UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
INSTITUTO METRÓPOLE DIGITAL
DIM0514 ARQUITETURA DE SOFTWARE

Janeto Erick da Costa Lima <janetoerick18@gmail.com>

1. Introdução

O presente relatório descreve o desenvolvimento de um projeto de gerenciamento de estacionamento, no qual se utiliza a ferramenta SysADL para a modelagem arquitetural do sistema. O SysADL é uma linguagem de modelagem que permite representar a arquitetura de sistemas de forma clara e precisa, facilitando o entendimento das interações entre os componentes do sistema.

O objetivo deste projeto é criar uma solução eficiente para a gestão de um estacionamento, com funcionalidades como controle de entrada e saída de veículos, monitoramento de ocupação das vagas e integração com sistemas de pagamento. Através da utilização do SysADL, busca-se obter uma visão abrangente da arquitetura do sistema, garantindo que os requisitos funcionais e não funcionais sejam atendidos.

Neste contexto, o relatório aborda o processo de modelagem arquitetural do sistema de estacionamento, detalhando as decisões de design, a estrutura dos componentes do sistema, bem como os principais desafios e soluções adotadas para a implementação da arquitetura.

2. Modelagem de Requisitos

Os requisitos foram elaborados a partir da análise detalhada das necessidades do estacionamento, incluindo a gestão de vagas, o controle de entrada e saída de veículos, a interação com os usuários e a integração com sistemas de pagamento. Na tabela 1, estão listados todos os requisitos, contendo seu código, nome, descrição e suas relações. Ao todo, são 17 requisitos funcionais e 9 requisito não funcional.

ID	Nome	Descrição	Relações
1	ControlarCancelaRF	O sistema deve ser capaz de controlar a cancela.	-

1.1	ControlarCancelaManualmenteRF	O sistema deve permitir que um usuário autorizado acione a cancela manualmente.	comp. 1
1.1.1	AbrirCancelaRF	O sistema deve abrir a cancela.	der. 1.1, der. 1.2.1, der. 1.2.7
1.1.2	FecharCancelaRF	O sistema deve fechar a cancela.	der. 1.1, der 1.1.3
1.1.3	VerificarPassagemVeiculoRF	O sistema deve ser capaz de verificar se o veículo passou pela cancela.	der. 1.1.1
1.2	ControlarCancelaAutomaticamenteRF	O sistema deve ser capaz de acionar a cancela.	comp. 1
1.2.1	FornecerTicketRF	O sistema deve fornecer ao usuário um ticket de acesso.	der 1.2
1.2.2	CapturarHorarioTicketRF	O sistema deve capturar o horário em que o ticket foi retirado.	der. 1.2.1
1.2.3	CalcularValorDoTicketRF	O sistema deve calcular o valor total do ticket com base na hora de entrada e saída.	der. 1.2.2
1.2.4	PagarTicketRF	O sistema deve permitir que o usuário pague o ticket.	der. 1.2.3
1.2.5	EscanearTicketRF	O sistema deve escanear o ticket.	der. 1.2
1.2.6	VerificarStatusDoTicketRF	O sistema deve ser capaz de verificar os status do ticket.	der. 1.2.5
1.2.7	ConfirmarPagamentoTicketRF	O sistema deve informar se o ticket está pago.	der. 1.2.6
2	GerenciarVagasRF	O sistema deve gerenciar a quantidade de vagas livres e	der. 2.2

		ocupadas.	
2.1	ExibirVagasRF	O sistema deve ser capaz de exibir a quantidade de vagas disponíveis.	der. 2
2.2	InformarPresencaDeVeiculoRF	O sistema deve informar a presença do veículo na vaga.	der. 2.3
2.3	MonitorarPresencaDeVeiculoRF	O sistema deve monitorar se há veículo em cada vaga.	-
3	QualidadeRNF	O sistema deve apresentar qualidade no seu funcionamento.	-
3.1	DisponibilidadeRNF	O sistema deve operar 24 horas por dia, 7 dias por semana.	comp. 3
3.2	DesempenhoRNF	O sistema deve ser capaz de processar múltiplos processos ao mesmo tempo de forma eficiente.	comp. 3
3.3	EscalabilidadeRNF	O sistema deve ser capaz de suportar um aumento de usabilidade sem comprometer o desempenho.	comp. 3
3.4	SegurançaRNF	O sistema deve manipular as informações de forma criptografada para não comprometer os dados dos usuários.	comp. 3
3.5	UsabilidadeRNF	O sistema deve ser de fácil compreensão a todo tipo de usuário.	comp. 3
3.6	ManutenibilidadeRNF	O sistema deve ser de fácil manutenção.	comp. 3
3.7	ToleranciaAFalhasRNF	O sistema deve ser tolerante a falhas no	comp. 3

		funcionamento de seus sensores.	
3.8	ModificabilidadeRNF	O sistema deve ser de fácil alteração.	comp. 3

Tabela 1. Requisitos do sistema

Na tabela, vemos que ControlarCancelaRF divide-se em dois requisitos: ControlarCancelaManualmenteRF (*id.1.1*) e ControlarCancelaAutomaticamenteRF (*id.1.2*). Essa divisão do requisito foi feita para que o sistema tivesse a autonomia de controlar a abertura e fechamento das cancelas mediante situações pré estabelecidas. Além disso, deve permitir que um usuário autorizado consiga acionar manualmente a abertura e o fechamento, sendo esta alternativa importante para casos de indisponibilidade do sistema.

Partindo de *ControlarCancelaManualmenteRF* (*id.1.1*), este se deriva em dois outros requisitos: *AbrirCancelaRF* (*id. 1.1.1*) e *FecharCancelaRF* (*id.1.1.2*), que diz respeito a abertura e fechamento da cancela, respectivamente. Para contribuir com a automação do sistema foi modelado o requisito VerificarPassagemVeiculoRF (*id.1.1.3*), no qual possibilita o fechamento da cancela, após sua abertura, quando um veículo atravessa-la.

Com relação à *ControlarCancelaAutomaticamenteRF* (*id.1.2*), é feita a derivação em outros dois requisitos que possibilitam o sistema a concretizar a abertura da cancela, são eles: *FornecerTicketRF* (*id.1.2.1*) e *EscanearTicketRF* (*id.1.2.5*). Como o controle do estacionamento será feito mediante ticket de acesso, o sistema deverá realizar o controle da cancela a partir da entrega e verificação dos tickets, sendo necessário a captura da hora de entrada do carro (*id.1.2.2*) para calcular o valor de estacionamento (*id.1.2.3*), possibilitando o pagamento do ticket pelo usuário (*id.1.2.4*).

3. Diagrama de Requisitos

O diagrama de requisitos teve dois núcleos de relacionamentos, um dos núcleos diz respeito ao controle das cancelas e do pagamento, e o outro tem relação ao gerenciamento das vagas. Primeiramente será exibido os núcleos separadamente, para melhor visualização, e no final será apresentado o diagrama de requisitos completo.

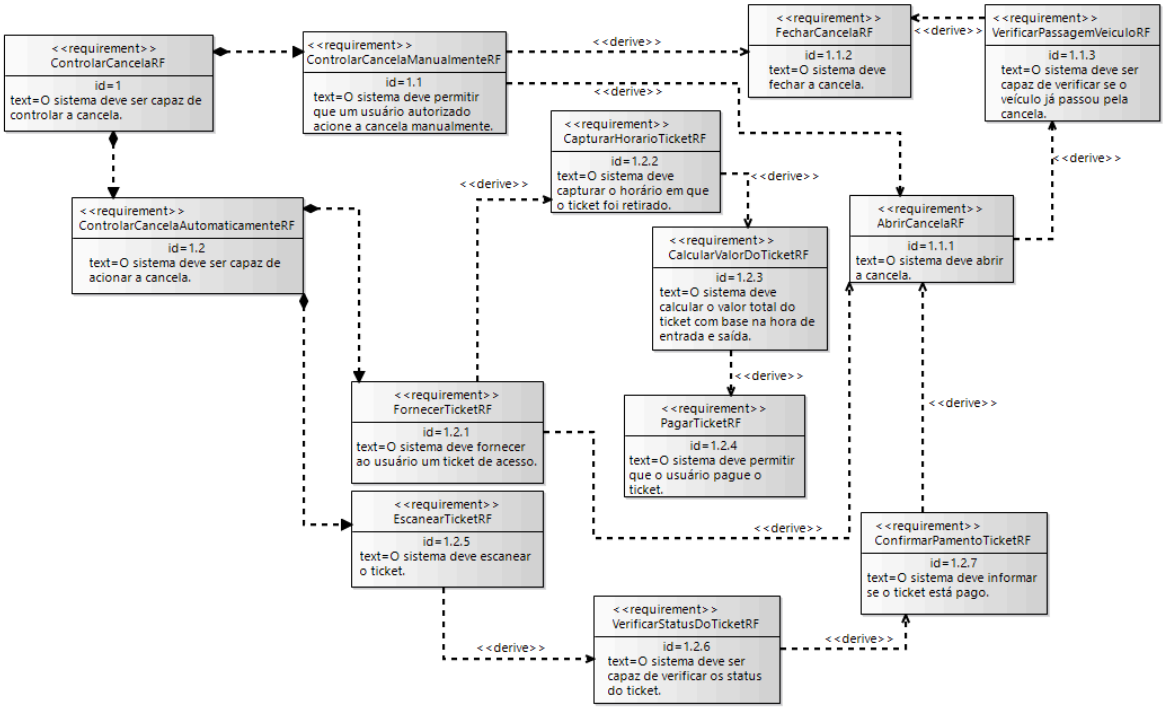

Figura 1. Requisitos relacionados com acionar cancela

A figura 1 mostra os relacionamentos de composição entre o requisito *1* e suas composições *1.1* e *1.2*, com o complemento da composição do requisito *1.2* nos requisitos *1.2.1* e *1.2.5*. Ainda, os relacionamentos de derivação dos demais requisitos no núcleo.

Com relação a figura 2, é mostrado os requisitos referentes a gerência do controle de vagas dentro do estacionamento. Isto é feito com o requisito *2*, os requisitos que o derivam, *2.1*, e de quem ele deriva (*2.2* e *2.3*).

