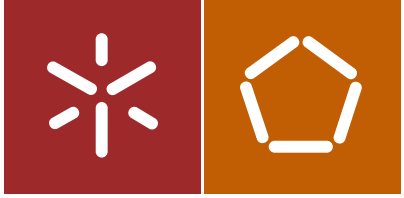
Universidade do Minho
Escola de Engenharia

Francisco Alberto de Castro e Dias

Gerador de código para uma Framework C++ a partir de diagramas de estado UML

Gerador de código para uma Framework C++ a partir de diagramas de estado UML
Francisco Alberto de Castro e Dias
UMinho | 2016

dezembro de 2016



Universidade do Minho
Escola de Engenharia

Francisco Alberto de Castro e Dias

Gerador de código para uma Framework C++ a partir de diagramas de estado UML

Dissertação de Mestrado
Ciclo de Estudos Integrados Conducentes ao Grau de
Mestre em Engenharia Eletrónica Industrial e de Computadores

Trabalho efetuado sob a orientação do
Professor Doutor Sérgio Lopes

dezembro de 2016

DECLARAÇÃO

Nome: Francisco Alberto de Castro e Dias

Endereço eletrónico: Dias.FranciscoA@gmail.com Telefone: 915778490

Bilhete de Identidade/Cartão do Cidadão: 12850217

Título da dissertação: Gerador de código para uma Framework C++ a partir de diagramas de estado UML

Orientador:

Professor Doutor Sérgio Lopes

Ano de conclusão: 2016

Mestrado Integrado em Engenharia Eletrónica Industrial e de Computadores

Universidade do Minho, _____/_____/_____

Assinatura:

Resumo

No auge da era informática e dada a demanda que existe por parte da sociedade de uma resposta rápida para a implementação de sistemas complexos, a geração de código de forma automática, é uma mais valia no processo de desenvolvimento de software. A linguagem de modelação UML vem, por sua vez, estabelecer uma norma no desenvolvimento de software e permitir ao desenvolvedor visualizar o produto do seu trabalho recorrendo a diagramas estruturais e comportamentais.

O objetivo desta dissertação tem por base a junção destes dois conceitos, geração de código a partir de diagramas de UML mais especificamente diagramas de estado. Embora existam soluções comerciais que conseguem gerar código a partir de modelos UML este para além de volumoso e mais complexo, é um código que não tem em consideração a sua manutenção e atualização de forma manual, sendo pouco otimizado e de difícil leitura e edição. Este torna-se então o segundo objetivo deste projeto, a geração de código que instancia uma *framework* de máquinas de estado finitas. Esta particularidade, vai tornar o código fonte gerado mais simples: quer em complexidade de leitura; quer em quantidade de linhas de código, tornando o processo de geração muito mais eficaz e rápido.

É desenvolvida uma ferramenta para realizar o processo de geração de código. Esta recebe como entrada um ficheiro XMI que é o formato *standard* para partilha de dados de diagramas UML e converte-o numa estrutura de objetos, posteriormente escrita para um ficheiro de texto sobre a forma de código, no formato C++. O código resultante é compilado com a *framework* de máquinas de estado, criando um ficheiro binário completo pronto a executar.

O resultado da aplicação prática da ferramenta permite comprovar que o código gerado, é mais fácil de ler e editar, dada a camada de abstração que a *framework* oferece para criar e executar uma máquina de estados. A framework foi concebida para ser utilizada por programadores, e isso permite reduzir muito a quantidade de código gerado em comparação com outras soluções comerciais e tornar a geração de código mais rápida. O formato escolhido para a ferramenta é um ficheiro executável que pode ser facilmente incorporado em qualquer *software* UML, independentemente da existência de um mecanismo de extensão ou das suas caraterísticas.

Palavras chave: UML, Geração de Código, Framework FSM, XMI, Máquinas de Estado

Abstract

At the height of the computer era and given the society's demand for a rapid response when implementing complex systems, automatic code generation, becomes an important asset in the software development process. The UML modeling language, establish a new standard on the software development and allow the developer to visualize the result of his work using structural and behavioral diagrams.

The objective of this dissertation is based on the combination of these two concepts, code generation from UML diagrams, specifically state diagrams. While there are commercial solutions that can generate code from UML models, the code generated by these tools, in addiction to being very bulky and complex does not take into account the manual maintenance and updating process of the code, resulting in a less optimized structure which is difficult to read and edit . This, there for, becomes the second objective of this project, code generated is to instantiate a finite state machines' framework. This feature will result in a simple, well structured source code regarding reading complexity and the number of lines of code needed, making the generation process much more efficient and fast.

A tool to make the code generation process was developed. Said tool receives as input a XMI file format, which is the standard format for sharing UML diagrams' data, and converts it into a structure of objects that later writes to a text file, following the C++ code format. The resulting code is then compiled with the libraries of the state machine framework, resulting in a complete binary file ready to run.

The results of the practical application of the tool allowed to prove that the generated code was easier to read and edit, given the abstraction layer provided by the framework. The framework was designed to be used by programmers, and this contributed to greatly reduced the amount of code generated compared to other commercial solutions, making the generation process much faster.

The format chosen for the tool was an executable file that can be easily embedded in any UML software, regardless of the existence of an extension mechanism or its characteristics.

KEYWORDS: UML, Code Generation, Framework FSM, XMI, State Machines

Índice Geral

Índice de Figuras

1. Introdução

Neste capítulo é apresentada uma contextualização teórica dos objetivos e contribuições desta dissertação, assim como a motivação para o desenvolvimento do projeto e uma planificação dos restantes capítulos do documento.

1.1 Motivação

Com o aumento da complexidade dos sistemas informáticos, torna-se necessária a sua planificação, através de diagramas. Assim surge, no ano de 1996, a UML (Unified Modeling Language). Esta linguagem de modelação tem como objetivo ajudar o desenvolvedor na organização do seu raciocínio e na comunicação do produto do seu trabalho através de diagramas estruturais e comportamentais. Entre outros, os diagramas de estados UML são responsáveis pela representação comportamental do programa informático [1]. Na atualidade, a geração de código a partir deste tipo de diagramas continua a ser um desafio, dado às limitações de formalização dos elementos de componente dinâmica da linguagem UML. As demais ferramentas de geração de código *open-source* e comerciais existentes utilizam, na maioria, modelos de geração de código que não foram concebidos para serem utilizados por programadores, mas sim pelo próprio motor de geração de código. Sendo o código gerado por estas, código nativo que corre diretamente em cima de um sistema operativo, este torna-se facilmente extenso e de difícil leitura.

Ainda que hierarquizado este código fonte gerado é de difícil manutenção pois a forma como é implementado obriga a alteração de código em diferentes níveis do modelo de classes. Contudo, do ponto de vista de um processo de desenvolvimento puramente assente em modelos, i.e., Model-driven Engineering (MDE), não há qualquer necessidade ou interesse em que o código gerado seja de fácil leitura ou manutenção uma vez que esta é sempre efetuada ao nível dos diagramas UML e nunca ao nível do código. Contudo, na prática, mesmo que o desenvolvimento de um software seja puramente assente em modelos, a manutenção deste é efetuada quase sempre ao nível do código.

Neste âmbito, o projeto visa apresentar uma alternativa no que diz respeito ao código fonte gerado. Contrariamente ao que acontece com a maior parte dos geradores de código, este utiliza como base uma *framework* de máquinas de estados concebida para ser utilizada por programadores [2]. Esta framework permite simplificar e minimizar a quantidade do código necessário para implementar máquinas de estado finitas. Estas características facilitam o desenvolvimento de um

gerador de código, que geralmente é uma ferramenta complexa, e contribuem para o aumento da velocidade de geração e para a simplicidade de manutenção do mesmo.

1.2 Objetivos e contribuições

São analisadas ferramentas comerciais que permitem geração de código a partir de diagramas UML, tentando perceber quais são as suas abordagens ao âmbito de criação de código. O *output* destas ferramentas é uma mais valia para perceber o que pode ser melhorado, por forma a que deste projeto resulte uma ferramenta compreensível de mais fácil utilização.

É avaliada a geração de código para a Framework FSM [2]. Esta avaliação tem diversas componentes incluindo, não só a análise da complexidade do gerador de código, mas também aspetos mais dependentes da *framework* como a eficiência da implementação e a facilidade de leitura do código gerado.

Como objetivo geral, a ferramenta desenvolvida vai possibilitar a geração de código eficiente e de fácil leitura que permite que a manutenção posterior possa ser feita de forma manual, sem ser necessária a utilização de ferramentas adicionais. O código gerado é também mais simples e compacto diminuído subsequentemente o tempo necessário para a geração do mesmo.

A ferramenta utiliza o conjunto de especificações do UML2 na conversão de diagramas de estados para código, tendo em conta as limitações que a *framework* ainda apresenta ao UML. Ainda que a framework não permita a criação de máquinas de estados hierárquicas, no seu desenvolvimento a ferramenta tem em conta esta futura implementação e está preparada para facilmente ser atualizada de forma a evoluir conjuntamente com a *framework*.

Um dos principais pontos do projeto é a criação de uma ferramenta funcional capaz de gerar código C++, a partir de diagramas de estado UML. Por isso, a ferramenta destina-se ao desenvolvimento de software e sistemas intensivos em software, e é facilmente incorporável em qualquer programa *open-source* ou comercial de criação de diagramas UML.

1.3 Estrutura da dissertação

Os restantes capítulos desta dissertação estão organizados da seguinte forma:

No capítulo 2 (estado da arte) é descrita a arquitetura do UML e a sua importância no desenvolvimento de software. É ainda introduzido o conceito de máquinas de estado e são

especificados os seus elementos constituintes. Prossegue-se: uma análise à abordagem de geração de código por parte de ferramentas comerciais; exposição de métodos de geração de código e; por fim, apresentação e descrição do funcionamento da Framework FSM.

No capítulo 3 (conceção da solução) é explicada e argumentada a abordagem à resolução da problemática proposta nesta dissertação (plugin vs. executável). É apresentada uma proposta para utilização de XMI para intercâmbio de dados entre os *softwares* envolvidos, sendo por fim feita a decomposição da conceção da ferramenta a desenvolver em fases (fase de leitura e posterior fase de escrita, vantagens da abordagem adotada).

No capítulo 4 (Implementação da ferramenta de geração de código) são apresentadas as opções tecnológicas utilizadas durante o desenvolvimento da ferramenta, seguindo-se a descrição do processo de implementação subdividido nas fases descritas durante o processo de conceção.

No capítulo 5 (Resultados e teste da ferramenta) é analisado o código gerado pela ferramenta desenvolvida: para uma máquina de estados de um sistema de Login e; para uma máquina de estado de um micro-ondas. Ainda são apresentados alguns procedimentos pós geração do código para que este possa ser corretamente utilizado.

No capítulo 6 (Conclusão e discussão dos resultados obtidos) são: discutidas e propostas considerações relativas à compilação do código da ferramenta; tiradas conclusões face aos objetivos propostos; e apresentadas sugestões para trabalho e implementações futuras.

2 Estado da arte

2.1 Máquinas de estado UML

2.1.1 UML

UML é uma língua de modelação geralmente utilizada na área de desenvolvimento de software que surgiu com o intuito de normalizar a forma como sistemas são representados graficamente. Esta linguagem é proposta e apresentada pela primeira vez em 1994 tendo sido três anos mais tarde adotada como norma pelo OMG, que desde então é responsável pela sua manutenção e atualização.

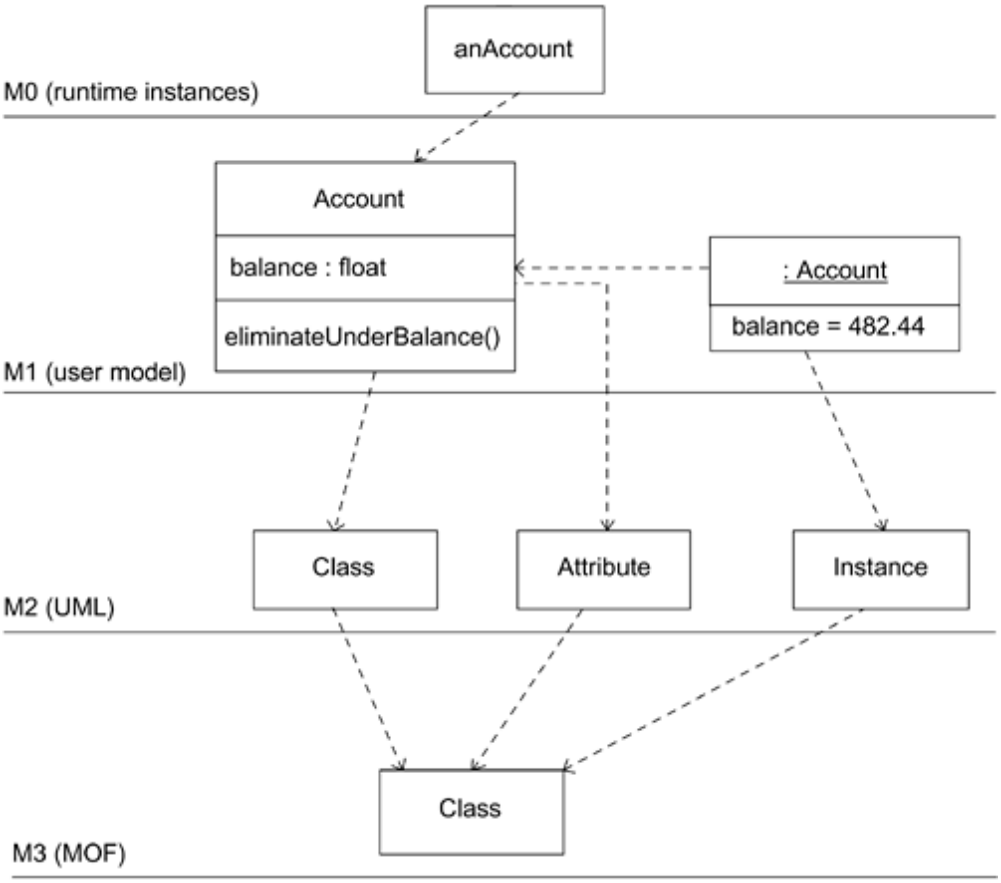


Figura 1 - Arquitetura em camadas da UML

Esta linguagem de modelação tem por base uma arquitetura hierárquica em 4 camadas (M3-M0), apresentada na Figura 1. A camada M3, denominada Meta-Object Facility (MOF), é a mais abstrata de todas e define a linguagem base que permite definir metamodelos OMG, e por isso é

também denominada de meta-metamodelo. A segunda camada (M2) é o metamodelo que neste caso representa o UML e nela são definidos todos os elementos integrantes da linguagem, como por exemplo, Class, Property, State, Transtion, etc. Estes elementos permitem construir o modelo estrutural e comportamental de um sistema/software. Este modelo é a terceira camada (M1) desta arquitetura e contém instâncias das classes do metamodelo (M2) para criar um modelo concreto do sistema a desenvolver. Por fim, a camada concreta desta hierarquia, é a camada M0, na qual se encontram representados os objetos que realmente existem quando o sistema/software está em funcionamento/execução.

Os diagramas UML subdividem-se em duas categorias diferentes: a categoria dos diagramas estruturais e; a categoria dos diagramas comportamentais. Os diagramas estruturais representam o aspeto estático do sistema, ou seja, a estrutura – "o quê". Ao passo que os diagramas comportamentais, representam o aspeto dinâmico do sistema, aquilo que muda no decorrer da execução do mesmo e que consiste na forma como o sistema responde ao longo do tempo – "o como".

Um exemplo de diagramas estruturais é o diagrama de classes. Este é o tipo mais usual de diagramas de UML e define: relações; associações; interfaces e; colaborações entre objetos. Na sua essência, um diagrama de classes apresenta uma visão orientada a objetos do sistema e define as suas caraterísticas.

Contrariamente ao aspeto estático do diagrama de classes existe, por exemplo, os diagramas de máquinas de estados. Estes pertencem á categoria de diagramas comportamentais. Qualquer sistema em tempo real, deve reagir a eventos internos ou externos a si, sendo estes responsáveis por alterações no estado do sistema. Os diagramas de máquina de estados apresentam, assim, a reação orientada a eventos de um dado sistema.

O UML é utilizado por uma vasta gama de ferramentas CASE (Computer-Aid Software Engineering) que permitem criação e manipulação de diagramas, e em alguns casos geração automática de código fonte. Várias destas ferramentas, sobretudo as *open-source*, integram bibliotecas que implementam o metamodelo UML, como é o caso da UML2 [3] para o Eclipse IDE.

A utilização do UML por uma grande variedade de softwares de diferentes fabricantes torna necessária a criação de uma forma fácil de trocar modelos UML e a informação da sua representação gráfica entres diferentes ferramentas. Neste contexto, o OMG cria duas especificações de elevada importância para o UML e quaisquer outros metamodelos baseados em MOF:

- o XMI, acrónimo para XML Metadata Interchange, que é uma norma ISO, que permite a troca de informação relativa a modelos de forma independente da sua representação gráfica;
- o Diagram Definition, que, ao contrário da anterior, permite o intercâmbio de diagramas, nomeadamente da disposição dos seus elementos representativos (nodos e arcos) num espaço bidimensional.

A OCL linguagem de restrição de objeto é uma linguagem de texto precisa que fornece expressões de restrição e consulta de objeto em qualquer modelo MOF ou meta-modelo que não possa ser expressado por notação diagramática.

2.1.2 Máquinas de estado

Uma máquina de estados finitos (MEF) é um modelo de computação matemático que pode ser usado para modelar um grande número de problemas computacionais em áreas como a automação eletrónica, protocolos de comunicação, *parsing* de linguagens, entre outras áreas de engenharia. Uma MEF representa de forma abstrata uma máquina que, num dado momento, se encontra num de um conjunto finito de estados. O estado no qual a máquina se encontra é designado de **estado atual**. A MEF transita para um novo estado assim que um dado **evento** (trigger) ou **condição** desencadeie uma **transição** (do atual para o novo estado).

O conceito de máquina de estados é mais ubíquo do que se possa imaginar e embora seja usado para a conceção de software, esta não é a sua única aplicação. O modelo de máquina de estados pode ser utilizado para descrever qualquer comportamento que responda a eventos, desde um simples "acender de um candeeiro" até a uma mais complexa "inteligência artificial" de um objeto num jogo de computador. De forma mais geral, serve para implementar sistemas reativos.

Segundo a versão UML2, as máquinas de estado apresentam a estrutura representada na Figura 2 por um diagrama MOF (camada M2). Na figura é possível identificar os vários tipos de elementos que constituem uma máquina de estados.

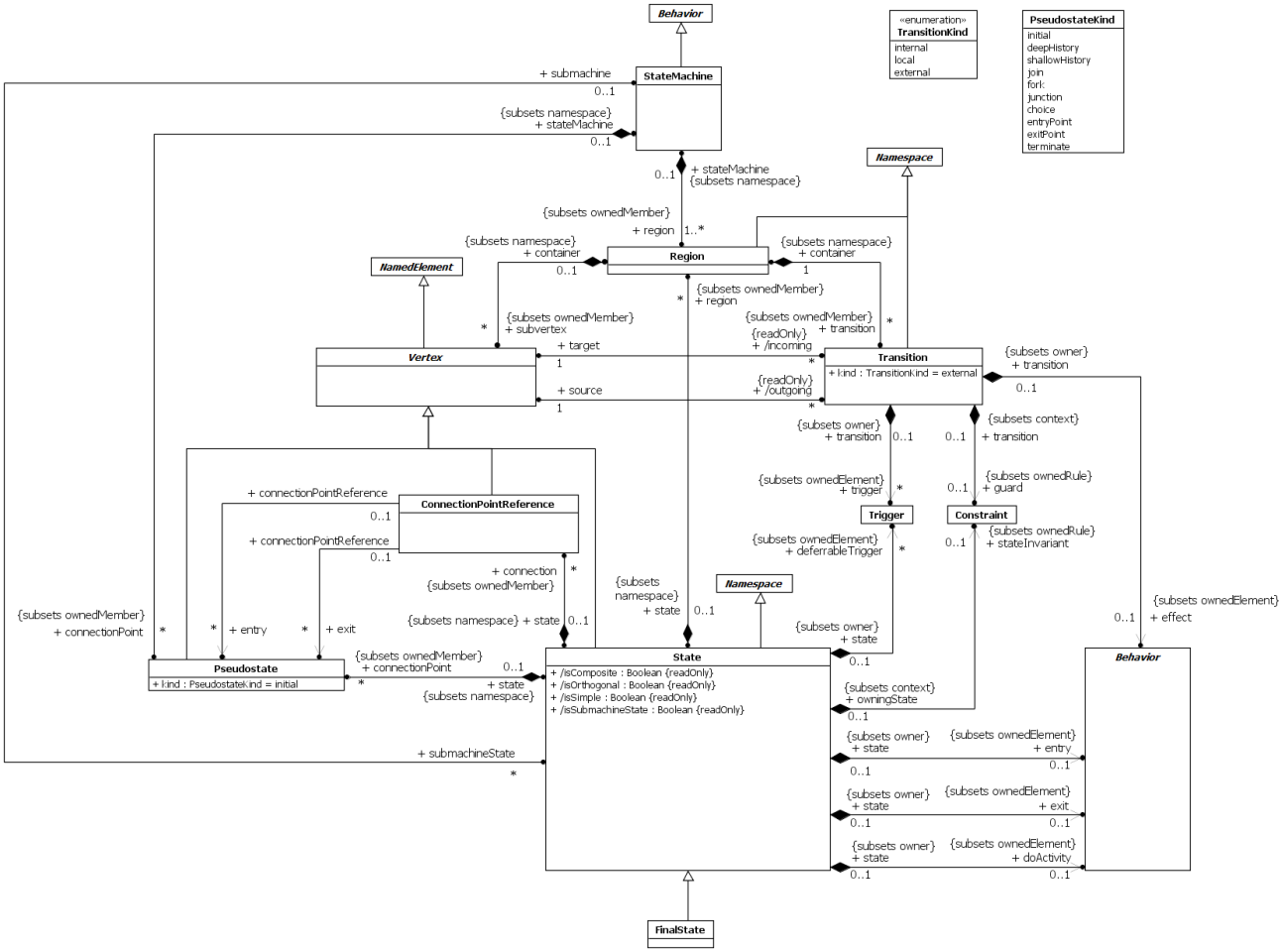


Figura 2 - Diagrama de classes representativo do modelo de máquina de estados

Os principais elementos são: *StateMachine* (**máquina de estados**), *State* (**estado**), *Transition* (**transição**), *Behavior* (**comportamento/ação**), *Trigger* (**evento**) e *Condition* (**condição**) [1]. Estes são seguidamente apresentados de uma forma mais detalhada. O elemento *Region,* embora não sendo um componente visual no modelo UML, está presente no seu meta-modelo e serve para delimitar máquinas de estado (uma máquina de estados está contida sempre numa Region) e modelar máquinas de estado hierárquicas com processamento paralelo, quer ao nível de topo ou ao nível de estados compostos.

2.1.3 Estado

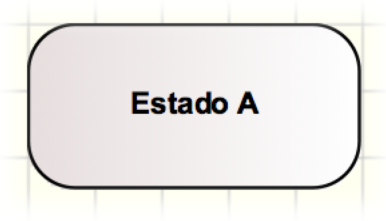


Figura 3 - Representação de estado em UML

Representado na Figura 3, um estado, é o elemento base da máquina de estados-finitos. No limite de simplicidade uma máquina de estados pode ser definida por apenas um estado. Nestes casos a máquinas de estados é denominada de combinatória, pois como o estado é constante, a saída depende apenas da entrada.

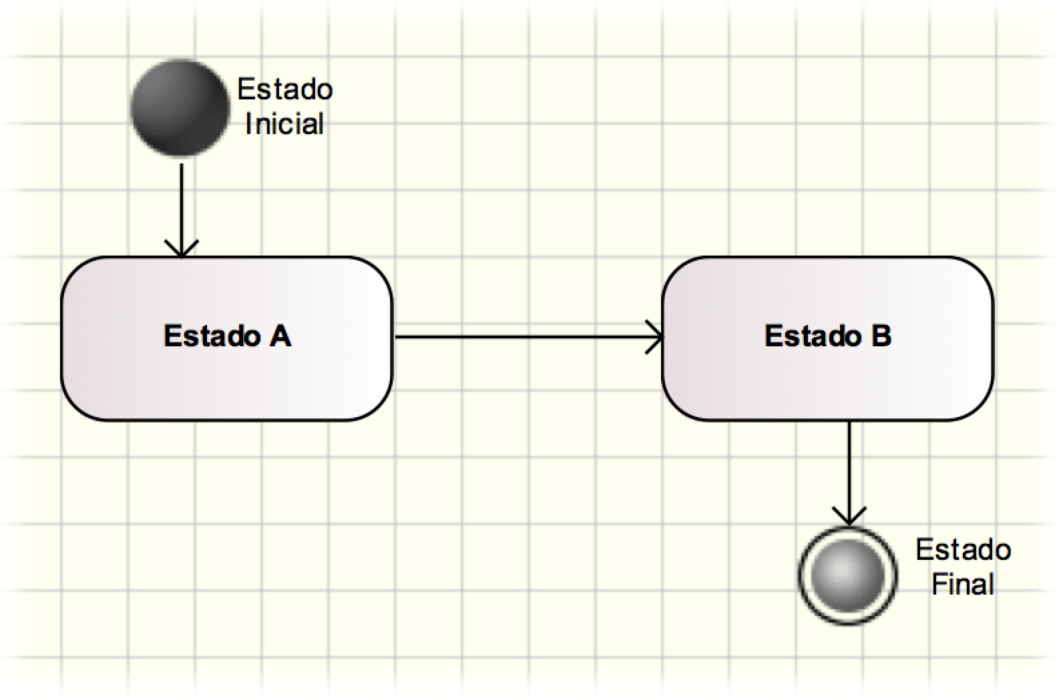


Figura 4 - Exemplo de máquina de estados para demonstrar transições e pseudo-estados

Existem dois tipos de "estados" mais específicos. Um deles é chamado de "*estado inicial"* e é obrigatório em qualquer máquina de estados. Este estado, não é um estado na realidade (não possui qualquer ação de entrada, saída ou permanência, e tem associada uma transição imediata e obrigatória, ou seja, sem evento), mas sim um apontador para um estado (o inicial), e como tal é denominado de *pseudo-estado*. O outro é o "*estado final*", que ao contrário dos convencionais não permite transições de saída. Por isso, é um estado de repouso que termina o funcionamento da máquina de estados. À semelhança do "estado inicial", este não possui quaisquer ações nem

transições. Estas restrições não são visíveis no digrama da Figura 4, mas são expressos por restrições OCL.

2.1.4 Transição

Apresentada na Figura 4, a transição é um elemento associado aos estados como chegada ou partida. A sua existência é essencial para o funcionamento de uma máquina de estados, por mais simples que esta seja. É sempre pelo menos necessário que exista uma transição para o estado

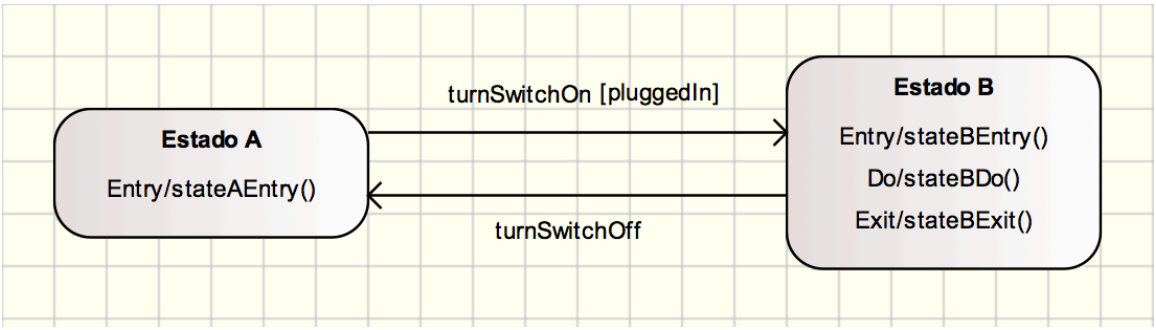


Figura 5 - Exemplo de máquina com presença de ações, condições e eventos

inicial. Qualquer transição é relacional, ou seja, relaciona sempre dois estados, um de partida e um de chegada, pelo que são unidirecionais. Tendo como exemplo a Figura 4, a transição muda a MEF do "*Estado A*" para o "*Estado B*" e nunca no sentido oposto. Uma transição não implica, contudo, uma mudança de estado, uma vez que o estado de partida e o estado de chegada podem ser o mesmo.

2.1.5 Evento

Eventos (ver a representação na Figura 5) são componentes que se associam às transições. Estes componentes são representativos de estímulos exteriores à máquina de estados. São os eventos que alimentam as entradas da máquina de estados e provocam a sua reação, sendo responsáveis por desencadear as transições entre estados. Cada transição pode possuir um ou mais eventos que a desencadeiam e para que a transição seja acionada basta que um dos seus eventos associados ocorra. Os eventos de uma máquina de estados podem ser equiparados aos mecanismos sensoriais de um ser humano, e podem ir desde uma simples comutação de um interruptor, passando por informação de sensores (ex.: proximidade, calor, fim-de-curso), até saídas de outras máquinas de estado.

2.1.6 Ação

A ação é um elemento associado aos estados e às transições, que altera as saídas, ou realiza o trabalho computacional, da MEF. Uma máquina que não possua quaisquer ações não realiza qualquer tarefa para além da permuta silenciosa entre estados. Como é possível observar na Figura 5, um estado pode ter 3 ações a ele associado, de três tipos diferentes. Os vários tipos de ações diferem no momento em que são executadas, desta forma existe: a ação de permanência (do), a ação de entrada (entry) e a ação de saída (exit).

A "*ação de permanência*" é uma ação que é executada em paralelo com outras ações de um estado composto ou máquina de estados interna (*submachine*), depois da ação de entrada e até terminar ou até à saída do estado (abortada antes da ação de saída). Esta ação não resulta de qualquer evento exterior e pode ser usada para realizar tarefas que devem ocorrer de forma independente dos estímulos.

A "*ação de entrada*", ao contrário da anterior, é executada sequencialmente a eventuais ações de um estado composto ou máquina de estados interna, à entrada no estado. Contrariamente à anterior, esta ação, é reativa uma vez que ocorre como resposta direta ao evento que provoca a transição.

A "*ação de saída*", é semelhante à anterior, mas executada assim que ocorre uma transição para fora do estado.

2.1.7 Condição

A condição também denominada "*condição de guarda*", é um elemento das transições e define condições que têm que se verificar para que a transição seja desencadeada. Uma vez que é um elemento opcional, por defeito, a condição é sempre válida. Contudo, se existir pode impedir que uma determinada transição ocorra independentemente de o evento que a desencadeia ter sido verificado.

Na Figura 5 é possível verificar a existência de uma condição na transição do "*Estado A*" para o "*Estado B*". Esta condição (*pluggedIn*) verifica, neste caso, se o sistema está ligado à corrente. Caso não se verifique a condição a transição entre estados não ocorre e o estado atual mantém-se inalterado.

No exemplo da Figura 5 partindo do princípio que o estado inicial é o "*Estado A*" podemos verificar a seguinte sequência de acontecimentos: É executada a ação de entrada do "*Estado A*" (stateAEntry) e a máquina fica à espera de novo evento. Aquando da ocorrência do evento ligar interruptor (*turnSwitchOn*), desencadeia-se a transição para o "*Estado B*". Como não há ação de saída no "Estado A" e a transição também não tem ação, é executada a ação de entrada no novo estado (stateBEntry). Terminando a fase reativa da mudança de estado, entra-se na fase passiva que é caracterizada pela execução da ação de permanência (do: stateBDo). Esta ação é executada até que surja o evento desligar interruptor (*turnSwitchOff*). Quando esse evento ocorre, a ação de permanência cessa e inicia-se a ação de saída do "*Estado B*" (exit: stateBExit), repetindo-se o ciclo.

2.2 Geração de código a partir de diagramas de estado

Existe no mercado soluções para geração de código fonte a partir de diagramas de estado UML. Esta secção analisa o código gerado por ferramentas de apoio ao desenvolvimento de software e tem como objetivo uma comparação direta com o projeto desenvolvido nesta tese. Os programas comerciais em análise são o **UModel** (Altova), o **Enterprise Architect** (Sparx Systems) e **Visual Paradigm**, estes apresentam compatibilidade com UML2, possuem um sistema de geração de código mais desenvolvido e maturo, e são utilizados para o desenvolvimento de software comercial.
Para cada um destes programas, o estudo consiste na criação de uma máquina de estados comum e geração do respetivo código em Java, de forma a comparar a estrutura de classes utilizada e desta forma interpretar como cada programa aborda a geração de código.

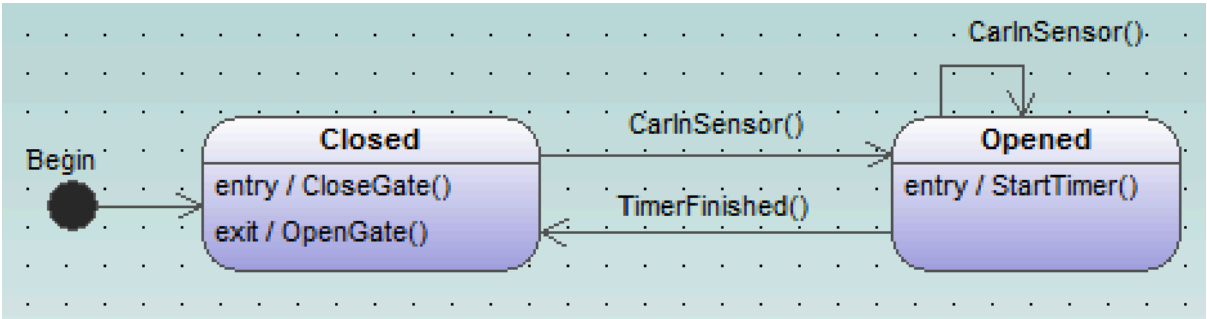


Figura 6 - Máquina de estados teste de um sistema de portão automático

O diagrama de estados utilizado é o controlo de um simples portão automático. Na Figura 6 é possível verificar que este é composto por dois estados, o ***Opened*** (portão aberto) e o estado ***Closed*** (portão fechado). Quando a máquina de estados inicia ocorre uma transição para o estado ***Closed***. A

entrada neste estado leva à execução da ação de entrada ***CloseGate*** que dá a ordem ao motor para fechar o portão. Se o carro se aproximar do sensor, o evento ***CarInSensor***, desencadeia a transição de estado:

- executando a ação de saída ***OpenGate*** que tem como função abrir o portão;
- mudando o estado para ***Opened***.

Após a transição para o estado ***Opened*** é executada a ação de entrada, ***StartTimer*** que inicia um temporizador. Neste ponto dois eventos podem desencadear transições:

- se o carro continuar no sensor, o estado transita para ele próprio, reiniciando o temporizador;
- se o temporizador chegar ao final é emitido o evento ***TimerFinished*** que provoca a transição para o estado ***Closed***, reiniciando-se o ciclo.

Para possibilitar a análise do código gerado são gerados diagramas de classes representantes da estrutura de implementação da MEF de cada software, uma vez que a geração de código a partir de modelos UML se faz, regra geral, ao nível M1 que é nível dos classificadores.

2.2.1 UModel

O diagrama de classes do código gerado pelo UModel (Anexo I) é apresentado na, Figura 7.

Verifica-se que todo o código é gerado dentro de uma única classe – SimpleGateController – sendo todas as outras classes internas (nested). Esta abordagem exibe uma hierarquia de classes, do tipo estado, interna à classe StateMachine, tornando-as privadas no âmbito de cada máquina de estados. Esta abordagem obriga a que na eventualidade de serem criadas máquina de estados diferentes ou até máquinas de estado hierárquicas, com ações e estados semelhantes, o código referente aos estados (classes) e ás ações (métodos) tenha de ser repetido para cada nova máquina de estados. As três classes mais acima no digrama prendem-se com o suporte para regiões de execução paralela, e não são analisadas. Após instanciada, a máquina de estados aguarda que os seus métodos relativos aos eventos (*CarInSensor* e *TimerFinished*) presentes da classe *SimpleGateController* sejam invocados, como input do exterior.

Os estados (*Opened* e *Closed*) e o pseudo-estado inicial (*CSimpleGate*) são traduzidos em classes que implementam a interface IState, que é composta por 3 métodos (comuns a todos os

estados) que servem para obter: o nome; o ID e a região do estado. Existe também uma classe (*TStateId*) com um enumerador para cada estado, que serve para identificar os estados no processamento interno de eventos e transições.

