

List<String> getCities();

}

public class IMakeCarImpl implements IMakeCarReservation{

public Boolean createReservation(Car car, Date start, Date end, Client client){

InitialContext context = new InitialContext();

ReservationFacadeRemote remoteInterface = (ReservationFacadeRemote) context.lookup(ReservationFacadeRemote.class.getName());

return remoteInterface.create(new Reservation(car, start, end, client);

}

public List<Car> getAvailableCars(RentalChain rentalChain, Customer customer){

InitialContext context = new InitialContext();

CarFacadeRemote remoteInterface = (CarFacadeRemote) context.lookup(CarFacadeRemote.class.getName());

return remoteInterface.findCarByCustomer(customer);

}

public List<RentalChain> getRentalChains(String city){

InitialContext context = new InitialContext();

RentalChainFacadeRemote remoteInterface = (RentalChainFacadeRemote) context.lookup(RentalChainFacadeRemote.class.getName());

return remoteInterface.findRentalChainByCity(city);

}

public List<String> getCities(){

InitialContext context = new InitialContext();

CityFacadeRemote remoteInterface = (CityFacadeRemote) context.lookup(CityFacadeRemote.class.getName());

return remoteInterface.findAll();

}

}

@Remote

public interface ReservationFacadeRemote{

Boolean create(Reservation reservation);

}

@Remote

public interface CarFacadeRemote{

List<Car> findCarByCustomer(Customer customer);

}

@Remote

public interface RentalChainFacadeRemote{

List<RentalChain> findRentalChainByCity(String city);

}

@Remote

public interface CityFacadeRemote{

List<String> findAll();

}

@Stateless

public class ReservationFacadeRemoteImpl implements ReservationFacadeRemote{

@PersistenceContext(unitName = "mackenzie-ejbPU")

private EntityManager em;

public EntityManager getEntityManager(){

return em;

}

public Boolean create(Reservation reservation){

getEntityManager().persist(reservation);

}

}

@Stateless

public class CarFacadeRemoteImpl implements CarFacadeRemote{

@PersistenceContext(unitName = "mackenzie-ejbPU")

private EntityManager em;

public EntityManager getEntityManager(){

return em;

}

public List<Car> findCarByCustomer(Customer customer){

return getEntityManager().createNamedQuery("searchCarByCustomer").setParameter("customerId", customer.getId()).getResultList();

}

}

@Stateless

public class RentalChainFacadeRemoteImpl implements RentalChainFacadeRemote{

@PersistenceContext(unitName = "mackenzie-ejbPU")

private EntityManager em;

public EntityManager getEntityManager(){

return em;

}

public List<RentalChain> findRentalChainByCity(String city){

return getEntityManager().createNamedQuery("searchRentalChainByCity").setParameter("city", city).getResultList();

}

}

@Stateless

public class CityFacadeRemoteImpl implements CityFacadeRemote{

@PersistenceContext(unitName = "mackenzie-ejbPU")

private EntityManager em;

public EntityManager getEntityManager(){

return em;

}

public List<String> findAll(){

return getEntityManager().createNamedQuery("retrieveAllCities").getResultList();

}

}

**Interface Source Code: ICarReservation**

public interface ICarManagement{

Boolean registerCar(Car car);

Boolean deleteCar(Car car);

Boolean updateCar(Car car);

Car retrieveCar(Object id);

}

public class ICarManagementImpl implements ICarManagement{

public Boolean registerCar(Car car){

InitialContext context = new InitialContext();

CarFacadeRemote remoteInterface = (CarFacadeRemote) context.lookup(CarFacadeRemote());

return remoteInterface.create(car);

}

public Boolean deleteCar(Car car){

InitialContext context = new InitialContext();

CarFacadeRemote remoteInterface = (CarFacadeRemote) context.lookup(CarFacadeRemote());

return remoteInterface.remove(car);

}

public Boolean deleteCar(Car car){

InitialContext context = new InitialContext();

CarFacadeRemote remoteInterface = (CarFacadeRemote) context.lookup(CarFacadeRemote());

return remoteInterface.update(car);

}

public Car retrieve(Object id){

InitialContext context = new InitialContext();

CarFacadeRemote remoteInterface = (CarFacadeRemote) context.lookup(CarFacadeRemote());

return remoteInterface.find(id);

}

}

@Remote

public interface CarFacadeRemote{

Boolean create(Car car);

Boolean remove(Car car);

Boolean update(Car car);

Car find(Object id);

}

@Stateless

public class CarFacadeRemoteImpl implements CarFacadeRemote{

@PersistenceContext(unitName = "mackenzie-ejbPU")

private EntityManager em;

public EntityManager getEntityManager(){

return em;

}

public Boolean create(Car car){

return getEntityManager().persist(car);

}

public remove(Car car){

return getEntityManager().remove(getEntityManager().merge(car));

}

public update(Car car){

return getEntityManager().merge(car);

}

public Car find(Object id){

return getEntityManager().find(Car.class, id);

}

}