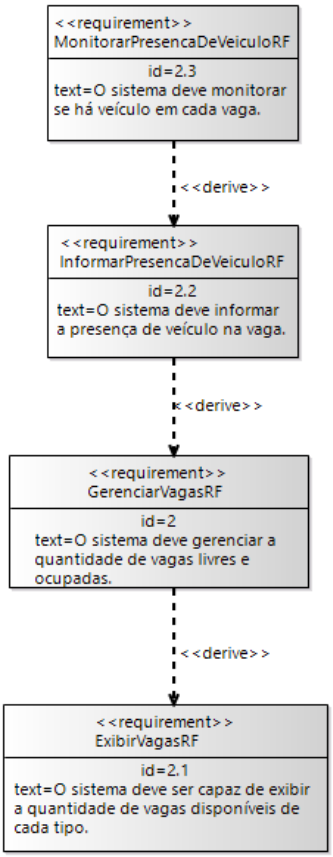

Figura 2. Requisitos relacionados com gerenciar vagas

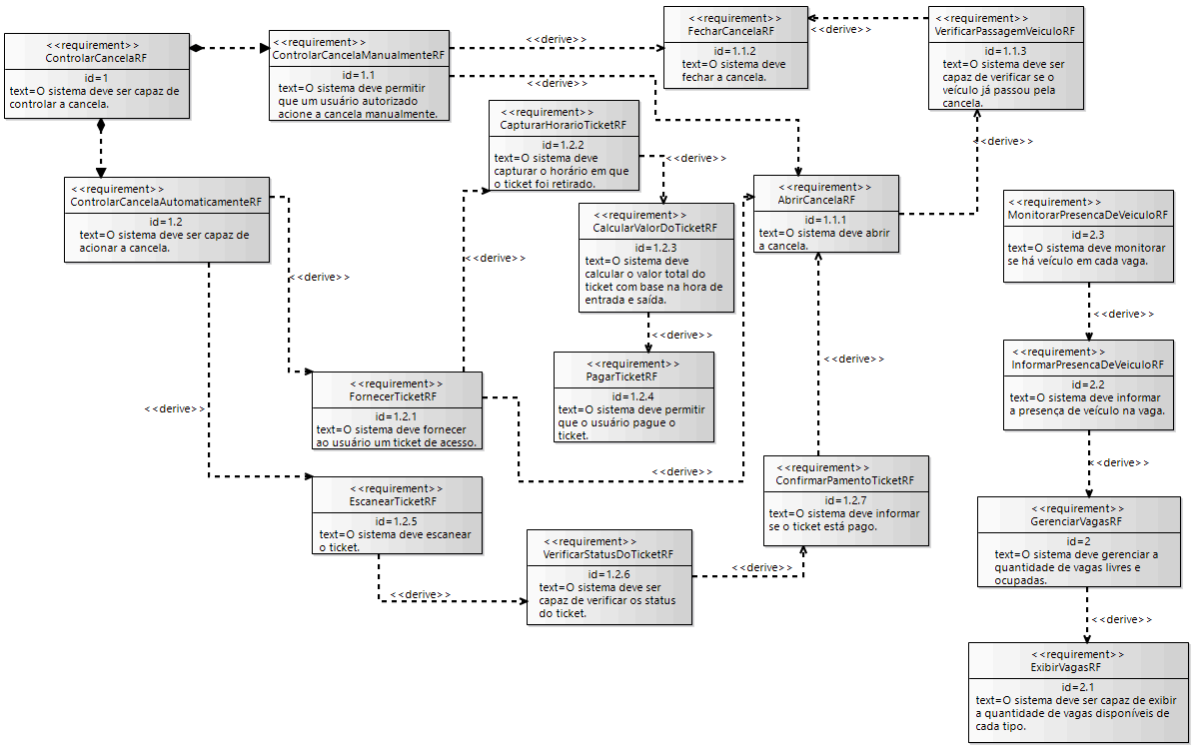

Figura 3. Diagrama de Requisitos Funcionais

Além dos já citados, todos presentes na figura 3, há também a presença dos requisitos não funcionais mostrados na figura 4 a seguir, na qual apresenta todas as necessidades que o sistema deve ter para garantir a qualidade. Muitos destes, serão citados ao longo do relatório como necessidade para algumas decisões tomadas.

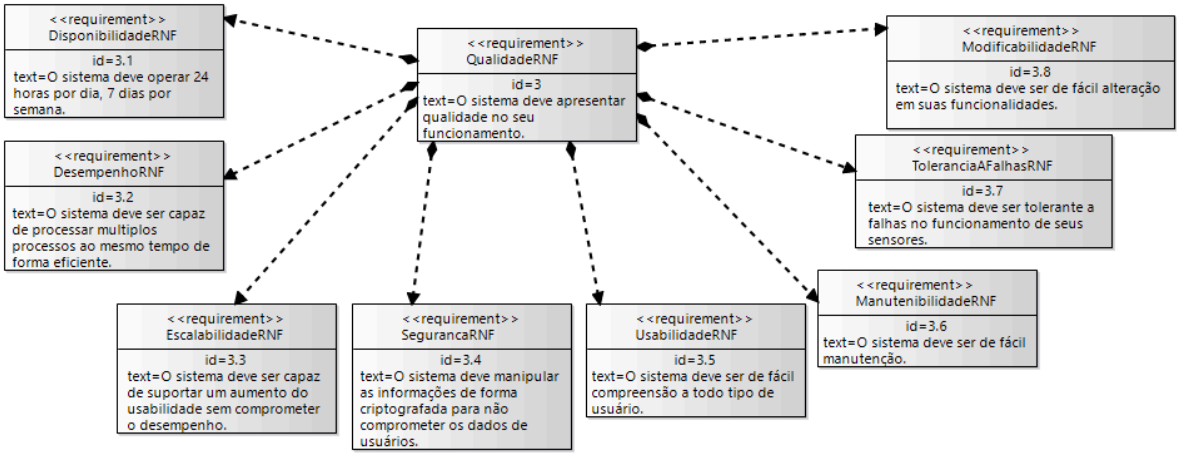

Figura 4. Diagrama de Requisitos Não Funcionais

4. Visão Estrutural

Para a modelagem da visão estrutural do sistema de estacionamento, foram separados quatro processos de construção para separar os principais elementos da arquitetura, ou seja, os tipos de valor, portas, conectores e componentes, os quais são essenciais para o desenvolvimento da arquitetura de software no SysADL.

Para cada etapa de criação dos elementos foi criado um Diagrama de Definição de Bloco (BDD), para definir os respectivos elementos ligados a eles. Esses diagramas serão apresentados nas subseções seguintes.

4.1. Diagrama de Valores

Para a construção da arquitetura utilizando o SysADL, os valores são os elementos atômicos necessários para iniciar a modelagem estrutural do sistema. Esse diagrama faz referência a todos os valores que serão utilizados. A representação dos valores na linguagem são feitos através dos elementos gráficos: tipos de valor (Value Type), tipo de dado (dataType) e enumeração (enumeration). Além disso, pode ter um acréscimo de dimensões (dimensions) e unidades (unit), no qual, não é utilizado no sistema proposto. O diagrama de valores do sistema é mostrado na figura 5.

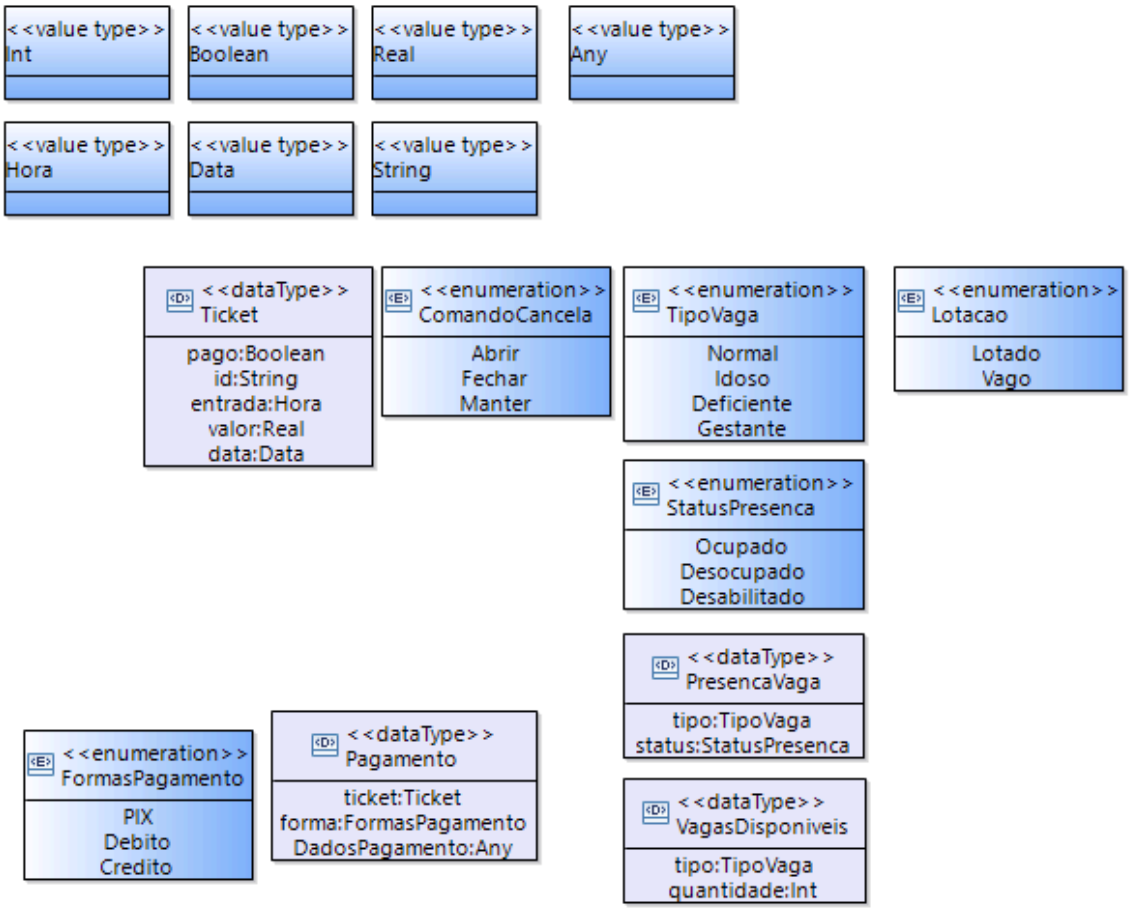

Figura 5. BDD de valores

Foram utilizados quatro dataType: *Ticket*, *VagasDisponiveis, PresencaVaga* e *Pagamento*. *O Ticket* é utilizado no núcleo de controle das cancelas e pagamento, tendo os elementos: pago (verifica se o ticket foi pago), id, entrada (horário de entrada do veículo, momento em que é imprimido), valor (valor à ser pago do Ticket) e data (data da impressão do ticket); *VagasDisponiveis* no núcleo de gerência de vagas e na exibição da quantidade de vagas, com os elementos: tipo e quantidade; *PresencaVaga* tem como objetivo representar os dados do sensor de presença, com os elementos: tipo e status*; Pagamento* é utilizado na confirmação de pagamento do ticket, com os elementos: ticket (informações do ticket a ser pago), forma (forma de pagamento) e DadosPagamento (os dados necessário para confirmar pagamento). Com isso foram criadas três enumerations, *ComandoCancela*, *Vaga* e *FormasPagamento*, que tem como foco os tipos de comando que a cancela deverá receber (abrir, fechar e manter), os tipos de vagas possíveis de ter no estacionamento (normal, idoso, deficiente e gestante) e as formas de pagamento aceitáveis pelo sistema, respectivamente. Além disso, há a presença de sete valueType: *Int, Boolean, Real, String, Hora, Data* e *Any*.