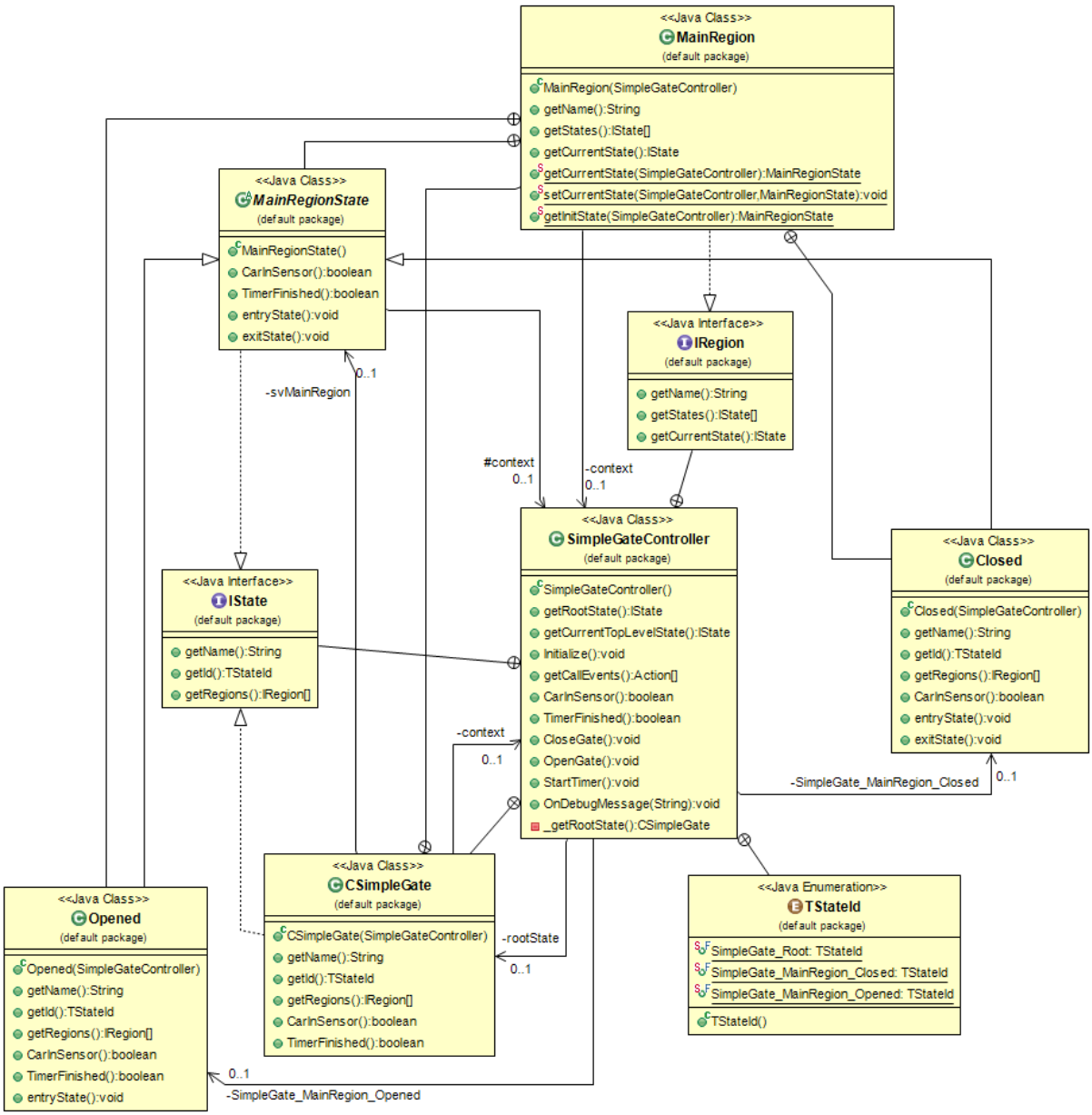


Figura 7 - Diagrama de classes do código gerado pelo software UModel

As ações da MEF (*OpenGate*, *CloseGate* e *StartTimer*) são implementadas como métodos na classe principal (*SimpleGateController*), e cada classe estado tem um método por cada tipo de ação (*entry*, *exit* e *do*) que contém. Cada um destes métodos invoca um método da classe principal correspondente à ação específica.

Os eventos da MEF (*CarInSensor* e *TimerFinished*) aparecem a dois níveis: como métodos da classe principal e como métodos de cada estado em que desencadeiam uma transição de saída. Estes métodos são inicialmente executados através da classe *SimpleGateController* que, por sua vez, invoca o respetivo método do estado atual da máquina de estados que se encarrega da transição.

Não são geradas classes que traduzam o conceito de transição. As transições entre estados são implementadas como métodos da classe de máquina de estados a que estão associados [4]. Estes métodos são os métodos dos próprios eventos e, toda a lógica que permite verificar a transição entre estados, bem como o estado final da transição, encontra-se definida no seu interior. Estes métodos estão definidos em dois níveis, na classe principal *SimpleStateController* e na classe que implementa o estado de origem, para que este método possa ser executado localmente de forma a desencadear a transição.

2.2.2 Enterprise Architect

Na Figura 8, é possível observar o diagrama de classes do código gerado (Anexo IV) pelo

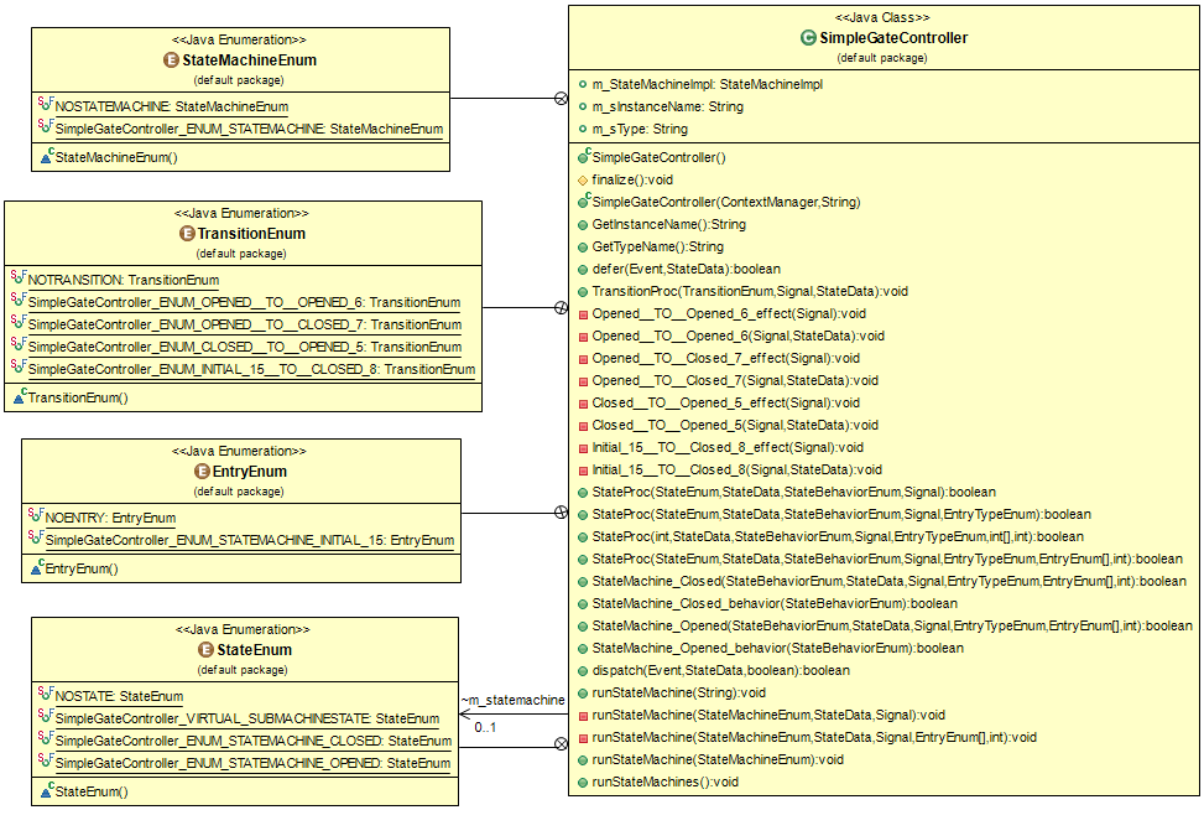


Figura 8 - Diagrama de classes do código gerado pelo software EnterpriseArchitect

software *Enterprise Architect* [5], relativo ao diagrama de máquina de estados apresentado anteriormente. É possível concluir que o código fonte relativo ao diagrama de máquina de estados não apresenta uma estrutura organizada em classes.

Os estados (*Opened* e *Closed*) e o pseudo-estado inicial são traduzidos num enumerador utilizado pela classe principal *SimpleGateController* durante a execução do método *StateProc*, para que através de um *switch case* identifique o estado atual.

As ações da MEF (*OpenGate*, *CloseGate* e *StartTimer*) não existem no contexto do código gerado, o código correspondente a estas ações deve ser definido na ferramenta e é copiado na integra para o interior do método de cada estado – *StateMachine_[estado]_behavior*. O corpo deste método é constituído por um *switch case* que separa a lógica de cada tipo de ação (*entry*, *do* e *exit*). O método *StateMachine_[estado],* invoca o método anterior e é usado para inicialização de variáveis da própria máquina de estados.

Os eventos da MEF (*CarInSensor* e *TimerFinished*) aparecem definidos através de um enumerador dentro da classe principal e são passados à máquina de estados através do método *dispatch*. Este método filtra os eventos enviados à máquina de estados selecionando apenas aqueles associados com o estado atual e chama o método *TransitionProc*.

As transições da MEF, são traduzidas como um binómio enumerador e método. Este binómio é descodificado no método *TransitionProc*, que possui como argumentos de entrada o enumerador da respetiva transição e o evento que a desencadeia, e cuja função é estabelecer a relação entre o enumerador e um dos métodos de transição. Os métodos de transição seguem a seguinte nomenclatura *[estado de saída]_TO_[estado de entrada]*, e são responsáveis por verificar a lógica de transição (condições) e invocar o respetivo método *[estado de saída]_TO_[estado de entrada]_effect*, que executa o código da ação associada à transição caso exista.

2.2.3 Visual Paradigm

O estudo do diagrama de classes (Figura 9) permite concluir que o código fonte gerado (ver Anexo III) pelo software utiliza uma estrutura hierarquizada em classes. O código gerado por esta ferramenta é mais organizado que as analisadas anteriormente. A abordagem do *Visual Paradigm* passa por criar uma série de ficheiros separando os vários conceitos, sendo todo o código gerado

utilizado pela classe *FSMContext* responsável pela gestão e funcionamento de toda a máquina de estados.

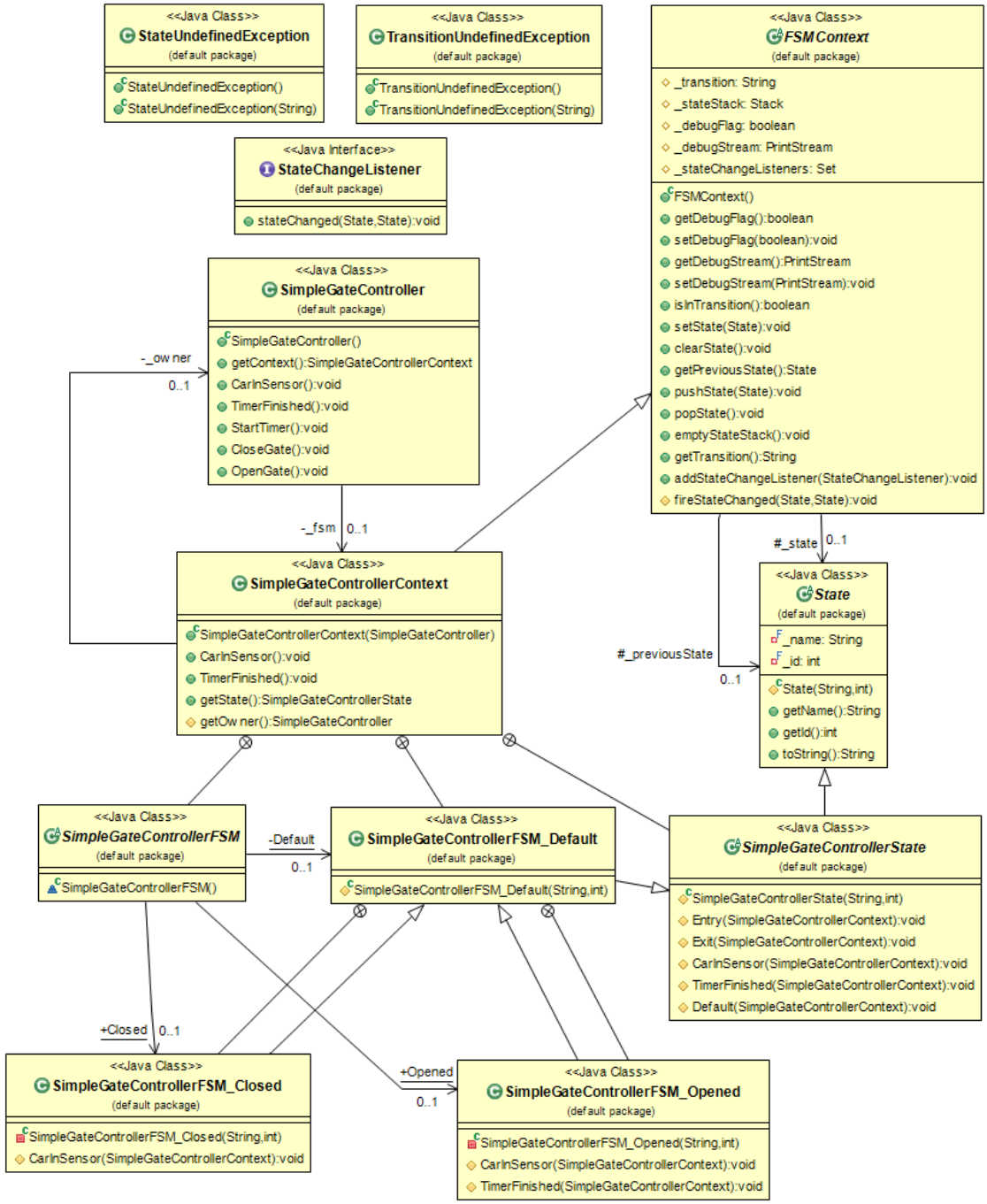


Figura 9 - Diagrama de classes do código gerado pelo software Visual Paradigm

Os estados (*Opened* e *Closed*) e o pseudo-estado inicial são traduzidos como classes internas (*nested*) à classe *SimpleGateControllerContext* e estendem a classe *SimpleGateControllerState.* Esta última possui métodos virtuais para todas as ações e eventos.

As ações da MEF (*OpenGate*, *CloseGate* e *StartTimer*) são implementados sobre a forma de métodos contidos na classe *SimpleGateController*. A classe *SimpleGateController* instancia a máquina de estados implementada pela *SimpleGateControllerContext* e passa-lhe como argumento a sua referência, para que esta possa chamar as ações a seu devido tempo. Esta abordagem torna, os métodos referentes ás ações, públicos permitindo que seja possível forçar a sua execução.

Os eventos da MEF (*CarInSensor* e *TimerFinished*) aparecem também definidos como métodos na classe *SimpleGateController*, este é o modo de entrada de eventos na máquina de estados. Para que um dado evento seja processado o seu respetivo método deve ser invocado, que por sua vez passa essa invocação para um nível abaixo na classe *SimpleGateControllerContext*.

As transições da MEF, são traduzidas como métodos, de forma semelhante ao que acontece com os eventos assumindo, portanto, o mesmo nome que os eventos que as desencadeiam. Uma vez que o evento é processado, ele desencadeia a invocação do método homónimo na classe *SimpleGateControllerContext*, esta classe contém toda a lógica para processar a transição correspondente.

É também de importância salientar que embora os *softwares* comerciais possibilitem o uso de ficheiros UML (*XML*) desenvolvidos em ferramentas de terceiros para a geração do código fonte, estes não podem ser usados *out-of-the-box* uma vez que cada *software* requer uma preparação prévia e específica do mesmo. Este requisito foi verificado utilizando o ficheiro de exportação *XMI* do *UModel* e realizando a importação do mesmo no *Enterprise Architect*, este processo resultou em máquinas de estados incompletas e, por vezes, falhas na importação da mesma.

2.3 Métodos de geração de código

Como se pode ver na secção anterior, existe uma enorme diferença entre uma máquina de estados e o código que a implementa. Isto é expectável sempre que se utilizem linguagens de modelação e de programação que não partilham os mesmos conceitos nem elementos (ou constructos), como é o caso de utilizar uma linguagem de programação de propósito genérico para implementar MEFs. Isto significa que a geração de código não pode ser feita *ad-hoc* e, por isso, existem vários métodos que estruturam o processo. Existem alguns métodos fundamentais de geração de código fonte utilizados na atualidade que são apresentados, de forma breve, mais abaixo.

A geração de código estabelece uma relação entre dois mundos diferentes: a modelação e o código. É importante notar que esta questão é ortogonal à arquitetura de camadas do UML, que se situa inteiramente na esfera da modelação. Para que se possa perceber melhor os conceitos presentes nesta dissertação é útil definir o que se entende por modelo e o que se entende por código. Um modelo é uma abstração da realidade programática que necessita de passar por uma série de transformações para que possa ser executado. Por outro lado, o código é algo que se encontra num nível de abstração inferior e que pode ser diretamente compilado e/ou executado. A geração de código a partir de modelos UML faz-se geralmente, e em particular para as máquinas de estado, ao nível M1 que é nível dos classificadores.

O método de *TEMPLATES + FILTERS* e o de *TEMPLATES + METAMODEL* [6] aplicam-se diretamente à linguagem de modelação. Recorrem á informação armazenada nos modelos para a partir destes obter a informação relacional necessária e gerar código com base nestas relações.

O método de *FRAME PROCESSING* [6], utiliza frames de código predefinidos. Esta abordagem pode surgir de duas formas. Uma das formas é um processador de frames (quadros) baseado em *script*, na qual as frames são instanciadas e parametrizadas de forma simples, através do preenchimento de parâmetros intrínsecos às mesmas. A outra forma é a utilização de um processador de frames adaptativo, que "injeta" código em zonas especificas de algumas instâncias de frames. Estas duas abordagens não são exclusivas, podendo ser utilizadas em simultâneo. De forma oposta ao *TEMPLATES + FILTERS* e *TEMPLATES + METAMODEL*, o *Frame Processing* segue um modo imperativo de programação.

API-BASED GENERATION [6], é um método que utiliza uma API para a qual o código é gerado. Este método é mais dirigido para programas pequenos e simples e o código gerado encontra-se limitado à linguagem previamente estipulada pela API.

Por vezes a definição de modelo e de código é um pouco arbitrária, desta forma, o código fonte pode ser visto como um "modelo" (mais baixo-nível) quando comparado com código binário resultante do processo de compilação. Os dois métodos que se seguem, contrariamente aos anteriormente apresentados, seguem esta nova abordagem e aplicam-se a código fonte.

O método *INLINE CODE GENERATION* [6] resume-se à pura substituição textual de componentes presentes no código fonte. É utilizado, por exemplo, durante a compilação, onde expressões como *#DEFINE*, *#IFDEF*, *#ENDIF* das linguagens C e C++ são pré-processadas e substituídas pelo código fonte correspondente.

O *CODE ATTRIBUTES* [6] é um método que geralmente usa geradores baseados em APIs (*API-BASED GENERATORS*) para a sua implementação e é aplicado em linguagens de *script* (de mais alto nível). A grande vantagem deste método é que possibilita que qualquer anotação seja adicionada ao código fonte, desde que o gerador de código perceba a anotação utilizada.

2.4 Framework FSM

Existem, atualmente, algumas frameworks para máquinas de estados [7], contudo este projeto visa trabalhar com a framework FSM [2], que está alinhada com um subconjunto da versão mais recente de linguagem de modelação UML2. Nesta secção encontram-se descritos os vários componentes que fazem parte da framework utilizada neste projeto. É ainda apresentado um exemplo da sua utilização, baseado numa máquina de estados hipotética e seu respetivo código.

2.4.1 Descrição

A Framework de máquinas de estado usada para este projeto apresenta uma estrutura simples e minimalista, amiga do programador que permite que o código de instanciação seja de leitura fácil e humanamente editável. É composta por um total de seis classes e duas interfaces (Figura 10).

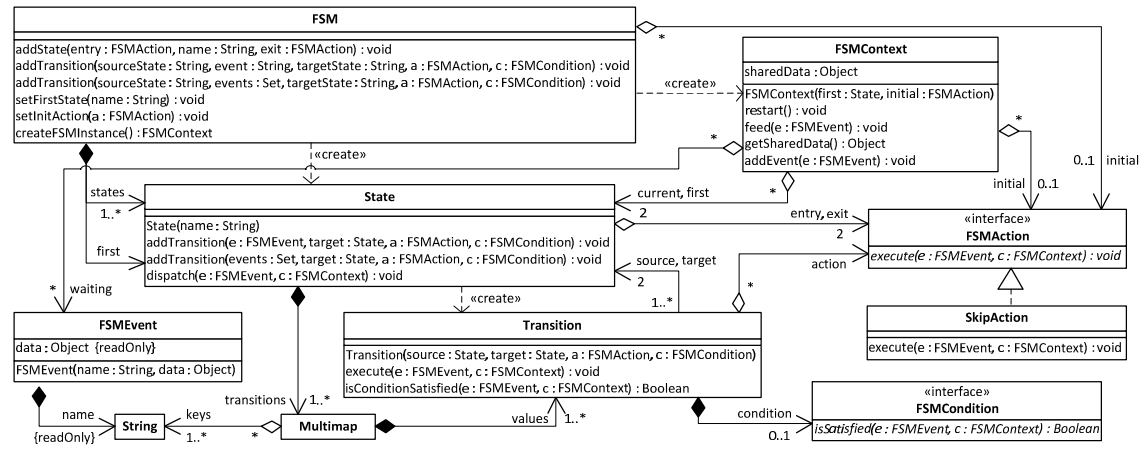


Figura 10 - Diagrama de classes da estrutura da Framework FSM

As interfaces são:

- "*FSMAction*", que define a estrutura requerida por todas as ações utilizadas na máquina de estados.
- "*FSMCondition*", define a estrutura necessária de todas as condições atribuídas ás transições na máquina de estados.

Duas das classes representam o modelo base de qualquer máquina de estados, apresentadas com os nomes:

- "*State*", esta classe modela os estados, possui como atributos (1) as transições que podem ocorrer a partir desse estado, (2) os eventos que as podem desencadear e (3) e as ações que devem ser executadas aquando da entrada e da saída do estado em causa; possui métodos para desencadear eventos.
- "*Transition*", esta classe modela as transições que ligam os vários estados da máquina de estados e possui como atributos o estado de partida e o estado de chegada da transição, assim como a ação (opcional) que deve ser executada aquando da transição. Possui ainda um método para executar a transição.

As restantes classes são:

- "*SkipAction*", esta classe é implementada através da interface "*FSMAction*" e representa a "ação vazia", não possuindo quaisquer atributos ou métodos. Esta ação é a ação executada por defeito para entrada e na saída dos vários estados bem como no arranque da "*FSMContext*".
- "*FSMContext*", esta classe possui como atributos, o estado atual da máquina de estados, o seu estado inicial, a ação de arranque (a ser executada antes da entrada no primeiro estado) e um objeto genérico que permite a partilha de dados entre os vários estados da FSM; os métodos desta classe permitem iniciar e reiniciar a máquina de estados bem como desencadear eventos internos para serem transmitidos à máquina de estados (é o meio de comunicação da máquina com o exterior).
- "*FSMEvent*", esta é representativa de um evento e possui um único atributo que é o objeto genérico de dados partilhado por todas as máquinas de estado que possuam o mesmo contexto; não possui quais quer métodos.
- "*FSM*", possui como atributos uma lista de todos os estados, bem como o estado inicial de arranque da máquina de estados e a ação inicial a ser executada pelos "*FSMContext*"

instanciados. Os métodos desta classe permitem configurar a máquina de estados e criar novas instâncias da mesma com um ou mais "*FSMContext*" a ela associados.

Esta última classe, "*FSM*", é a classe raiz de estrutura da Framework e encapsula toda a estrutura com o objetivo de criar um maior nível de abstração e uma maior facilidade de utilização. É a partir desta classe que são instanciados estados, contextos e adicionadas transições aos respetivos estados. A classe "*FSMContext*" tem como função preservar recursos e diminuir o *footprint* em memória da máquina de estados uma vez que permite que máquinas de estado semelhantes possuam a mesma estrutura e difiram apenas no contexto – *FSMContext*.

2.4.2 Utilização

Quando utilizada, a *Framework*, requere que sejam seguidas algumas normas de forma a que o código seja executado corretamente: (1) declarar as classes representativas das ações, estas são implementadas a partir da interface *FSMAction* e instanciam objetos que são responsáveis pela execução das ações da máquina de estados; (2) declarar as condições, há semelhança das anteriores estas utilizam também uma interface – *FSMCondition*, as condições vão ser utilizadas para verificar se o evento de transição deve ou não ser desencadeado; (3) declarar dois enumeradores um para os estados e outro para os eventos, os enumeradores tem como função associar a estes um número de identificação; no corpo do código principal da função "*main*" devem (4) instanciar-se objetos para as ações e condições declaradas anteriormente; (5) instanciar e inicializar os vários estados com o respetivo identificador e as ações de entrada e de saída a eles associadas; (6) adicionar a cada estado as respetivas transições para os restantes estados; (7) instanciar e inicializar um novo contexto, a partir da classe *FSMContext* com a informação relativa ao estado inicial, ação inicial e ainda o objeto para partilha de dados [2].

Uma vez completamente inicializada, a MEF pode ser controlada alimentado o seu contexto com o enumerador dos eventos ou através de dados externos. O contexto pode ser ainda usado para reiniciar a máquina de estados em qualquer momento, sendo que esta volta ao seu estado inicial e a ação de entrada do estado inicial é de novo executada.

2.4.3 Exemplo

Como exemplo de utilização da Framework é utilizado o mesmo diagrama de estados apresentado na Figura 6 para analisar os programas comerciais. Na Figura 11 é possível observar o diagrama das classes necessárias gerar para o correto funcionamento da Framework, as únicas classes geradas são as classes referentes a ações (implementadas pela *FSMAction*) e dois enumeradores (*TStateId* e *EventID*) os restantes elementos são instanciações de objetos que se encontram no corpo da função *main*.

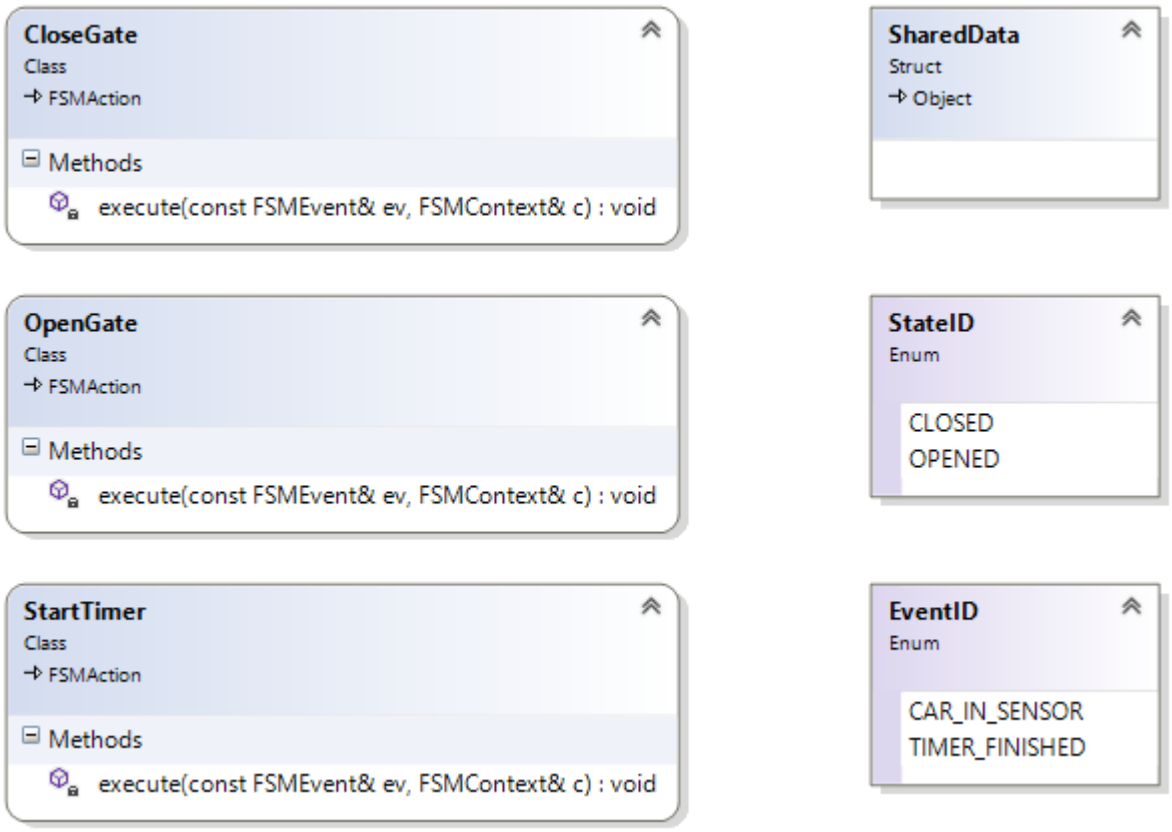


Figura 11 - Diagrama de classes do código necessário para a Framework

O código fonte pode ser observado na Figura 12. Como explicado na secção anterior, a utilização necessita que algumas regras sejam seguidas, assim sendo o ficheiro deve começar pela declaração das classes representativas das várias ações utilizadas pela máquina de estados. Essas classes são *CloseGate*, *OpenGate* e *StartTimer*, e implementam o método *execute* que contém o código correspondente à ação. Neste exemplo não existem condições, por isso, estas não se encontram declaradas, mas seguem o exemplo das anteriores implementando o método *isSastified*.

Os enumeradores que facilitam a interpretação do código seguem-se às declarações das ações e das condições. Por fim, no corpo da função *main*, são instanciadas as ações dos vários estados, seguindo-se a instanciação dos estados (*Opened* e *Closed*) com as ações associadas. As transições são adicionadas depois, recorrendo ao método *addTransition*. Estas associam os estados e podem ou não ter uma condição de transição. O último passo é instanciação de um contexto com o estado inicial *Closed* e sem qualquer ação de inicialização.

```
struct SharedData : public Object {
};

class CloseGate : public FSMAction {
        void execute(const FSMEvent &ev, FSMContext &c) {
                ///TODO: INSERT CODE HERE!!
        }
};

class OpenGate : public FSMAction {
        void execute(const FSMEvent &ev, FSMContext &c) {
                ///TODO: INSERT CODE HERE!!
        }
};

class StartTimer : public FSMAction {
        void execute(const FSMEvent &ev, FSMContext &c) {
                ///TODO: INSERT CODE HERE!!
        }
};

enum StateID { CLOSED = 1, OPENED };

enum EventID { CAR_IN_SENSOR = 1, TIMER_FINISHED };

int main() {
        CloseGate closeGate;
        OpenGate openGate;
        StartTimer startTimer;


        State closed(CLOSED, closeGate, openGate);
        State opened(OPENED, startTimer);

        closed.addTransition(CAR_IN_SENSOR, opened);
        opened.addTransition(CAR_IN_SENSOR, opened);
        opened.addTransition(TIMER_FINISHED, closed);

        SharedData sd;

        FSMContext c(closed, FSMSkipAction::instance, &sd);
        //ADD FSM FEED CODE HERE
        return 0;
}
```

Figura 12 - Código necessário gerar para a correta utilização da Framework FSM

Como breve comparação com os *softwares* comerciais anteriormente utilizados, é de notar que a Framework aborda, estados, transições, ações e condições como objetos, instanciados a partir das respetivas classes. O código resultante é de fácil leitura e possui zonas específicas para ser editado, o que proporciona uma melhor experiência durante a manutenção do mesmo.

3 Conceção da solução

Este capítulo visa criar uma ideia mais aprofundada e concreta da forma como o problema foi abordado e o porquê das decisões tomadas no decorrer do desenvolvimento da ferramenta de geração de código.

3.1 Abordagem

Inicialmente, a abordagem considerada para a ferramenta passava pela criação de um *plugin* para uma ferramenta CASE *open-source* de criação de diagramas UML. Este caminho foi alvo de alterações no decorrer do trabalho, alterações que tiveram na sua origem a melhoria de algumas limitações e correção de alguns problemas que surgiram durante a planificação do *plugin*.

O primeiro problema, análise de *software* de terceiros, é um dos maiores desafios quando se desenvolve um *plugin*. A maior parte dos programas *open-source* disponíveis na internet são mantidos por comunidades, que embora tenham de seguir regras de programação, apresentam uma eficiência e organização menor que empresas dedicadas. Assim sendo, o tempo necessário para esta etapa de desenvolvimento depende muito da documentação existente e fornecida pela comunidade, bem como a capacidade desta em manter um código limpo, robusto e sobretudo estável. Este último ponto é crítico pois obriga a que o *plugin* necessite de ser atualizado de forma a manter-se em conformidade com a plataforma sobre a qual corre e este processo requere constante investimento de tempo de análise de uma ferramenta muito mais complexa que o âmbito do projeto.

Entrando em conta com o conceito de atualizações, e uma vez que este projeto está sempre sujeito a melhorias, assim também está a interface com o *software* hóspede. Na sua natureza um *plugin* é composto por duas componentes fundamentais: a interface com o *software* hóspede; e o processamento funcional independente. Ainda que o *plugin* não seja atualizado na sua componente funcional a componente de interface necessita de ser constantemente atualizada para se manter em conformidade com os requisitos do programa para o qual o *plugin* é desenvolvido. Esta necessidade torna-se dispendiosa em termos de tempo e de mão-de-obra.

Embora as razões acima mencionadas sejam por si só justificativas de uma mudança de abordagem, a maior limitação na criação de um *plugin* é a dependência da longevidade do *software*

hospedeiro. Quando se trabalha com programas *open-source*, um dos maiores problemas com que nos podemos deparar é que, com o tempo, a comunidade que o mantém diminua ou deixe de existir. Tal facto pode tornar o *plugin* obsoleto com o passar do tempo. Outra possibilidade é que o projeto se torne tão grande que deixe de ser *open-source* para passar a ter uma licença comercial, o que provavelmente poria em risco a viabilidade do *plugin* quer por questões de *software* quer legais.

Em alternativa à abordagem inicial, optou-se por desenvolver uma ferramenta que possa funcionar de forma independente, utilizando um formato homologado pelo OMG para leitura de modelos produzidos por qualquer *software CASE* de desenho de diagramas UML. Esse formato é o XMI e a ferramenta a desenvolver terá a forma de um binário executável. Assim, a integração com qualquer ferramenta UML fica o mais simples possível: basta duplicar a interface de geração de código XMI e acrescentar uma chamada ao sistema para executar outro programa. Esse programa é a ferramenta que aqui se propõe, e que recebe como argumento o caminho do ficheiro XMI recém-gerado (i.e., a entrada), conforme ilustrado na Figura 13. Seguindo o modelo da figura verifica-se que o ponto de partida da geração de código é o modelo UML o que faz com que o método de geração de código utilizado se enquadre no *TEMPLATES + METAMODEL* [6], onde através da aplicação de *templates* textuais se faz uma conversão a partir da informação do metamodelo.

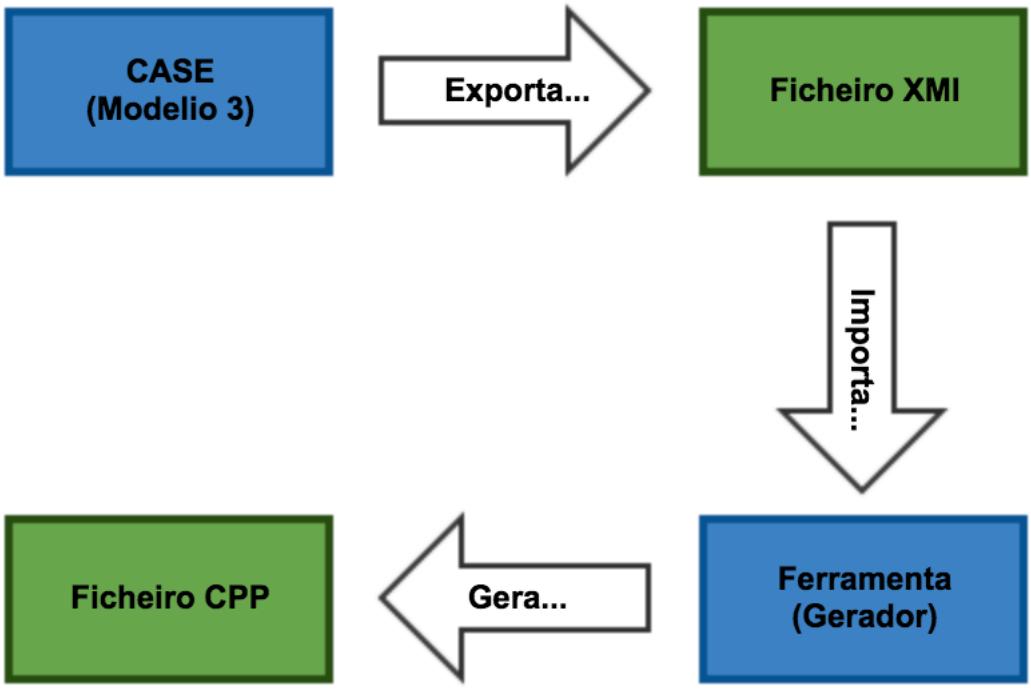


Figura 13 - As várias etapas (quadrados) e eventos (setas) da geração de código.