4.2. Diagrama de Portas

As portas são elementos acoplados nos componentes, no qual retratam a entrada e saída de dados. Para cada tipo de dado manipulado no sistema foram criadas suas portas de entrada e saída. O diagrama apresentando as portas é mostrado na figura 6. Ao todo foram feitas vinte e quatro portas, sendo seis destas, compostas.

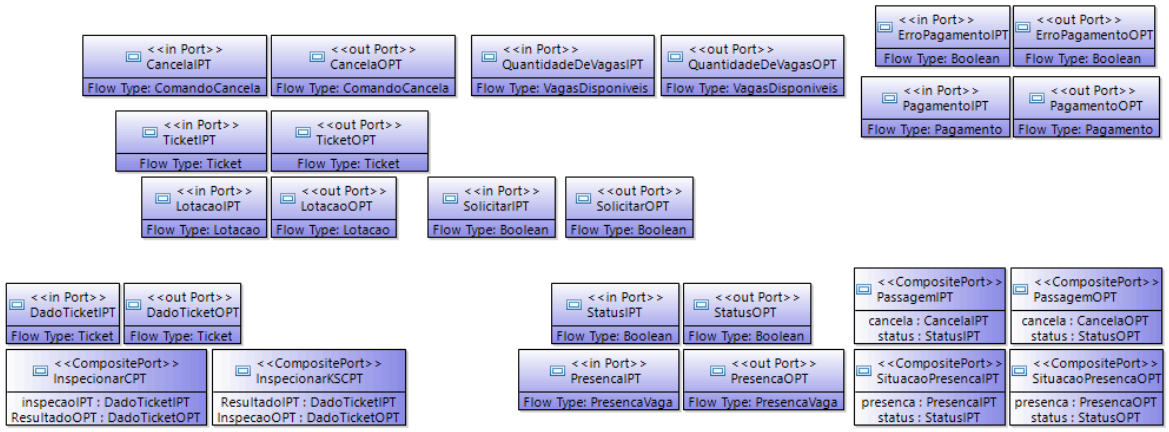

Figura 6. BDD de portas

4.3. Diagrama de Conectores

Os conectores são a representação da manipulação dos dados pelas portas citadas anteriormente, neles são retratados quais valores são passados em cada tipo de porta. No sistema proposto é feito o uso de nove conectores simples e três conectores compostos, a apresentação deles está presente no diagrama da figura 7.

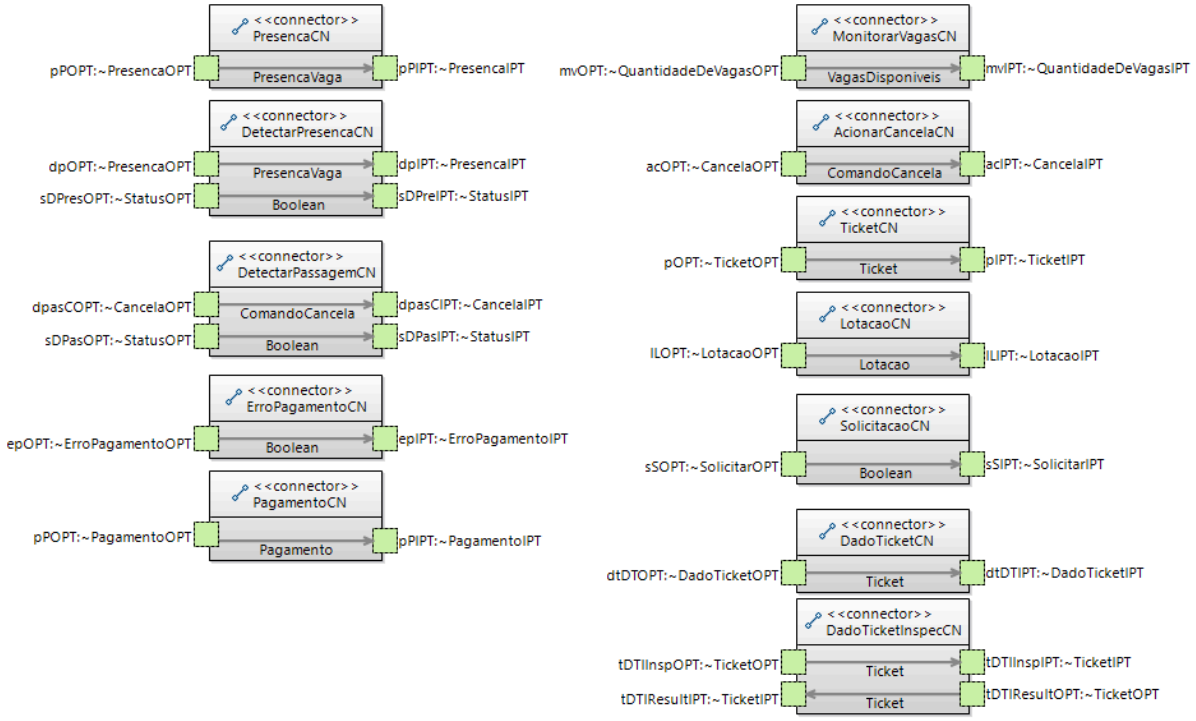

Figura 7. BDD de conectores

4.4. Diagrama de Componentes

Para a construção deste diagrama foi pensado inicialmente em utilizar a estratégia centralizada, para ter o controle das informações concentradas em um único local com algumas iterações feitas fora do seu interior. Contudo, ao começar a construção da parte comportamental do sistema, foi visto que esse modelo centralizado não era adequado para sustentar a qualidade proposta nos requisitos não funcionais.

Com isso, foi remontado um novo BDD para os componentes, no qual fazia a distribuição das ações do sistema de forma mais distributiva, para garantir a qualidade. Além disso, foram empregadas alguns estilos de arquitetura para agregar no melhor funcionamento geral do projeto. Ao todo foram utilizados, nessa nova visão, 23 componentes, no qual é apresentado na figura 8.

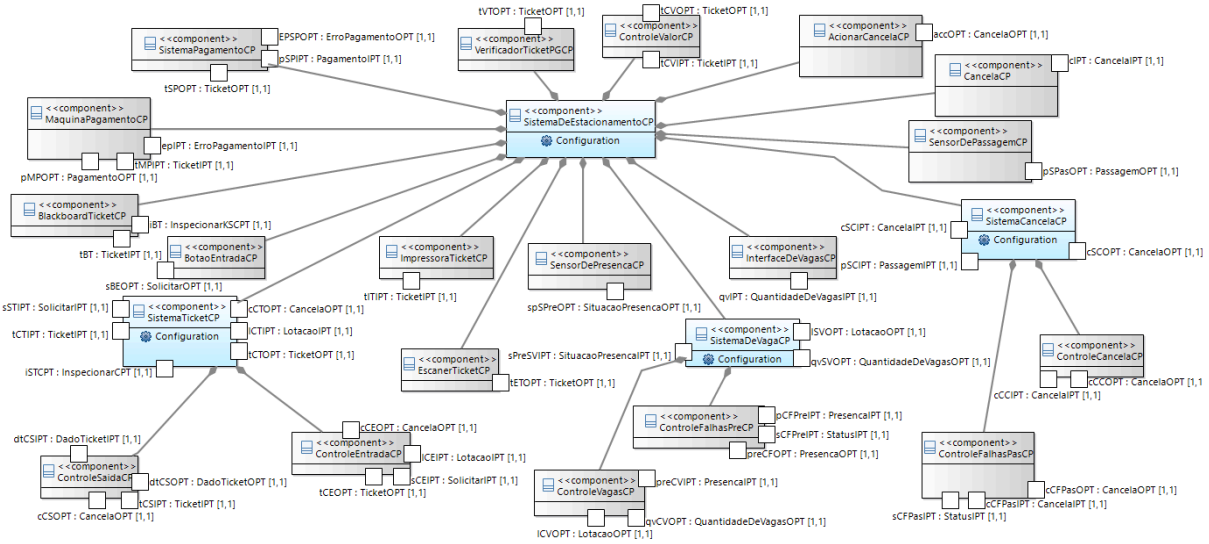

Figura 8. BDD de Componentes

Para o controle das cancelas foram modelados seis componentes, *CancelaCP*, *AcionarCancelaCP, SensorDePassagemCP, SistemaCancelaCP, ControleFalhasPasCP* e *ControleCancelaCP.* No qual tem como propósito fazer o controle das cancelas do estacionamento tanto de forma manual, como também de forma automática, utilizando de dados adquiridos pelos componentes de fronteira. Além disso, há um controle das vagas feito por cinco componentes, *InterfaceDeVagasCP, SensorDePresencaCP, SistemaDeVagaCP, ControleFalhasPreCP* e *ControleVagasCP.* Este é responsável por gerenciar a quantidade de vagas disponíveis no local, proporcionando a exibição das vagas disponíveis por tipos.

Ademais, foram modelados também, um núcleo para controle dos tickets (*EscanerTicketCP, ImpressoraTicketCP, BotaoEntradaCP, SistemaTicketCP, ControleSaidaCP* e *ControleEntradaCP*) e outro para controle do pagamento dos tickets(*MaquinaPagamentoCP, VerificadorTicketPGCP, ControleValorCP* e *SistemaPagamentoCP*), no qual ambos utilizam de uma base de dados disponibilizado pelo componente *BlackboardTicketCP* para garantir uma escalabilidade no uso dos dados dos tickets.

Foram montados quatro IBDs a partir dos componentes citados. Os componentes que tem um IBD referente a ele é marcado na imagem da figura 8 com uma cor azul, são eles: *SistemaDeEstacionamentoCP, SistemaCancelaCP, SistemaDeVagaCP* e *SistemaTicketCP*. Dentre eles, o principal componente dessa arquitetura é o SistemaDeEstacionamentoCP.