3.2 Análise da entrada XMI

Para desenhar a aplicação foi necessário analisar o formato XMI. O XMI é um formato XML específico para modelos UML. Procede-se a uma breve apresentação da linguagem de marcação XML.

3.2.1 XML

O acrónimo XML, do inglês *eXtensible Markup Language*, refere-se a uma linguagem de marcação utilizada para a descrição de estruturas de dados e é concebida com o propósito de facilitar a transação de informação através da internet.

Esta linguagem surge na década de 1990 com a necessidade de criar uma linguagem de marcação capaz de unir a flexibilidade da SGML (Linguagem Padronizada de Marcação Genérica) com a simplicidade do HTML (Linguagem de Marcação de Hipertexto). O principal objetivo era criar uma linguagem unicamente textual que fosse interpretada a um alto nível (por *software*), e com algumas características importantes:

- Separar o conteúdo (dados) da formatação (aparência do documento);
- Facilmente legível, tanto para humanos como para computadores;
- Ser auto-descritiva;
- Capacidade ilimitada de criação de marcações (*tags*), os elementos básicos da linguagem e que não são pré-definidos;
- Possibilidade de serem usados como forma de validação de estruturas de dados;
- Possibilitar a hierarquização de dados;

No exemplo da Figura 14, é possível ter uma ideia da linguagem de marcação XML.

Na primeira linha aparece o prólogo com a versão XML e codificação de caracteres utilizadas. Às marcações (*tags*) “casa”, “nome”, “divisórias”, “divisória”, “ligações” e “ligação” dá-se o nome de elementos. O elemento “casa” é a base (ou raiz) da hierarquia do documento e possui algumas características a ele associadas, a enumerar: “local”, “pisos” e “orientação”. Estas características chamam-se atributos. Este elemento tem subelementos ou elementos filho, e termina no final do documento com a utilização da tag de finalização com o mesmo nome (</casa>). Na terceira linha

surge o elemento "nome". Aqui podemos observar que a existência de atributos é opcional, e que um elemento folha (i.e., sem filhos) pode conter apenas texto ("Casa de Férias"). Os dois exemplos restantes que requerem atenção são o elemento "divisória", que apresenta uma mistura de atributos e conteúdo de texto, e o elemento "ligação" que, contém apenas atributos utilizando uma marcação de tag única (*self-closing*). Esta última marcação não permite criação de hierarquia.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<casa local="Porto" pisos="2" orientação="Norte">
   <nome>Casa de Férias</nome>
   <divisórias>
      <divisória comprimento="2" largura="2" piso="1">Despensa</divisória>
      <divisória comprimento="3" largura="2" piso="1">Cozinha</divisória>
      <divisória comprimento="3" largura="3" piso="2">Quarto</divisória>
   </divisórias>
   <ligações>
      <ligação divisória1="Cozinha" divisória2="Despensa" />
      <ligação divisória1="Cozinha" divisória2="Quarto" />
   </ligações>
</casa>
```

Figura 14 - Exemplo de formatação de um ficheiro XML

Esta análise é útil, porque o XMI é um formato XML em que os elementos têm um formato pré-definido, nomeadamente dos elementos do meta-modelo UML, tal como acontece com o HTML para a *web*.

3.2.2 Ficheiro de Exportação (XMI e UML)

Como mencionado anteriormente, os ficheiros exportados pelas ferramentas CASE UML têm a extensão "*xmi*" ou "*uml*". Neste subcapítulo é apresentada uma análise detalhada de um destes ficheiros.

Na Figura 15 podemos observar um diagrama de estados e na Figura 16 uma versão simplificada do ficheiro XMI exportado pela aplicação de modelação *Modelio* [8] (versão 3.4.1). O diagrama de estados corresponde ao de uma porta automática. O elemento raiz é denominado de <*packagedElement*> e possui três atributos: tipo, id e nome. O "*type*" é um atributo recorrente utilizado em vários elementos ao longo do ficheiro e especifica o tipo de elemento da máquina de estados que está a ser considerado. O "*id*" é um identificador (único) alfanumérico presente em todos

os elementos existentes no ficheiro UML. O "*name*" é um identificador também, mas é atribuído durante o processo de criação do modelo e tem como objetivo tornar percetível o elemento em causa.

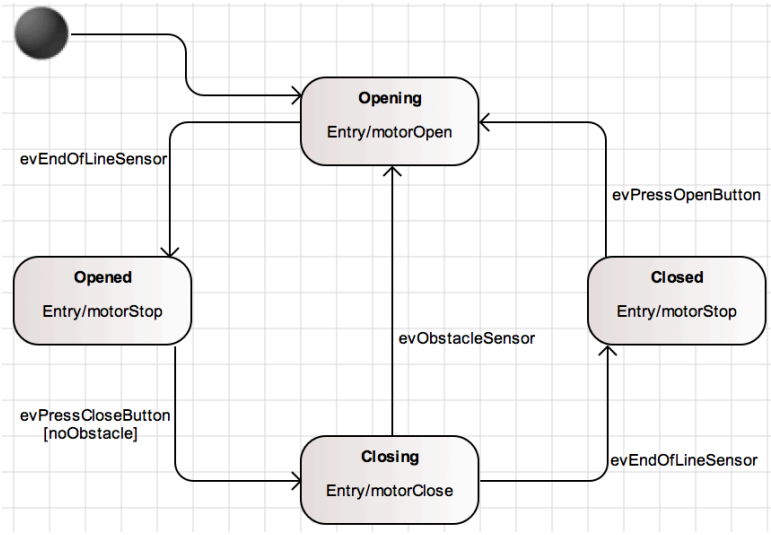


Figura 15 - Exemplo de diagrama de máquina de estados

Dentro do elemento raiz existe um elemento <*region*>. Este e outros subelementos (<*subvertex*>, <*entry*>, <*transition*> e <*trigger*>) correspondem a terminações de associações (*association ends*) do metamodelo UML da Figura 2. A exceção é <*body*> que é um elemento da metaclasse <*OpaqueBehaviour*>, que é a classe concreta da metaclasse "*Behaviour*" para comportamentos simples. A terminação da associação <*region*> identifica o elemento "*Region*" que delimita uma máquina de estados. Cada um dos elementos tem no atributo "tipo" a classe do metamodelo UML à qual corresponde (*State*, *Behaviour*, *Transition* e *Trigger*).
No nível hierárquico abaixo do anterior é possível identificar dois tipos de elementos: <*subvertex*> e <*transition*>.

<*Subvertex*> é um fim de associação para diferentes elementos e tanto pode representar estados como pseudo-estados (estado inicial). o atributo "*type*" indica que quatro são estados e um deles é o pseudo-estado inicial. Dentro dos estados podem existir elementos dos tipos: <*entry*>, <*exit*> e <*doActivity*>, que representam respetivamente ações de entrada, saída e permanência. No caso em análise existem apenas ações de entrada. O nome atribuído a estas ações é apresentado num grau hierárquico inferior em elementos <*body*>.

Os elementos <*transition*> representam as transições e possuem dois atributos essenciais: "*source*" e "*target*". Estes atributos contêm uma cópia do identificador único dos estados de partida e de chegada da transição. Esta informação é importante para estabelecer a relação entre os estados e as respetivas transições. No nível hierárquico inferior à transição está contida a informação referente aos eventos que a desencadeiam assim como a condição de guarda da transição. Os eventos são

```
<packagedElement xmi:type="uml:StateMachine" xmi:id="" name="">
   <region xmi:id="">
      <subvertex xmi:type="uml:State" xmi:id="" name="Closed">
         <entry xmi:type="uml:OpaqueBehavior" xmi:id="">
            <body>motorStop</body>
         </entry>
      </subvertex>
      <subvertex xmi:type="uml:State" xmi:id="" name="Opening">
         <entry xmi:type="uml:OpaqueBehavior" xmi:id="">
            <body>motorOpen</body>
         </entry>
      </subvertex>
      <subvertex xmi:type="uml:State" xmi:id="" name="Closing">
         <entry xmi:type="uml:OpaqueBehavior" xmi:id="">
            <body>motorClosed</body>
         </entry>
      </subvertex>
      <subvertex xmi:type="uml:State" xmi:id="" name="Opened">
         <entry xmi:type="uml:OpaqueBehavior" xmi:id="">
            <body>motorStop</body>
         </entry>
      </subvertex>
      <subvertex xmi:type="uml:Pseudostate" xmi:id="" name=""/>
      <transition xmi:type="uml:Transition" xmi:id="" source="" target="" >
         <trigger xmi:type="uml:Trigger" xmi:id="" name="evPressOpenButton"/>
      </transition>
      <transition xmi:type="uml:Transition" xmi:id="" source="" target="" >
         <trigger xmi:type="uml:Trigger" xmi:id="" name="evEndOfLineSensor"/>
      </transition>
      <transition xmi:type="uml:Transition" xmi:id="" source="" target="" >
         <trigger xmi:type="uml:Trigger" xmi:id="" name="evEndOfLineSensor"/>
      </transition>
      <transition xmi:type="uml:Transition" xmi:id="" source="" target="" >
         <trigger xmi:type="uml:Trigger" xmi:id="" name="evObstacleSensor"/>
      </transition>
      <transition xmi:type="uml:Transition" xmi:id="" source="" target="" >
         <ownedRule xmi:type="uml:Constraint" xmi:id="" constrainedElement="">
         </ownedRule>
         <trigger xmi:type="uml:Trigger" xmi:id="" name="evPressCloseButton"/>
      </transition>
      <transition xmi:type="uml:Transition" xmi:id="" source="" target=""/>
   </region>
</packagedElement>
```

Figura 16 - Exemplo de um ficheiro XMI utilizando a marcação UML

elementos *<trigger>* identificados pelo atributo nome. As condições de guarda são elementos *<ownedRule>* e possuem o atributo "*type*", que permite identificar o tipo de regra (condição de guarda, ação de transição ou ação pós-transição) e o "nome" que contém a (descrição da) condição.

3.3 Decomposição em fases

Com o intuito de criar um projeto modular e flexível, com um código limpo e passível de ser atualizado de forma fácil, torna-se importante conceber um sistema bipartido (Figura 17), composto por duas fases distintas. Esta necessidade parte da diferença existente entre a sequência estrutural do código C++ (declarações, inicializações) e a utilizada no ficheiro XMI. A programação C++ utiliza o

Figura 17 - Estrutura bipartida da ferramenta (a amarelo as duas fases, a azul a estrutura de comunicação)

binómio – declaração e inicialização – que faz com que uma escrita *on-the-fly* necessite que o ficheiro seja percorrido várias vezes para cada elemento. As leituras do ficheiro em disco são operações lentas que devem ser evitadas a todo o custo para que o desempenho da ferramenta seja o mais otimizado possível. Esta separação, em duas fases, permite ainda dividir o problema em partes mais pequenas, que são tornadas independentes através da criação de uma estrutura de dados gerada pela primeira fase e utilizada pela segunda. Esta estrutura não é considerada uma fase *per-si* pois não possui qualquer ação a ela associada, serve apenas de comunicação entre as duas fases do processo. Para a criação da estrutura da MEF, serão utilizadas classes parecidas às classes da própria *Framework FSM*. O nome das classes terá o nome do elemento UML que lhes corresponde, de acordo com o seguinte mapeamento:

- Máquinas de estado → *FSM*
- Estados → *FSMState*
- Transições → *FSMTransition*
- Ações → *FSMAction*

- Condições → *FSMCondition*
- Eventos → *FSMEvent*

Numa primeira fase – Interpretação – a aplicação deve ser capaz de abrir o ficheiro XMI ou UML e proceder à sua análise. Esta análise passa pela identificação e interpretação dos dados textuais contidos no ficheiro e criação de uma estrutura orientada a objetos com a informação interpretada. Esta estrutura assenta num conjunto de classes criadas especificamente para armazenar a informação e estabelecer ligações entre os vários elementos. O componente de interpretação percorre a estrutura na ordem hierárquica em que o ficheiro XMI está organizado. No fluxograma da Figura 18 encontra-se definida a ordem de acontecimentos na interpretação do ficheiro.

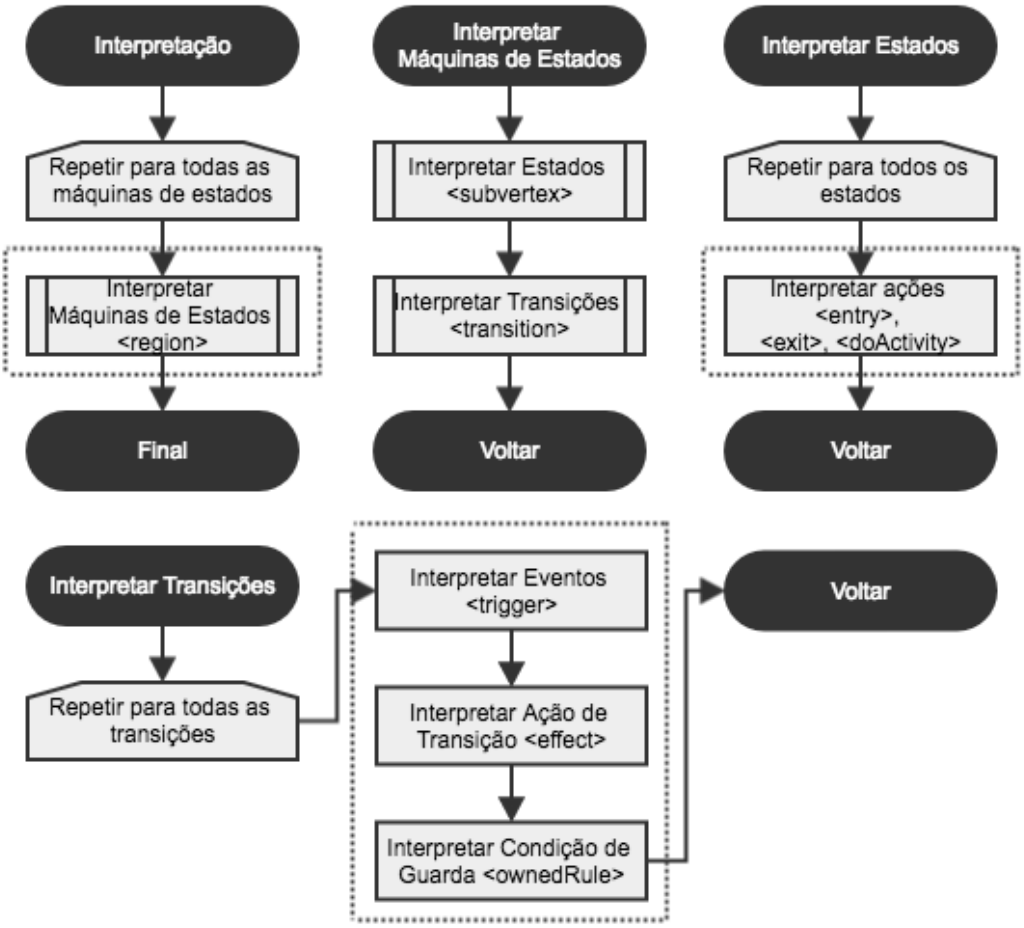


Figura 18 - Fluxograma de conceção do componente de interpretação (pseudocódigo)

O programa deverá aceder ao ficheiro XMI e procurar por marcações do tipo <*region*> (máquinas de estados) e depois para cada uma delas interpreta os estados e as transições. Os

estados são localizados utilizando a marcação <*subvertex*> e posteriormente para cada um é localizada a informação referente ás ações de entrada, saída e corpo do estado representadas pelas marcações <*entry*>, <*exit*> e <*doActivity*>, respetivamente. Para as transições é feita uma localização dos elementos <*transition*> e para cada um deles é instanciado um conjunto de eventos (<*trigger*>), a condição de guarda (<*ownedRule*>) associada à transição e a ação de transição (<*effect*>). Após executar estes passos e toda a informação textual ser extraída para as respetivas instancias dos vários objetos, é executada uma função de criação de referências que vai percorrer toda a estrutura e estabelecer relações entre os vários objetos.

Numa segunda fase – Escrita – após a estrutura da máquina de estados ser criada, esta informação é escrita seguindo o modelo de funcionamento da *Framework* dando origem a um ficheiro **.cpp*. Este componente percorre a estrutura na ordem em que a informação deve ser escrita para o ficheiro, de forma a corresponder com o modo de funcionamento da *Framework FSM*. No fluxograma da Figura 19, encontra-se definida a ordem de acontecimentos na escrita do código.

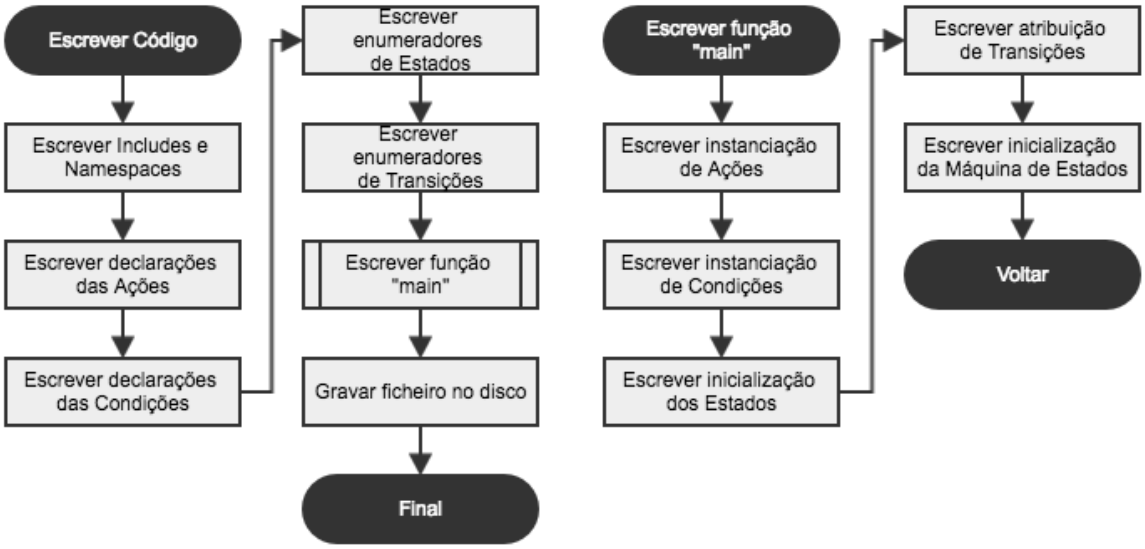


Figura 19 - Fluxograma de conceção do componente de escrita (pseudocódigo)

De forma a evitar os contínuos acessos ao disco para escrever o código gerado, este é mantido em memória num *buffer* ("*output*") que só no final do processo é que escreve todo o texto para o ficheiro especificado. O módulo de escrita começa por escrever as classes que definem as ações e as condições, estas derivam das classes *FSMAction* e *FSMCondition* da *framework*,

respetivamente. Os componentes de máquina de estados que se seguem são os estados e eventos. Estes são definidos como enumeradores, é ainda acrescentada uma função com a associação entre enumerador e respetivo texto com nome de estados e eventos, o propósito destes métodos é apenas para efeitos de *debug*. Durante o processo de escrita da função *main*, inicialmente são declaradas as variáveis para as várias ações e condições e posteriormente, inicializados os estados com as atribuições das ações de entrada e saída. A última etapa, na fase de escrita, é criar as transições de cada estado e adicionar-lhes ações e condições declaradas anteriormente.

Esta abordagem vai permitir que mais tarde seja mais simples atualizar o código da aplicação uma vez que é possível atualizar a fase de interpretação de forma independente da fase de escrita do código. Esta abordagem conta ainda com a capacidade de desenvolver classes de escrita para outras linguagens de programação nomeadamente Java.

4 Implementação

Neste capítulo encontra-se explicada em detalhe a ferramenta de geração de código C++ a partir de um ficheiro XMI.

Este capítulo está dividido em três partes essenciais: a primeira tem como objetivo explicar o esqueleto de classes responsáveis por criar os objetos da máquina de estados; a segunda parte explica em detalhe a fase de interpretação do ficheiro XMI, que transforma a informação textual nele contida numa estrutura relacional de objetos; e por fim a terceira parte que descreve as classes que geram o código C++ a partir da estrutura gerada na fase anterior. No diagrama da Figura 20 encontram-se (da esquerda para a direita, segundo a ordem de intervenção no processo de geração de código) todas as classes implementadas no decorrer desta dissertação.

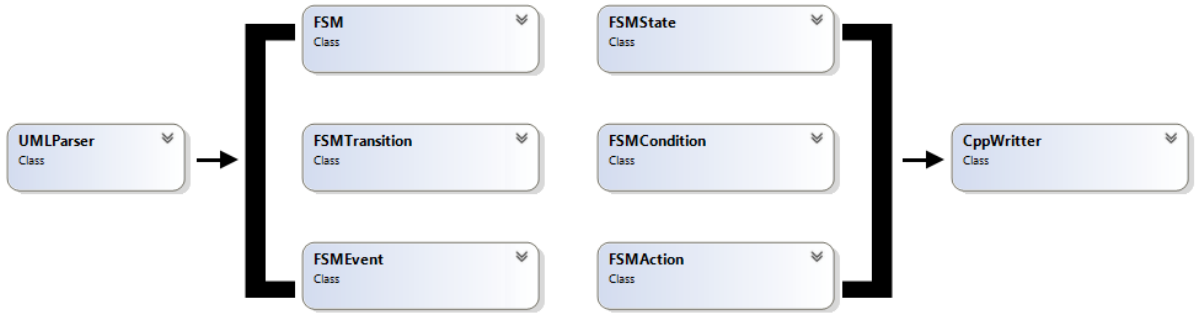


Figura 20 - Classes intervenientes no processo de geração de código

4.1 Opções tecnológicas

Para ajudar no processo de interpretação, é utilizada uma biblioteca para efetuar a leitura do ficheiro. Esta biblioteca encontra-se incluída no conjunto de bibliotecas fornecidas pelo *Mono .NET* e possui um conjunto de funções essenciais para interpretar ficheiros XML. Os métodos necessários fazem uso de uma sintaxe denominada *XPath*, que é própria para definir partes de um documento XML. De entre os métodos disponíveis é importante destacar os mais utilizados:

SelectNodes(): este método faz uso da referência que lhe é passada por argumentos e seleciona todos os elementos que possuam as características definidas. No exemplo abaixo, a sintaxe XPath faz retornar todos os elementos do tipo *subvertex* cujo atributo "*xmi:type*" seja "*uml:State*". Por

outras palavras o método retorna uma lista de elementos representativos de estados do diagrama de máquina de estados utilizado.

```
"./subvertex[@xmi:type='uml:State']"
```

SelectSingleNode(); muito semelhante ao método anterior este retorna apenas um elemento que corresponda às especificações contidas no caminho *XPath*. Caso vários elementos sejam encontrados apenas o primeiro é retornado. No exemplo abaixo, a sintaxe *XPath* permite não só selecionar o primeiro elemento do tipo *<subvertex>* cujo atributo "*xmi:type"* seja "*uml:Pseudostate*" – refere-se aqui ao estado inicial – como retornar o seu identificador único. Isto é possível recorrendo à expressão "*@xmi:id*", o uso da arroba permite aceder a atributos dos elementos.

```
"./subvertex[@xmi:type='uml:Pseudostate']/@xmi:id
```

4.2 Classes Auxiliares

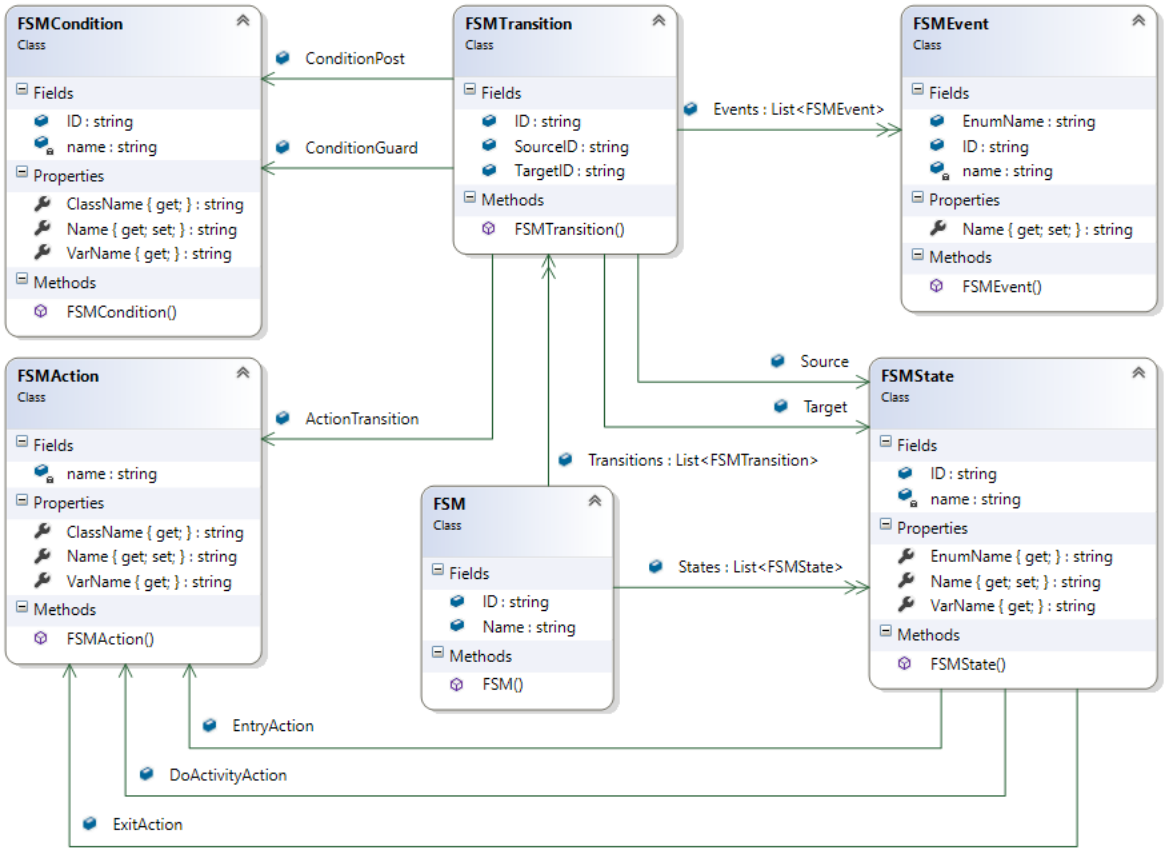


Figura 21 - Relação entre as várias classes estruturais da FSM

As classes que tornam possível o armazenamento de informação relacional da máquina de estados são seis – *FSM*, *FSMState*, *FSMAction*, *FSMTransition*, *FSMEvent*, *FSMCondition* – que representam respetivamente, a máquina de estados. Este modelo de classes auxiliares vai permitir contruir a estrutura com a informação necessária à geração de código para a *framework*.

Na Figura 21 é possível visualizar, a relação entre as várias classes estruturais. A classe *FSM* estabelece relações com as classes *FSMTransition* e *FSMState*, possuindo uma lista para cada uma destas delas. A classe *FSMState* apresenta três relações com a classe *FSMAction* que representam as ações de entrada, saída e do corpo do estado. Por fim, a classe *FSMTransition* estrutura toda a máquina de estados e apresenta relação com todas as outras classes. A uma transição está associada: uma lista de eventos (*FSMEvent*); dois estados (*FSMState*) um de partida e outro de chegada; uma ação (*FSMAction*) que ocorre aquando a transição e; duas condições (*FSMCondition*). Seguidamente apresenta-se uma análise mais detalhada de cada classe.

As classes auxiliares não possuem métodos para além do método construtor que possibilita a instanciação do elemento que representam. Este método construtor não possui qualquer argumento, e serve apenas para o processo de instanciação.

A classe *FSM* (Figura 21 – em baixo ao centro) representa uma máquina de estados. Ela é composta por: um campo denominado *ID*, que corresponde ao identificador único contido no ficheiro UML; um *name* que é gerado de forma automática e sequencial; e os campos *states* e *transitions* que são duas listas (conjuntos) para guardar, respetivamente, os estados e as transições existentes numa dada máquina de estados. De forma a simplificar os campos necessários, o estado inicial será colocado em primeiro lugar da lista de estados.

A classe *FSMState* (Figura 21 – em baixo à direita) instancia todos os estados presentes na máquina de estados finitos. Em relação aos campos presentes nesta classe, existem: *DoActivityAction*, *EntryAction* e *ExitAction* que são as ações de permanência, entrada e saída do estado; o campo *ID* para o identificador único; o campo *name* que contem o nome não formatado do estado. As propriedades *Name, EnumName* e *VarName* relacionam-se com o campo *name*. A primeira serve apenas para fazer leitura e escrita no campo *name* com o nome do estado tal como consta no ficheiro

UML; a propriedade *EnumName* retorna o campo *name* com uma formatação em maiúsculas e com os espaços substituídos por *underscores* e; a propriedade *VarName* retorna o campo *name* com formatação *CamelCase* e inicial minúscula.

A classe *FSMAction* (Figura 21 – em baixo à esquerda) representa as ações associadas aos estados e transições e possui a informação relativa ao nome da ação. À semelhança do que acontece anteriormente, possui um campo *name* que contem a mesma informação que o ficheiro UML. As propriedades *Name, EnumName* e *VarName* relacionam-se com o campo *name*. A primeira serve apenas para fazer leitura e escrita no campo *name* com o nome da ação tal como consta no ficheiro UML; a propriedade *ClassName* retorna o campo *name* com uma formatação *CamelCase e* inicial maiúscula*;* a propriedade *VarName* retorna o campo *name* com formatação *CamelCase* e inicial minúscula.

A classe *FSMTransition* (Figura 21 – em cima ao centro) instancia objetos do tipo transição. e possui vários campos que serão descritos seguidamente. O campo *ID*, já presente em algumas classes anteriores, armazena o identificador alfanumérico contido no ficheiro UML. Os campos *SourceID* e *Source* representam respetivamente o *ID* do estado de partida da transição e o próprio objeto *FSMState*. Os campos *TargetID* e *Target,* à semelhança dos anteriores, representam o identificador e o objeto *FSMState* do estado de chegada da transição. *Events* representa uma lista de objetos do tipo *FSMEvent* (descritos seguidamente), ao passo que *ConditionGuard*, *ConditionPost* e *ActionTransition* são os objetos condição guarda e ações associadas à transição.

A classe *FSMEvent* (Figura 21 – em cima à direita) instância eventos para serem associados a transições. A informação relativa aos eventos que é necessária armazenar encontra-se no campo *ID* que, mais uma vez, corresponde ao identificador único do evento no ficheiro UML e no campo *name* que há semelhança do que acontece com os estados se pode apresentar com algumas formatações diferentes, obtidas com recurso a propriedades. A propriedade *Name* escreve e lê diretamente no campo *name* e corresponde ao nome lido do ficheiro, e outra denominada *EnumName* que apresenta uma formatação toda em maiúsculas e onde os espaços são substituídos por *underscores*. Este nome é usado para criação de um enumerador que permite a fácil identificação do tipo de cada evento.

A última das classes auxiliares a ser descrita é a classe *FSMCondition* (Figura 21 – em cima à esquerda) responsável pela instanciação dos objetos condição. Esta classe segue uma estrutura muito semelhante à classe *FSMAction*, à exceção de que esta possui mais um campo *ID* para guardar o seu identificador único que no caso da classe *FSMAction* não existe.

Finda a análise da estrutura de classes auxiliares é necessário esclarecer alguns aspetos que são importantes ter em conta.

- Nem todas as classes possuem identificadores *ID*. A classe *FSMAction* não necessita destes identificadores, pois os objetos que são instanciados no momento da leitura são ligados de imediato aos estados ou transições correspondentes.
- Nem todas as classes tiram partido do *ID* lido no ficheiro UML, a classe *FSMCondition* e *FSMEvent* não utilizam os *ID* durante todo o processo de geração de código. À semelhança da classe *FSMAction* os objetos instanciados são imediatamente ligados ás transições respetivas. É de notar que, ainda assim, esta informação é armazenada pois poderá ser importante em futuras intervenções.
- A ligação entre os estados e transições é efetuada à *posteriori* recorrendo a um método presente na classe *UMLParser*, descrito na fase de interpretação.

4.3 Fase de interpretação do XMI

4.3.1 Estrutura da classe

Esta classe (Figura 22) é a responsável pela leitura e interpretação textual do ficheiro UML (formato XMI). Ela é composta essencialmente por métodos, contendo apenas um campo denominado de *namespaceManager* que é necessário para a leitura dos dados XML contidos no ficheiro. Este campo é um objeto que guarda uma lista de *namespaces* que devem ser ignorados durante a pesquisa no ficheiro, sendo eles: "*xmi*" e "*uml*" (Figura 16).

Quanto aos métodos presentes nesta classe, existe um total de vinte métodos, dezoito dos quais são métodos internos (privados) à classe, sendo que a única interação possível com o utilizador é através dos métodos: *UMLParser* que é o construtor da classe e recebe como argumento um documento XML (XMLDocument) e; *ParseStateMachines* que recebe o elemento raiz do documento a partir do qual é iniciada a interpretação dos dados. Os métodos podem ser divididos em dois tipos

diferentes: métodos *Get* (procura) e; métodos *Parse* (interpretação). Os métodos de procura – *GetStateMachines*; *GetStates*; *GetTransitions*; *GetEvents*; *GetStartStateID* e; *GetConditionGuard* – são métodos que acedem diretamente a elementos específicos do ficheiro no disco através da utilização da biblioteca XML e retornam listas de elementos XML com a informação referente aos vários elementos, à exceção dos dois últimos que recolhem apenas um elemento XML. Por sua vez os métodos de interpretação – *ParseConditionGuard*, *ParseEvents*, *ParseStates*, *ParseState*, *ParseTransitions*, *ParseTransition* – são métodos que apenas interpretam os dados obtidos pelos métodos de procura e devolvem objetos interpretados, representados pelas classes auxiliares. Existem, contudo, quatro exceções nesta arquitetura – *ParseActionDoActivity*, *ParseActionEntry*, *ParseActionExit*, *ParseActionTransition* – são métodos que realizam tanto procura como interpretação dos elementos

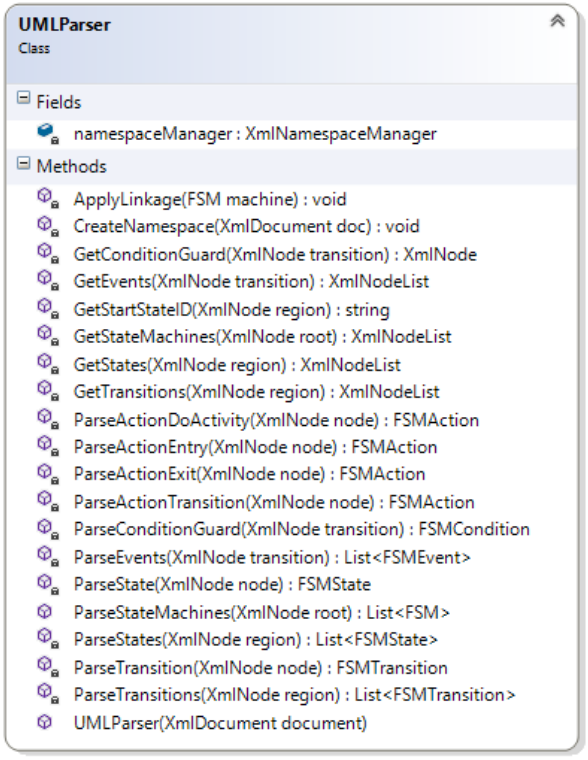


Figura 22 - Classe UMLParser

das ações, uma vez que as *FSMAction* não possuem campo de identificação – *ID*.

Os dois métodos restantes são: *CreateNamespace* que configura o campo *namespaceManager* para que a procura dos elementos XML utilizando a linguagem *XPath* seja bem-sucedida e; *ApplyLinkage* que tem como argumento um objeto FSM (máquina de estados) e estabelece a ligação entre os objetos *FSMState* e *FSMTransition*, recorrendo para isso aos *ID*

(identificadores de cada objeto). Este segundo método é o último a ser executado pois necessita que todos os objetos tenham sido instanciados.

4.3.2 Relação entre métodos

Por cada elemento *State* e *Transition* de uma máquina de estados UML existem três métodos denominados *Parse[elemento-plural]*, *Get[elemento-plural]*, *Parse[elemento-singular]* (como por exemplo: *ParseStates*; *GetStates*; e *ParseState*), para os elementos *Event* e *StateMachine* existe um conjunto de dois métodos denominados *Get[elemento-plural]* e *Parse[elemento-plural],* para os elementos *Condition* existem dois métodos *Get[elemento-singular]* e *Parse[elemento-singular]* e, por fim para os elementos do tipo *Action* existem vários métodos que seguem a estrutura *Parse[elemento-singular][tipo]*. Esta nomenclatura é importante para perceber a forma como estes são executados de forma hierárquica. Assim sendo, a lista de tarefas para os vários (três) métodos é, respetivamente:

1. Instanciação uma lista de um dado tipo de entidade e passa para o próximo método um elemento XML.
2. Faz a pesquisa no ficheiro XMI por todas as ocorrências dessa entidade, retornando uma lista de elementos XML, se se tratar de apenas um elemento retorna um só elemento XML.
3. *Parse[elemento-singular]* – converte o(s) elemento(s) XML em objetos para preencher a estrutura relacional da máquina de estados.

Tendo em conta a lista de tarefas acima descrita, nos elementos que apenas possuem dois métodos *Event* e *StateMachine*, os métodos correspondem às tarefas 2 e 3, respetivamente. No caso de elementos com apenas um método o mesmo método realiza ambas as tarefas 2 e 3. Esta interação entre os vários métodos pode ser observada no fluxograma da (Figura 23).

Depois da classe ser instanciada *UMLParser*, o primeiro conjunto de métodos a ser executado é o *ParseStateMachines* e o *GetStateMachines*. O método *ParseStateMachines* recebe como argumento o elemento raiz do documento XMI, instancia uma lista (conjunto) máquinas de estados recorrendo à classe auxiliar *FSM* e chama o método *GetStateMachines* que procura o documento por elementos do tipo <region> –representativos de máquinas de estados – devolvendo uma lista de *XMLNode*, sendo que apenas o primeiro resultado é interpretado. Estes dois métodos apresentam uma

estrutura quase semelhante à acima descrita, contudo uma vez que a *Framework* não possui capacidade de lidar com máquinas de estados hierárquicas não existe método para interpretar a máquina de estados uma a uma (*ParseStateMachine*) pois só existe uma máquina de estados.

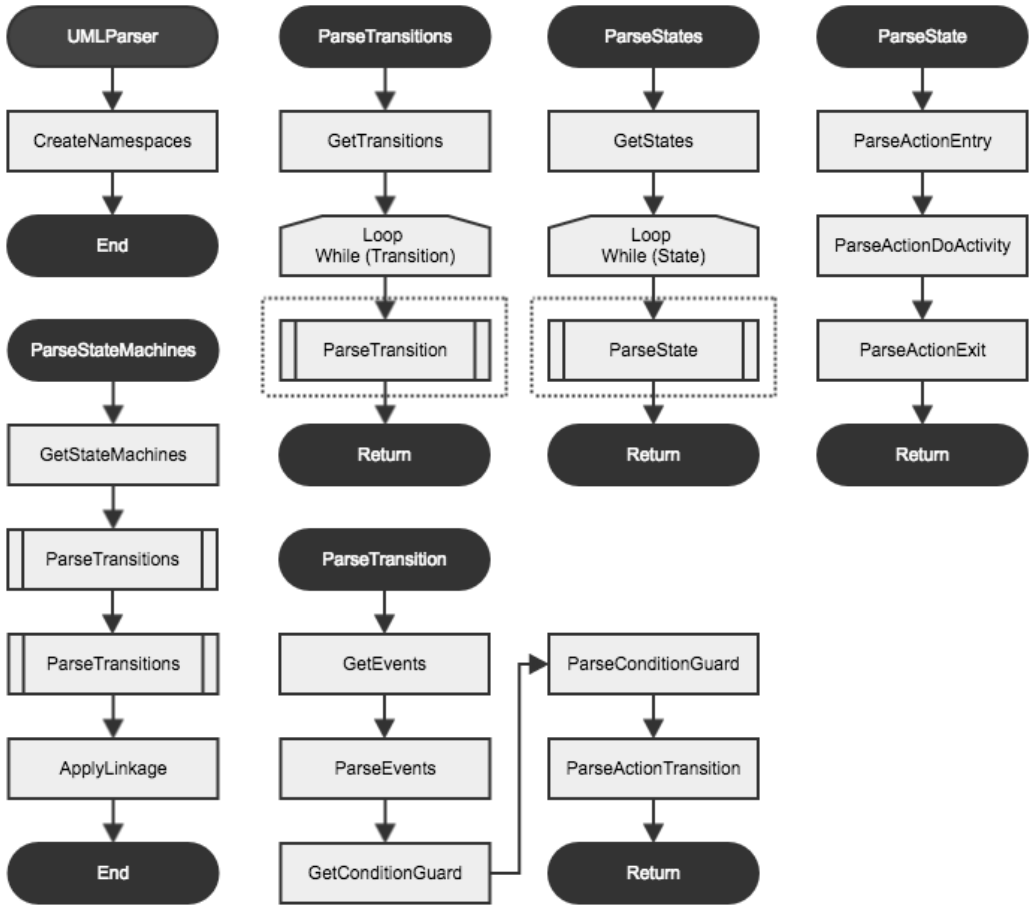


Figura 23 - Fluxograma do componente de interpretação do XMI

De seguida o conjunto de métodos que é executado na fase de interpretação é o *ParseStates*, *GetStates* e o *ParseState*, conforme acima descrito. O método *ParseStates* é alimentado por um argumento do tipo *XMLNode* de uma <*region*> e instancia uma lista de estados recorrendo à classe auxiliar *FSMState*. De seguida chama o método *GetStates* que procura no interior no *XMLNode* por todas as ocorrências de elementos <*subvertex*> – representativos de estados e pseudo-estados – devolvendo-os em forma de uma lista (conjunto) de *XMLNode*. Os elementos dessa lista são depois interpretados um a um através da função *ParseState* que instancia vários objetos do tipo *FSMState*.

Uma vez processados os estados, é executado o método *GetStartStateID* que tem como objetivo encontrar o ID do estado inicial. Este método procura por entre os elementos <*subvertex*> até encontrar um cujo atributo "*uml:type*" seja "*Pseudostate*" e identifica o estado para o qual o pseudo-estado transita. Este é o estado inicial e como tal é deslocado para o primeiro lugar da lista de estados.

O próximo conjunto de métodos a ser executado é: *ParseActionEntry*, *ParseActionExit* e *ParseActionDoActivity*. Estes métodos dizem respeito à interpretação das ações de cada estado. Todos eles recebem um argumento do tipo *XMLNode* de um <subvertex> e instanciam uma nova ação recorrendo à classe auxiliar *FSMAction*. O método *ParseAction[tipo]* procura pelo elemento <[tipo]> do estado e preenche o campo *name* da nova ação com o conteúdo (<*body*>) do elemento XMI. Apesar da *Framework FSM* não suportar ações do tipo <*doActivity*>, o código para interpretação foi na mesma implementado para mais tarde ser possível uma fácil atualização do projeto. Estes elementos após a instanciação são adicionados aos estados aos campos respetivos, não sendo necessário liga-los mais tarde.

As transições são interpretadas pelo seguinte conjunto de métodos: *ParseTransitions*, *GetTransitions* e *ParseTransition*. O primeiro método é alimentado por um argumento do tipo *XMLNode* de uma <*region*> e instancia uma lista de transições recorrendo à classe auxiliar *FSMTransition*. De seguida chama o método *GetTransitions* que procura no interior do elemento com por todas as ocorrências de *XMLNode* <*transition*> – estes elementos são representativos de transições entre estados – devolvendo-os em forma de uma lista (conjunto). Os elementos são depois interpretados um a um através da função *ParseTransition* que instancia vários objetos do tipo *FSMTransition*. A ligação entre estados e transições é feita mais tarde.

Para cada transição é chamado um conjunto de métodos para instanciar e inicializar eventos e condições. O conjunto de métodos responsáveis pela interpretação de eventos é composto pelos métodos *GetEvents* e *ParseEvents*. O primeiro, procura no interior do *XMLNode* de uma <*transition*> por elementos com o nome <*trigger*> em seguida é executado método *ParseEvents* que procede à instanciação de objetos *FSMEvent*. Estes objetos são posteriormente inicializados com a variável nome, presente no atributo *name* do elemento *XMI*. Estes objetos do tipo evento são adicionados diretamente à transição ao campo correspondente – "*events"*.

Como a cada transição apenas pode ser associada uma condição de guarda, os métodos utilizados para a interpretação de condições apenas procuram por um elemento do tipo <*ownedRule*>. Esta tarefa é levada a cabo pelo método *GetConditionGuard*. O *XMLNode* retornado é passado por argumento ao método *ParseConditionGuard* que instancia objetos *FSMCondition* com toda a informação, necessária à geração de código, contida no *XMLNode*. Estes objetos são adicionados diretamente à transição ao campo correspondente – "*conditionGuard"*.

O mesmo ocorre com as ações de transição onde o *XMLNode* da transição é percorrido pelo método *ParseActionTransition*, que devolve o elemento <*effect*> correspondente e procede à instanciação de um objeto *FSMAction* que é adicionado ao campo correspondente – "*actionTransition*". A ação de transição do pseudo-estado não é suportada atualmente e é capacidade que pode ser introduzida no futuro.

Depois de todos os objetos terem sido instanciados e inicializados com os seus nomes e respetivos identificadores únicos, é executado o método *ApplyLinkage*. Este método tem como objetivo ligar os objetos *FSMTransition*, das transições criadas anteriormente aos vários objetos *FSMState* de forma a criar uma estrutura virtual relacionada em que cada transição passa a possuir um apontador para os respetivos estados de partida e chegada. Se a ligação fosse feita *on-the-fly* o ficheiro não poderia ser lido de forma sequencial e teria de ser lido aleatoriamente, e a pesquisa de estados teria de ser feita através dos seus identificadores. Fazer comparações de identificadores utilizando a biblioteca XML e diretamente do disco é uma tarefa em si muito dispendiosa e pouco eficiente do ponto de vista programático.

4.4 Fase de geração de código

4.4.1 Estrutura da classe

Com o término da fase de interpretação inicia-se a fase de escrita. A classe responsável por esta fase é a *CppWriter* (Figura 24). Esta é composta, essencialmente, por métodos contendo apenas um campo denominado de *outputString* que é um *buffer* para o qual todo o código gerado é escrito antes de ser exportado para o ficheiro *.cpp de saída.

Quanto aos métodos presentes nesta classe, existe um total de vinte e seis métodos, vinte e quatro dos quais são métodos internos (privados) à classe, sendo que a única interação possível com

o utilizador é através dos métodos: *CppWriter* que é o construtor da classe não possuindo nenhum argumento e; *WriteStateMachine* que recebe um objeto FSM e o local onde deve ser criado o ficheiro *.Cpp. Os métodos podem ser divididos em dois tipos diferentes: métodos *Get* (procura) e; métodos

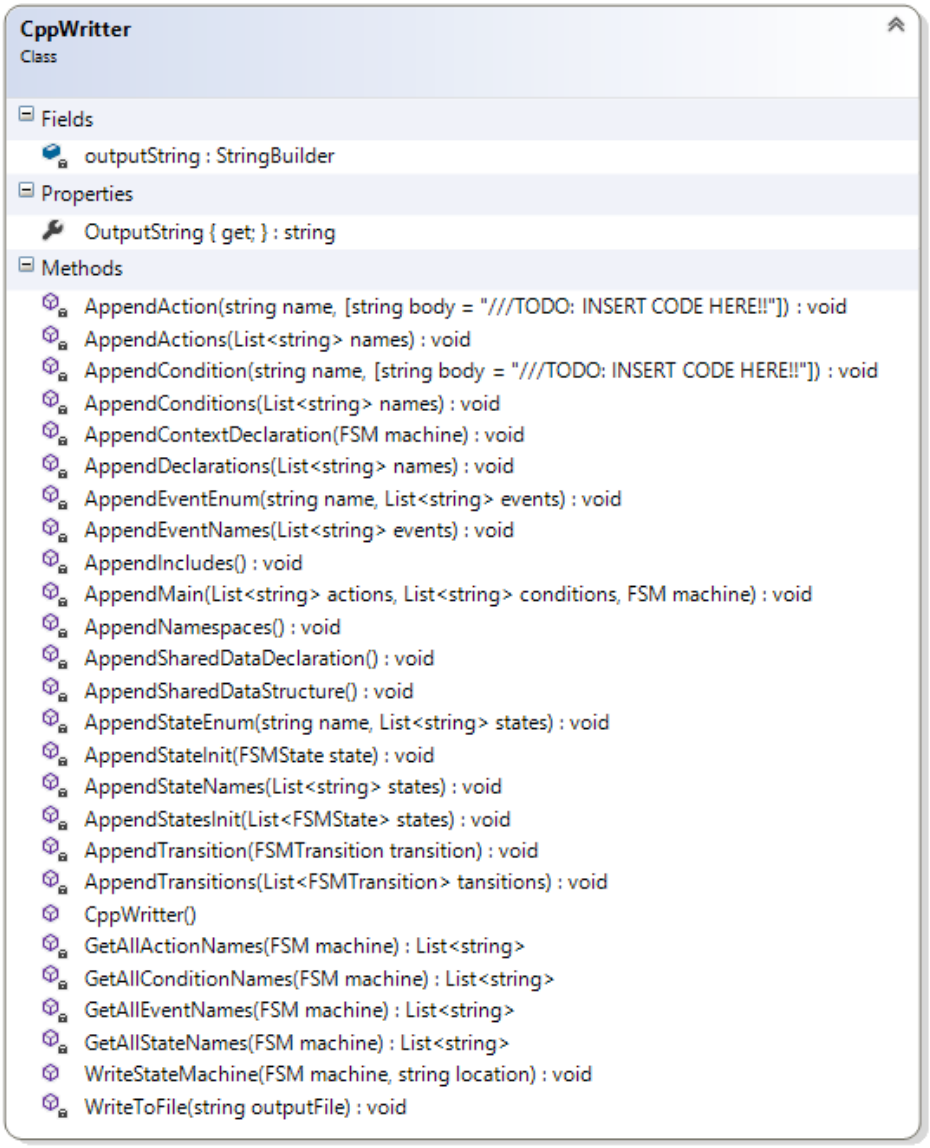


Figura 24 - Classe CppWriter

Append (adição de texto). Os métodos de procura – *GetAllActionNames*; *GetAllConditionNames, GetAllEventNames* e; *GetAllStateNames* – são métodos que acedem à estrutura *FSM* e procuram todos os elementos de um dado tipo retornando uma lista de nomes devidamente formatados. Por sua vez os métodos de adição de texto – *Append[*]* – são métodos que interagem diretamente com o *buffer* de saída e adicionando-lhe o código gerado. Existem, contudo, quatro exceções nesta arquitetura – *AppendConditions*, *AppendActions*, *AppendTransitions*, *AppendStatesInit* – são métodos que invocam

de forma recursiva os seus homónimos no singular (*AppendCondition, AppendAction, AppendTransition, AppendStateInit*).

Por fim, o método *WriteToFile* é o ultimo a ser executado e a sua tarefa é escrever para o disco todo o texto existente no *buffer,* recebendo como argumentos o caminho e nome do ficheiro a ser gerado.

4.4.2 Relação entre métodos

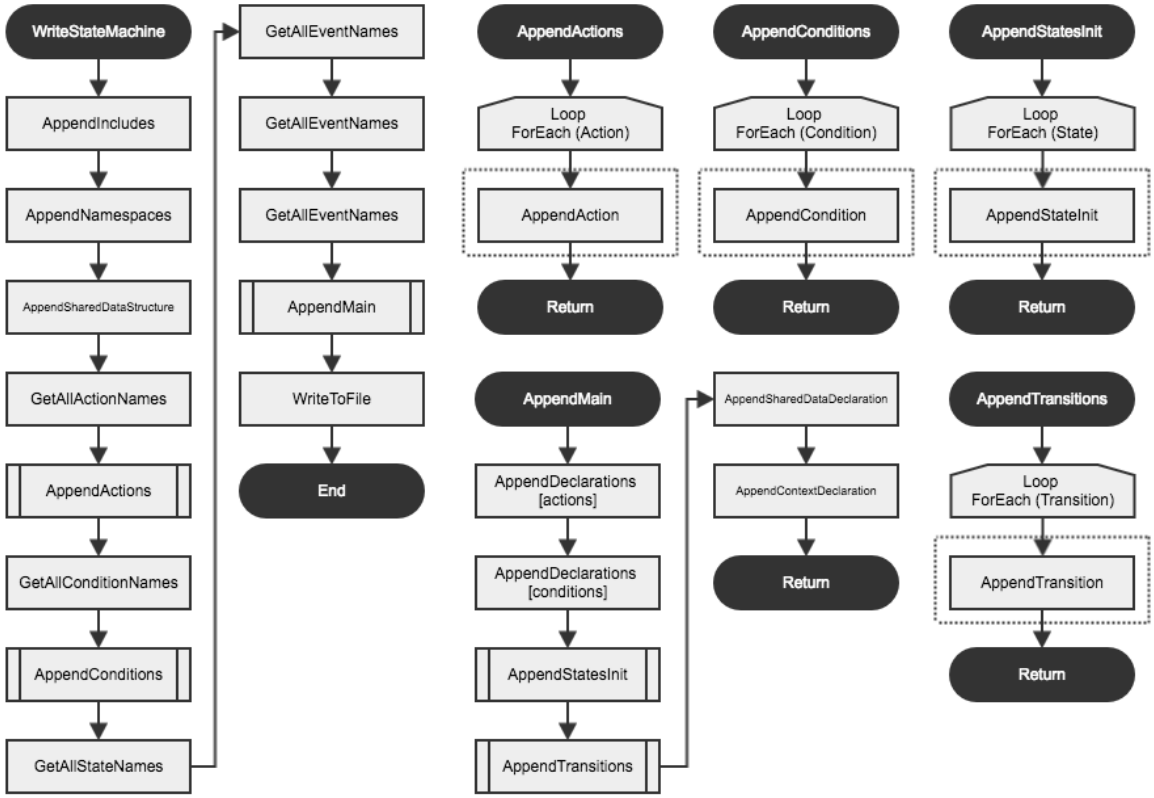


Figura 25 - Fluxograma do componente de escrita do ficheiro Cpp

Seguindo a ordem de acontecimentos do fluxograma da Figura 25, os primeiros métodos de escrita a serem executados, são uma série de métodos estáticos que não possuem quaisquer argumentos de entrada. Eles apenas existem para manter todo o código organizado. O primeiro destes métodos é o *AppendIncludes*, que adiciona as linhas de inclusão de ficheiros cabeçalho das bibliotecas necessárias para que o código gerado possa ser compilado. Estas bibliotecas compreendem a *Framework*, e a biblioteca padrão C++. O próximo método é o *AppendNamespaces* que tem como

objetivo adicionar as linhas "*using*" dos *namespaces* da *Framework FSM* e da biblioteca padrão do C++ ao código gerado.

Terminado o processo de inicial, é executado o método *AppendSharedDataStructure*, que é responsável por gerar o código correspondente à definição da estrutura de dados para partilhar informação entre as várias ações e condições da máquina de estados. Esta estrutura é criada por derivação da classe *Object*, não possuindo quaisquer membros.

A etapa que se segue diz respeito à geração do código pertencente à máquina de estado propriamente dita. Para que tal seja possível o primeiro passo é percorrer os estados criados na fase anterior (4.2) e criar uma lista de nomes de todas as ações dos mesmos. Este processo é levado a cabo pelo método *GetAllActionNames*, que é essencial para os dois métodos seguintes. Os dois métodos seguintes são invocados de forma hierárquica e visam utilizar a recursividade da escrita de forma eficiente de modo a criar código que possa ser reutilizado. O primeiro método é o *AppendActions*, *este* método recebe como argumentos a lista gerada pela função anterior e para cada nome na lista executa o método *AppendAction*. Este último método recebe como argumento um nome de uma ação e procede à geração do código de declaração de uma classe com o mesmo nome, que deriva da classe *FSMAction* da *Framework FSM*, e implementa o método *execute*. Este método fica vazio e é um *placeholder*, onde mais tarde deve ser, manualmente, introduzido o código da ação.

A *GetAllConditionNames* percorre toda as transições e devolve uma lista que contém os nomes de todas as condições. O método é o *AppendConditions*, *este* método recebe a lista e para cada nome executa o método *AppendCondition*. Este último método, gera o código para uma classe com o mesmo nome, que deriva de *FSMCondition* da *Framework FSM* e implementa o método *isSatisfied*. Este método fica vazio (apenas devolve *true* para permitir a compilação) e é um *placeholder*, onde mais tarde deve ser, manualmente, introduzido o código da condição.

Depois de definidas as ações e as condições, é necessário gerar o código dos enumerados referentes aos estados e eventos. De forma idêntica ás ações e ás condições são utilizados os métodos *GetAllStateNames* e *GetAllEventNames* para obter respetivamente a lista de nomes de todos os estados e eventos existentes na MEF. Estas listas são passadas respetivamente aos métodos *AppendStateEnum* e *AppendEventEnum*, que geram os respetivos enumeradores: os dos estados para

utilizar na invocação dos construtores dos objetos estado, e os dos eventos para utilizar na adição de transições.

Todo o código gerado até este momento é global ao ficheiro fonte e trata apenas de tipos de objetos/dados que são necessários. Falta gerar o código que cria objetos e os interliga para formar a MEF e a por em execução. Foi escolhido escrever esse código na função *main*, permitindo assim a execução do código gerado. A escrita desse código é iniciada com a chamada do método *AppendMain*. Este método é estrutural (ou seja, é este método que vai invocar os demais métodos a partir deste ponto) e serve para manter o código organizado, sendo o corpo gerado pelos métodos descritos a seguir.

O primeiro passo do corpo da função *main* é instanciar as condições e as ações, o que é levado a cabo pelo método *AppendDeclarations*. O método recebe uma lista de nomes e é executado por duas vezes. Na primeira iteração a lista de nomes utilizada é a lista de condições e na segunda iteração é passada, nos argumentos, a lista de nomes das ações. Note-se que as linhas de código que criam estes objetos só diferem no nome da classe, e desta forma o mesmo método serve propósitos diferentes durante a declaração de variáveis. O nome das variáveis é obtido recorrendo à propriedade *VarName* dos respetivos elementos.

O passo que se segue na escrita da função *main* é a instanciação dos estados através do método *AppendStatesInit*. Este método recebe a lista de estados e gera a invocação do construtor com os argumentos correspondentes às ações de entrada ou de saída, previamente declaradas, recorrendo para isso à propriedade *VarName* de cada estado que devolve o nome devidamente formatado. Caso não haja ação associada, à entrada ou saída do estado, é atribuída uma ação pré-definida pela *Framework* que salta o processo de execução sem que haja qualquer ação. Note-se que a *Framework* não possui a implementação para ações de permanência.

Depois de inicializados os estados, a cada um deles são associadas as transições respetivas. O método *AppendTransitions*, trata deste processo, recebendo uma lista de transições *FSMTransition* e gerando uma linha por cada transição (recorrendo ao método *AppendTransition*), que a adiciona ao respetivo estado de partida tendo como argumentos o evento que a desencadeia, o estado de destino, a ação que lhe está associada e a respetiva condição de guarda.

Depois de a MEF estar definida, de acordo com a *Framework FSM*, é necessário criar o objeto responsável pela troca de dados entre as várias ações e condições. Isso é feito pelo método *AppendSharedDataDeclaration*, a informação adicionada por este método é *hard-coded* uma vez que a nome é sempre o mesmo, contudo é mantido num método separado caso seja preciso fazer alterações em trabalho futuro. Por fim, o último passo para que a máquina de estados possa ser executada é a criação de um contexto. O método responsável por esta instanciação é o *AppendContextDeclaration*, e para isso cria um objeto do tipo *FSMContext* e passa-lhe como argumentos o nome do estado inicial e o objeto de dados criado na linha anterior. A ação de transição inicial passada como argumento é uma ação vazia, sendo que este processo terá de ser implementado em trabalho futuro.

5 Resultados

Neste capítulo são apresentados e comentados, os resultados de código gerado com a ferramenta desenvolvida para dois exemplos de diagramas de máquina de estados. Os exemplos são crescentes em complexidade, de forma a provar que a ferramenta está apta para geração de código, nas mais diversas circunstâncias.

5.1 Máquina de estados de *Login*

5.1.1 Modelo

O primeiro exemplo, ilustrado na Figura 26, é o de uma máquina de estados de um sistema de login. É uma máquina simples, composta apenas por três estados – *Login*, *Administrator* e *Block*. O estado inicial é o estado de *Login*. Aquando da transição para este estado é executada uma ação de entrada – *CheckBlocked*. Esta ação tem como função verificar o número de vezes que o código foi mal introduzido. Se o número de vezes ultrapassar um dado valor estipulado, a ação desencadeia o evento – *BlockedEvent* – que provocará a transição para o estado *Block*, estado que por não ter ações nem transições de partida a ele associado, bloqueia o sistema na tentativa de evitar uma fraude.

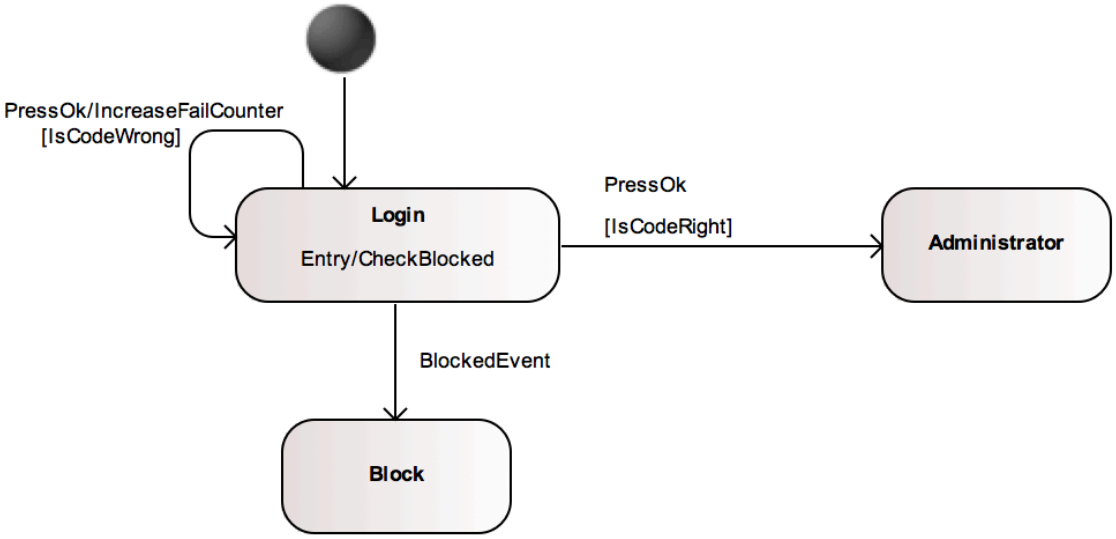


Figura 26 - Primeiro exemplo; diagrama de estados de um sistema de autenticação

Outro evento que pode ocorrer quando o estado atual é o estado *Login* é o *PressOk*, que corresponde ao ato de confirmação das credenciais introduzidas por parte do utilizador. Contudo, o evento *PressOk*, por si só não provoca nenhuma transição pois as transições por ele desencadeadas

têm associadas uma condição de guarda e conforme a resposta da condição de guarda o estado final da transição é diferente. Caso a condição de guarda *IsCodeRight* seja verdadeira é desencadeada a transição para o estado *Administrator*, significando que o utilizador consegue autenticar-se com sucesso no sistema e está agora em modo de administrador. Por outro lado, se a condição *IsCodeWrong* for verdadeira, é desencadeada a transição para o mesmo estado (*Login*). Durante a transição é executada a ação *IncreaseFailCounter* a ela associada. Esta transição e ação formam um ciclo de contagem do número de tentativas falhadas.

5.1.2 Código Gerado

Utilizando o ficheiro de exportação XMI do diagrama apresentado na Figura 26 (Anexo V), como entrada, a ferramenta gera o código da Figura 27, correspondente á sua implementação instanciando a *Framework FSM*.

```
class CheckBlocked : public FSMAction {
        void execute(const FSMEvent &ev, FSMContext &c) {
                ///TODO: INSERT CODE HERE!!
        }
};

class IncreaseFailCounter : public FSMAction {
        void execute(const FSMEvent &ev, FSMContext &c) {
                ///TODO: INSERT CODE HERE!!
        }
};

class IsCodeRight : public FSMCondition {
        bool isSatisfied(const FSMEvent &ev, FSMContext &c) {
                ///TODO: INSERT CODE HERE!!
                return true;
        }
};

class IsCodeWrong : public FSMCondition {
        bool isSatisfied(const FSMEvent &ev, FSMContext &c) {
                ///TODO: INSERT CODE HERE!!
                return true;
        }
};
```

Figura 27 - Declaração de ações e condições; exemplo 1

As ações e condições são corretamente geradas, concretamente:

- Ação: [entrada no estado *Login*] – *CheckBlocked*
- Ação: [*Login* → *Login*] – *IncreaseFailCounter*
- Condição [*Login* → *Administrator*] – *IsCodeRight*
- Condição [*Login* → *Login*] – *IsCodeWrong*

```
enum StateID { LOGIN = 1, ADMINISTRATOR, BLOCK };

enum EventID { PRESS_OK = 1, BLOCKED_EVENT };
```

Figura 28 - Declaração dos enumeradores para estados e eventos; exemplo 1

Depois, aparecem os enumeradores dos estados e eventos. Existem três estados e dois eventos possíveis:

- Estados: *Login*, *Administrator*, *Block*
- Eventos: *PressOk*, *BlockedEvent*

Todos os estados e eventos são bem gerados e seguem as regras de formatação textual necessárias, da Figura 28.

```
int main() {

        CheckBlocked checkBlocked;
        IncreaseFailCounter increaseFailCounter;

        IsCodeRight isCodeRight;
        IsCodeWrong isCodeWrong;

        [...]
}
```

Figura 29 - Corpo do método "main" primeira fase, declaração de ações e condições; exemplo 1

O corpo do método *main*, por ser demasiado extenso, foi dividido em várias partes. Na primeira (Figura 29), pode observar-se a criação dos objetos das ações e condições acima definidas. Primeiro são declaradas as ações e depois as condições, sendo estes dois blocos separados por uma linha em branco para facilitar a leitura.

```
int main() {

        [...]

        State login(LOGIN, checkBlock);
        State administrator(ADMINISTRATOR);
        State block(BLOCK);

        [...]
}
```

Figura 30 - Corpo do método "main" segunda fase, inicialização dos estados; exemplo 1

A segunda parte do corpo do método *main* é a criação dos objetos estado, utilizando o construtor da classe. Esta instanciação, inicializa o enumerador correspondente e associa as respetivas ações de entrada e saída, caso existam. No caso em concreto (Figura 30), apenas o estado de *login* apresenta uma ação de entrada – *CheckBlocked*.

```
int main() {

        [...]

        login.addTransition(PRESS_OK, administrator, FSMSkipAction::instance,
                            isCodeRight);
        login.addTransition(BLOCKED_EVENT, block);
        login.addTransition(PRESS_OK, login, increaseFailCounter,
                            isCodeWrong);

        [...]
}
```

Figura 31 - Corpo de método "main" terceira fase, associação das transições e respetivas ações aos estados; exemplo 1

Depois de criados os estados, são-lhes adicionadas as transições. Comparando o diagrama UML da Figura 26 com o código gerado na Figura 31, é possível verificar que as três transições que partem do estado *Login*, são adicionadas corretamente, bem como as respetivas condições e ações, concretamente a ação *IncreaseFailCounter* da transição entre o estado *Login* e ele próprio.

```
int main() {
        [...]
        SharedData sd;
        FSMContext c(login, FSMSkipAction::instance, &sd);
        //ADD FSM FEED CODE HERE
        return 0;
}
```

Figura 32 - Corpo do método "main" quarta fase, inicialização de contexto; exemplo 1

A fase final do corpo do método *main* é a declaração do objeto de partilha de dados – *SharedData* – e a inicialização de um novo contexto para a máquina de estados (Figura 32), com o respetivo estado inicial, ação inicial e o próprio objeto para partilha de dados. No exemplo apresentado, o estado inicial é o estado *Login* e não existe ação de inicialização.

5.2 Máquina de estados de um Micro-ondas

5.2.1 Modelo

O segundo exemplo (Figura 33) é um pouco mais complexo e lida com o funcionamento de um micro-ondas. O estado inicial do sistema é o estado *Idle*, este estado é responsável por fazer aparecer as horas no ecrã do aparelho. Deste estado podem ocorrer duas transições para o estado *Full Power On* ou *Half Power On*, dependendo do botão de intensidade de cozedura que é pressionado. Esses botões geram os eventos *Haft Power Button* e *Full Power Button*, que também permitem trocar directamente entre os estados *Haft Power On* e *Full Power On*, tornando-os, portanto, intermutáveis.

Após a rotação do temporizador o micro-ondas passa para o estado *Set Time*, que é responsável pela atualização do intervalo de tempo no visor. Este estado pode transitar para ele próprio ou para um dos estados *Operation Disabled* ou *Operation Enabled*, conforme a porta esteja aberta ou fechada, respetivamente.

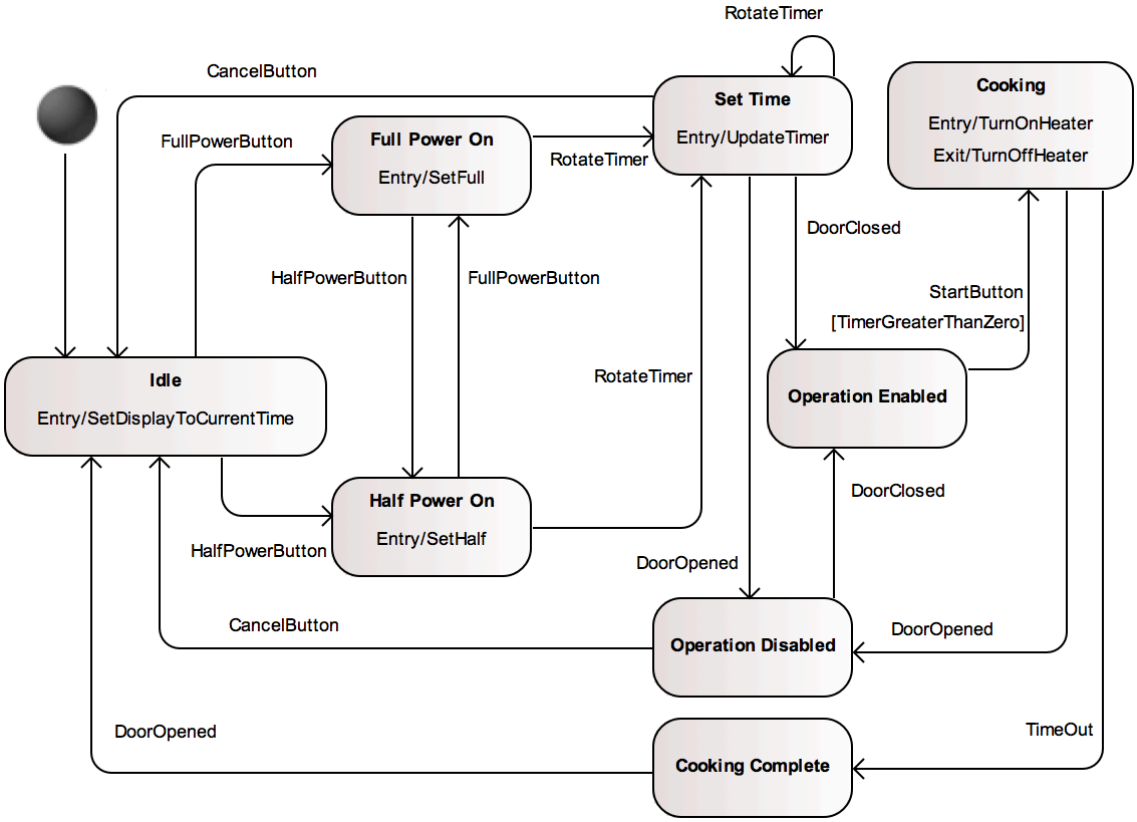


Figura 33 - Segundo exemplo diagrama de estados de um micro-ondas

Quando a máquina de estados se encontra no estado *Operation Enabled*, ou seja, pronto a operar, o evento *StartButton* – botão de início – desencadeia a transição que leva o micro-ondas a transitar para o estado *Cooking*, iniciando-se o processo de cozedura (*TurnOnHeater*). A transição anterior possui condição de guarda *TimeGreaterThanZero*, que é verdadeira caso o temporizador tenha um tempo superior a zero segundos. Uma vez no estado *Cooking*, o evento *TimeOut* lançado quando o temporizador atinge o valor zero desencadeia a transição para o estado *CookingComplete*, finalizando o processo de cozedura (*TurnOffHeater*). O estado *Cooking Complete* transita de novo para o estado inicial, após a abertura da porta, fechando desta forma o ciclo.

5.2.2 Código Gerado

Utilizando o ficheiro de exportação XMI do diagrama apresentado na Figura 33 (ver Anexo VII), foi gerado o código (ver Anexo VIII) correspondente á sua implementação para a *Framework FSM*, seguindo-se a sua demonstração.

As ações e condições geradas podem ser observados na Figura 34, e correspondem às do ficheiro XMI.