A modelagem do IBD referente ao SistemaDeEstacionamento é mostrada na figura 9 a seguir.

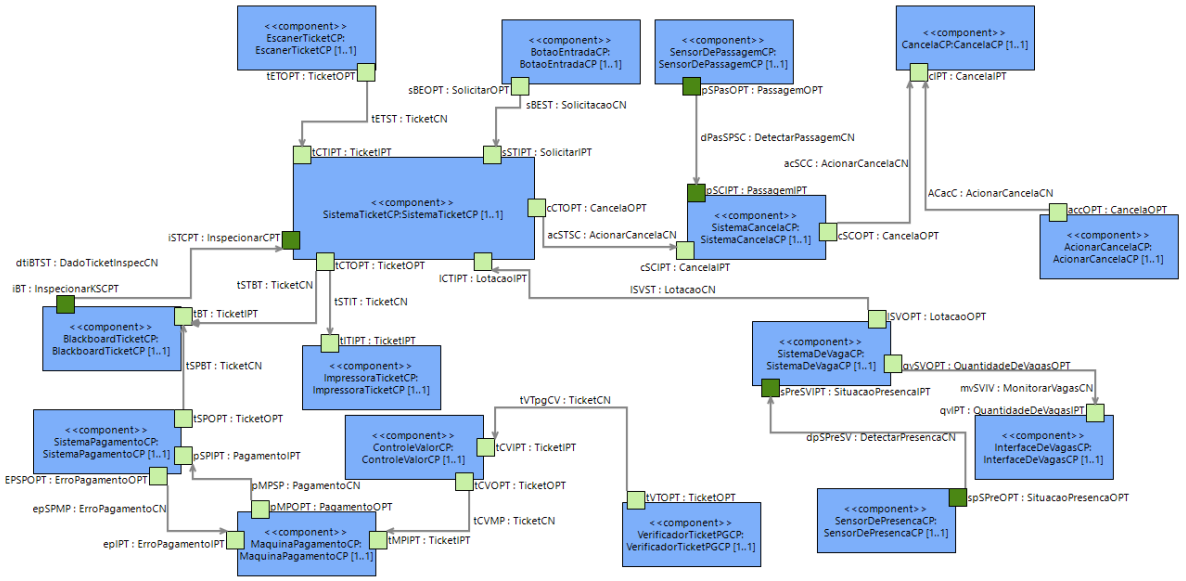

Figura 9. IBD do SistemaDeEstacionamentoCP

Na figura é mostrado o funcionamento do sistema em linhas gerais, como os componentes em geral, de núcleos diferentes, se relacionam entre si.

Tomando como ponto de partida, o núcleo relacionado ao controle das cancelas. Temos o componente *CancelaCP* que recebe o comando (abrir, fechar e manter) dos componentes *AcionarCancelaCP* e *SistemaCancelaCP*, nos quais representam a chamada de forma manual e automática, respectivamente. O comando vindo de *AcionarCancelaCP* representa o acionamento da cancela por um funcionário autorizado para tal ação, já o vindo de *SistemaCancelaCP* é um comando pré processado que toma como base um pedido de modificar o estado da cancela dos componentes *SensorDePassagemCP* e *SistemaTicketCP*. O *SensorDePassagemCP* identifica a passagem do carro pela cancela e solicita ao sistema da cancela que feche-a, enquanto que o *SistemaTicketCP* faz a solicitação de abertura sempre que alguém quiser entrar e sair do estacionamento.

O IBD de *SistemaCancelaCP* é apresentado na figura 10 a seguir. No qual apresenta os componentes referentes ao processamento das solicitações mencionadas anteriormente.

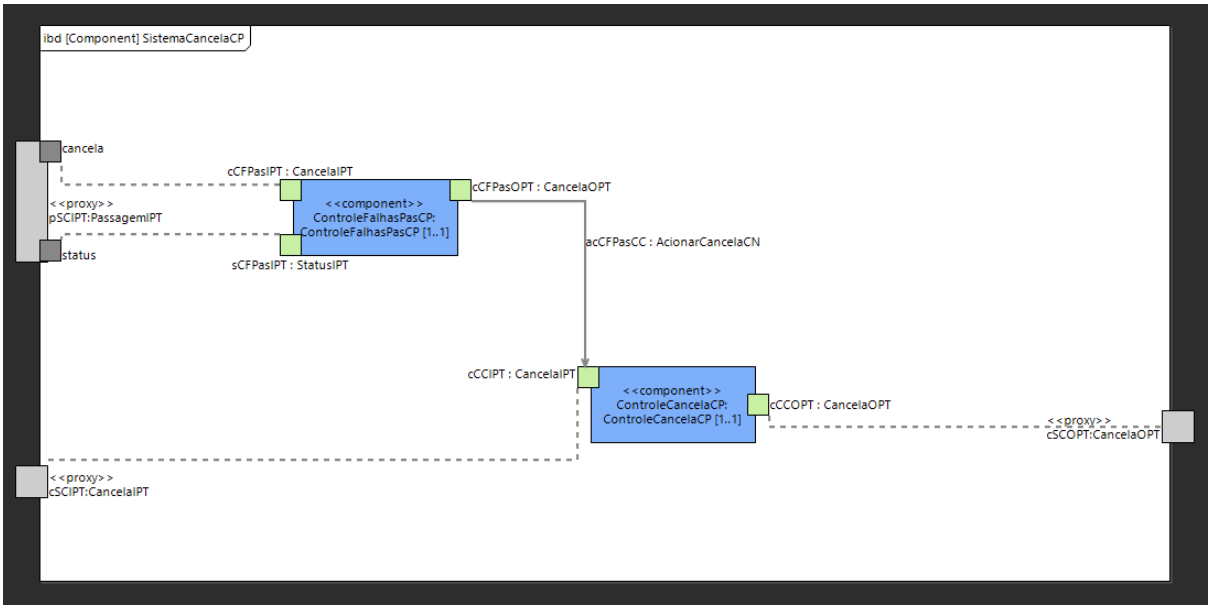

Figura 10. IBD do SistemaCancelaCP

Os dados vindos do *SensorDePassagemCP* passam inicialmente pelo componente *ControleFalhasPasCP* que garante o bom funcionamento do sensor aplicando o controle de falhas de forma concorrente em todos os sensores, confirmado a integridade do componente, é repassado a solicitação sem mudanças para o *ControleCancelaCP.* Porém, caso haja algum problema no sensor, é modificado o comando para o *ControleCancelaCP* ficar ciente do mau funcionamento. Pelo outro lado, a solicitação vinda do *SistemaTicketCP* parte direto para o *ControleCancelaCP* para ser repassado para a cancela. A presença do componente *ControleCancelaCP* é para garantir a escalabilidade em sistemas de estacionamento mais robustos.

Partindo para o núcleo referente ao ticket, há os componentes: *BotaoEntradaCP* no qual é responsável pela solicitação de entrada no estacionamento de um usuário. Essa solicitação é recebida pelo *SistemaTicketCP*, que avalia o status de lotação do estacionamento (disponibilizado pelo *SistemaDeVagaCP*) e caso haja vaga disponível, cria um ticket e manda para o componente *ImpressoraTicketCP* para imprimir e ser recebido pelo cliente. Ao criar o ticket, o *SistemaTicketCP* se comunica com o componente *BlackboardTicketCP* ,este é responsável por armazenar os dados de todos os tickets criados, mandando as informações do novo ticket para serem guardadas na base de dados. Além disso, faz a solicitação ao *SistemaCancelaCP* para a abertura da cancela.

Ademais, temos o *EscanerTicketCP* que manda os dados de um ticket para o *SistemaTicketCP*, com o objetivo de acionar a cancela para saída de veículo.

O IBD de *SistemaTicketCP* é apresentado na figura 11. Nele há os componentes *ControleEntradaCP* e *ControleSaidaCP*.

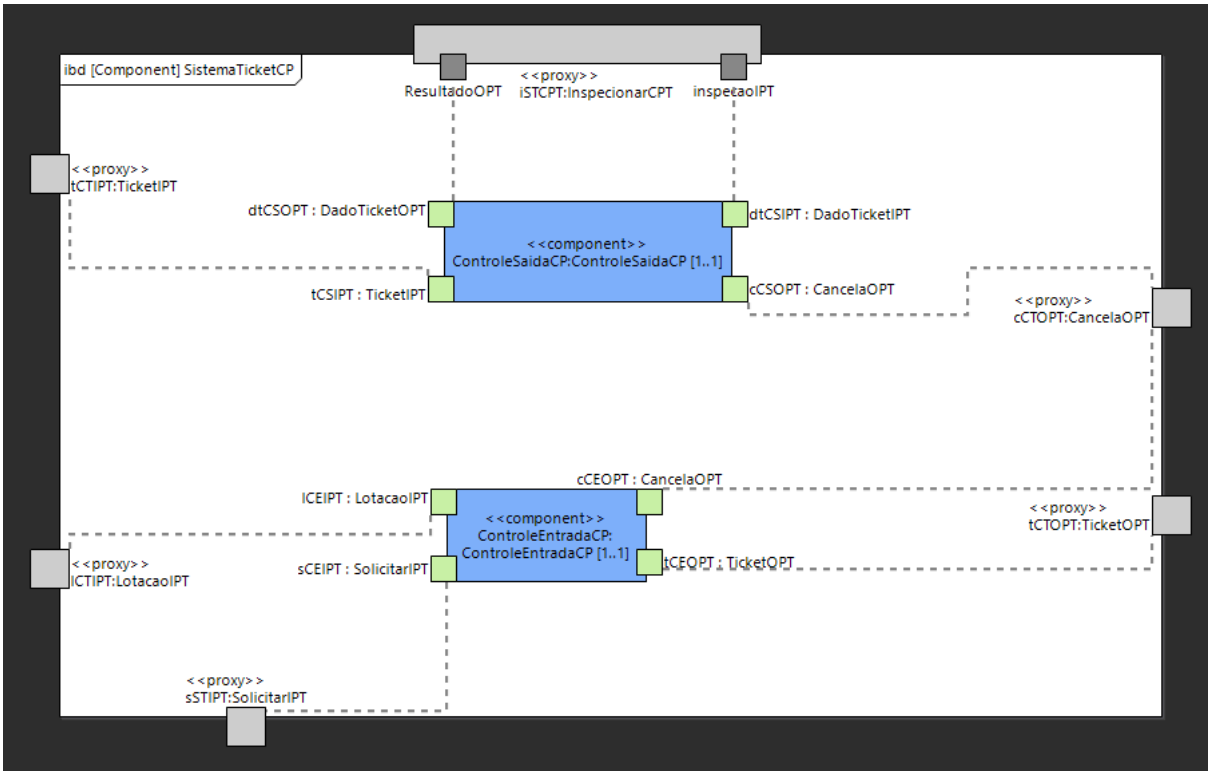

Figura 11. IBD do SistemaTicketCP

A solicitação vinda do *BotaoEntradaCP* é recebida pelo *ControleEntradaCP*, este, analisa a situação da lotação através das informações disponibilizadas pelo *SistemaDeVagaCP*. Caso apresente ao menos uma vaga livre, o *ControleEntradaCP* criará um novo ticket e mandará para a *ImpressoraTicketCP* e *BlackboardTicketCP*, para atualizar a base de dados, possibilitando o pagamento do ticket posteriormente. Ao mesmo tempo, é mandado uma solicitação para acionar a abertura da cancela para o *SistemaCancelaCP*. Com relação ao *ControleSaidaCP,* este, recebe informações de ticket do *EscanerTicketCP* e manda uma solicitação de checagem para o *BlackboardTicketCP* e recebe o dado sobre o ticket que apresenta no sistema de armazenamento, confirmando se o ticket foi pago. Caso tenha sido pago, manda uma solicitação para acionar a abertura da cancela.