```
class SetDisplayToCurrentTime : public FSMAction {
        void execute(const FSMEvent &ev, FSMContext &c) {
                ///TODO: INSERT CODE HERE!!
        }
};

class SetFull : public FSMAction {   //...

class UpdateTimer : public FSMAction {   //...

class TurnOnHeater : public FSMAction {   //...

class TurnOffHeater : public FSMAction {   //...

class SetHalf : public FSMAction {   //...

class TimerGreaterThanZero : public FSMCondition {
        bool isSatisfied(const FSMEvent &ev, FSMContext &c) {
                ///TODO: INSERT CODE HERE!!
                return true;
        }
};
```

Figura 34 - Declaração de ações e condições; exemplo 2

Ação: [entrada no estado *Idle*] – *SetDisplayToCurrentTime*

- Ação: [entrada no estado *Full Power On*] – *SetFull*
- Ação: [entrada no estado *Set Timer*] – *UpdateTimer*
- Ação: [entrada no estado *Cooking*] – *TurnOnHeater*
- Ação: [saída do estado *Cooking*] – *TurnOffHeater*
- Ação: [entrada no estado *Half Power On*] – *SetHalf*
- Condição [*Operation Enabled* → *Cooking*] – *TimerGreaterThanZero*

```
enum StateID { IDLE = 1, FULL_POWER_ON, SET_TIME, COOKING, COOKING_COMPLETE,
               OPERATION_ENABLED, OPERATION_DISABLED, HALF_POWER_ON };

enum EventID { HALF_POWER_BUTTON = 1, ROTATE_TIMER, DOOR_OPENED, DOOR_CLOSED,
               CANCEL_BUTTON, TIME_OUT, START_BUTTON, FULL_POWER_BUTTON };
```

Figura 35 - Declaração dos enumeradores para estados e eventos; exemplo 2

Depois das ações e as transições, aparecem os enumeradores dos estados e eventos. Existem três estados e dois eventos possíveis:

- Estados: *Idle, Full Power On, Set Timer, Cooking, Set Timer, Cooking Complete, Operation Enabled, Operation Disabled, Half Power On*
- Eventos: *HalfPowerButton, RotateTimer, DoorOpened, DoorClosed, TimeOut, StartButton, FullPowerButton*

Todos os estados e eventos são bem gerados e seguem as regras de formatação textual pretendidas (Figura 35).

```
int main() {

        SetDisplayToCurrentTime setDisplayToCurrentTime;
        SetFull setFull;
        UpdateTimer updateTimer;
        TurnOnHeater turnOnHeater;
        TurnOffHeater turnOffHeater;
        SetHalf setHalf;

        TimerGreaterThanZero timerGreaterThanZero;

        [...]
}
```

Figura 36 - Corpo do método "main" primeira fase, declaração de ações e condições; exemplo 2

O corpo do método *main* por ser demasiado extenso foi dividido em vários blocos. No primeiro bloco podem observar-se (Figura 36) as instanciações das ações e condições acima definidas. Estas encontram-se separadas por uma linha em branco para facilitar a leitura, sendo que primeiro são declaradas as ações e depois as condições.

```
int main() {

        [...]

        State idle(IDLE, setDisplayToCurrentTime);
        State fullPowerOn(FULL_POWER_ON, setFull);
        State setTime(SET_TIME, updateTimer);
        State cooking(COOKING, turnOnHeater, turnOffHeater);
        State cookingComplete(COOKING_COMPLETE);
        State operationEnabled(OPERATION_ENABLED);
        State operationDisabled(OPERATION_DISABLED);
        State halfPowerOn(HALF_POWER_ON, setHalf);

        [...]
}
```

Figura 37 - Corpo do método "main" segunda fase, inicialização dos estados; exemplo 2

O segundo bloco do corpo do método *main* é a inicialização dos estados, que passa pela atribuição do enumerador correspondente e associação das ações de entrada e saída, caso existam.

```
int main() {

        [...]

        fullPowerOn.addTransition(HALF_POWER_BUTTON, halfPowerOn);
        fullPowerOn.addTransition(ROTATE_TIMER, setTime);
        setTime.addTransition(DOOR_OPENED, operationDisabled);
        setTime.addTransition(DOOR_CLOSED, operationEnabled);
        setTime.addTransition(CANCEL_BUTTON, idle);
        setTime.addTransition(ROTATE_TIMER, setTime);
        cooking.addTransition(TIME_OUT, cookingComplete);
        cooking.addTransition(DOOR_OPENED, operationDisabled);
        cookingComplete.addTransition(DOOR_OPENED, idle);
        operationEnabled.addTransition(START_BUTTON, cooking,
                FSMSkipAction::instance, timerGreaterThanZero);
        operationDisabled.addTransition(DOOR_CLOSED, operationEnabled);
        operationDisabled.addTransition(CANCEL_BUTTON, idle);
        halfPowerOn.addTransition(FULL_POWER_BUTTON, fullPowerOn);
        halfPowerOn.addTransition(ROTATE_TIMER, setTime);
        idle.addTransition(FULL_POWER_BUTTON, fullPowerOn);
        idle.addTransition(HALF_POWER_BUTTON, halfPowerOn);

        [...]
}
```

Figura 39 - Corpo de método "main" terceira fase, associação das transições e respetivas ações aos estados; exemplo 2

No caso em concreto (Figura 37) apenas os estados *Login*, *OperationEnabled* e *OperationDisabled*, não possuem ações associadas e o estado *Cooking* apresenta duas, uma de entrada – *TurnOnHeater* – e uma de saída – *TurnOffHeater*.

```
int main() {

        [...]

        SharedData sd;

        FSMContext c(idle, FSMSkipAction::instance, &sd);

        //ADD FSM FEED CODE HERE

        return 0;
}
```

Figura 38 - Corpo do método "main" quarta fase, inicialização de contexto; exemplo 2

Depois de declarados, são adicionadas aos estados, as transições (Figura 38). Conforme o diagrama UML da máquina de estados (e o XMI no Anexo VII), o código gerado apresenta um total de 16 transições possíveis entre estados, e estas são adicionadas corretamente. A única condição existente é *TimerGreaterThanZero*, que é corretamente associada à transição de *OperationEnabled* para *Cooking*.

O bloco final do corpo do método *main* é a declaração do objeto de partilha de dados – *SharedData* – e a inicialização de um novo contexto para a máquina de estados (Figura 39), com o respetivo estado inicial, ação de entrada e o próprio objeto para partilha de dados. No exemplo apresentado, o estado inicial é o estado *Idle* e não existe ação de inicialização.

5.3 Procedimentos Pós-Geração

Após ter sido gerado o código através da ferramenta, este, apesar de pronto a compilar não apresenta nenhuma função implementada. Essa etapa é manual, e deve proceder-se à substituição dos comentários ("///TODO: INSERT CODE HERE") das ações e condições, pela respetiva lógica a ser executada. A alimentação da máquina de estados pelos vários eventos, deve ser acrescentada, no corpo do método *main,* através da implementação de um ciclo, substituindo o comentário respetivo ("//ADD FSM FEED CODE HERE").

6 Conclusão

Neste capítulo tecem-se as observações e considerações finais ao projeto realizado. Este capítulo está dividido em três secções: a primeira analisa e esclarece alguns tópicos referente aos resultados obtidos; a segunda apresenta as conclusões do projeto tendo em conta uma visão retrospetiva de todo o processo; e, por fim, a terceira aponta algumas ideias no que diz respeito a trabalho futuro.

6.1 Discussão de resultados

A ferramenta desenvolvida foi testada com uma diversidade de diagramas de UML, tendo gerado código correto para uso com a *Framework FSM* e adequadamente formatado segundo as convenções. Os testes experimentais comprovam o funcionamento da mesma, pois o código compila e executa sem problemas. Os ficheiros XMI, e código C++ gerado encontram-se apresentados na integra na secção de anexos.

A ferramenta depende da *Framework MONO* que, no entanto, se encontra disponível para os sistemas operativos *Windows*, *Linux* e *OSX*. O IDE utilizado foi o *Xamarin Studio*, contudo a edição do código da ferramenta não está limitada pelo IDE utilizado uma vez que o projeto pode ser aberto e até compilado utilizando outros IDEs como por exemplo o *Visual Studio* (é necessária a instalação do *MONO* para *Visual Studio*, também disponível em versão *open-source*).

A compatibilidade da ferramenta com vários ficheiros XMI de diferentes aplicações é parcialmente assegurada sendo, por vezes, necessárias algumas alterações no código fonte no que diz respeito aos *namespaces* utilizados no ficheiro XMI. Contudo, e uma vez que o XMI não é suposto ser uniforme entre ferramentas, pois é apenas um conjunto de regras para gerar modelos UML e não um formato concreto [9], a compatibilidade da ferramenta com vários ficheiros XMI é limitada a alguns *softwares CASE*. Neste caso especifico foram efetuados testes bem-sucedidos utilizando o *software Modelio* (*software* para o qual o gerador foi desenvolvido) e ainda para ficheiros de exportação do *software UModel* da *Altova*.

Outro aspeto, de grande importância, diz respeito ao desempenho da aplicação. Com a finalidade de avaliar este aspeto, a aplicação foi sujeita a um teste de *stress*, que consistiu na geração

de código para um diagrama de grandes dimensões. O diagrama utilizado neste teste bem como o respetivo XMI e código C++ gerado encontram-se na integra nos anexos IX, X e XI respetivamente. Foi possível verificar que os dois componentes da ferramenta – interpretador do ficheiro XMI e gerador do ficheiro CPP – apresentam tempos de processamento muito baixos.

- Interpretador do ficheiro XMI – média de 500.000 iterações (62,42 milissegundos)
- Gerador do ficheiro CPP – media de 500.000 iterações (13,32 milissegundos)

Conclui-se que a ferramenta apresenta um bom desempenho computacional.

6.2 Conclusões

Num presente informático em constante necessidade de adaptação e mudança, é extremamente importante a rápida criação e atualização de código fonte no desenvolvimento de sistemas informáticos. Embora a geração automática de código seja um tema já com alguma abrangência nos *softwares* comerciais, esta abordagem é *software centered* e implica a reutilização do *software* de forma a que o código seja atualizado.

Os objetivos da dissertação no que diz respeito ao desenvolvimento de uma ferramenta capaz de apresentar uma alternativa de geração de código face aos demais *softwares* comerciais existentes, foram atingidos. A ferramenta oferece não só uma alternativa, mas também uma nova filosofia. É possível afirmar que a geração de código para *frameworks* específicas, especializadas num dado conjunto de comportamentos pode ser considerada como uma grande aposta futura do desenvolvimento de *software*, facilitando muito o desenvolvimento de ferramentas e permitindo que o código possa ser facilmente lido e interpretado e editado por humanos.

O principal desafio desta implementação foi a abordagem ao processo de geração de código. Esta abordagem levou à criação de uma ferramenta modular que separa a geração de código nas componentes de interpretação do ficheiro XMI e geração do código para um ficheiro C++. Esta abordagem tem a vantagem de possibilitar uma maior facilidade na manutenção do código da ferramenta, e permitir que o *debug* seja feito por partes tornando, por isso, também mais fácil a correção de erros.

Uma das limitações da ferramenta assenta na existência de várias versões de XMI e UML que podem ser utilizadas para a exportação do modelo UML, que por sua vez dependem do *software* de modelação utilizado. Uma vez que é difícil encontrar pontos comuns nas várias formatações utilizadas, esta limitação poderia ser ultrapassada criando tradutores específicos para cada *software CASE*. Com a criação de uma ferramenta intermédia que proceda à preparação do XMI para que este possa ser utilizado na ferramenta de geração de código.

Outro ponto que deve ser mencionado é que a ferramenta foi concebida para a *Framework FSM*, tendo algumas funcionalidades do UML relativas aos diagramas de estados (como implementação de pós-condições de transição, ações de loop do estado – *doActivity* – e máquinas de estado hierárquicas), sido deixadas de fora da implementação da mesma, uma vez que não poderiam ser testadas. Esta decisão foi tomada no início do desenvolvimento do código, contudo, e dado à arquitetura modular da ferramenta, esta pode ser facilmente estendida para incorporar essas funcionalidades no módulo de interpretação embora não tenham sido utilizadas durante o processo de escrita (como o que acontece por exemplo com as ações de repetição do estado, estas são lidas e interpretadas pelo módulo de interpretação e não são utilizadas no módulo de escrita).

6.3 Trabalho futuro

A ferramenta de geração de código está sempre sujeita a melhoramentos e acréscimo de funcionalidades, de forma a crescer juntamente com o desenvolvimento da *Framework FSM*.

As principais funcionalidades a serem acrescentadas passam pela:

- suporte completo das ações, nomeadamente a ação da transição inicial;
- implementação de geração de código para máquinas de estados hierárquicas, permitindo elaborar sistemas complexos por níveis;
- aumentar o leque de linguagens de exportação de forma a permitir geração de código em Java;
- possibilitar inserção de código ao nível da especificação UML, permitindo que as especificações das ações e condições sejam feitas de forma contextualizada, reduzindo a necessidade de edição posterior do código gerado;

- capacidade de engenharia reversa, para que alterações manuais no código reflitam alterações no ficheiro XMI e consequente atualização dinâmica do modelo UML utilizado (*round-trip engineering*).

Referências Bibliográficas

[1] (OMG), Object Management Group, "OMG Unified Modeling Language TM (OMG UML)," 1 Março 2015. [Online]. Available: http://www.omg.org/spec/UML/2.5.

[2] S. Lopes, S. Silva e J. Monteiro, "An easy-to-use and flexible Object-Oriented Framework for Extended Finite State Machines," *IEEE International Conference on Industrial Informatics,* pp. 35-40, 27 Julho 2012.

[3] Eclipse, "Eclipse Modeling - Model Development Tools," 2016. [Online]. Available: http://www.eclipse.org/modeling/mdt/?project=uml2#uml2.

[4] Altova, "Working with state machine code," 2016. [Online]. Available: http://manual.altova.com/umodel/umodelbasic/index.html?umworking_with_state_machine_cod.htm.

[5] Sparx Systems, "UML Tools for software development and modelling," SparxSystems, [Online]. Available: http://www.sparxsystems.com.au/. [Acedido em 2016].

[6] M. Voelter, "A Catalog of Patterns for Program Generation," 14 Abril 2003. [Online]. Available: http://www.voelter.de/data/pub/ProgramGeneration.pdf.

[7] P. Metz, J. O´Brien e W. Weber, "Code Generation Concepts for Statechart Diagrams of the UML v1.1," em *Object Technology Group (OTG) Conference*, Vienna, 1999.

[8] Modeliosoft, "Modelio Open-source - UML and BPMN free modeling tool," 2016. [Online]. Available: https://www.modelio.org/. [Acedido em 1 12 2015].

[9] B. Marchal, "Working XML: UML, XMI, and code generation, Part 2," 11 5 2004. [Online]. Available: https://www.ibm.com/developerworks/library/x-wxxm24.

[10] (OMG), Object Management Group, "XML Metadata Interchange (XMI) Specification," 4 Abril 2014. [Online]. Available: http://www.omg.org/spec/XMI/2.4.2.

[11] Sparx Systems, "SW Code Generation - State Machine Diagrams," 2011. [Online]. Available: http://www.sparxsystems.com/enterprise_architect_user_guide/9.3/software_engineering/code_generation___state_machin.html.

Anexos

Anexo I Código fonte gerado pela ferramenta UModel (Altova)

Anexo II Código fonte gerado pela ferramenta Enterprise Architect (SparxSystems)

Anexo III Código fonte gerado pela ferramenta Visual Paradigm

Anexo IV Código fonte de exemplo de utilização da Framework

Anexo V Ficheiro XMI do diagrama utilizado no primeiro exemplo dos resultados

Anexo VI Código fonte gerado para o primeiro exemplo dos resultados

Anexo VII Ficheiro XMI do diagrama utilizado no segundo exemplo dos resultados

Anexo VIII Código fonte gerado para o segundo exemplo dos resultados

Anexo IX Diagrama de máquina de estados utilizado para teste de stress da ferramenta

Anexo X Ficheiro XMI do diagrama utilizado para teste de stress

Anexo XI Código fonte gerado para o diagrama de teste de stress da ferramenta

Anexo I

Código fonte gerado pela ferramenta UModel (Altova)

```
1  ////////////////////////////////////////////////////////////////////////
2  //
3  // AirConditionController.java
4  //
5  // This file was generated by UModel 2017 Enterprise Edition
6  //
7  // YOU SHOULD NOT MODIFY THIS FILE, BECAUSE IT WILL BE
8  // OVERWRITTEN WHEN YOU RE-RUN CODE GENERATION.
9  //
10 // Refer to the UModel Documentation for further details.
11 // http://www.altova.com/umodel
12 //
13 ////////////////////////////////////////////////////////////////////////
14 package SimpleGate;
15
16 import java.awt.event.ActionEvent;
17 import javax.swing.AbstractAction;
18 import javax.swing.Action;
19
20 @SuppressWarnings({"serial", "unused"})
21
22 public class SimpleGateController
23 {
24     public enum TStateId
25     {
26         SimpleGate_Root,
27         SimpleGate_MainRegion_Closed,
28         SimpleGate_MainRegion_Opened
29     }
30
31     public interface IState
32     {
33         String getName();
34         TStateId getId();
35         IRegion[] getRegions();
36     }
37
38     public interface IRegion
39     {
40         String getName();
41         IState[] getStates();
42         IState getCurrentState();
43     }
44
45     // get the Root State
46     public IState getRootState() {
47         return _getRootState();
48     }
49     public IState getCurrentTopLevelState() {
50         return getRootState().getRegions()[0].getCurrentState();
51     }
52
```

```
53      // Initialize the State Machine
54      public void Initialize() {
55          OnDebugMessage( "BEGIN_INITIALIZE" );
56          SimpleGate_MainRegion_Closed = new CSimpleGate.MainRegion.Closed(this);
57          SimpleGate_MainRegion_Opened = new CSimpleGate.MainRegion.Opened(this);
58
59          rootState = new CSimpleGate(this);
60
61          CSimpleGate.MainRegion.setCurrentState(this,
              CSimpleGate.MainRegion.getInitState(this));
62          OnDebugMessage( "END_INITIALIZE" );
63      }
64
65      // get all possible call event actions
66      public Action[] getCallEvents() {
67          return new Action[] {
68              new AbstractAction("CarInSensor") {
69                  public void actionPerformed(ActionEvent e) {
70                      CarInSensor();
71                  }
72              },
73              new AbstractAction("TimerFinished") {
74                  public void actionPerformed(ActionEvent e) {
75                      TimerFinished();
76                  }
77              }
78          };
79      }
80
81      // call event method implementations
82      public boolean CarInSensor() {
83          OnDebugMessage("BEGIN_EVENT: CarInSensor");
84          boolean bHandled = _getRootState().CarInSensor();
85          OnDebugMessage("END_EVENT: CarInSensor");
86          return bHandled;
87      }
88      public boolean TimerFinished() {
89          OnDebugMessage("BEGIN_EVENT: TimerFinished");
90          boolean bHandled = _getRootState().TimerFinished();
91          OnDebugMessage("END_EVENT: TimerFinished");
92          return bHandled;
93      }
94
95      // Overwrite to handle entry/exit/do actions, transition effects,...:
96      public void CloseGate() {OnDebugMessage("ACTION: CloseGate");}
97      public void OpenGate() {OnDebugMessage("ACTION: OpenGate");}
98      public void StartTimer() {OnDebugMessage("ACTION: StartTimer");}
99
100     // Overwrite to handle debug messages:
101     public void OnDebugMessage(String str) {
102     }
103
```

```
104     // Internal Definitions and Implementations
105
106     // Root State Definition
107     private CSimpleGate rootState = null;
108     private CSimpleGate _getRootState() {
109         if(rootState == null)
110             Initialize();
111
112         return rootState;
113     }
114
115     // State Variable Definitions
116     private CSimpleGate.MainRegion.Closed SimpleGate_MainRegion_Closed = null;
117     private CSimpleGate.MainRegion.Opened SimpleGate_MainRegion_Opened = null;
118
119     // Implementation for StateMachine 'SimpleGate'
120     private static class CSimpleGate implements IState
121     {
122         private SimpleGateController context;
123         private MainRegion.MainRegionState svMainRegion; // State Variable for
              Region 'MainRegion'
124
125         public CSimpleGate(SimpleGateController context) {
126             this.context = context;
127         }
128         public String getName() {
129             return "SimpleGate";
130         }
131         public TStateId getId() {
132             return TStateId.SimpleGate_Root;
133         }
134         public IRegion[] getRegions() {
135             return new IRegion[] {
136                 new MainRegion(context)
137             };
138         }
139         public boolean CarInSensor() {
140             return MainRegion.getCurrentState(context).CarInSensor();
141         }
142         public boolean TimerFinished() {
143             return MainRegion.getCurrentState(context).TimerFinished();
144         }
145
146         // Implementation for Region 'MainRegion'
147         public static class MainRegion implements IRegion
148         {
149             private SimpleGateController context;
150
151             public MainRegion(SimpleGateController context) {
152                 this.context = context;
153             }
154             public String getName() {
```

```
155                 return "MainRegion";
156             }
157             public IState[] getStates() {
158                  return new IState[] {
159                     new Closed(context),
160                     new Opened(context)
161                 };
162             }
163             public IState getCurrentState() {
164                 return getCurrentState(context);
165             }
166             public static MainRegionState getCurrentState(SimpleGateController
                  context) {
167                 return context._getRootState().svMainRegion;
168             }
169             public static void setCurrentState(SimpleGateController context,
                  MainRegionState state) {
170                 context.OnDebugMessage( "SET_CURRENT_STATE: " + state.getName
                    () );
171                 context._getRootState().svMainRegion = state;
172                 context._getRootState().svMainRegion.entryState();
173             }
174             public static MainRegionState getInitState(SimpleGateController
                  context) {
175                 return context.SimpleGate_MainRegion_Closed;
176             }
177
178             // Base class for all states of Region 'MainRegion'
179             public static abstract class MainRegionState implements IState
180             {
181                 protected SimpleGateController context;
182                 public boolean CarInSensor() {return false;}
183                 public boolean TimerFinished() {return false;}
184                 public void entryState() {}
185                 public void exitState() {}
186             }
187
188             // Implementation for State 'Closed'
189             public static class Closed extends MainRegionState
190             {
191                 public Closed(SimpleGateController context) {
192                     this.context = context;
193                 }
194                 @Override
195                 public String getName() {
196                     return "Closed";
197                 }
198                 @Override
199                 public TStateId getId() {
200                     return TStateId.SimpleGate_MainRegion_Closed;
201                 }
202                 @Override
```

```
203                 public IRegion[] getRegions() {
204                     return new IRegion[] {
205                     };
206                 }
207                 @Override
208                 public boolean CarInSensor() {
209                     // Handle Closed ---> Opened
210                     context.OnDebugMessage( "TRANSITION: Closed ---> Opened");
211                     exitState();
212                     CSimpleGate.MainRegion.setCurrentState(context,
                         context.SimpleGate_MainRegion_Opened);
213                     return true;
214                 }
215                 @Override
216                 public void entryState() {
217                     context.CloseGate();
218                 }
219                 @Override
220                 public void exitState() {
221                     context.OpenGate();
222                 }
223             }
224 
225             // Implementation for State 'Opened'
226             public static class Opened extends MainRegionState
227             {
228                 public Opened(SimpleGateController context) {
229                     this.context = context;
230                 }
231                 @Override
232                 public String getName() {
233                     return "Opened";
234                 }
235                 @Override
236                 public TStateId getId() {
237                     return TStateId.SimpleGate_MainRegion_Opened;
238                 }
239                 @Override
240                 public IRegion[] getRegions() {
241                     return new IRegion[] {
242                     };
243                 }
244                 @Override
245                 public boolean CarInSensor() {
246                     // Handle Opened ---> Opened
247                     context.OnDebugMessage( "TRANSITION: Opened ---> Opened");
248                     CSimpleGate.MainRegion.setCurrentState(context,
                         context.SimpleGate_MainRegion_Opened);
249                     return true;
250                 }
251                 @Override
252                 public boolean TimerFinished() {
```

```
253                     // Handle Opened ---> Closed
254                     context.OnDebugMessage( "TRANSITION: Opened ---> Closed");
255                     CSimpleGate.MainRegion.setCurrentState(context,
                         context.SimpleGate_MainRegion_Closed);
256                     return true;
257                 }
258                 @Override
259                 public void entryState() {
260                     context.StartTimer();
261                 }
262             }
263         }
264     }
265 }
```

Anexo II

Código fonte gerado pela ferramenta Enterprise Architect (SparxSystems)

```
1  //package SimpleGate;   //Comment out for EA Statemachine simulation purpose
2
3  import java.util.*;
4
5
6  /**
7   * @author FranciscoDias
8   * @version 1.0
9   * @created 24-Oct-2016 18:09:42
10  */
11 public class SimpleGateController implements StateMachineContext {
12
13     /* Begin - EA generated code for StateMachine */
14
15
16     enum StateMachineEnum
17     {
18         NOSTATEMACHINE,
19         SimpleGateController_ENUM_STATEMACHINE
20     };
21
22     enum StateEnum
23     {
24         NOSTATE,
25         SimpleGateController_VIRTUAL_SUBMACHINESTATE,
26         SimpleGateController_ENUM_STATEMACHINE_CLOSED,
27         SimpleGateController_ENUM_STATEMACHINE_OPENED
28     };
29
30     enum TransitionEnum
31     {
32         NOTRANSITION,
33         SimpleGateController_ENUM_OPENED__TO__OPENED_6,
34         SimpleGateController_ENUM_OPENED__TO__CLOSED_7,
35         SimpleGateController_ENUM_CLOSED__TO__OPENED_5,
36         SimpleGateController_ENUM_INITIAL_15__TO__CLOSED_8
37     };
38
39     enum EntryEnum
40     {
41         NOENTRY,
42         SimpleGateController_ENUM_STATEMACHINE_INITIAL_15
43     };
44
45     public StateMachineImpl m_StateMachineImpl;
46
47     public SimpleGateController()
48     {
49         m_StateMachineImpl = null;
50     }
51
52     protected void finalize( ) throws Throwable
```

```
53      {
54      }
55
56      public SimpleGateController(ContextManager pManager, String sInstanceName)
57      {
58          //Initialize Region Variables
59          m_statemachine = StateEnum.NOSTATE;
60          m_sInstanceName = sInstanceName;
61          m_sType = "SimpleGateController";
62          m_StateMachineImpl = new StateMachineImpl(this, pManager);
63      }
64
65      public String m_sInstanceName;
66      public String m_sType;
67
68      public String GetInstanceName()
69      {
70          return m_sInstanceName;
71      }
72
73      public String GetTypeName()
74      {
75          return m_sType;
76      }
77
78      public boolean defer(Event event, StateData toState)
79      {
80          if (m_StateMachineImpl == null)
81              return false;
82
83          boolean bDefer = false;
84
85
86          if(!bDefer)
87          {
88              if(toState.parent_state != null)
89              {
90                  bDefer = defer(event, toState.parent_state);
91              }
92          }
93          return bDefer;
94      }
95
96      public void TransitionProc(TransitionEnum transition, Signal signal,
          StateData submachineState)
97      {
98          if (m_StateMachineImpl == null)
99              return;
100
101
102         switch (transition)
103         {
```

```
104             case SimpleGateController_ENUM_OPENED__TO__OPENED_6:
105                 m_StateMachineImpl.currentTransition.SetTransition
                      (transition.ordinal(), submachineState,
                      "SimpleGateController_Opened__TO__Opened_6", "{8AC257FB-
                      FBCA-4155-94E2-662CD993F3A6},");
106                 Opened__TO__Opened_6(signal, submachineState);
107                 break;
108             case SimpleGateController_ENUM_OPENED__TO__CLOSED_7:
109                 m_StateMachineImpl.currentTransition.SetTransition
                      (transition.ordinal(), submachineState,
                      "SimpleGateController_Opened__TO__Closed_7",
                      "{1C88528A-00DA-4766-9BF3-68EC4A5C43AB},");
110                 Opened__TO__Closed_7(signal, submachineState);
111                 break;
112             case SimpleGateController_ENUM_CLOSED__TO__OPENED_5:
113                 m_StateMachineImpl.currentTransition.SetTransition
                      (transition.ordinal(), submachineState,
                      "SimpleGateController_Closed__TO__Opened_5", "{F915A775-
                      AC46-4c41-8898-215C1BFC1F03},");
114                 Closed__TO__Opened_5(signal, submachineState);
115                 break;
116             case SimpleGateController_ENUM_INITIAL_15__TO__CLOSED_8:
117                 m_StateMachineImpl.currentTransition.SetTransition
                      (transition.ordinal(), submachineState,
                      "SimpleGateController_Initial_15__TO__Closed_8",
                      "{C7962E15-66B7-4030-AB76-56A8775D1D11},");
118                 Initial_15__TO__Closed_8(signal, submachineState);
119                 break;
120         }
121
122         m_StateMachineImpl.currentTransition.SetTransition
              (TransitionEnum.NOTRANSITION.ordinal(), null, "", "");
123     }
124
125     private void Opened__TO__Opened_6_effect(Signal signal)
126     {
127         GlobalFuncs.trace("[" + m_sInstanceName + ":" + m_sType + "] Transition
              Effect: Opened__TO__Opened_6");
128
129     }
130
131     private void Opened__TO__Opened_6(Signal signal, StateData submachineState)
132     {
133         if (m_StateMachineImpl == null)
134             return;
135
136         if(!m_StateMachineImpl.GetStateObject(submachineState,
              StateEnum.SimpleGateController_ENUM_STATEMACHINE_OPENED.ordinal
              ()).IsActiveState())
137         {
138             return;
139         }
```

```
140         StateProc(StateEnum.SimpleGateController_ENUM_STATEMACHINE_OPENED,
              submachineState, StateBehaviorEnum.EXIT, null);
141         Opened__TO__Opened_6_effect(signal);
142         m_StateMachineImpl.currentTransition.SetActive(m_StateMachineImpl);
143         StateProc(StateEnum.SimpleGateController_ENUM_STATEMACHINE_OPENED,
              submachineState, StateBehaviorEnum.ENTRY, signal,
              EntryTypeEnum.DefaultEntry);
144     }
145     private void Opened__TO__Closed_7_effect(Signal signal)
146     {
147         GlobalFuncs.trace("[" + m_sInstanceName + ":" + m_sType + "] Transition
              Effect: Opened__TO__Closed_7");
148
149     }
150
151     private void Opened__TO__Closed_7(Signal signal, StateData submachineState)
152     {
153         if (m_StateMachineImpl == null)
154             return;
155
156         if(!m_StateMachineImpl.GetStateObject(submachineState,
              StateEnum.SimpleGateController_ENUM_STATEMACHINE_OPENED.ordinal
              ()).IsActiveState())
157         {
158             return;
159         }
160         StateProc(StateEnum.SimpleGateController_ENUM_STATEMACHINE_OPENED,
              submachineState, StateBehaviorEnum.EXIT, null);
161         Opened__TO__Closed_7_effect(signal);
162         m_StateMachineImpl.currentTransition.SetActive(m_StateMachineImpl);
163         StateProc(StateEnum.SimpleGateController_ENUM_STATEMACHINE_CLOSED,
              submachineState, StateBehaviorEnum.ENTRY, signal,
              EntryTypeEnum.DefaultEntry);
164     }
165     private void Closed__TO__Opened_5_effect(Signal signal)
166     {
167         GlobalFuncs.trace("[" + m_sInstanceName + ":" + m_sType + "] Transition
              Effect: Closed__TO__Opened_5");
168
169     }
170
171     private void Closed__TO__Opened_5(Signal signal, StateData submachineState)
172     {
173         if (m_StateMachineImpl == null)
174             return;
175
176         if(!m_StateMachineImpl.GetStateObject(submachineState,
              StateEnum.SimpleGateController_ENUM_STATEMACHINE_CLOSED.ordinal
              ()).IsActiveState())
177         {
178             return;
179         }
```

```
180         StateProc(StateEnum.SimpleGateController_ENUM_STATEMACHINE_CLOSED,
              submachineState, StateBehaviorEnum.EXIT, null);
181         Closed__TO__Opened_5_effect(signal);
182         m_StateMachineImpl.currentTransition.SetActive(m_StateMachineImpl);
183         StateProc(StateEnum.SimpleGateController_ENUM_STATEMACHINE_OPENED,
              submachineState, StateBehaviorEnum.ENTRY, signal,
              EntryTypeEnum.DefaultEntry);
184     }
185     private void Initial_15__TO__Closed_8_effect(Signal signal)
186     {
187         GlobalFuncs.trace("[" + m_sInstanceName + ":" + m_sType + "] Transition
              Effect: Initial_15__TO__Closed_8");
188
189     }
190
191     private void Initial_15__TO__Closed_8(Signal signal, StateData
          submachineState)
192     {
193         if (m_StateMachineImpl == null)
194             return;
195
196         if(submachineState != null)
197             submachineState.IncrementActiveCount();
198         Initial_15__TO__Closed_8_effect(signal);
199         m_StateMachineImpl.currentTransition.SetActive(m_StateMachineImpl);
200         StateProc(StateEnum.SimpleGateController_ENUM_STATEMACHINE_CLOSED,
              submachineState, StateBehaviorEnum.ENTRY, signal,
              EntryTypeEnum.DefaultEntry);
201     }
202     public boolean StateProc(StateEnum state, StateData submachineState,
          StateBehaviorEnum behavior, Signal signal)
203     {
204         return StateProc(state, submachineState, behavior, signal,
              EntryTypeEnum.DefaultEntry, null, 0);
205     }
206
207     public boolean StateProc(StateEnum state, StateData submachineState,
          StateBehaviorEnum behavior, Signal signal, EntryTypeEnum enumEntryType)
208     {
209         return StateProc(state, submachineState, behavior, signal, enumEntryType,
              null, 0);
210     }
211
212     public boolean StateProc(int state, StateData submachineState,
          StateBehaviorEnum behavior, Signal signal, EntryTypeEnum enumEntryType, int
          [] entryArray, int nArrayCount)
213     {
214         ArrayList<EntryEnum> entryEnumList = new ArrayList<>();
215         if (entryArray != null)
216         {
217             for(int nEntryIndex : entryArray)
218             {
```

```
219                 entryEnumList.add(EntryEnum.values()[nEntryIndex]);
220             }
221         }
222         return StateProc(StateEnum.values()[state], submachineState, behavior,
              signal, enumEntryType, entryEnumList.toArray(new EntryEnum
              [entryEnumList.size()]), nArrayCount);
223     }
224 
225     public boolean StateProc(StateEnum state, StateData submachineState,
          StateBehaviorEnum behavior, Signal signal, EntryTypeEnum enumEntryType,
          EntryEnum[] entryArray, int nArrayCount)
226     {
227         switch (state)
228         {
229             case SimpleGateController_ENUM_STATEMACHINE_CLOSED:
230                 StateMachine_Closed(behavior, submachineState, signal,
                      enumEntryType, entryArray, nArrayCount);
231                 break;
232 
233             case SimpleGateController_ENUM_STATEMACHINE_OPENED:
234                 StateMachine_Opened(behavior, submachineState, signal,
                      enumEntryType, entryArray, nArrayCount);
235                 break;
236         }
237         return false;
238     }
239 
240     public boolean StateMachine_Closed(StateBehaviorEnum behavior, StateData
          submachineState, Signal signal, EntryTypeEnum enumEntryType, EntryEnum[]
          entryArray, int nArrayCount)
241     {
242         if (m_StateMachineImpl == null)
243             return false;
244 
245         StateData state = m_StateMachineImpl.GetStateObject(submachineState,
              StateEnum.SimpleGateController_ENUM_STATEMACHINE_CLOSED.ordinal());
246         switch (behavior)
247         {
248             case ENTRY:
249                 if(state.active_count > 0)
250                     return false;
251                 m_statemachine =
                      StateEnum.SimpleGateController_ENUM_STATEMACHINE_CLOSED;
252                 state.IncrementActiveCount();
253                 StateMachine_Closed_behavior(StateBehaviorEnum.ENTRY);
254 
255                 StateMachine_Closed_behavior(StateBehaviorEnum.DO);
256 
257                 if((enumEntryType == EntryTypeEnum.EntryPointEntry ||
                      enumEntryType == EntryTypeEnum.DefaultEntry) &&
                      state.IsActiveState())
258                     m_StateMachineImpl.deferInternalEvent
```