Falando agora do núcleo sobre vagas do sistema, existem três componentes principais: *SensorDePresencaCP, InterfaceDeVagasCP* e *SistemaDeVagaCP*. O *SensorDePresencaCP* é responsável por verificar se a vaga está ocupada ou desocupada, mandando ao *SistemaDeVagaCP* todas as informações obtidas. Por sua vez, o *SistemaDeVagaCP* analisa os dados obtidos por todos os sensores e obtém a quantidade de vagas disponíveis para cada tipo de vaga no estacionamento (normal, idoso, gestante e deficiente). Com isso, informa a quantidade de vagas ainda disponíveis no estacionamento para *InterfaceDeVagasCP(*componente de fronteira responsável por transparecer a quantidade de vagas aos usuários fora do estacionamento), ao mesmo tempo que notifica o *SistemaTicketCP* se o estacionamento está lotado (impossibilita a entrada de veículos) ou vago (permite a entrada de veículos).

O IBD de *SistemaDeVagaCP* é apresentado na figura 12. Nele há os componentes *ControleFalhasPreCP* e *ControleVagasCP*, responsáveis por garantir a integridade dos dados dos sensores de forma escalável e com controle de falhas.

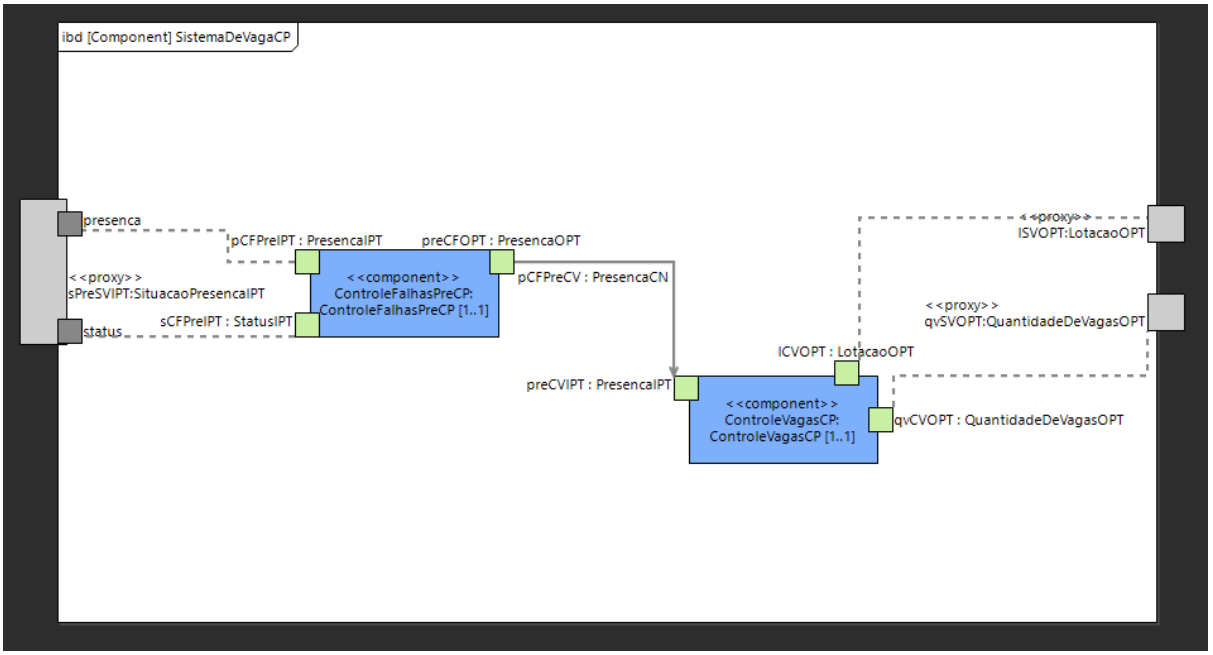

Figura 12. IBD do SistemaDeVagaCP

Semelhante ao controle de falhas apresentado na figura 10, sobre o controle acerca dos sensores de passagem, há também um controle de falhas específico para os sensores de presença nas vagas. O componente *ControleFalhasPreCP* recebe os dados do *SensorPresencaCP*, com o dado de presença e o status do sensor, caso o sensor apresente um bom funcionamento de acordo com seu status, as informações sobre presença são transferidas para o componente *ControleVagasCP*. Este, processa todas as informações de presença que recebe e calcula a quantidade de vagas que estão disponíveis dentro do estacionamento, separando por tipo de vaga. Essa informação é disponibilizada a todo momento a *InterfaceDeVagasCP*, além disso, notifica o *SistemaTicketCP* se o estacionamento apresenta a situação de lotação (caso a soma de vagas de todos os tipos disponíveis seja igual a zero).

Por fim, o núcleo referente ao pagamento do ticket. Tem como base quatro componentes: *VerificadorTicketPGCP, ControleValorCP, MaquinaPagamentoCP* e *SistemaPagamentoCP*. Foi pensado em utilizar o estilo Pipe-Filter, para manipular a forma como os componentes neste meio se comportam. Cada componente se liga ao próximo, adicionando mais uma informação à cadeia de informações transmitidas. Inicialmente o *VerificadorTicketPGCP* recebe a informação de ticket, disponibilizado pelo usuário, e manda essa informação para *ControleValorCP* que irá fazer o cálculo do valor a ser pago por aquele ticket, levando em consideração o horário de entrada, horário atual e conceitos da regra de negócio. Ao finalizar o cálculo, substitui o campo *valor* no tipo ticket e manda a informação para a

MaquinaPagamentoCP. Ao iniciar seu processamento, o usuário será informado sobre o valor a ser pago e, posteriormente, adicionará a forma de pagamento e informações de pagamento. Ao obter os dados, *MaquinaPagamentoCP* encaminha para *SistemaPagamentoCP*, este confirma o pagamento. Caso o pagamento seja bem sucedido, atualiza a informação do ticket pago para *BlackboardTicketCP,* caso haja erro, notifica o erro para *MaquinaPagamentoCP*.

5. Visão Comportamental

A visão comportamental é a última etapa do processo de modelagem da arquitetura do sistema. Para construir essa etapa vão ser criados três tipos de elementos possíveis (atividade, ação e *constraint*) com o objetivo de apresentar o comportamento de componentes do sistema, apresentados na figura 8.

O diagrama com todas as atividades, ações e *constraints* do sistema é apresentado na figura 13. Para a construção de cada atividade, foi necessário analisar quais as principais ações feitas pelos componentes de forma que resultasse nas funcionalidades especificadas para o projeto. Com isso, foram criadas seis atividades.

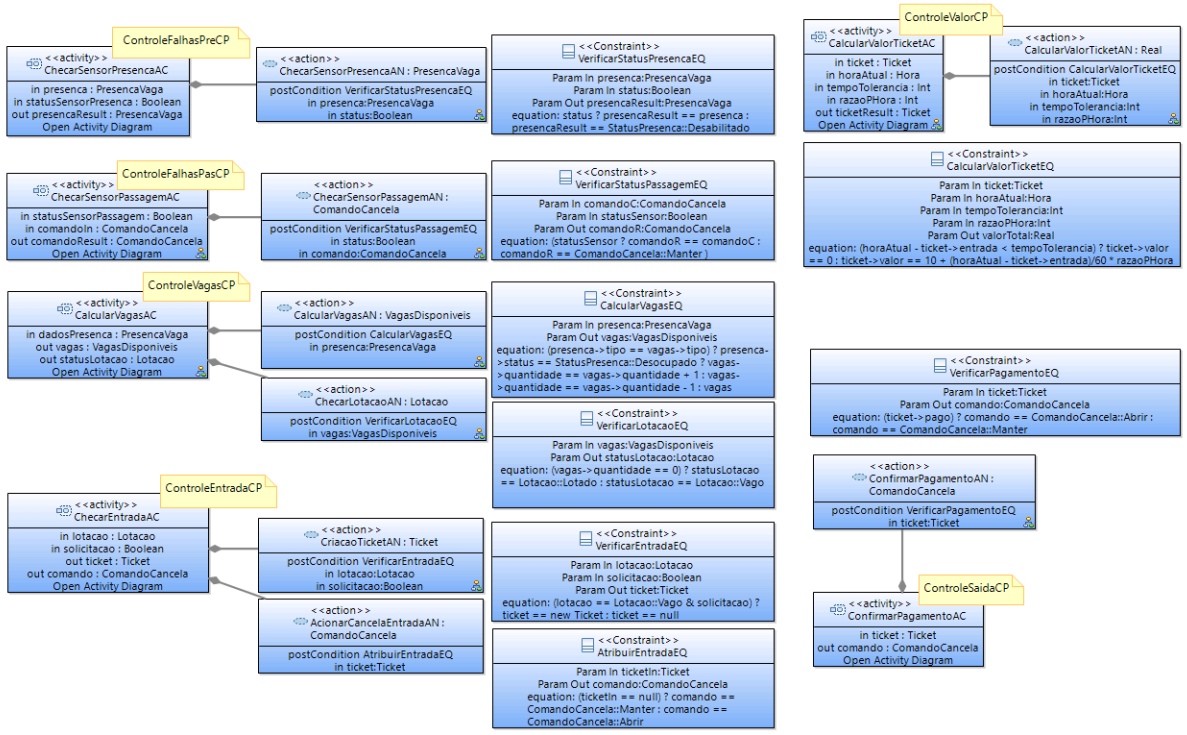

Figura 13. BDD comportamental dos componentes

5.1. Checar sensor de presença

Tomando como foco o controle de vagas, foi modelada a atividade *ChecarSensorPresencaAC.* Essa atividade está relacionada com o processamento dos dados recebidos pelo sensor de presença nas vagas do estacionamento e, a partir disso, verificar se o status do sensor está válido para ser considerado no sistema. Para isso, a ação *ChecarSensorPresencaAN* foi modelada com sua respectiva *constraint VerificarStatusPresencaEQ*. Esta ação usa, como pós-condição, a equação definida na *constraint* que busca garantir a integridade dos dados recebidos, caso seja comprovada sua integridade, repassa os dados de presença, porém, caso seja comprovado o contrário, modifica o status do sensor para desabilitado. O diagrama de atividade é apresentado na figura 14 e o diagrama paramétrico na figura 15.