```
                    (EventProxy.EventEnum.COMPLETION, null, state);
259                 break;
260             case EXIT:
261                 if(state.active_count == 0)
262                     return false;
263                 m_statemachine = StateEnum.NOSTATE;
264                 state.DecrementActiveCount();
265                 StateMachine_Closed_behavior(StateBehaviorEnum.EXIT);
266                 m_StateMachineImpl.removeInternalEvent(state);
267                 break;
268         }
269 
270         return true;
271     }
272 
273     public boolean StateMachine_Closed_behavior(StateBehaviorEnum behavior)
274     {
275         switch (behavior)
276         {
277             case ENTRY:
278             {
279                 GlobalFuncs.trace("[" + m_sInstanceName + ":" + m_sType + "]      ↵
                      Entry Behavior: StateMachine_Closed");
280             }
281             break;
282             case DO:
283             {
284                 GlobalFuncs.trace("[" + m_sInstanceName + ":" + m_sType + "] Do   ↵
                      Behavior: StateMachine_Closed");
285             }
286             break;
287             case EXIT:
288             {
289                 GlobalFuncs.trace("[" + m_sInstanceName + ":" + m_sType + "] Exit ↵
                       Behavior: StateMachine_Closed");
290             }
291             break;
292         }
293 
294         return true;
295     }
296     public boolean StateMachine_Opened(StateBehaviorEnum behavior, StateData      ↵
        submachineState, Signal signal, EntryTypeEnum enumEntryType, EntryEnum[]    ↵
        entryArray, int nArrayCount)
297     {
298         if (m_StateMachineImpl == null)
299             return false;
300 
301         StateData state = m_StateMachineImpl.GetStateObject(submachineState,      ↵
          StateEnum.SimpleGateController_ENUM_STATEMACHINE_OPENED.ordinal());
302         switch (behavior)
303         {
```

```
304             case ENTRY:
305                 if(state.active_count > 0)
306                     return false;
307                 m_statemachine =
                      StateEnum.SimpleGateController_ENUM_STATEMACHINE_OPENED;
308                 state.IncrementActiveCount();
309                 StateMachine_Opened_behavior(StateBehaviorEnum.ENTRY);
310 
311                 StateMachine_Opened_behavior(StateBehaviorEnum.DO);
312 
313                 if((enumEntryType == EntryTypeEnum.EntryPointEntry ||
                      enumEntryType == EntryTypeEnum.DefaultEntry) &&
                      state.IsActiveState())
314                     m_StateMachineImpl.deferInternalEvent
                       (EventProxy.EventEnum.COMPLETION, null, state);
315                 break;
316             case EXIT:
317                 if(state.active_count == 0)
318                     return false;
319                 m_statemachine = StateEnum.NOSTATE;
320                 state.DecrementActiveCount();
321                 StateMachine_Opened_behavior(StateBehaviorEnum.EXIT);
322                 m_StateMachineImpl.removeInternalEvent(state);
323                 break;
324         }
325 
326         return true;
327     }
328 
329     public boolean StateMachine_Opened_behavior(StateBehaviorEnum behavior)
330     {
331         switch (behavior)
332         {
333             case ENTRY:
334             {
335                 GlobalFuncs.trace("[" + m_sInstanceName + ":" + m_sType + "]
                      Entry Behavior: StateMachine_Opened");
336             }
337             break;
338             case DO:
339             {
340                 GlobalFuncs.trace("[" + m_sInstanceName + ":" + m_sType + "] Do
                      Behavior: StateMachine_Opened");
341             }
342             break;
343             case EXIT:
344             {
345                 GlobalFuncs.trace("[" + m_sInstanceName + ":" + m_sType + "] Exit
                       Behavior: StateMachine_Opened");
346             }
347             break;
348         }
```

```
349
350         return true;
351     }
352     public boolean dispatch(Event event, StateData toState, boolean bCheckOnly)
353     {
354         if (m_StateMachineImpl == null)
355             return false;
356
357         boolean bDispatched = false;
358         if(toState == null || !toState.IsActiveState() && !bCheckOnly)
359             return bDispatched;
360
361         switch (StateEnum.values()[toState.state_enum])
362         {
363             case SimpleGateController_VIRTUAL_SUBMACHINESTATE:
364                 if(event.eventEnum == EventProxy.EventEnum.COMPLETION)
365                 {
366                     if(!bCheckOnly)
367                     {
368                         m_StateMachineImpl.ReleaseSubmachineState(toState);
369                         for (int index = m_StateMachineImpl.lstStateData.size() -
                          1; index >= 0; index--)
370                         {
371                             StateData state = m_StateMachineImpl.lstStateData.get
                         (index);
372                             if (state == toState)
373                             {
374                                 m_StateMachineImpl.lstStateData.remove(state);
375                                 break;
376                             }
377                         }
378                         //delete toState;
379                         toState = null;
380                     }
381                     bDispatched = true;
382                 }
383                 break;
384             case SimpleGateController_ENUM_STATEMACHINE_CLOSED:
385                 switch (event.eventEnum)
386                 {
387                     case ENUM_CARINSENSOR:
388                         if(!bCheckOnly)
389                             TransitionProc
                         (TransitionEnum.SimpleGateController_ENUM_CLOSED__TO__OPENED_
                         5, event.signal, toState.submachine_state);
390                         bDispatched = true;
391                         break;
392                 }
393                 break;
394             case SimpleGateController_ENUM_STATEMACHINE_OPENED:
395                 switch (event.eventEnum)
396                 {
```

```
397                     case ENUM_CARINSENSOR:
398                         if(!bCheckOnly)
399                             TransitionProc
                        (TransitionEnum.SimpleGateController_ENUM_OPENED__TO__OPENED_
                        6, event.signal, toState.submachine_state);
400                         bDispatched = true;
401                         break;
402                     case ENUM_TIMERFINISHED:
403                         if(!bCheckOnly)
404                             TransitionProc
                        (TransitionEnum.SimpleGateController_ENUM_OPENED__TO__CLOSED_
                        7, event.signal, toState.submachine_state);
405                         bDispatched = true;
406                         break;
407                 }
408                 break;
409         }
410
411         if (!bDispatched && toState != null && toState.parent_state != null &&
              event.eventEnum != EventProxy.EventEnum.COMPLETION)
412         {
413             bDispatched = dispatch(event, toState.parent_state, true);
414             if (bDispatched && !bCheckOnly)
415             {
416                 /*1. Exit the current state; 2. Decrement the active count of the
                       parent state; 3. dispatch the event to parent state*/
417                 StateProc(StateEnum.values()[toState.state_enum],
                      toState.submachine_state, StateBehaviorEnum.EXIT, null);
418                 toState.parent_state.DecrementActiveCount();
419                 dispatch(event, toState.parent_state, false);
420                 event = null;
421             }
422         }
423
424         return bDispatched;
425     }
426
427     StateEnum m_statemachine;
428
429     @Override
430     public void runStateMachine(String statemachine)
431     {
432         if(statemachine.equals("SimpleGateController_StateMachine"))
433         {
434             runStateMachine
                  (StateMachineEnum.SimpleGateController_ENUM_STATEMACHINE);
435             return;
436         }
437     }
438
439     private void runStateMachine(StateMachineEnum statemachine, StateData
          submachineState, Signal signal)
```

```
440     {
441         runStateMachine(statemachine, submachineState, signal, null, 0);
442     }
443
444     private void runStateMachine(StateMachineEnum statemachine, StateData
          submachineState, Signal signal, EntryEnum[] entryArray, int nEntryCount)
445     {
446         if (m_StateMachineImpl == null)
447             return;
448
449         if(submachineState == null)
450         {
451             StateInitialData initialData = new StateInitialData
                (StateEnum.SimpleGateController_VIRTUAL_SUBMACHINESTATE.ordinal(),
                StateEnum.NOSTATE.ordinal(), 1, false,
                "SimpleGateController_VIRTUAL_SUBMACHINESTATE", "");
452             submachineState = new StateData(m_StateMachineImpl, initialData);
453             submachineState.IncrementActiveCount();
454             m_StateMachineImpl.lstStateData.add(submachineState);
455         }
456         switch (statemachine)
457         {
458             case SimpleGateController_ENUM_STATEMACHINE:
459                 {
460                     final int nArrayCount = 2;
461                     StateInitialData[] initialDataArray =
462                         {
463                             new StateInitialData
                         (StateEnum.SimpleGateController_ENUM_STATEMACHINE_CLOSED.ordi
                         nal(), StateEnum.NOSTATE.ordinal(), 0, false,
                         "SimpleGateController_StateMachine_Closed",
                         "{816F979E-667A-4b12-B661-E2AF10B72CD4}"),
464                             new StateInitialData
                         (StateEnum.SimpleGateController_ENUM_STATEMACHINE_OPENED.ordi
                         nal(), StateEnum.NOSTATE.ordinal(), 0, false,
                         "SimpleGateController_StateMachine_Opened",
                         "{9B058E1E-45B9-49ac-B756-7EDCA2E20F9A}")
465                         };
466
467                     m_StateMachineImpl.CreateStateObjects(initialDataArray,
                         nArrayCount, submachineState);
468                 }
469                 for(int i = 0; i < nEntryCount; i++)
470                 {
471                     switch(entryArray[i])
472                     {
473                     case SimpleGateController_ENUM_STATEMACHINE_INITIAL_15:
474                         TransitionProc
                         (TransitionEnum.SimpleGateController_ENUM_INITIAL_15__TO__CLO
                         SED_8, signal, submachineState);
475                         break;
476                     }
```

```
477                 }
478                 if(submachineState.IsActiveState())
479                     m_StateMachineImpl.deferInternalEvent
                        (EventProxy.EventEnum.COMPLETION, null, submachineState);
480                 break;
481         }
482
483     }
484
485     public void runStateMachine(StateMachineEnum statemachine)
486     {
487         if (m_StateMachineImpl == null)
488             return;
489
490         List<StateData> activeStateArray = new ArrayList<StateData>();
491         if(m_StateMachineImpl.GetCurrentStates(activeStateArray) > 0)
492             return;
493         switch (statemachine)
494         {
495             case SimpleGateController_ENUM_STATEMACHINE:
496                 {
497                     final int nArrayCount = 1;
498                     EntryEnum[] entryArray =
                        {EntryEnum.SimpleGateController_ENUM_STATEMACHINE_INITIAL_15}
                        ;
499                     runStateMachine(statemachine, null, null, entryArray,
                        nArrayCount);   //submachineState is NULL if run standalone
                        statemachine
500                 }
501                 break;
502         }
503     }
504
505     public void runStateMachines()
506     {
507         runStateMachine(StateMachineEnum.SimpleGateController_ENUM_STATEMACHINE);

508     }
509
510
511     /* End - EA generated code for StateMachine */
512 }//end SimpleGateController
```

Anexo III

Código fonte gerado pela ferramenta Visual Paradigm

```
 1
 2 //
 3 // The contents of this file are subject to the Mozilla Public
 4 // License Version 1.1 (the "License"); you may not use this file
 5 // except in compliance with the License. You may obtain a copy of
 6 // the License at http://www.mozilla.org/MPL/
 7 //
 8 // Software distributed under the License is distributed on an "AS
 9 // IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or
10 // implied. See the License for the specific language governing
11 // rights and limitations under the License.
12 //
13 // The Original Code is  State Machine Compiler(SMC).
14 //
15 // The Initial Developer of the Original Code is Charles W. Rapp.
16 // Portions created by Charles W. Rapp are
17 // Copyright (C) 2000 - 2003 Charles W. Rapp.
18 // All Rights Reserved.
19 //
20 // Contributor(s):
21 //
22 // statemap.java --
23 //
24 //  This package defines the FSMContext class which must be inherited by
25 //  any Java class wanting to use an smc-generated state machine.
26 //
27 // RCS ID
28 // $Id: FSMContext.java,v 1.7 2005/05/28 18:44:13 cwrapp Exp $
29 //
30 // Change Log
31 // $Log: FSMContext.java,v $
32 // Revision 1.7  2005/05/28 18:44:13  cwrapp
33 // Updated C++, Java and Tcl libraries, added CSharp, Python and VB.
34 //
35 // Revision 1.1  2005/02/21 19:03:38  charlesr
36 // Variable name clean up.
37 //
38 // Revision 1.0  2003/12/14 20:38:40  charlesr
39 // Initial revision
40 //
41
42 import java.io.PrintStream;
43 import java.io.Serializable;
44 import java.util.EmptyStackException;
45 import java.util.Set;
46 import java.util.HashSet;
47 import java.util.Iterator;
48
49 // statemap.FSMContext --
50 //
51 //  All Java classes wanting to use an SMC-generated state machine
52 //  must extend this class. Since FSMContext provides
```

```
53  //  functionality, it was not possible to implement it as an
54  //  interface. See the SMC FAQ for how a class can use a state
55  //  machine when that class is already extending another class.
56
57  public abstract class FSMContext
58      implements Serializable
59  {
60  // Member functions
61
62      public FSMContext()
63      {
64          // There is no state until the application explicitly
65          // sets the initial state.
66          _state = null;
67          _transition = "";
68          _previousState = null;
69          _stateStack = null;
70          _debugFlag = false;
71          _debugStream = System.err;
72      }
73
74      // When debug is set to true, the state machine
75      // will print messages to the console.
76      public boolean getDebugFlag()
77      {
78          return(_debugFlag && _debugStream != null);
79      }
80
81      public void setDebugFlag(boolean flag)
82      {
83          _debugFlag = flag;
84          return;
85      }
86
87      // Write the debug output to this stream.
88      public PrintStream getDebugStream()
89      {
90          return (_debugStream == null ? System.err : _debugStream);
91      }
92
93      public void setDebugStream(PrintStream stream)
94      {
95          _debugStream = stream;
96          return;
97      }
98
99      // Is this state machine in a transition? If state is null,
100     // then true; otherwise, false.
101     public boolean isInTransition()
102     {
103         return(_state == null ? true : false);
104     }
```

```
105
106     public void setState(State state)
107     {
108         if (getDebugFlag() == true)
109         {
110             getDebugStream().println("NEW STATE    : " +
111                                      state.getName());
112         }
113
114         State lOldState = _state;
115         _state = state;
116         fireStateChanged(lOldState, state);
117
118         return;
119     }
120
121     public void clearState()
122     {
123         _previousState = _state;
124         _state = null;
125         fireStateChanged(_previousState, null);
126
127         return;
128     }
129
130     public State getPreviousState()
131         throws NullPointerException
132     {
133         if (_previousState == null)
134         {
135             throw (new NullPointerException());
136         }
137         else
138         {
139             return(_previousState);
140         }
141     }
142
143     public void pushState(State state)
144     {
145         if (_state == null)
146         {
147             throw (new NullPointerException());
148         }
149
150         if (getDebugFlag() == true)
151         {
152             getDebugStream().println("PUSH TO STATE: " +
153                                      state.getName());
154         }
155
156         if (_stateStack == null)
```

```
157         {
158             _stateStack = new java.util.Stack();
159         }
160
161         _stateStack.push(_state);
162         State lOldState = _state;
163         _state = state;
164         fireStateChanged(lOldState, state);
165
166         return;
167     }
168
169     public void popState()
170         throws EmptyStackException
171     {
172         if (_stateStack == null ||
173             _stateStack.isEmpty() == true)
174         {
175             if (getDebugFlag() == true)
176             {
177                 getDebugStream().println("POPPING ON EMPTY STATE STACK.");
178             }
179
180             throw (new EmptyStackException());
181         }
182         else
183         {
184             // The pop method removes the top element
185             // from the stack and returns it.
186             State lOldState = _state;
187             _state = (State) _stateStack.pop();
188             fireStateChanged(lOldState, _state);
189
190             if (_stateStack.isEmpty() == true)
191             {
192                 _stateStack = null;
193             }
194
195             if (getDebugFlag() == true)
196             {
197                 getDebugStream().println("POP TO STATE : " +
198                                          _state.getName());
199             }
200         }
201
202         return;
203     }
204
205     public void emptyStateStack()
206     {
207         _stateStack.clear();
208         _stateStack = null;
```

```
209
210         return;
211     }
212
213     public String getTransition()
214     {
215         return(_transition);
216     }
217
218 // Member data
219
220     // The current state.
221     transient protected State _state;
222
223     // The current transition *name*. Used for debugging
224     // purposes.
225     transient protected String _transition;
226
227     // Remember what state a transition left.
228     // Do no persist the previous state because an FSM should be
229     // serialized while in transition.
230     transient protected State _previousState;
231
232     // This stack is used when a push transition is taken.
233     transient protected java.util.Stack _stateStack;
234
235     // When this flag is set to true, this class will print
236     // out debug messages.
237     transient protected boolean _debugFlag;
238
239     // Write debug output to this stream.
240     transient protected PrintStream _debugStream;
241
242     // Listener
243     transient protected Set _stateChangeListeners = new HashSet();
244
245     public void addStateChangeListener(StateChangeListener aStateChangeListener)
        {
246         _stateChangeListeners.add(aStateChangeListener);
247     }
248
249     protected void fireStateChanged(State aOldState, State aNewState) {
250         for (Iterator iter = _stateChangeListeners.iterator(); iter.hasNext();) {
251             StateChangeListener listener = (StateChangeListener) iter.next();
252             listener.stateChanged(aOldState, aNewState);
253         }
254     }
255 }
256
```

```
1  public class SimpleGateController {
2      private SimpleGateControllerContext _fsm;
3
4      public SimpleGateController() {
5          _fsm = new SimpleGateControllerContext(this);
6      }
7
8      public SimpleGateControllerContext getContext() {
9          return _fsm;
10     }
11
12     public void CarInSensor() {
13         _fsm.CarInSensor();
14         throw new RuntimeException("Not Implemented!");
15     }
16
17     public void TimerFinished() {
18         _fsm.TimerFinished();
19         throw new RuntimeException("Not Implemented!");
20     }
21
22     public void CloseGate() {
23         throw new RuntimeException("Not Implemented!");
24     }
25
26     public void OpenGate() {
27         throw new RuntimeException("Not Implemented!");
28     }
29
30 }
```

```
 1
 2  public final class SimpleGateControllerContext
 3      extends FSMContext
 4  {
 5  //---------------------------------------------------------------
 6  // Member methods.
 7  //
 8
 9      public SimpleGateControllerContext(SimpleGateController owner)
10      {
11          super();
12
13          _owner = owner;
14          setState(SimpleGateControllerFSM.Closed);
15          SimpleGateControllerFSM.Closed.Entry(this);
16      }
17
18      public synchronized void CarInSensor()
19      {
20          _transition = "CarInSensor";
21          getState().CarInSensor(this);
22          _transition = "";
23          return;
24      }
25
26      public synchronized void TimerFinished()
27      {
28          _transition = "TimerFinished";
29          getState().TimerFinished(this);
30          _transition = "";
31          return;
32      }
33
34      public SimpleGateControllerState getState()
35          throws StateUndefinedException
36      {
37          if (_state == null)
38          {
39              throw(
40                  new StateUndefinedException());
41          }
42
43          return ((SimpleGateControllerState) _state);
44      }
45
46      protected SimpleGateController getOwner()
47      {
48          return (_owner);
49      }
50
51  //---------------------------------------------------------------
52  // Member data.
```

```
53  //
54
55      transient private SimpleGateController _owner;
56
57  //-------------------------------------------------------------
58  // Inner classes.
59  //
60
61      public static abstract class SimpleGateControllerState
62          extends State
63      {
64      //-------------------------------------------------------------
65      // Member methods.
66      //
67
68          protected SimpleGateControllerState(String name, int id)
69          {
70              super (name, id);
71          }
72
73          protected void Entry(SimpleGateControllerContext context) {}
74          protected void Exit(SimpleGateControllerContext context) {}
75
76          protected void CarInSensor(SimpleGateControllerContext context)
77          {
78              Default(context);
79          }
80
81          protected void TimerFinished(SimpleGateControllerContext context)
82          {
83              Default(context);
84          }
85
86          protected void Default(SimpleGateControllerContext context)
87          {
88              throw (
89                  new TransitionUndefinedException(
90                      "State: " +
91                      context.getState().getName() +
92                      ", Transition: " +
93                      context.getTransition()));
94          }
95
96      //-------------------------------------------------------------
97      // Member data.
98      //
99      }
100
101     /* package */ static abstract class SimpleGateControllerFSM
102     {
103     //-------------------------------------------------------------
104     // Member methods.
```

```
105     //
106
107     //------------------------------------------------------------
108     // Member data.
109     //
110
111         //------------------------------------------------------
112         // Statics.
113         //
114         public static
              SimpleGateControllerFSM_Default.SimpleGateControllerFSM_Opened Opened;
115         public static
              SimpleGateControllerFSM_Default.SimpleGateControllerFSM_Closed Closed;
116         private static SimpleGateControllerFSM_Default Default;
117
118         static
119         {
120             Opened = new
                  SimpleGateControllerFSM_Default.SimpleGateControllerFSM_Opened
                  ("SimpleGateControllerFSM.Opened", 0);
121             Closed = new
                  SimpleGateControllerFSM_Default.SimpleGateControllerFSM_Closed
                  ("SimpleGateControllerFSM.Closed", 1);
122             Default = new SimpleGateControllerFSM_Default
                  ("SimpleGateControllerFSM.Default", -1);
123         }
124
125     }
126
127     protected static class SimpleGateControllerFSM_Default
128         extends SimpleGateControllerState
129     {
130     //------------------------------------------------------------
131     // Member methods.
132     //
133
134         protected SimpleGateControllerFSM_Default(String name, int id)
135         {
136             super (name, id);
137         }
138
139     //------------------------------------------------------------
140     // Inner classse.
141     //
142
143
144         private static final class SimpleGateControllerFSM_Opened
145             extends SimpleGateControllerFSM_Default
146         {
147         //------------------------------------------------------
148         // Member methods.
149         //
```

```
150
151             private SimpleGateControllerFSM_Opened(String name, int id)
152             {
153                 super (name, id);
154             }
155
156             protected void CarInSensor(SimpleGateControllerContext context)
157             {
158
159
160                 return;
161             }
162
163             protected void TimerFinished(SimpleGateControllerContext context)
164             {
165
166
167                 (context.getState()).Exit(context);
168                 context.setState(SimpleGateControllerFSM.Closed);
169                 (context.getState()).Entry(context);
170                 return;
171             }
172
173         //--------------------------------------------------------
174         // Member data.
175         //
176         }
177
178         private static final class SimpleGateControllerFSM_Closed
179             extends SimpleGateControllerFSM_Default
180         {
181         //--------------------------------------------------------
182         // Member methods.
183         //
184
185             private SimpleGateControllerFSM_Closed(String name, int id)
186             {
187                 super (name, id);
188             }
189
190             protected void CarInSensor(SimpleGateControllerContext context)
191             {
192
193                 if (ConditionTest())
194                 {
195
196                     (context.getState()).Exit(context);
197                     // No actions.
198                     context.setState(SimpleGateControllerFSM.Opened);
199                     (context.getState()).Entry(context);
200                 }
201                 else
```

```
202                 {
203                     super.CarInSensor(context);
204                 }
205 
206                 return;
207             }
208 
209         //-------------------------------------------------------
210         // Member data.
211         //
212         }
213 
214     //-------------------------------------------------------------
215     // Member data.
216     //
217     }
218 }
219 
```

```
 1
 2  //
 3  // The contents of this file are subject to the Mozilla Public
 4  // License Version 1.1 (the "License"); you may not use this file
 5  // except in compliance with the License. You may obtain a copy of
 6  // the License at http://www.mozilla.org/MPL/
 7  //
 8  // Software distributed under the License is distributed on an "AS
 9  // IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or
10  // implied. See the License for the specific language governing
11  // rights and limitations under the License.
12  //
13  // The Original Code is State Machine Compiler (SMC).
14  //
15  // The Initial Developer of the Original Code is Charles W. Rapp.
16  // Portions created by Charles W. Rapp are
17  // Copyright (C) 2000 - 2005 Charles W. Rapp.
18  // All Rights Reserved.
19  //
20  // Contributor(s):
21  //
22  // statemap.java --
23  //
24  //  This package defines the fsmContext class which must be inherited by
25  //  any Java class wanting to use an smc-generated state machine.
26  //
27  // RCS ID
28  // $Id: State.java,v 1.6 2005/06/03 19:58:29 cwrapp Exp $
29  //
30  // CHANGE LOG
31  // (See the bottom of this file.)
32  //
33
34  import java.io.Serializable;
35
36  // statemap.State --
37  //
38  //  The reason why Java has an abstract State class while C++ and
39  //  Tcl do not is because:
40  //  1) in C++, FSMContext is a template class and so the
41  //     app-specific state class name can be used to generate an
42  //     app-specific fsmContext class,
43  //  2) Tcl is weakly typed so all it needs is a state name.
44  //  Since Java does not have templates and is strongly typed,
45  //  an abstract State class is needed for FSMContext's _state
46  //  and _state_stack declarations.
47  //
48  //  Another reason why I don't have a state class in C++ and Tcl
49  //  when I could have one is because there is nothing to the class.
50
51  public abstract class State
52      implements Serializable
```

```
53  {
54  //-----------------------------------------------------------------
55  // Member functions
56  //
57
58      protected State(String name, int id)
59      {
60          _name = name;
61          _id = id;
62      }
63
64      public String getName()
65      {
66          return (_name);
67      }
68
69      public int getId()
70      {
71          return (_id);
72      }
73
74      public String toString()
75      {
76          return (_name);
77      }
78
79  //-----------------------------------------------------------------
80  // Member data
81  //
82
83      private final String _name;
84      private final int _id;
85  }
86
87  //
88  // CHANGE LOG
89  // $Log: State.java,v $
90  // Revision 1.6  2005/06/03 19:58:29  cwrapp
91  // Further updates for release 4.0.0
92  //
93  // Revision 1.5  2005/05/28 18:44:13  cwrapp
94  // Updated C++, Java and Tcl libraries, added CSharp, Python and VB.
95  //
96  // Revision 1.0  2003/12/14 20:39:41  charlesr
97  // Initial revision
98  //
99
```

```
1
2  public interface StateChangeListener {
3     public void stateChanged(State aOldState, State aNewState);
4  }
```

```
 1
 2  //
 3  // The contents of this file are subject to the Mozilla Public
 4  // License Version 1.1 (the "License"); you may not use this file
 5  // except in compliance with the License. You may obtain a copy of
 6  // the License at http://www.mozilla.org/MPL/
 7  //
 8  // Software distributed under the License is distributed on an "AS
 9  // IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or
10  // implied. See the License for the specific language governing
11  // rights and limitations under the License.
12  //
13  // The Original Code is  State Machine Compiler(SMC).
14  //
15  // The Initial Developer of the Original Code is Charles W. Rapp.
16  // Portions created by Charles W. Rapp are
17  // Copyright (C) 2000 - 2003 Charles W. Rapp.
18  // All Rights Reserved.
19  //
20  // Contributor(s):
21  //
22  // statemap.java --
23  //
24  //  This package defines the FSMContext class which must be inherited by
25  //  any Java class wanting to use an smc-generated state machine.
26  //
27  // RCS ID
28  // $Id: StateUndefinedException.java,v 1.4 2005/05/28 18:44:13 cwrapp Exp $
29  //
30  // Change Log
31  // $Log: StateUndefinedException.java,v $
32  // Revision 1.4  2005/05/28 18:44:13  cwrapp
33  // Updated C++, Java and Tcl libraries, added CSharp, Python and VB.
34  //
35  // Revision 1.0  2003/12/14 20:39:59  charlesr
36  // Initial revision
37  //
38
39  /**
40   * A <code>StateUndefinedException</code> is thrown by
41   * an SMC-generated state machine whenever a transition is taken
42   * and there is no state currently set. This occurs when a
43   * transition is issued from with a transition action.
44   */
45  public final class StateUndefinedException
46      extends RuntimeException
47  {
48      /**
49       * Default constructor.
50       */
51      public StateUndefinedException()
52      {
```

```
53          super();
54      }
55
56      /**
57       * Constructs a <code>StateUndefinedException</code>
58       * with a detail message.
59       * @param reason the detail message.
60       */
61      public StateUndefinedException(String reason)
62      {
63          super(reason);
64      }
65  }
66
```

```
 1
 2  //
 3  // The contents of this file are subject to the Mozilla Public
 4  // License Version 1.1 (the "License"); you may not use this file
 5  // except in compliance with the License. You may obtain a copy of
 6  // the License at http://www.mozilla.org/MPL/
 7  //
 8  // Software distributed under the License is distributed on an "AS
 9  // IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or
10  // implied. See the License for the specific language governing
11  // rights and limitations under the License.
12  //
13  // The Original Code is  State Machine Compiler(SMC).
14  //
15  // The Initial Developer of the Original Code is Charles W. Rapp.
16  // Portions created by Charles W. Rapp are
17  // Copyright (C) 2000 - 2003 Charles W. Rapp.
18  // All Rights Reserved.
19  //
20  // Contributor(s):
21  //
22  // statemap.java --
23  //
24  //  This package defines the FSMContext class which must be inherited by
25  //  any Java class wanting to use an smc-generated state machine.
26  //
27  // RCS ID
28  // $Id: TransitionUndefinedException.java,v 1.4 2005/05/28 18:44:13 cwrapp Exp $
29  //
30  // Change Log
31  // $Log: TransitionUndefinedException.java,v $
32  // Revision 1.4  2005/05/28 18:44:13  cwrapp
33  // Updated C++, Java and Tcl libraries, added CSharp, Python and VB.
34  //
35  // Revision 1.0  2003/12/14 20:40:47  charlesr
36  // Initial revision
37  //
38
39  /**
40   * A <code>TransitionUndefinedException</code> is thrown by
41   * an SMC-generated state machine whenever a transition is taken
42   * which:
43   * <ol>
44   *   <li>Is not explicitly defined in the current state.
45   *   <li>Is not explicitly defined in the current FSM's default
46   *       state.
47   *   <li>There is no Default transition in the current state.
48   * </ol>
49   */
50  public final class TransitionUndefinedException
51      extends RuntimeException
52  {
```

```
53      /**
54       * Constructs a <code>TransitionUndefinedException</code>
55       * with no detail message.
56       */
57      public TransitionUndefinedException()
58      {
59          super();
60      }
61
62      /**
63       * Constructs a <code>TransitionUndefinedException</code>
64       * with a detail message.
65       * @param reason the detail message.
66       */
67      public TransitionUndefinedException(String reason)
68      {
69          super(reason);
70      }
71  }
72
```

Anexo IV

Código fonte de exemplo de utilização da Framework

```cpp
#include <iostream>
#include <cstdlib>
#include <ctime>

#include "../lib/FSM.h"
#include "../lib/State.h"
#include "../lib/Transition.h"
#include "../lib/FSMEvent.h"

using namespace UMinho::FSM;
using namespace std;

struct SharedData : public Object {
};

class CloseGate : public FSMAction {
    void execute(const FSMEvent &ev, FSMContext &c) {
        ///TODO: INSERT CODE HERE!!
    }
};

class OpenGate : public FSMAction {
    void execute(const FSMEvent &ev, FSMContext &c) {
        ///TODO: INSERT CODE HERE!!
    }
};

class StartTimer : public FSMAction {
    void execute(const FSMEvent &ev, FSMContext &c) {
        ///TODO: INSERT CODE HERE!!
    }
};

enum StateID { CLOSED = 1, OPENED };

enum EventID { CAR_IN_SENSOR = 1, TIMER_FINISHED };

string stateName(StateID s) {
    static string names[] = { "CLOSED", "OPENED" };
    return names[(int) s-1];
}

string eventName(EventID e) {
    static string names[] = { "CAR_IN_SENSOR", "TIMER_FINISHED" };
    return names[(int) e-1];
}

int main() {
    CloseGate closeGate;
    OpenGate openGate;
    StartTimer startTimer;

```

```
53
54      State closed(CLOSED, closeGate, openGate);
55      State opened(OPENED, startTimer);
56
57      closed.addTransition(CAR_IN_SENSOR, opened);
58      opened.addTransition(CAR_IN_SENSOR, opened);
59      opened.addTransition(TIMER_FINISHED, closed);
60
61      SharedData sd;
62
63      FSMContext c(closed, FSMSkipAction::instance, &sd);
64
65      //ADD FSM FEED CODE HERE
66
67      return 0;
68  }
69
70
```

Anexo V

Ficheiro XMI do diagrama utilizado no primeiro exemplo dos resultados

```
1  <?xml version="1.0" encoding="UTF-8"?><uml:Model xmlns:uml="http://www.omg.org/
    spec/UML/20110701" xmlns:xmi="http://schema.omg.org/spec/XMI/2.1"
    xmi:version="2.1" xmi:id="_dXi0EKHrEeaxS6uaubzTAg" name="exemple1">
2    <eAnnotations xmi:id="_dXi0EaHrEeaxS6uaubzTAg" source="Objing">
3      <contents xmi:type="uml:Property" xmi:id="_dXi0EqHrEeaxS6uaubzTAg"
        name="exporterVersion">
4        <defaultValue xmi:type="uml:LiteralString" xmi:id="_dXi0E6HrEeaxS6uaubzTAg"
         value="3.0.0"/>
5      </contents>
6    </eAnnotations>
7    <ownedComment xmi:type="uml:Comment" xmi:id="_dXi0FKHrEeaxS6uaubzTAg">
8      <body/>
9    </ownedComment>
10   <packagedElement xmi:type="uml:StateMachine" xmi:id="_dXjbIKHrEeaxS6uaubzTAg"
      name="State Machine">
11     <ownedComment xmi:type="uml:Comment" xmi:id="_dXjbIaHrEeaxS6uaubzTAg">
12       <body/>
13     </ownedComment>
14     <region xmi:id="_dXjbIqHrEeaxS6uaubzTAg">
15       <subvertex xmi:type="uml:State" xmi:id="_dXjbI6HrEeaxS6uaubzTAg"
         name="Login">
16         <entry xmi:type="uml:OpaqueBehavior" xmi:id="_dXjbJKHrEeaxS6uaubzTAg">
17           <body>CheckBlocked</body>
18         </entry>
19       </subvertex>
20       <subvertex xmi:type="uml:State" xmi:id="_dXjbJaHrEeaxS6uaubzTAg"
         name="Administrator"/>
21       <subvertex xmi:type="uml:State" xmi:id="_dXjbJqHrEeaxS6uaubzTAg"
         name="Block"/>
22       <subvertex xmi:type="uml:Pseudostate" xmi:id="_dXjbJ6HrEeaxS6uaubzTAg"
         name="Initial State"/>
23       <transition xmi:type="uml:Transition" xmi:id="_dXjbKKHrEeaxS6uaubzTAg"
         name="Transition" source="_dXjbI6HrEeaxS6uaubzTAg"
         target="_dXjbJaHrEeaxS6uaubzTAg" guard="_dXjbKaHrEeaxS6uaubzTAg">
24         <ownedRule xmi:type="uml:Constraint" xmi:id="_dXjbKaHrEeaxS6uaubzTAg"
          name="IsCodeRight" constrainedElement="_dXjbKKHrEeaxS6uaubzTAg">
25           <specification xmi:type="uml:LiteralString"
            xmi:id="_dXjbKqHrEeaxS6uaubzTAg" value="IsCodeRight"/>
26         </ownedRule>
27         <trigger xmi:type="uml:Trigger" xmi:id="_dXjbK6HrEeaxS6uaubzTAg"
          name="PressOk" event="_dXjbNKHrEeaxS6uaubzTAg"/>
28       </transition>
29       <transition xmi:type="uml:Transition" xmi:id="_dXjbLKHrEeaxS6uaubzTAg"
         name="Transition2" source="_dXjbI6HrEeaxS6uaubzTAg"
         target="_dXjbJqHrEeaxS6uaubzTAg">
30         <trigger xmi:type="uml:Trigger" xmi:id="_dXjbLaHrEeaxS6uaubzTAg"
          name="BlockedEvent" event="_dXjbNaHrEeaxS6uaubzTAg"/>
31       </transition>
32       <transition xmi:type="uml:Transition" xmi:id="_dXjbLqHrEeaxS6uaubzTAg"
         name="Transition1" source="_dXjbI6HrEeaxS6uaubzTAg"
         target="_dXjbI6HrEeaxS6uaubzTAg" guard="_dXjbL6HrEeaxS6uaubzTAg">
33         <ownedRule xmi:type="uml:Constraint" xmi:id="_dXjbL6HrEeaxS6uaubzTAg"
```

```
            name="IsCodeWrong" constrainedElement="_dXjbLqHrEeaxS6uaubzTAg">
34           <specification xmi:type="uml:LiteralString"
              xmi:id="_dXjbMKHrEeaxS6uaubzTAg" value="IsCodeWrong"/>
35         </ownedRule>
36         <effect xmi:type="uml:OpaqueBehavior" xmi:id="_dXjbMaHrEeaxS6uaubzTAg">
37           <body>IncreaseFailCounter</body>
38         </effect>
39         <trigger xmi:type="uml:Trigger" xmi:id="_dXjbMqHrEeaxS6uaubzTAg"
            name="PressOk" event="_dXjbNKHrEeaxS6uaubzTAg"/>
40       </transition>
41       <transition xmi:type="uml:Transition" xmi:id="_dXjbM6HrEeaxS6uaubzTAg"
          name="Transition" source="_dXjbJ6HrEeaxS6uaubzTAg"
          target="_dXjbI6HrEeaxS6uaubzTAg"/>
42     </region>
43   </packagedElement>
44   <packagedElement xmi:type="uml:SignalEvent" xmi:id="_dXjbNKHrEeaxS6uaubzTAg"
      name="PressOk"/>
45   <packagedElement xmi:type="uml:SignalEvent" xmi:id="_dXjbNaHrEeaxS6uaubzTAg"
      name="BlockedEvent"/>
46 </uml:Model>
47
```

Anexo VI

Código fonte gerado para o primeiro exemplo dos resultados