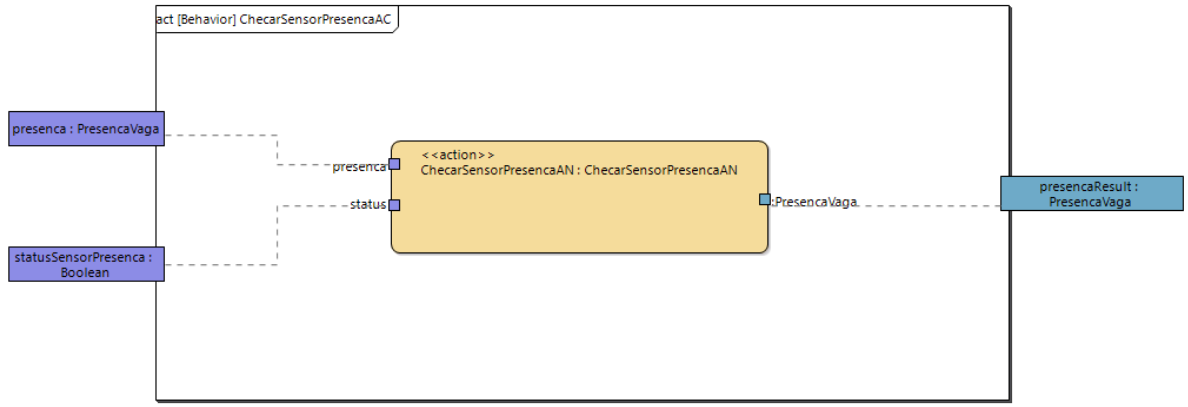

Figura 14. Diagrama de atividade de ChecarSensorPresencaAC

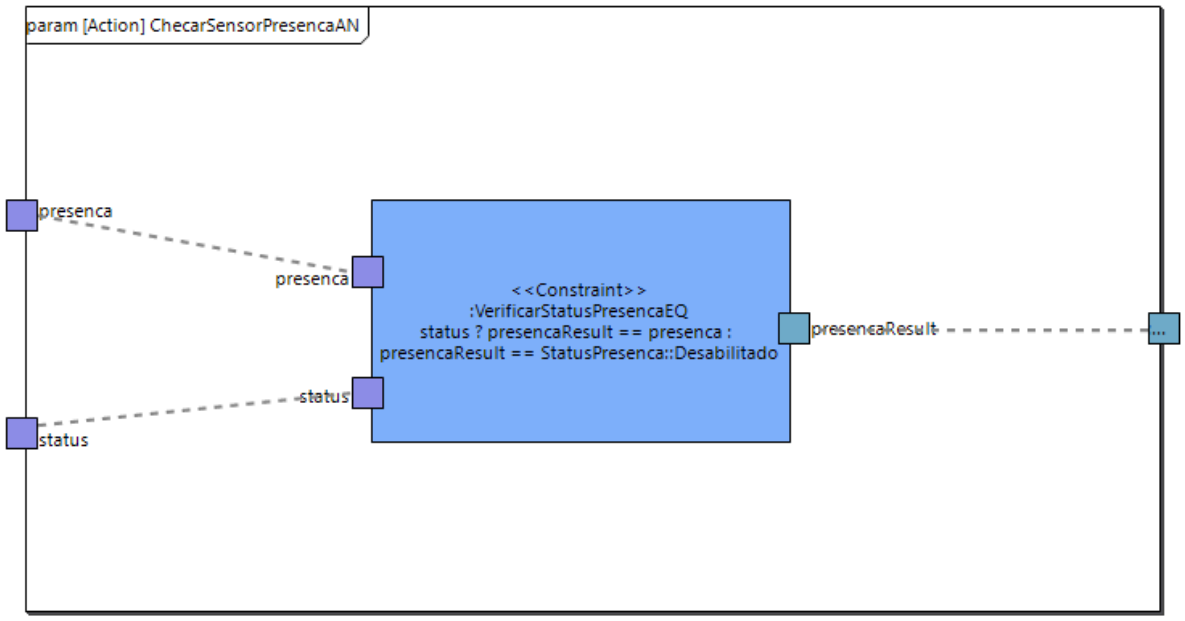

Figura 15. Diagrama paramétrico de ChecarSensorPresencaAN

5.2. Checar sensor de passagem

Para o controle automático da cancela a partir do sensor de passagem, foi modelada a atividade *ChecarSensorPassagemAC* para o processo de checagem do sensor, que faz a checagem do status, avaliando se as informações mandadas por ele são relevantes. Para isso, foi criada a ação *ChecarSensorPassagemAN* que faz uso da *constraint* nomeada de *VerificarStatusPassagemEQ*. Esta ação usa, como pós-condição, a equação definida na *constraint* que busca garantir a integridade dos dados recebidos, semelhante a atividade apresentada anteriormente. Caso seja comprovada sua integridade, repassa o comando da cancela, porém, caso apresente uma falha nos seus status, modifica o comando mandado para 'Manter' o estado da cancela.

O diagrama de atividade é apresentado na figura 16 e o diagrama paramétrico na figura 17.

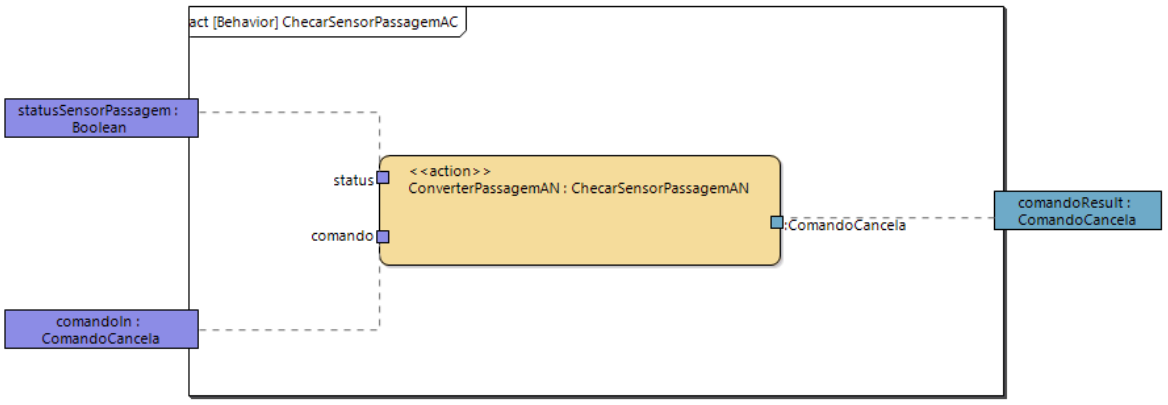

Figura 16. Diagrama de atividade de ChecarSensorPassagemAC

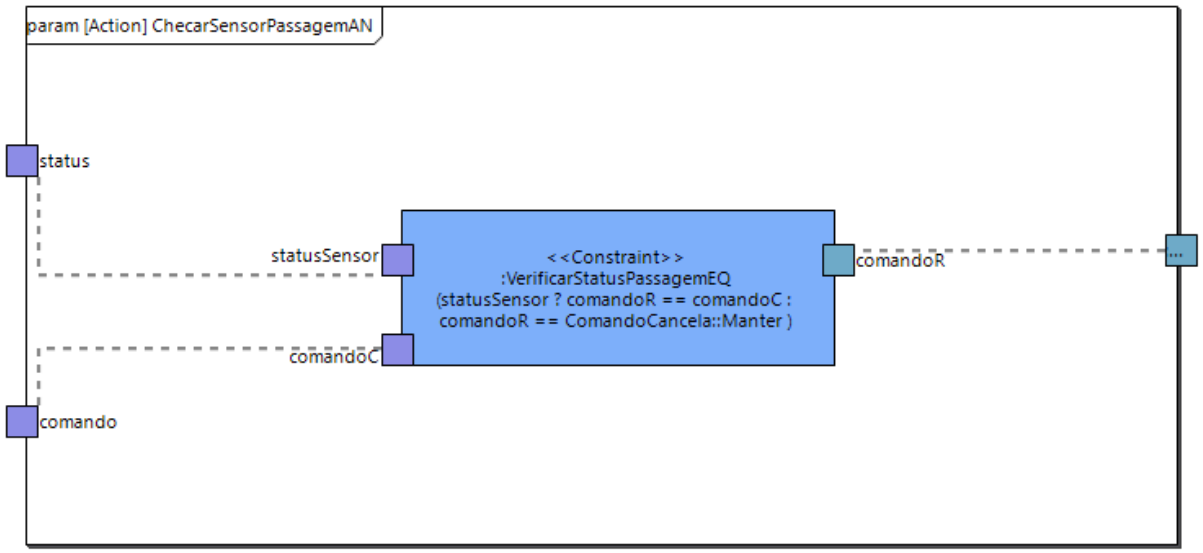

Figura 17. Diagrama paramétrico de ChecarSensorPassagemAN

5.3. Calcular vagas

No controle de quantidade de vagas, foi modelada a atividade *CalcularVagasAC* para o processo de calcular a quantidade de vagas por tipo e verificar o status de lotação do estacionamento. Para isso, foi criada a ação *CalcularVagasAN* que faz uso da *constraint CalcularVagasEQ*, responsável por fazer a contagem de vagas disponíveis no ambiente. Esta ação usa a equação definida na *constraint* que busca incrementar e decrementar a quantidade de vaga em cada tipo disponível, do qual além de disponibilizar essa informação, repassa o resultado para outra ação. Essa outra ação foi nomeada de *ChecarLotacaoAN* que foi modelada com a *constraint VerificarLotacaoEQ*. O objetivo desta, é a partir dos dados recebidos, verificar se o ambiente está considerado como lotado para segurança, caso sim, transmite que está "lotado", do contrário, transmite que está "vago".