```cpp
#include <iostream>
#include <cstdlib>
#include <ctime>

#include "../lib/FSM.h"
#include "../lib/State.h"
#include "../lib/Transition.h"
#include "../lib/FSMEvent.h"

using namespace UMinho::FSM;
using namespace std;

struct SharedData : public Object {
};

class CheckBlocked : public FSMAction {
    void execute(const FSMEvent &ev, FSMContext &c) {
        ///TODO: INSERT CODE HERE!!
    }
};

class IncreaseFailCounter : public FSMAction {
    void execute(const FSMEvent &ev, FSMContext &c) {
        ///TODO: INSERT CODE HERE!!
    }
};

class IsCodeRight : public FSMCondition {
    bool isSatisfied(const FSMEvent &ev, FSMContext &c) {
        ///TODO: INSERT CODE HERE!!
        return true;
    }
};

class IsCodeWrong : public FSMCondition {
    bool isSatisfied(const FSMEvent &ev, FSMContext &c) {
        ///TODO: INSERT CODE HERE!!
        return true;
    }
};

enum StateID { LOGIN = 1, ADMINISTRATOR, BLOCK };

enum EventID { PRESS_OK = 1, BLOCKED_EVENT };

string stateName(StateID s) {
    static string names[] = { "LOGIN", "ADMINISTRATOR", "BLOCK" };
    return names[(int) s-1];
}

string eventName(EventID e) {
    static string names[] = { "PRESS_OK", "BLOCKED_EVENT" };
```

```
53      return names[(int) e-1];
54  }
55
56  int main() {
57      CheckBlocked checkBlocked;
58      IncreaseFailCounter increaseFailCounter;
59
60      IsCodeRight isCodeRight;
61      IsCodeWrong isCodeWrong;
62
63      State login(LOGIN, checkBlocked);
64      State administrator(ADMINISTRATOR);
65      State block(BLOCK);
66
67      login.addTransition(PRESS_OK, administrator, FSMSkipAction::instance,
          isCodeRight);
68      login.addTransition(BLOCKED_EVENT, block);
69      login.addTransition(PRESS_OK, login, increaseFailCounter, isCodeWrong);
70
71      SharedData sd;
72
73      FSMContext c(login, FSMSkipAction::instance, &sd);
74
75      //ADD FSM FEED CODE HERE
76
77      return 0;
78  }
79
80
```

Anexo VII

Ficheiro XMI do diagrama utilizado no segundo exemplo dos resultados

```
<?xml version="1.0" encoding="UTF-8"?><uml:Model xmlns:uml="http://www.omg.org/spec/UML/20110701" xmlns:xmi="http://schema.omg.org/spec/XMI/2.1" xmi:version="2.1" xmi:id="_J9z1IKFwEeaSG4aYJfaebQ" name="example2">
 <eAnnotations xmi:id="_J90cMKFwEeaSG4aYJfaebQ" source="Objing">
  <contents xmi:type="uml:Property" xmi:id="_J90cMaFwEeaSG4aYJfaebQ" name="exporterVersion">
   <defaultValue xmi:type="uml:LiteralString" xmi:id="_J90cMqFwEeaSG4aYJfaebQ" value="3.0.0"/>
  </contents>
 </eAnnotations>
 <packagedElement xmi:type="uml:StateMachine" xmi:id="_J90cM6FwEeaSG4aYJfaebQ" name="State Machine">
  <region xmi:id="_J90cNKFwEeaSG4aYJfaebQ">
   <subvertex xmi:type="uml:State" xmi:id="_J90cNaFwEeaSG4aYJfaebQ" name="Full Power On">
    <entry xmi:type="uml:OpaqueBehavior" xmi:id="_J90cNqFwEeaSG4aYJfaebQ">
     <body>SetFull</body>
    </entry>
   </subvertex>
   <subvertex xmi:type="uml:State" xmi:id="_J90cN6FwEeaSG4aYJfaebQ" name="Set Time">
    <entry xmi:type="uml:OpaqueBehavior" xmi:id="_J90cOKFwEeaSG4aYJfaebQ">
     <body>UpdateTimer</body>
    </entry>
   </subvertex>
   <subvertex xmi:type="uml:State" xmi:id="_J90cOaFwEeaSG4aYJfaebQ" name="Cooking">
    <entry xmi:type="uml:OpaqueBehavior" xmi:id="_J90cOqFwEeaSG4aYJfaebQ">
     <body>TurnOnHeater</body>
    </entry>
    <exit xmi:type="uml:OpaqueBehavior" xmi:id="_J90cO6FwEeaSG4aYJfaebQ">
     <body>TurnOffHeater</body>
    </exit>
   </subvertex>
   <subvertex xmi:type="uml:State" xmi:id="_J90cPKFwEeaSG4aYJfaebQ" name="Cooking Complete"/>
   <subvertex xmi:type="uml:State" xmi:id="_J90cPaFwEeaSG4aYJfaebQ" name="Operation Enabled"/>
   <subvertex xmi:type="uml:State" xmi:id="_J90cPqFwEeaSG4aYJfaebQ" name="Operation Disabled"/>
   <subvertex xmi:type="uml:State" xmi:id="_J90cP6FwEeaSG4aYJfaebQ" name="Half Power On">
    <entry xmi:type="uml:OpaqueBehavior" xmi:id="_J90cQKFwEeaSG4aYJfaebQ">
     <body>SetHalf</body>
    </entry>
   </subvertex>
   <subvertex xmi:type="uml:State" xmi:id="_J90cQaFwEeaSG4aYJfaebQ" name="Idle">
    <entry xmi:type="uml:OpaqueBehavior" xmi:id="_J90cQqFwEeaSG4aYJfaebQ">
     <body>SetDisplayToCurrentTime</body>
    </entry>
   </subvertex>
```

```
40        <subvertex xmi:type="uml:Pseudostate" xmi:id="_J90cQ6FwEeaSG4aYJfaebQ"
           name="Initial State"/>
41        <transition xmi:type="uml:Transition" xmi:id="_J90cRKFwEeaSG4aYJfaebQ"
           name="Transition" source="_J90cNaFwEeaSG4aYJfaebQ"
           target="_J90cP6FwEeaSG4aYJfaebQ">
42          <trigger xmi:type="uml:Trigger" xmi:id="_J90cRaFwEeaSG4aYJfaebQ"
            name="HalfPowerButton" event="_J90caKFwEeaSG4aYJfaebQ"/>
43        </transition>
44        <transition xmi:type="uml:Transition" xmi:id="_J90cRqFwEeaSG4aYJfaebQ"
           name="Transition1" source="_J90cNaFwEeaSG4aYJfaebQ"
           target="_J90cN6FwEeaSG4aYJfaebQ">
45          <trigger xmi:type="uml:Trigger" xmi:id="_J90cR6FwEeaSG4aYJfaebQ"
            name="RotateTimer" event="_J90caaFwEeaSG4aYJfaebQ"/>
46        </transition>
47        <transition xmi:type="uml:Transition" xmi:id="_J90cSKFwEeaSG4aYJfaebQ"
           name="Transition1" source="_J90cN6FwEeaSG4aYJfaebQ"
           target="_J90cPqFwEeaSG4aYJfaebQ">
48          <trigger xmi:type="uml:Trigger" xmi:id="_J90cSaFwEeaSG4aYJfaebQ"
            name="DoorOpened" event="_J90caqFwEeaSG4aYJfaebQ"/>
49        </transition>
50        <transition xmi:type="uml:Transition" xmi:id="_J90cSqFwEeaSG4aYJfaebQ"
           name="Transition2" source="_J90cN6FwEeaSG4aYJfaebQ"
           target="_J90cPaFwEeaSG4aYJfaebQ">
51          <trigger xmi:type="uml:Trigger" xmi:id="_J90cS6FwEeaSG4aYJfaebQ"
            name="DoorClosed" event="_J90ca6FwEeaSG4aYJfaebQ"/>
52        </transition>
53        <transition xmi:type="uml:Transition" xmi:id="_J90cTKFwEeaSG4aYJfaebQ"
           name="Transition3" source="_J90cN6FwEeaSG4aYJfaebQ"
           target="_J90cQaFwEeaSG4aYJfaebQ">
54          <trigger xmi:type="uml:Trigger" xmi:id="_J90cTaFwEeaSG4aYJfaebQ"
            name="CancelButton" event="_J90cbqFwEeaSG4aYJfaebQ"/>
55        </transition>
56        <transition xmi:type="uml:Transition" xmi:id="_J90cTqFwEeaSG4aYJfaebQ"
           name="Transition" source="_J90cN6FwEeaSG4aYJfaebQ"
           target="_J90cN6FwEeaSG4aYJfaebQ">
57          <trigger xmi:type="uml:Trigger" xmi:id="_J90cT6FwEeaSG4aYJfaebQ"
            name="RotateTimer" event="_J90caaFwEeaSG4aYJfaebQ"/>
58        </transition>
59        <transition xmi:type="uml:Transition" xmi:id="_J90cUKFwEeaSG4aYJfaebQ"
           name="Transition" source="_J90cOaFwEeaSG4aYJfaebQ"
           target="_J90cPKFwEeaSG4aYJfaebQ">
60          <trigger xmi:type="uml:Trigger" xmi:id="_J90cUaFwEeaSG4aYJfaebQ"
            name="TimeOut" event="_J90cbaFwEeaSG4aYJfaebQ"/>
61        </transition>
62        <transition xmi:type="uml:Transition" xmi:id="_J90cUqFwEeaSG4aYJfaebQ"
           name="Transition1" source="_J90cOaFwEeaSG4aYJfaebQ"
           target="_J90cPqFwEeaSG4aYJfaebQ">
63          <trigger xmi:type="uml:Trigger" xmi:id="_J90cU6FwEeaSG4aYJfaebQ"
            name="DoorOpened" event="_J90caqFwEeaSG4aYJfaebQ"/>
64        </transition>
65        <transition xmi:type="uml:Transition" xmi:id="_J90cVKFwEeaSG4aYJfaebQ"
           name="Transition" source="_J90cPKFwEeaSG4aYJfaebQ"
```

```
          target="_J90cQaFwEeaSG4aYJfaebQ">
66         <trigger xmi:type="uml:Trigger" xmi:id="_J90cVaFwEeaSG4aYJfaebQ"
           name="DoorOpened" event="_J90caqFwEeaSG4aYJfaebQ"/>
67       </transition>
68       <transition xmi:type="uml:Transition" xmi:id="_J90cVqFwEeaSG4aYJfaebQ"
          name="Transition" source="_J90cPaFwEeaSG4aYJfaebQ"
          target="_J90cOaFwEeaSG4aYJfaebQ" guard="_J90cV6FwEeaSG4aYJfaebQ">
69         <ownedRule xmi:type="uml:Constraint" xmi:id="_J90cV6FwEeaSG4aYJfaebQ"
           name="TimerGreaterThanZero"
           constrainedElement="_J90cVqFwEeaSG4aYJfaebQ">
70           <specification xmi:type="uml:LiteralString"
            xmi:id="_J90cWKFwEeaSG4aYJfaebQ" value="TimerGreaterThanZero"/>
71         </ownedRule>
72         <trigger xmi:type="uml:Trigger" xmi:id="_J90cWaFwEeaSG4aYJfaebQ"
           name="StartButton" event="_J90cbKFwEeaSG4aYJfaebQ"/>
73       </transition>
74       <transition xmi:type="uml:Transition" xmi:id="_J90cWqFwEeaSG4aYJfaebQ"
          name="Transition" source="_J90cPqFwEeaSG4aYJfaebQ"
          target="_J90cPaFwEeaSG4aYJfaebQ">
75         <trigger xmi:type="uml:Trigger" xmi:id="_J90cW6FwEeaSG4aYJfaebQ"
           name="DoorClosed" event="_J90ca6FwEeaSG4aYJfaebQ"/>
76       </transition>
77       <transition xmi:type="uml:Transition" xmi:id="_J90cXKFwEeaSG4aYJfaebQ"
          name="Transition1" source="_J90cPqFwEeaSG4aYJfaebQ"
          target="_J90cQaFwEeaSG4aYJfaebQ">
78         <trigger xmi:type="uml:Trigger" xmi:id="_J90cXaFwEeaSG4aYJfaebQ"
           name="CancelButton" event="_J90cbqFwEeaSG4aYJfaebQ"/>
79       </transition>
80       <transition xmi:type="uml:Transition" xmi:id="_J90cXqFwEeaSG4aYJfaebQ"
          name="Transition" source="_J90cP6FwEeaSG4aYJfaebQ"
          target="_J90cNaFwEeaSG4aYJfaebQ">
81         <trigger xmi:type="uml:Trigger" xmi:id="_J90cX6FwEeaSG4aYJfaebQ"
           name="FullPowerButton" event="_J90cZ6FwEeaSG4aYJfaebQ"/>
82       </transition>
83       <transition xmi:type="uml:Transition" xmi:id="_J90cYKFwEeaSG4aYJfaebQ"
          name="Transition1" source="_J90cP6FwEeaSG4aYJfaebQ"
          target="_J90cN6FwEeaSG4aYJfaebQ">
84         <trigger xmi:type="uml:Trigger" xmi:id="_J90cYaFwEeaSG4aYJfaebQ"
           name="RotateTimer" event="_J90caaFwEeaSG4aYJfaebQ"/>
85       </transition>
86       <transition xmi:type="uml:Transition" xmi:id="_J90cYqFwEeaSG4aYJfaebQ"
          name="Transition" source="_J90cQaFwEeaSG4aYJfaebQ"
          target="_J90cNaFwEeaSG4aYJfaebQ">
87         <trigger xmi:type="uml:Trigger" xmi:id="_J90cY6FwEeaSG4aYJfaebQ"
           name="FullPowerButton" event="_J90cZ6FwEeaSG4aYJfaebQ"/>
88       </transition>
89       <transition xmi:type="uml:Transition" xmi:id="_J90cZKFwEeaSG4aYJfaebQ"
          name="Transition1" source="_J90cQaFwEeaSG4aYJfaebQ"
          target="_J90cP6FwEeaSG4aYJfaebQ">
90         <trigger xmi:type="uml:Trigger" xmi:id="_J90cZaFwEeaSG4aYJfaebQ"
           name="HalfPowerButton" event="_J90caKFwEeaSG4aYJfaebQ"/>
91       </transition>
```

```
92        <transition xmi:type="uml:Transition" xmi:id="_J90cZqFwEeaSG4aYJfaebQ"
           name="Transition" source="_J90cQ6FwEeaSG4aYJfaebQ"
           target="_J90cQaFwEeaSG4aYJfaebQ"/>
93      </region>
94    </packagedElement>
95    <packagedElement xmi:type="uml:SignalEvent" xmi:id="_J90cZ6FwEeaSG4aYJfaebQ"
       name="FullPowerButton"/>
96    <packagedElement xmi:type="uml:SignalEvent" xmi:id="_J90caKFwEeaSG4aYJfaebQ"
       name="HalfPowerButton"/>
97    <packagedElement xmi:type="uml:SignalEvent" xmi:id="_J90caaFwEeaSG4aYJfaebQ"
       name="RotateTimer"/>
98    <packagedElement xmi:type="uml:SignalEvent" xmi:id="_J90caqFwEeaSG4aYJfaebQ"
       name="DoorOpened"/>
99    <packagedElement xmi:type="uml:SignalEvent" xmi:id="_J90ca6FwEeaSG4aYJfaebQ"
       name="DoorClosed"/>
100   <packagedElement xmi:type="uml:SignalEvent" xmi:id="_J90cbKFwEeaSG4aYJfaebQ"
       name="StartButton"/>
101   <packagedElement xmi:type="uml:SignalEvent" xmi:id="_J90cbaFwEeaSG4aYJfaebQ"
       name="TimeOut"/>
102   <packagedElement xmi:type="uml:SignalEvent" xmi:id="_J90cbqFwEeaSG4aYJfaebQ"
       name="CancelButton"/>
103 </uml:Model>
104
```

Anexo VIII

Código fonte gerado para o segundo exemplo dos resultados

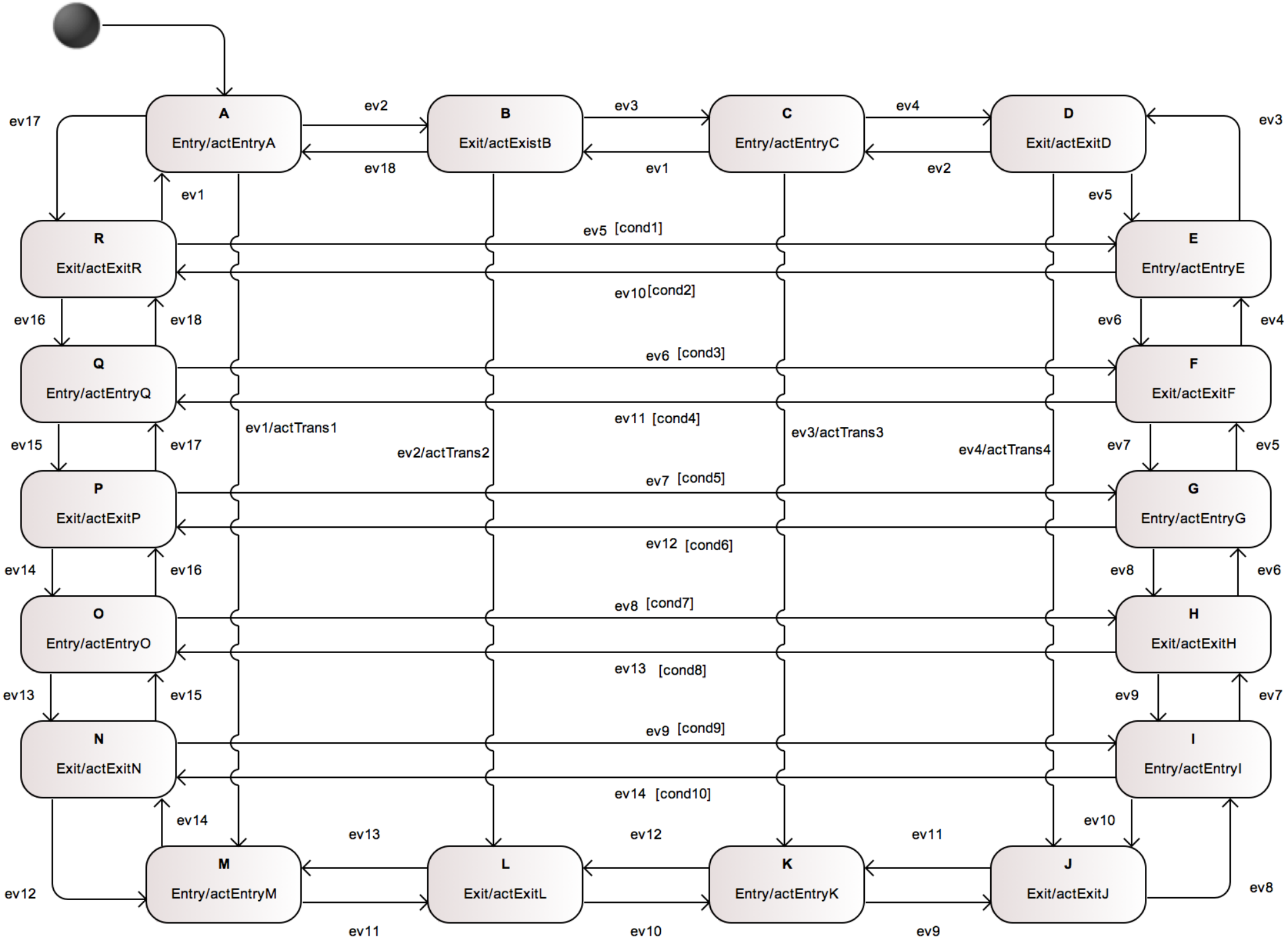
```cpp
#include <iostream>
#include <cstdlib>
#include <ctime>

#include "../lib/FSM.h"
#include "../lib/State.h"
#include "../lib/Transition.h"
#include "../lib/FSMEvent.h"

using namespace UMinho::FSM;
using namespace std;

struct SharedData : public Object {
};

class SetDisplayToCurrentTime : public FSMAction {
    void execute(const FSMEvent &ev, FSMContext &c) {
        ///TODO: INSERT CODE HERE!!
    }
};

class SetFull : public FSMAction {
    void execute(const FSMEvent &ev, FSMContext &c) {
        ///TODO: INSERT CODE HERE!!
    }
};

class UpdateTimer : public FSMAction {
    void execute(const FSMEvent &ev, FSMContext &c) {
        ///TODO: INSERT CODE HERE!!
    }
};

class TurnOnHeater : public FSMAction {
    void execute(const FSMEvent &ev, FSMContext &c) {
        ///TODO: INSERT CODE HERE!!
    }
};

class TurnOffHeater : public FSMAction {
    void execute(const FSMEvent &ev, FSMContext &c) {
        ///TODO: INSERT CODE HERE!!
    }
};

class SetHalf : public FSMAction {
    void execute(const FSMEvent &ev, FSMContext &c) {
        ///TODO: INSERT CODE HERE!!
    }
};

class TimerGreaterThanZero : public FSMCondition {
```

```
53      bool isSatisfied(const FSMEvent &ev, FSMContext &c) {
54          ///TODO: INSERT CODE HERE!!
55          return true;
56      }
57  };
58
59  enum StateID { IDLE = 1, FULL_POWER_ON, SET_TIME, COOKING, COOKING_COMPLETE,
      OPERATION_ENABLED, OPERATION_DISABLED, HALF_POWER_ON };
60
61  enum EventID { HALF_POWER_BUTTON = 1, ROTATE_TIMER, DOOR_OPENED, DOOR_CLOSED,
      CANCEL_BUTTON, TIME_OUT, START_BUTTON, FULL_POWER_BUTTON };
62
63  string stateName(StateID s) {
64      static string names[] = { "IDLE", "FULL_POWER_ON", "SET_TIME", "COOKING",
          "COOKING_COMPLETE", "OPERATION_ENABLED", "OPERATION_DISABLED",
          "HALF_POWER_ON" };
65      return names[(int) s-1];
66  }
67
68  string eventName(EventID e) {
69      static string names[] = { "HALF_POWER_BUTTON", "ROTATE_TIMER", "DOOR_OPENED",
           "DOOR_CLOSED", "CANCEL_BUTTON", "TIME_OUT", "START_BUTTON",
          "FULL_POWER_BUTTON" };
70      return names[(int) e-1];
71  }
72
73  int main() {
74      SetDisplayToCurrentTime setDisplayToCurrentTime;
75      SetFull setFull;
76      UpdateTimer updateTimer;
77      TurnOnHeater turnOnHeater;
78      TurnOffHeater turnOffHeater;
79      SetHalf setHalf;
80
81      TimerGreaterThanZero timerGreaterThanZero;
82
83      State idle(IDLE, setDisplayToCurrentTime);
84      State fullPowerOn(FULL_POWER_ON, setFull);
85      State setTime(SET_TIME, updateTimer);
86      State cooking(COOKING, turnOnHeater, turnOffHeater);
87      State cookingComplete(COOKING_COMPLETE);
88      State operationEnabled(OPERATION_ENABLED);
89      State operationDisabled(OPERATION_DISABLED);
90      State halfPowerOn(HALF_POWER_ON, setHalf);
91
92      fullPowerOn.addTransition(HALF_POWER_BUTTON, halfPowerOn);
93      fullPowerOn.addTransition(ROTATE_TIMER, setTime);
94      setTime.addTransition(DOOR_OPENED, operationDisabled);
95      setTime.addTransition(DOOR_CLOSED, operationEnabled);
96      setTime.addTransition(CANCEL_BUTTON, idle);
97      setTime.addTransition(ROTATE_TIMER, setTime);
98      cooking.addTransition(TIME_OUT, cookingComplete);
```

```
 99      cooking.addTransition(DOOR_OPENED, operationDisabled);
100      cookingComplete.addTransition(DOOR_OPENED, idle);
101      operationEnabled.addTransition(START_BUTTON, cooking,
           FSMSkipAction::instance, timerGreaterThanZero);
102      operationDisabled.addTransition(DOOR_CLOSED, operationEnabled);
103      operationDisabled.addTransition(CANCEL_BUTTON, idle);
104      halfPowerOn.addTransition(FULL_POWER_BUTTON, fullPowerOn);
105      halfPowerOn.addTransition(ROTATE_TIMER, setTime);
106      idle.addTransition(FULL_POWER_BUTTON, fullPowerOn);
107      idle.addTransition(HALF_POWER_BUTTON, halfPowerOn);
108
109      SharedData sd;
110
111      FSMContext c(idle, FSMSkipAction::instance, &sd);
112
113      //ADD FSM FEED CODE HERE
114
115      return 0;
116  }
117
118
```

Anexo IX

Diagrama de máquina de estados utilizado para teste de stress da ferramenta


A
Entry/actEntryA
B
Exit/actExistB
C
Entry/actEntryC
D
Exit/actExitD
R
Exit/actExitR
E
Entry/actEntryE
Q
Entry/actEntryQ
F
Exit/actExitF
P
Exit/actExitP
G
Entry/actEntryG
O
Entry/actEntryO
H
Exit/actExitH
N
Exit/actExitN
I
Entry/actEntryI
M
Entry/actEntryM
L
Exit/actExitL
K
Entry/actEntryK
J
Exit/actExitJ
ev1
ev2
ev3
ev4
ev5 [cond1]
ev6 [cond3]
ev7 [cond5]
ev8 [cond7]
ev9 [cond9]
ev10 [cond2]
ev11 [cond4]
ev12 [cond6]
ev13 [cond8]
ev14 [cond10]
ev1/actTrans1
ev2/actTrans2
ev3/actTrans3
ev4/actTrans4

Anexo X

Ficheiro XMI do diagrama utilizado para teste de stress

```
<?xml version="1.0" encoding="UTF-8"?><uml:Model xmlns:uml="http://www.omg.org/spec/UML/20110701" xmlns:xmi="http://schema.omg.org/spec/XMI/2.1" xmi:version="2.1" xmi:id="_vp5aAKRwEeahGMeIhypmvg" name="stresstest">
 <eAnnotations xmi:id="_vp5aAaRwEeahGMeIhypmvg" source="Objing">
  <contents xmi:type="uml:Property" xmi:id="_vp5aAqRwEeahGMeIhypmvg" name="exporterVersion">
   <defaultValue xmi:type="uml:LiteralString" xmi:id="_vp5aA6RwEeahGMeIhypmvg" value="3.0.0"/>
  </contents>
 </eAnnotations>
 <packagedElement xmi:type="uml:StateMachine" xmi:id="_vp5aBKRwEeahGMeIhypmvg" name="State Machine">
  <region xmi:id="_vp5aBaRwEeahGMeIhypmvg">
   <subvertex xmi:type="uml:State" xmi:id="_vp5aBqRwEeahGMeIhypmvg" name="A">
    <entry xmi:type="uml:OpaqueBehavior" xmi:id="_vp5aB6RwEeahGMeIhypmvg">
     <body>actEntryA</body>
    </entry>
   </subvertex>
   <subvertex xmi:type="uml:State" xmi:id="_vp5aCKRwEeahGMeIhypmvg" name="B">
    <exit xmi:type="uml:OpaqueBehavior" xmi:id="_vp5aCaRwEeahGMeIhypmvg">
     <body>actExistB</body>
    </exit>
   </subvertex>
   <subvertex xmi:type="uml:State" xmi:id="_vp5aCqRwEeahGMeIhypmvg" name="C">
    <entry xmi:type="uml:OpaqueBehavior" xmi:id="_vp5aC6RwEeahGMeIhypmvg">
     <body>actEntryC</body>
    </entry>
   </subvertex>
   <subvertex xmi:type="uml:State" xmi:id="_vp5aDKRwEeahGMeIhypmvg" name="D">
    <exit xmi:type="uml:OpaqueBehavior" xmi:id="_vp5aDaRwEeahGMeIhypmvg">
     <body>actExitD</body>
    </exit>
   </subvertex>
   <subvertex xmi:type="uml:State" xmi:id="_vp5aDqRwEeahGMeIhypmvg" name="R">
    <exit xmi:type="uml:OpaqueBehavior" xmi:id="_vp5aD6RwEeahGMeIhypmvg">
     <body>actExitR</body>
    </exit>
   </subvertex>
   <subvertex xmi:type="uml:State" xmi:id="_vp5aEKRwEeahGMeIhypmvg" name="Q">
    <entry xmi:type="uml:OpaqueBehavior" xmi:id="_vp5aEaRwEeahGMeIhypmvg">
     <body>actEntryQ</body>
    </entry>
   </subvertex>
   <subvertex xmi:type="uml:State" xmi:id="_vp5aEqRwEeahGMeIhypmvg" name="P">
    <exit xmi:type="uml:OpaqueBehavior" xmi:id="_vp5aE6RwEeahGMeIhypmvg">
     <body>actExitP</body>
    </exit>
   </subvertex>
   <subvertex xmi:type="uml:State" xmi:id="_vp5aFKRwEeahGMeIhypmvg" name="O">
    <entry xmi:type="uml:OpaqueBehavior" xmi:id="_vp5aFaRwEeahGMeIhypmvg">
     <body>actEntryO</body>
    </entry>
```

```
48        </subvertex>
49        <subvertex xmi:type="uml:State" xmi:id="_vp5aFqRwEeahGMeIhypmvg" name="E">
50          <entry xmi:type="uml:OpaqueBehavior" xmi:id="_vp5aF6RwEeahGMeIhypmvg">
51            <body>actEntryE</body>
52          </entry>
53        </subvertex>
54        <subvertex xmi:type="uml:State" xmi:id="_vp5aGKRwEeahGMeIhypmvg" name="F">
55          <exit xmi:type="uml:OpaqueBehavior" xmi:id="_vp5aGaRwEeahGMeIhypmvg">
56            <body>actExitF</body>
57          </exit>
58        </subvertex>
59        <subvertex xmi:type="uml:State" xmi:id="_vp5aGqRwEeahGMeIhypmvg" name="G">
60          <entry xmi:type="uml:OpaqueBehavior" xmi:id="_vp5aG6RwEeahGMeIhypmvg">
61            <body>actEntryG</body>
62          </entry>
63        </subvertex>
64        <subvertex xmi:type="uml:State" xmi:id="_vp5aHKRwEeahGMeIhypmvg" name="H">
65          <exit xmi:type="uml:OpaqueBehavior" xmi:id="_vp5aHaRwEeahGMeIhypmvg">
66            <body>actExitH</body>
67          </exit>
68        </subvertex>
69        <subvertex xmi:type="uml:State" xmi:id="_vp5aHqRwEeahGMeIhypmvg" name="N">
70          <exit xmi:type="uml:OpaqueBehavior" xmi:id="_vp5aH6RwEeahGMeIhypmvg">
71            <body>actExitN</body>
72          </exit>
73        </subvertex>
74        <subvertex xmi:type="uml:State" xmi:id="_vp5aIKRwEeahGMeIhypmvg" name="M">
75          <entry xmi:type="uml:OpaqueBehavior" xmi:id="_vp5aIaRwEeahGMeIhypmvg">
76            <body>actEntryM</body>
77          </entry>
78        </subvertex>
79        <subvertex xmi:type="uml:State" xmi:id="_vp5aIqRwEeahGMeIhypmvg" name="L">
80          <exit xmi:type="uml:OpaqueBehavior" xmi:id="_vp5aI6RwEeahGMeIhypmvg">
81            <body>actExitL</body>
82          </exit>
83        </subvertex>
84        <subvertex xmi:type="uml:State" xmi:id="_vp5aJKRwEeahGMeIhypmvg" name="K">
85          <entry xmi:type="uml:OpaqueBehavior" xmi:id="_vp5aJaRwEeahGMeIhypmvg">
86            <body>actEntryK</body>
87          </entry>
88        </subvertex>
89        <subvertex xmi:type="uml:State" xmi:id="_vp5aJqRwEeahGMeIhypmvg" name="J">
90          <exit xmi:type="uml:OpaqueBehavior" xmi:id="_vp5aJ6RwEeahGMeIhypmvg">
91            <body>actExitJ</body>
92          </exit>
93        </subvertex>
94        <subvertex xmi:type="uml:State" xmi:id="_vp5aKKRwEeahGMeIhypmvg" name="I">
95          <entry xmi:type="uml:OpaqueBehavior" xmi:id="_vp5aKaRwEeahGMeIhypmvg">
96            <body>actEntryI</body>
97          </entry>
98        </subvertex>
99        <subvertex xmi:type="uml:Pseudostate" xmi:id="_vp5aKqRwEeahGMeIhypmvg"
```

```
          name="Initial State"/>
100       <transition xmi:type="uml:Transition" xmi:id="_vp5aK6RwEeahGMeIhypmvg"
          name="Transition" source="_vp5aBqRwEeahGMeIhypmvg"
          target="_vp5aCKRwEeahGMeIhypmvg">
101        <trigger xmi:type="uml:Trigger" xmi:id="_vp5aLKRwEeahGMeIhypmvg"
           name="ev2" event="_vp5aqaRwEeahGMeIhypmvg"/>
102       </transition>
103       <transition xmi:type="uml:Transition" xmi:id="_vp5aLaRwEeahGMeIhypmvg"
          name="Transition1" source="_vp5aBqRwEeahGMeIhypmvg"
          target="_vp5aDqRwEeahGMeIhypmvg">
104        <trigger xmi:type="uml:Trigger" xmi:id="_vp5aLqRwEeahGMeIhypmvg"
           name="ev17" event="_vp5auKRwEeahGMeIhypmvg"/>
105       </transition>
106       <transition xmi:type="uml:Transition" xmi:id="_vp5aL6RwEeahGMeIhypmvg"
          name="Transition2" source="_vp5aBqRwEeahGMeIhypmvg"
          target="_vp5aIKRwEeahGMeIhypmvg">
107        <effect xmi:type="uml:OpaqueBehavior" xmi:id="_vp5aMKRwEeahGMeIhypmvg">
108          <body>actTrans1</body>
109        </effect>
110        <trigger xmi:type="uml:Trigger" xmi:id="_vp5aMaRwEeahGMeIhypmvg"
           name="ev1" event="_vp5aqKRwEeahGMeIhypmvg"/>
111       </transition>
112       <transition xmi:type="uml:Transition" xmi:id="_vp5aMqRwEeahGMeIhypmvg"
          name="Transition" source="_vp5aCKRwEeahGMeIhypmvg"
          target="_vp5aCqRwEeahGMeIhypmvg">
113        <trigger xmi:type="uml:Trigger" xmi:id="_vp5aM6RwEeahGMeIhypmvg"
           name="ev3" event="_vp5aqqRwEeahGMeIhypmvg"/>
114       </transition>
115       <transition xmi:type="uml:Transition" xmi:id="_vp5aNKRwEeahGMeIhypmvg"
          name="Transition1" source="_vp5aCKRwEeahGMeIhypmvg"
          target="_vp5aBqRwEeahGMeIhypmvg">
116        <trigger xmi:type="uml:Trigger" xmi:id="_vp5aNaRwEeahGMeIhypmvg"
           name="ev18" event="_vp5auaRwEeahGMeIhypmvg"/>
117       </transition>
118       <transition xmi:type="uml:Transition" xmi:id="_vp5aNqRwEeahGMeIhypmvg"
          name="Transition2" source="_vp5aCKRwEeahGMeIhypmvg"
          target="_vp5aIqRwEeahGMeIhypmvg">
119        <effect xmi:type="uml:OpaqueBehavior" xmi:id="_vp5aN6RwEeahGMeIhypmvg">
120          <body>actTrans2</body>
121        </effect>
122        <trigger xmi:type="uml:Trigger" xmi:id="_vp5aOKRwEeahGMeIhypmvg"
           name="ev2" event="_vp5aqaRwEeahGMeIhypmvg"/>
123       </transition>
124       <transition xmi:type="uml:Transition" xmi:id="_vp5aOaRwEeahGMeIhypmvg"
          name="Transition" source="_vp5aCqRwEeahGMeIhypmvg"
          target="_vp5aDKRwEeahGMeIhypmvg">
125        <trigger xmi:type="uml:Trigger" xmi:id="_vp5aOqRwEeahGMeIhypmvg"
           name="ev4" event="_vp5aq6RwEeahGMeIhypmvg"/>
126       </transition>
127       <transition xmi:type="uml:Transition" xmi:id="_vp5aO6RwEeahGMeIhypmvg"
          name="Transition1" source="_vp5aCqRwEeahGMeIhypmvg"
          target="_vp5aCKRwEeahGMeIhypmvg">
```