O diagrama de atividade é apresentado na figura 18 e os diagramas paramétricos nas figuras 19 e 20.

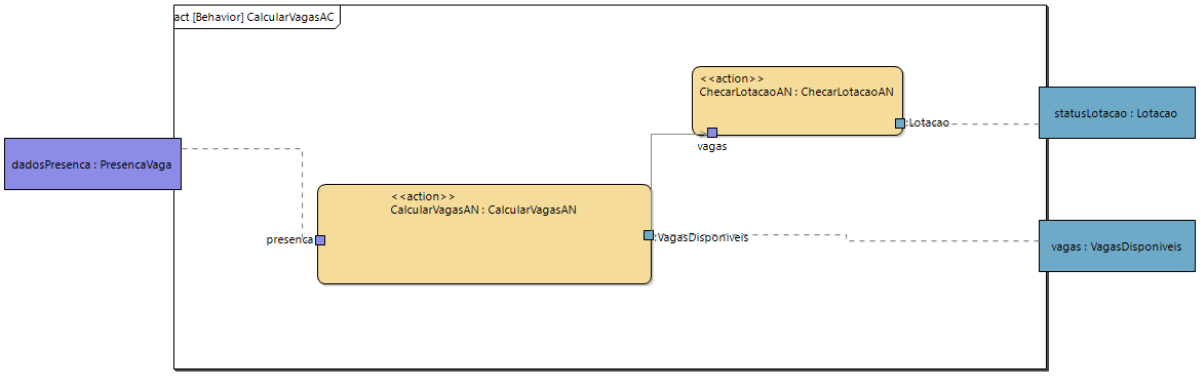

Figura 18. Diagrama de atividade de CalcularVagasAC

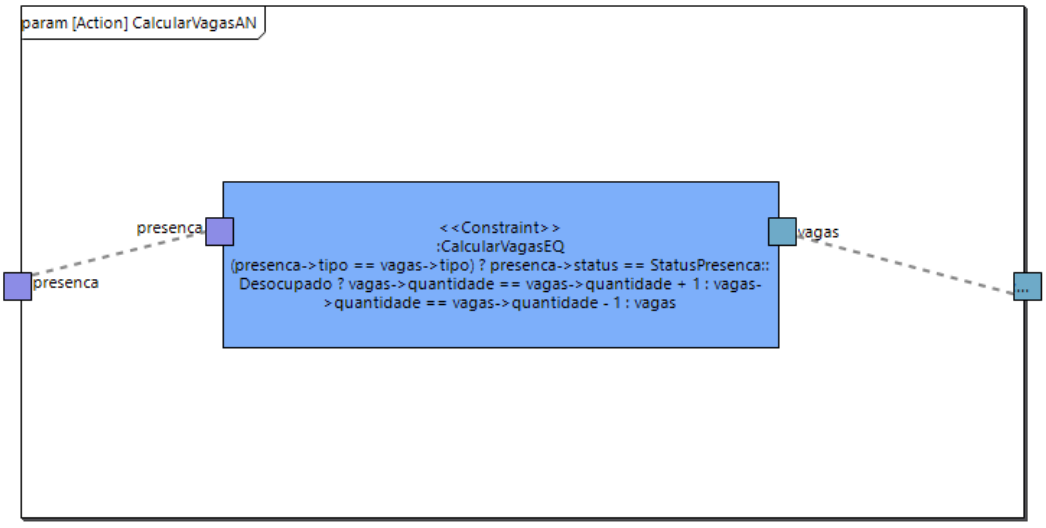

Figura 19. Diagrama paramétrico de CalcularVagasAN

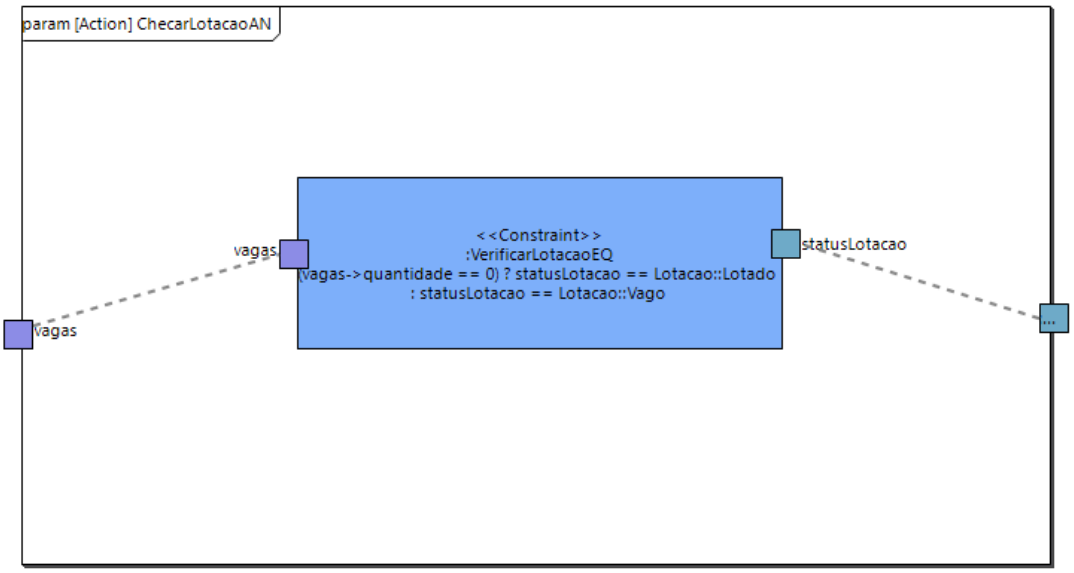

Figura 20. Diagrama paramétrico de ChecarLotacaoAN

5.4. Checar entrada

No controle de entrada no estacionamento, foi modelada a atividade *ChecarEntradaAC* para o processo de fazer a verificação de lotação e criar um novo ticket para fazer a solicitação de abertura da cancela. Para isso, foi criada a ação *CriacaoTicketAN* que faz uso da *constraint VerificarEntradaEQ*, responsável por criar um novo ticket caso haja vaga disponível no estacionamento. Esta ação usa, como pós-condição, a equação definida na *constraint* que busca garantir a entrada do carro apenas se houver uma vaga disponível, chamando uma outra ação. Essa outra ação foi nomeada de *AcionarCancelaEntradaAN* que foi modelada com a *constraint AtribuirEntradaEQ*. O objetivo desta, é a partir do dado recebido de ticket, produzir um dado de acionamento da cancela. Para isso, verifica a integridade do dado do ticket, caso tenha o dado normal, produz o comando de abertura da cancela, do contrário, produz o dado de 'manter' o estado da cancela.

O diagrama de atividade é apresentado na figura 21 e os diagramas paramétricos nas figuras 22 e 23.

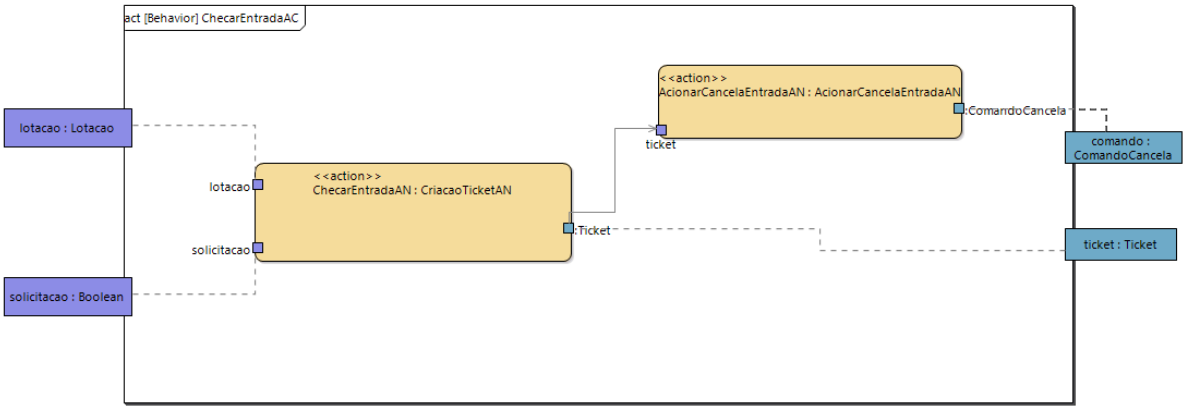

Figura 21. Diagrama de atividade de ChecarEntradaAC

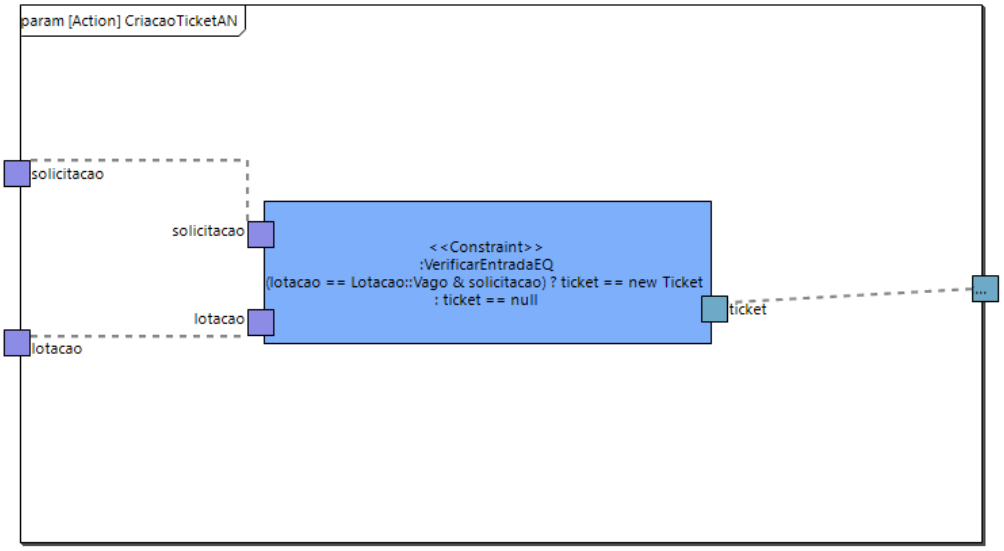

Figura 22. Diagrama paramétrico de CriacaoTicketAN

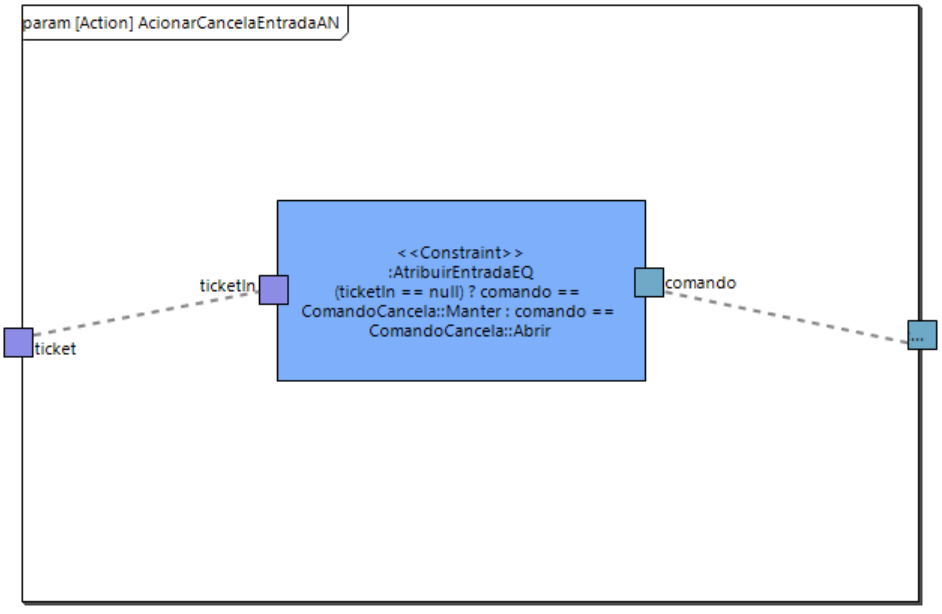

Figura 23. Diagrama paramétrico de AcionarCancelaEntradaAN

5.5. Confirmar pagamento

Para o controle de saída do estacionamento, foi modelada a atividade *ConfirmarPagamentoAC* para o processo de checagem de pagamento do ticket. Para isso, foi criada a ação *ConfirmarPagamentoAN* que faz uso da *constraint VerificarPagamentoEQ*. Esta ação usa, como pós-condição, a equação definida na *constraint* que busca garantir a o pagamento do ticket, verificando o atributo de referência. Caso seja comprovado seu pagamento, cria o comando de abrir a cancela, do contrário, cria o comando de manter o estado da cancela.

O diagrama de atividade é apresentado na figura 24 e o diagrama paramétrico na figura 25.

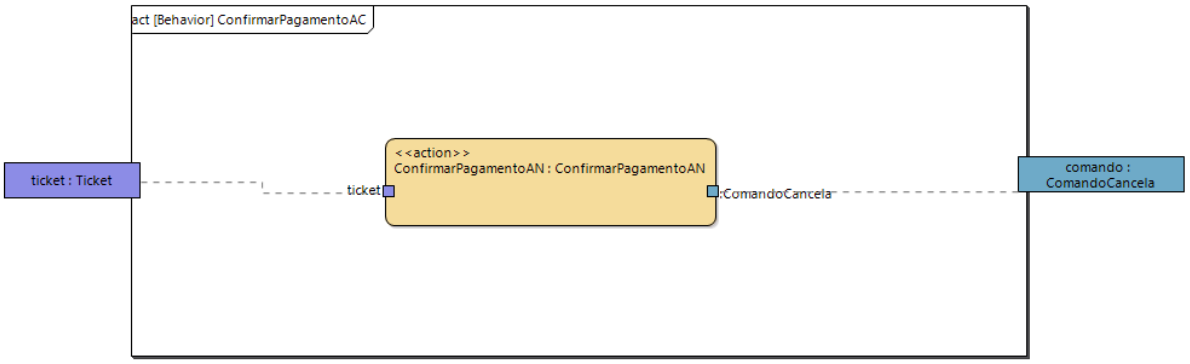

Figura 24. Diagrama de atividade de ConfirmarPagamentoAC

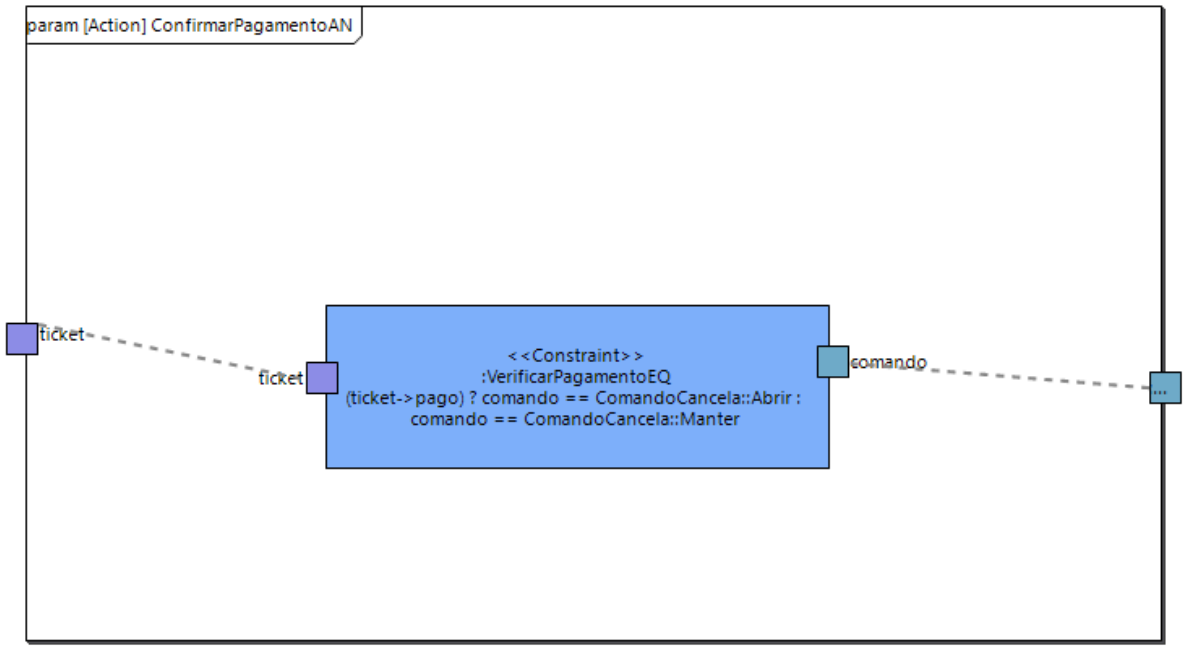

Figura 25. Diagrama paramétrico de ConfirmarPagamentoAN

5.6. Calcular valor de Ticket

Para o calcular o valor a ser pago pelo ticket, foi modelada a atividade *CalcularValorTicketAC*. Para isso, foi criada a ação *CalcularValorTicketAN* que faz uso da *constraint CalcularValorTicketEQ*. A atividade recebe de entrada quatro valores que repassa para a ação, são eles: ticket, horaAtual (horário atual para fazer o cálculo do tempo de estacionamento), tempoTolerancia (quantidade de minutos para não pagar o estacionamento) e razaoPHora (número base para aplicar a razão do preço por hora). A ação, por sua vez, faz uso da equação definida na *constraint* que calcula o valor a ser pago pelo usuário do ticket. A equação e campos apresentados foram pensados para possibilitar o estacionamento à ter mais de um tipo possível de ticket, proporcionando uma versatilidade maior para momentos diferentes.

O diagrama de atividade é apresentado na figura 26 e o diagrama paramétrico na figura 27.

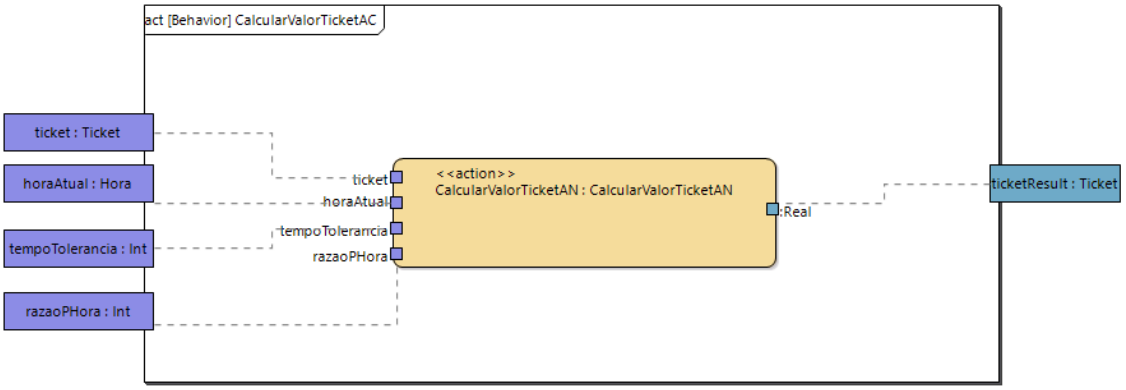

Figura 26. Diagrama de atividade de CalcularValorTicketAC

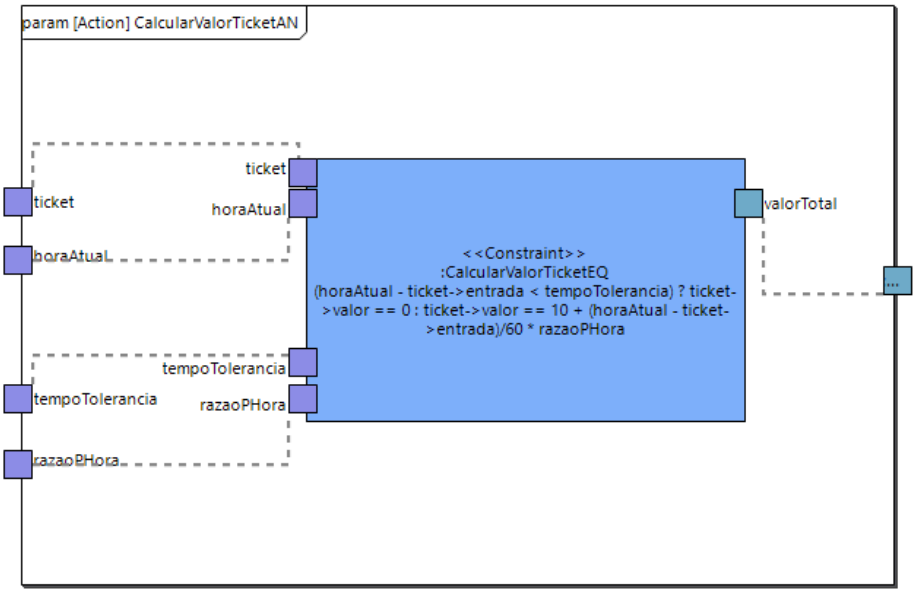

Figura 27. Diagrama paramétrico de CalcularValorTicketAN