```
128           <trigger xmi:type="uml:Trigger" xmi:id="_vp5aPKRwEeahGMeIhypmvg"
               name="ev1" event="_vp5aqKRwEeahGMeIhypmvg"/>
129         </transition>
130         <transition xmi:type="uml:Transition" xmi:id="_vp5aPaRwEeahGMeIhypmvg"
           name="Transition2" source="_vp5aCqRwEeahGMeIhypmvg"
           target="_vp5aJKRwEeahGMeIhypmvg">
131           <effect xmi:type="uml:OpaqueBehavior" xmi:id="_vp5aPqRwEeahGMeIhypmvg">
132             <body>actTrans3</body>
133           </effect>
134           <trigger xmi:type="uml:Trigger" xmi:id="_vp5aP6RwEeahGMeIhypmvg"
            name="ev3" event="_vp5aqqRwEeahGMeIhypmvg"/>
135         </transition>
136         <transition xmi:type="uml:Transition" xmi:id="_vp5aQKRwEeahGMeIhypmvg"
           name="Transition" source="_vp5aDKRwEeahGMeIhypmvg"
           target="_vp5aFqRwEeahGMeIhypmvg">
137           <trigger xmi:type="uml:Trigger" xmi:id="_vp5aQaRwEeahGMeIhypmvg"
            name="ev5" event="_vp5arKRwEeahGMeIhypmvg"/>
138         </transition>
139         <transition xmi:type="uml:Transition" xmi:id="_vp5aQqRwEeahGMeIhypmvg"
           name="Transition1" source="_vp5aDKRwEeahGMeIhypmvg"
           target="_vp5aCqRwEeahGMeIhypmvg">
140           <trigger xmi:type="uml:Trigger" xmi:id="_vp5aQ6RwEeahGMeIhypmvg"
            name="ev2" event="_vp5aqaRwEeahGMeIhypmvg"/>
141         </transition>
142         <transition xmi:type="uml:Transition" xmi:id="_vp5aRKRwEeahGMeIhypmvg"
           name="Transition2" source="_vp5aDKRwEeahGMeIhypmvg"
           target="_vp5aJqRwEeahGMeIhypmvg">
143           <effect xmi:type="uml:OpaqueBehavior" xmi:id="_vp5aRaRwEeahGMeIhypmvg">
144             <body>actTrans4</body>
145           </effect>
146           <trigger xmi:type="uml:Trigger" xmi:id="_vp5aRqRwEeahGMeIhypmvg"
            name="ev4" event="_vp5aq6RwEeahGMeIhypmvg"/>
147         </transition>
148         <transition xmi:type="uml:Transition" xmi:id="_vp5aR6RwEeahGMeIhypmvg"
           name="Transition" source="_vp5aDqRwEeahGMeIhypmvg"
           target="_vp5aBqRwEeahGMeIhypmvg">
149           <trigger xmi:type="uml:Trigger" xmi:id="_vp5aSKRwEeahGMeIhypmvg"
            name="ev1" event="_vp5aqKRwEeahGMeIhypmvg"/>
150         </transition>
151         <transition xmi:type="uml:Transition" xmi:id="_vp5aSaRwEeahGMeIhypmvg"
           name="Transition1" source="_vp5aDqRwEeahGMeIhypmvg"
           target="_vp5aEKRwEeahGMeIhypmvg">
152           <trigger xmi:type="uml:Trigger" xmi:id="_vp5aSqRwEeahGMeIhypmvg"
            name="ev16" event="_vp5at6RwEeahGMeIhypmvg"/>
153         </transition>
154         <transition xmi:type="uml:Transition" xmi:id="_vp5aS6RwEeahGMeIhypmvg"
           name="Transition2" source="_vp5aDqRwEeahGMeIhypmvg"
           target="_vp5aFqRwEeahGMeIhypmvg" guard="_vp5aTKRwEeahGMeIhypmvg">
155           <ownedRule xmi:type="uml:Constraint" xmi:id="_vp5aTKRwEeahGMeIhypmvg"
            name="cond1" constrainedElement="_vp5aS6RwEeahGMeIhypmvg">
156             <specification xmi:type="uml:LiteralString"
              xmi:id="_vp5aTaRwEeahGMeIhypmvg" value="cond1"/>
```

```
157          </ownedRule>
158          <trigger xmi:type="uml:Trigger" xmi:id="_vp5aTqRwEeahGMeIhypmvg"
              name="ev5" event="_vp5arKRwEeahGMeIhypmvg"/>
159        </transition>
160        <transition xmi:type="uml:Transition" xmi:id="_vp5aT6RwEeahGMeIhypmvg"
             name="Transition" source="_vp5aEKRwEeahGMeIhypmvg"
             target="_vp5aDqRwEeahGMeIhypmvg">
161          <trigger xmi:type="uml:Trigger" xmi:id="_vp5aUKRwEeahGMeIhypmvg"
              name="ev18" event="_vp5auaRwEeahGMeIhypmvg"/>
162        </transition>
163        <transition xmi:type="uml:Transition" xmi:id="_vp5aUaRwEeahGMeIhypmvg"
             name="Transition1" source="_vp5aEKRwEeahGMeIhypmvg"
             target="_vp5aEqRwEeahGMeIhypmvg">
164          <trigger xmi:type="uml:Trigger" xmi:id="_vp5aUqRwEeahGMeIhypmvg"
              name="ev15" event="_vp5atqRwEeahGMeIhypmvg"/>
165        </transition>
166        <transition xmi:type="uml:Transition" xmi:id="_vp5aU6RwEeahGMeIhypmvg"
             name="Transition2" source="_vp5aEKRwEeahGMeIhypmvg"
             target="_vp5aGKRwEeahGMeIhypmvg" guard="_vp5aVKRwEeahGMeIhypmvg">
167          <ownedRule xmi:type="uml:Constraint" xmi:id="_vp5aVKRwEeahGMeIhypmvg"
              name="cond3" constrainedElement="_vp5aU6RwEeahGMeIhypmvg">
168            <specification xmi:type="uml:LiteralString"
               xmi:id="_vp5aVaRwEeahGMeIhypmvg" value="cond3"/>
169          </ownedRule>
170          <trigger xmi:type="uml:Trigger" xmi:id="_vp5aVqRwEeahGMeIhypmvg"
              name="ev6" event="_vp5araRwEeahGMeIhypmvg"/>
171        </transition>
172        <transition xmi:type="uml:Transition" xmi:id="_vp5aV6RwEeahGMeIhypmvg"
             name="Transition" source="_vp5aEqRwEeahGMeIhypmvg"
             target="_vp5aEKRwEeahGMeIhypmvg">
173          <trigger xmi:type="uml:Trigger" xmi:id="_vp5aWKRwEeahGMeIhypmvg"
              name="ev17" event="_vp5auKRwEeahGMeIhypmvg"/>
174        </transition>
175        <transition xmi:type="uml:Transition" xmi:id="_vp5aWaRwEeahGMeIhypmvg"
             name="Transition1" source="_vp5aEqRwEeahGMeIhypmvg"
             target="_vp5aFKRwEeahGMeIhypmvg">
176          <trigger xmi:type="uml:Trigger" xmi:id="_vp5aWqRwEeahGMeIhypmvg"
              name="ev14" event="_vp5ataRwEeahGMeIhypmvg"/>
177        </transition>
178        <transition xmi:type="uml:Transition" xmi:id="_vp5aW6RwEeahGMeIhypmvg"
             name="Transition2" source="_vp5aEqRwEeahGMeIhypmvg"
             target="_vp5aGqRwEeahGMeIhypmvg" guard="_vp5aXKRwEeahGMeIhypmvg">
179          <ownedRule xmi:type="uml:Constraint" xmi:id="_vp5aXKRwEeahGMeIhypmvg"
              name="cond5" constrainedElement="_vp5aW6RwEeahGMeIhypmvg">
180            <specification xmi:type="uml:LiteralString"
               xmi:id="_vp5aXaRwEeahGMeIhypmvg" value="cond5"/>
181          </ownedRule>
182          <trigger xmi:type="uml:Trigger" xmi:id="_vp5aXqRwEeahGMeIhypmvg"
              name="ev7" event="_vp5arqRwEeahGMeIhypmvg"/>
183        </transition>
184        <transition xmi:type="uml:Transition" xmi:id="_vp5aX6RwEeahGMeIhypmvg"
             name="Transition" source="_vp5aFKRwEeahGMeIhypmvg"
```

```
          target="_vp5aEqRwEeahGMeIhypmvg">
185        <trigger xmi:type="uml:Trigger" xmi:id="_vp5aYKRwEeahGMeIhypmvg"
            name="ev16" event="_vp5at6RwEeahGMeIhypmvg"/>
186       </transition>
187       <transition xmi:type="uml:Transition" xmi:id="_vp5aYaRwEeahGMeIhypmvg"
          name="Transition1" source="_vp5aFKRwEeahGMeIhypmvg"
          target="_vp5aHqRwEeahGMeIhypmvg">
188        <trigger xmi:type="uml:Trigger" xmi:id="_vp5aYqRwEeahGMeIhypmvg"
            name="ev13" event="_vp5atKRwEeahGMeIhypmvg"/>
189       </transition>
190       <transition xmi:type="uml:Transition" xmi:id="_vp5aY6RwEeahGMeIhypmvg"
          name="Transition2" source="_vp5aFKRwEeahGMeIhypmvg"
          target="_vp5aHKRwEeahGMeIhypmvg" guard="_vp5aZKRwEeahGMeIhypmvg">
191        <ownedRule xmi:type="uml:Constraint" xmi:id="_vp5aZKRwEeahGMeIhypmvg"
            name="cond7" constrainedElement="_vp5aY6RwEeahGMeIhypmvg">
192          <specification xmi:type="uml:LiteralString"
             xmi:id="_vp5aZaRwEeahGMeIhypmvg" value="cond7"/>
193        </ownedRule>
194        <trigger xmi:type="uml:Trigger" xmi:id="_vp5aZqRwEeahGMeIhypmvg"
            name="ev8" event="_vp5ar6RwEeahGMeIhypmvg"/>
195       </transition>
196       <transition xmi:type="uml:Transition" xmi:id="_vp5aZ6RwEeahGMeIhypmvg"
          name="Transition" source="_vp5aFqRwEeahGMeIhypmvg"
          target="_vp5aGKRwEeahGMeIhypmvg">
197        <trigger xmi:type="uml:Trigger" xmi:id="_vp5aaKRwEeahGMeIhypmvg"
            name="ev6" event="_vp5araRwEeahGMeIhypmvg"/>
198       </transition>
199       <transition xmi:type="uml:Transition" xmi:id="_vp5aaaRwEeahGMeIhypmvg"
          name="Transition1" source="_vp5aFqRwEeahGMeIhypmvg"
          target="_vp5aDKRwEeahGMeIhypmvg">
200        <trigger xmi:type="uml:Trigger" xmi:id="_vp5aaqRwEeahGMeIhypmvg"
            name="ev3" event="_vp5aqqRwEeahGMeIhypmvg"/>
201       </transition>
202       <transition xmi:type="uml:Transition" xmi:id="_vp5aa6RwEeahGMeIhypmvg"
          name="Transition2" source="_vp5aFqRwEeahGMeIhypmvg"
          target="_vp5aDqRwEeahGMeIhypmvg" guard="_vp5abKRwEeahGMeIhypmvg">
203        <ownedRule xmi:type="uml:Constraint" xmi:id="_vp5abKRwEeahGMeIhypmvg"
            name="cond2" constrainedElement="_vp5aa6RwEeahGMeIhypmvg">
204          <specification xmi:type="uml:LiteralString"
             xmi:id="_vp5abaRwEeahGMeIhypmvg" value="cond2"/>
205        </ownedRule>
206        <trigger xmi:type="uml:Trigger" xmi:id="_vp5abqRwEeahGMeIhypmvg"
            name="ev10" event="_vp5asaRwEeahGMeIhypmvg"/>
207       </transition>
208       <transition xmi:type="uml:Transition" xmi:id="_vp5ab6RwEeahGMeIhypmvg"
          name="Transition" source="_vp5aGKRwEeahGMeIhypmvg"
          target="_vp5aGqRwEeahGMeIhypmvg">
209        <trigger xmi:type="uml:Trigger" xmi:id="_vp5acKRwEeahGMeIhypmvg"
            name="ev7" event="_vp5arqRwEeahGMeIhypmvg"/>
210       </transition>
211       <transition xmi:type="uml:Transition" xmi:id="_vp5acaRwEeahGMeIhypmvg"
          name="Transition1" source="_vp5aGKRwEeahGMeIhypmvg"
```

```
          target="_vp5aFqRwEeahGMeIhypmvg">
212        <trigger xmi:type="uml:Trigger" xmi:id="_vp5acqRwEeahGMeIhypmvg"
           name="ev4" event="_vp5aq6RwEeahGMeIhypmvg"/>
213       </transition>
214       <transition xmi:type="uml:Transition" xmi:id="_vp5ac6RwEeahGMeIhypmvg"
          name="Transition2" source="_vp5aGKRwEeahGMeIhypmvg"
          target="_vp5aEKRwEeahGMeIhypmvg" guard="_vp5adKRwEeahGMeIhypmvg">
215        <ownedRule xmi:type="uml:Constraint" xmi:id="_vp5adKRwEeahGMeIhypmvg"
           name="cond4" constrainedElement="_vp5ac6RwEeahGMeIhypmvg">
216         <specification xmi:type="uml:LiteralString"
            xmi:id="_vp5adaRwEeahGMeIhypmvg" value="cond4"/>
217        </ownedRule>
218        <trigger xmi:type="uml:Trigger" xmi:id="_vp5adqRwEeahGMeIhypmvg"
           name="ev11" event="_vp5asqRwEeahGMeIhypmvg"/>
219       </transition>
220       <transition xmi:type="uml:Transition" xmi:id="_vp5ad6RwEeahGMeIhypmvg"
          name="Transition" source="_vp5aGqRwEeahGMeIhypmvg"
          target="_vp5aHKRwEeahGMeIhypmvg">
221        <trigger xmi:type="uml:Trigger" xmi:id="_vp5aeKRwEeahGMeIhypmvg"
           name="ev8" event="_vp5ar6RwEeahGMeIhypmvg"/>
222       </transition>
223       <transition xmi:type="uml:Transition" xmi:id="_vp5aeaRwEeahGMeIhypmvg"
          name="Transition1" source="_vp5aGqRwEeahGMeIhypmvg"
          target="_vp5aGKRwEeahGMeIhypmvg">
224        <trigger xmi:type="uml:Trigger" xmi:id="_vp5aeqRwEeahGMeIhypmvg"
           name="ev5" event="_vp5arKRwEeahGMeIhypmvg"/>
225       </transition>
226       <transition xmi:type="uml:Transition" xmi:id="_vp5ae6RwEeahGMeIhypmvg"
          name="Transition2" source="_vp5aGqRwEeahGMeIhypmvg"
          target="_vp5aEqRwEeahGMeIhypmvg" guard="_vp5afKRwEeahGMeIhypmvg">
227        <ownedRule xmi:type="uml:Constraint" xmi:id="_vp5afKRwEeahGMeIhypmvg"
           name="cond6" constrainedElement="_vp5ae6RwEeahGMeIhypmvg">
228         <specification xmi:type="uml:LiteralString"
            xmi:id="_vp5afaRwEeahGMeIhypmvg" value="cond6"/>
229        </ownedRule>
230        <trigger xmi:type="uml:Trigger" xmi:id="_vp5afqRwEeahGMeIhypmvg"
           name="ev12" event="_vp5as6RwEeahGMeIhypmvg"/>
231       </transition>
232       <transition xmi:type="uml:Transition" xmi:id="_vp5af6RwEeahGMeIhypmvg"
          name="Transition" source="_vp5aHKRwEeahGMeIhypmvg"
          target="_vp5aKKRwEeahGMeIhypmvg">
233        <trigger xmi:type="uml:Trigger" xmi:id="_vp5agKRwEeahGMeIhypmvg"
           name="ev9" event="_vp5asKRwEeahGMeIhypmvg"/>
234       </transition>
235       <transition xmi:type="uml:Transition" xmi:id="_vp5agaRwEeahGMeIhypmvg"
          name="Transition1" source="_vp5aHKRwEeahGMeIhypmvg"
          target="_vp5aGqRwEeahGMeIhypmvg">
236        <trigger xmi:type="uml:Trigger" xmi:id="_vp5agqRwEeahGMeIhypmvg"
           name="ev6" event="_vp5araRwEeahGMeIhypmvg"/>
237       </transition>
238       <transition xmi:type="uml:Transition" xmi:id="_vp5ag6RwEeahGMeIhypmvg"
          name="Transition2" source="_vp5aHKRwEeahGMeIhypmvg"
```

```
          target="_vp5aFKRwEeahGMeIhypmvg" guard="_vp5ahKRwEeahGMeIhypmvg">
239        <ownedRule xmi:type="uml:Constraint" xmi:id="_vp5ahKRwEeahGMeIhypmvg"
            name="cond8" constrainedElement="_vp5ag6RwEeahGMeIhypmvg">
240          <specification xmi:type="uml:LiteralString"
              xmi:id="_vp5ahaRwEeahGMeIhypmvg" value="cond8"/>
241        </ownedRule>
242        <trigger xmi:type="uml:Trigger" xmi:id="_vp5ahqRwEeahGMeIhypmvg"
            name="ev13" event="_vp5atKRwEeahGMeIhypmvg"/>
243      </transition>
244      <transition xmi:type="uml:Transition" xmi:id="_vp5ah6RwEeahGMeIhypmvg"
          name="Transition" source="_vp5aHqRwEeahGMeIhypmvg"
          target="_vp5aFKRwEeahGMeIhypmvg">
245        <trigger xmi:type="uml:Trigger" xmi:id="_vp5aiKRwEeahGMeIhypmvg"
            name="ev15" event="_vp5atqRwEeahGMeIhypmvg"/>
246      </transition>
247      <transition xmi:type="uml:Transition" xmi:id="_vp5aiaRwEeahGMeIhypmvg"
          name="Transition1" source="_vp5aHqRwEeahGMeIhypmvg"
          target="_vp5aIKRwEeahGMeIhypmvg">
248        <trigger xmi:type="uml:Trigger" xmi:id="_vp5aiqRwEeahGMeIhypmvg"
            name="ev12" event="_vp5as6RwEeahGMeIhypmvg"/>
249      </transition>
250      <transition xmi:type="uml:Transition" xmi:id="_vp5ai6RwEeahGMeIhypmvg"
          name="Transition2" source="_vp5aHqRwEeahGMeIhypmvg"
          target="_vp5aKKRwEeahGMeIhypmvg" guard="_vp5ajKRwEeahGMeIhypmvg">
251        <ownedRule xmi:type="uml:Constraint" xmi:id="_vp5ajKRwEeahGMeIhypmvg"
            name="cond9" constrainedElement="_vp5ai6RwEeahGMeIhypmvg">
252          <specification xmi:type="uml:LiteralString"
              xmi:id="_vp5ajaRwEeahGMeIhypmvg" value="cond9"/>
253        </ownedRule>
254        <trigger xmi:type="uml:Trigger" xmi:id="_vp5ajqRwEeahGMeIhypmvg"
            name="ev9" event="_vp5asKRwEeahGMeIhypmvg"/>
255      </transition>
256      <transition xmi:type="uml:Transition" xmi:id="_vp5aj6RwEeahGMeIhypmvg"
          name="Transition" source="_vp5aIKRwEeahGMeIhypmvg"
          target="_vp5aHqRwEeahGMeIhypmvg">
257        <trigger xmi:type="uml:Trigger" xmi:id="_vp5akKRwEeahGMeIhypmvg"
            name="ev14" event="_vp5ataRwEeahGMeIhypmvg"/>
258      </transition>
259      <transition xmi:type="uml:Transition" xmi:id="_vp5akaRwEeahGMeIhypmvg"
          name="Transition1" source="_vp5aIKRwEeahGMeIhypmvg"
          target="_vp5aIqRwEeahGMeIhypmvg">
260        <trigger xmi:type="uml:Trigger" xmi:id="_vp5akqRwEeahGMeIhypmvg"
            name="ev11" event="_vp5asqRwEeahGMeIhypmvg"/>
261      </transition>
262      <transition xmi:type="uml:Transition" xmi:id="_vp5ak6RwEeahGMeIhypmvg"
          name="Transition" source="_vp5aIqRwEeahGMeIhypmvg"
          target="_vp5aIKRwEeahGMeIhypmvg">
263        <trigger xmi:type="uml:Trigger" xmi:id="_vp5alKRwEeahGMeIhypmvg"
            name="ev13" event="_vp5atKRwEeahGMeIhypmvg"/>
264      </transition>
265      <transition xmi:type="uml:Transition" xmi:id="_vp5alaRwEeahGMeIhypmvg"
          name="Transition1" source="_vp5aIqRwEeahGMeIhypmvg"
```

```
          target="_vp5aJKRwEeahGMeIhypmvg">
266        <trigger xmi:type="uml:Trigger" xmi:id="_vp5alqRwEeahGMeIhypmvg"
            name="ev10" event="_vp5asaRwEeahGMeIhypmvg"/>
267       </transition>
268       <transition xmi:type="uml:Transition" xmi:id="_vp5al6RwEeahGMeIhypmvg"
          name="Transition" source="_vp5aJKRwEeahGMeIhypmvg"
          target="_vp5aIqRwEeahGMeIhypmvg">
269        <trigger xmi:type="uml:Trigger" xmi:id="_vp5amKRwEeahGMeIhypmvg"
            name="ev12" event="_vp5as6RwEeahGMeIhypmvg"/>
270       </transition>
271       <transition xmi:type="uml:Transition" xmi:id="_vp5amaRwEeahGMeIhypmvg"
          name="Transition1" source="_vp5aJKRwEeahGMeIhypmvg"
          target="_vp5aJqRwEeahGMeIhypmvg">
272        <trigger xmi:type="uml:Trigger" xmi:id="_vp5amqRwEeahGMeIhypmvg"
            name="ev9" event="_vp5asKRwEeahGMeIhypmvg"/>
273       </transition>
274       <transition xmi:type="uml:Transition" xmi:id="_vp5am6RwEeahGMeIhypmvg"
          name="Transition" source="_vp5aJqRwEeahGMeIhypmvg"
          target="_vp5aJKRwEeahGMeIhypmvg">
275        <trigger xmi:type="uml:Trigger" xmi:id="_vp5anKRwEeahGMeIhypmvg"
            name="ev11" event="_vp5asqRwEeahGMeIhypmvg"/>
276       </transition>
277       <transition xmi:type="uml:Transition" xmi:id="_vp5anaRwEeahGMeIhypmvg"
          name="Transition1" source="_vp5aJqRwEeahGMeIhypmvg"
          target="_vp5aKKRwEeahGMeIhypmvg">
278        <trigger xmi:type="uml:Trigger" xmi:id="_vp5anqRwEeahGMeIhypmvg"
            name="ev8" event="_vp5ar6RwEeahGMeIhypmvg"/>
279       </transition>
280       <transition xmi:type="uml:Transition" xmi:id="_vp5an6RwEeahGMeIhypmvg"
          name="Transition" source="_vp5aKKRwEeahGMeIhypmvg"
          target="_vp5aJqRwEeahGMeIhypmvg">
281        <trigger xmi:type="uml:Trigger" xmi:id="_vp5aoKRwEeahGMeIhypmvg"
            name="ev10" event="_vp5asaRwEeahGMeIhypmvg"/>
282       </transition>
283       <transition xmi:type="uml:Transition" xmi:id="_vp5aoaRwEeahGMeIhypmvg"
          name="Transition1" source="_vp5aKKRwEeahGMeIhypmvg"
          target="_vp5aHKRwEeahGMeIhypmvg">
284        <trigger xmi:type="uml:Trigger" xmi:id="_vp5aoqRwEeahGMeIhypmvg"
            name="ev7" event="_vp5arqRwEeahGMeIhypmvg"/>
285       </transition>
286       <transition xmi:type="uml:Transition" xmi:id="_vp5ao6RwEeahGMeIhypmvg"
          name="Transition2" source="_vp5aKKRwEeahGMeIhypmvg"
          target="_vp5aHqRwEeahGMeIhypmvg" guard="_vp5apKRwEeahGMeIhypmvg">
287        <ownedRule xmi:type="uml:Constraint" xmi:id="_vp5apKRwEeahGMeIhypmvg"
            name="cond10" constrainedElement="_vp5ao6RwEeahGMeIhypmvg">
288          <specification xmi:type="uml:LiteralString"
             xmi:id="_vp5apaRwEeahGMeIhypmvg" value="cond10"/>
289        </ownedRule>
290        <trigger xmi:type="uml:Trigger" xmi:id="_vp5apqRwEeahGMeIhypmvg"
            name="ev14" event="_vp5ataRwEeahGMeIhypmvg"/>
291       </transition>
292       <transition xmi:type="uml:Transition" xmi:id="_vp5ap6RwEeahGMeIhypmvg"
```

```
          name="Transition" source="_vp5aKqRwEeahGMeIhypmvg"
          target="_vp5aBqRwEeahGMeIhypmvg"/>
293     </region>
294   </packagedElement>
295   <packagedElement xmi:type="uml:SignalEvent" xmi:id="_vp5aqKRwEeahGMeIhypmvg"
       name="ev1"/>
296   <packagedElement xmi:type="uml:SignalEvent" xmi:id="_vp5aqaRwEeahGMeIhypmvg"
       name="ev2"/>
297   <packagedElement xmi:type="uml:SignalEvent" xmi:id="_vp5aqqRwEeahGMeIhypmvg"
       name="ev3"/>
298   <packagedElement xmi:type="uml:SignalEvent" xmi:id="_vp5aq6RwEeahGMeIhypmvg"
       name="ev4"/>
299   <packagedElement xmi:type="uml:SignalEvent" xmi:id="_vp5arKRwEeahGMeIhypmvg"
       name="ev5"/>
300   <packagedElement xmi:type="uml:SignalEvent" xmi:id="_vp5araRwEeahGMeIhypmvg"
       name="ev6"/>
301   <packagedElement xmi:type="uml:SignalEvent" xmi:id="_vp5arqRwEeahGMeIhypmvg"
       name="ev7"/>
302   <packagedElement xmi:type="uml:SignalEvent" xmi:id="_vp5ar6RwEeahGMeIhypmvg"
       name="ev8"/>
303   <packagedElement xmi:type="uml:SignalEvent" xmi:id="_vp5asKRwEeahGMeIhypmvg"
       name="ev9"/>
304   <packagedElement xmi:type="uml:SignalEvent" xmi:id="_vp5asaRwEeahGMeIhypmvg"
       name="ev10"/>
305   <packagedElement xmi:type="uml:SignalEvent" xmi:id="_vp5asqRwEeahGMeIhypmvg"
       name="ev11"/>
306   <packagedElement xmi:type="uml:SignalEvent" xmi:id="_vp5as6RwEeahGMeIhypmvg"
       name="ev12"/>
307   <packagedElement xmi:type="uml:SignalEvent" xmi:id="_vp5atKRwEeahGMeIhypmvg"
       name="ev13"/>
308   <packagedElement xmi:type="uml:SignalEvent" xmi:id="_vp5ataRwEeahGMeIhypmvg"
       name="ev14"/>
309   <packagedElement xmi:type="uml:SignalEvent" xmi:id="_vp5atqRwEeahGMeIhypmvg"
       name="ev15"/>
310   <packagedElement xmi:type="uml:SignalEvent" xmi:id="_vp5at6RwEeahGMeIhypmvg"
       name="ev16"/>
311   <packagedElement xmi:type="uml:SignalEvent" xmi:id="_vp5auKRwEeahGMeIhypmvg"
       name="ev17"/>
312   <packagedElement xmi:type="uml:SignalEvent" xmi:id="_vp5auaRwEeahGMeIhypmvg"
       name="ev18"/>
313 </uml:Model>
314
```

Anexo XI

Código fonte gerado para o diagrama de teste de stress da ferramenta

```cpp
#include <iostream>
#include <cstdlib>
#include <ctime>

#include "../lib/FSM.h"
#include "../lib/State.h"
#include "../lib/Transition.h"
#include "../lib/FSMEvent.h"

using namespace UMinho::FSM;
using namespace std;

struct SharedData : public Object {
};

class ActEntryA : public FSMAction {
    void execute(const FSMEvent &ev, FSMContext &c) {
        ///TODO: INSERT CODE HERE!!
    }
};

class ActExistB : public FSMAction {
    void execute(const FSMEvent &ev, FSMContext &c) {
        ///TODO: INSERT CODE HERE!!
    }
};

class ActEntryC : public FSMAction {
    void execute(const FSMEvent &ev, FSMContext &c) {
        ///TODO: INSERT CODE HERE!!
    }
};

class ActExitD : public FSMAction {
    void execute(const FSMEvent &ev, FSMContext &c) {
        ///TODO: INSERT CODE HERE!!
    }
};

class ActExitR : public FSMAction {
    void execute(const FSMEvent &ev, FSMContext &c) {
        ///TODO: INSERT CODE HERE!!
    }
};

class ActEntryQ : public FSMAction {
    void execute(const FSMEvent &ev, FSMContext &c) {
        ///TODO: INSERT CODE HERE!!
    }
};

class ActExitP : public FSMAction {
```

```
53      void execute(const FSMEvent &ev, FSMContext &c) {
54          ///TODO: INSERT CODE HERE!!
55      }
56  };
57
58  class ActEntryO : public FSMAction {
59      void execute(const FSMEvent &ev, FSMContext &c) {
60          ///TODO: INSERT CODE HERE!!
61      }
62  };
63
64  class ActEntryE : public FSMAction {
65      void execute(const FSMEvent &ev, FSMContext &c) {
66          ///TODO: INSERT CODE HERE!!
67      }
68  };
69
70  class ActExitF : public FSMAction {
71      void execute(const FSMEvent &ev, FSMContext &c) {
72          ///TODO: INSERT CODE HERE!!
73      }
74  };
75
76  class ActEntryG : public FSMAction {
77      void execute(const FSMEvent &ev, FSMContext &c) {
78          ///TODO: INSERT CODE HERE!!
79      }
80  };
81
82  class ActExitH : public FSMAction {
83      void execute(const FSMEvent &ev, FSMContext &c) {
84          ///TODO: INSERT CODE HERE!!
85      }
86  };
87
88  class ActExitN : public FSMAction {
89      void execute(const FSMEvent &ev, FSMContext &c) {
90          ///TODO: INSERT CODE HERE!!
91      }
92  };
93
94  class ActEntryM : public FSMAction {
95      void execute(const FSMEvent &ev, FSMContext &c) {
96          ///TODO: INSERT CODE HERE!!
97      }
98  };
99
100 class ActExitL : public FSMAction {
101     void execute(const FSMEvent &ev, FSMContext &c) {
102         ///TODO: INSERT CODE HERE!!
103     }
104 };
```

```
105
106 class ActEntryK : public FSMAction {
107     void execute(const FSMEvent &ev, FSMContext &c) {
108         ///TODO: INSERT CODE HERE!!
109     }
110 };
111
112 class ActExitJ : public FSMAction {
113     void execute(const FSMEvent &ev, FSMContext &c) {
114         ///TODO: INSERT CODE HERE!!
115     }
116 };
117
118 class ActEntryI : public FSMAction {
119     void execute(const FSMEvent &ev, FSMContext &c) {
120         ///TODO: INSERT CODE HERE!!
121     }
122 };
123
124 class ActTrans1 : public FSMAction {
125     void execute(const FSMEvent &ev, FSMContext &c) {
126         ///TODO: INSERT CODE HERE!!
127     }
128 };
129
130 class ActTrans2 : public FSMAction {
131     void execute(const FSMEvent &ev, FSMContext &c) {
132         ///TODO: INSERT CODE HERE!!
133     }
134 };
135
136 class ActTrans3 : public FSMAction {
137     void execute(const FSMEvent &ev, FSMContext &c) {
138         ///TODO: INSERT CODE HERE!!
139     }
140 };
141
142 class ActTrans4 : public FSMAction {
143     void execute(const FSMEvent &ev, FSMContext &c) {
144         ///TODO: INSERT CODE HERE!!
145     }
146 };
147
148 class Cond1 : public FSMCondition {
149     bool isSatisfied(const FSMEvent &ev, FSMContext &c) {
150         ///TODO: INSERT CODE HERE!!
151         return true;
152     }
153 };
154
155 class Cond3 : public FSMCondition {
156     bool isSatisfied(const FSMEvent &ev, FSMContext &c) {
```

```
157         ///TODO: INSERT CODE HERE!!
158         return true;
159     }
160 };
161 
162 class Cond5 : public FSMCondition {
163     bool isSatisfied(const FSMEvent &ev, FSMContext &c) {
164         ///TODO: INSERT CODE HERE!!
165         return true;
166     }
167 };
168 
169 class Cond7 : public FSMCondition {
170     bool isSatisfied(const FSMEvent &ev, FSMContext &c) {
171         ///TODO: INSERT CODE HERE!!
172         return true;
173     }
174 };
175 
176 class Cond2 : public FSMCondition {
177     bool isSatisfied(const FSMEvent &ev, FSMContext &c) {
178         ///TODO: INSERT CODE HERE!!
179         return true;
180     }
181 };
182 
183 class Cond4 : public FSMCondition {
184     bool isSatisfied(const FSMEvent &ev, FSMContext &c) {
185         ///TODO: INSERT CODE HERE!!
186         return true;
187     }
188 };
189 
190 class Cond6 : public FSMCondition {
191     bool isSatisfied(const FSMEvent &ev, FSMContext &c) {
192         ///TODO: INSERT CODE HERE!!
193         return true;
194     }
195 };
196 
197 class Cond8 : public FSMCondition {
198     bool isSatisfied(const FSMEvent &ev, FSMContext &c) {
199         ///TODO: INSERT CODE HERE!!
200         return true;
201     }
202 };
203 
204 class Cond9 : public FSMCondition {
205     bool isSatisfied(const FSMEvent &ev, FSMContext &c) {
206         ///TODO: INSERT CODE HERE!!
207         return true;
208     }
```

```
209 };
210
211 class Cond10 : public FSMCondition {
212     bool isSatisfied(const FSMEvent &ev, FSMContext &c) {
213         ///TODO: INSERT CODE HERE!!
214         return true;
215     }
216 };
217
218 enum StateID { A = 1, B, C, D, R, Q, P, O, E, F, G, H, N, M, L, K, J, I };
219
220 enum EventID { EV2 = 1, EV17, EV1, EV3, EV18, EV4, EV5, EV16, EV15, EV6, EV14,
      EV7, EV13, EV8, EV10, EV11, EV12, EV9 };
221
222 string stateName(StateID s) {
223     static string names[] = { "A", "B", "C", "D", "R", "Q", "P", "O", "E", "F",
        "G", "H", "N", "M", "L", "K", "J", "I" };
224     return names[(int) s-1];
225 }
226
227 string eventName(EventID e) {
228     static string names[] = { "EV2", "EV17", "EV1", "EV3", "EV18", "EV4", "EV5",
        "EV16", "EV15", "EV6", "EV14", "EV7", "EV13", "EV8", "EV10", "EV11",
        "EV12", "EV9" };
229     return names[(int) e-1];
230 }
231
232 int main() {
233     ActEntryA actEntryA;
234     ActExistB actExistB;
235     ActEntryC actEntryC;
236     ActExitD actExitD;
237     ActExitR actExitR;
238     ActEntryQ actEntryQ;
239     ActExitP actExitP;
240     ActEntryO actEntryO;
241     ActEntryE actEntryE;
242     ActExitF actExitF;
243     ActEntryG actEntryG;
244     ActExitH actExitH;
245     ActExitN actExitN;
246     ActEntryM actEntryM;
247     ActExitL actExitL;
248     ActEntryK actEntryK;
249     ActExitJ actExitJ;
250     ActEntryI actEntryI;
251     ActTrans1 actTrans1;
252     ActTrans2 actTrans2;
253     ActTrans3 actTrans3;
254     ActTrans4 actTrans4;
255
256     Cond1 cond1;
```

```
257     Cond3 cond3;
258     Cond5 cond5;
259     Cond7 cond7;
260     Cond2 cond2;
261     Cond4 cond4;
262     Cond6 cond6;
263     Cond8 cond8;
264     Cond9 cond9;
265     Cond10 cond10;
266
267     State a(A, actEntryA);
268     State b(B, FSMSkipAction::instance, actExistB);
269     State c(C, actEntryC);
270     State d(D, FSMSkipAction::instance, actExitD);
271     State r(R, FSMSkipAction::instance, actExitR);
272     State q(Q, actEntryQ);
273     State p(P, FSMSkipAction::instance, actExitP);
274     State o(O, actEntryO);
275     State e(E, actEntryE);
276     State f(F, FSMSkipAction::instance, actExitF);
277     State g(G, actEntryG);
278     State h(H, FSMSkipAction::instance, actExitH);
279     State n(N, FSMSkipAction::instance, actExitN);
280     State m(M, actEntryM);
281     State l(L, FSMSkipAction::instance, actExitL);
282     State k(K, actEntryK);
283     State j(J, FSMSkipAction::instance, actExitJ);
284     State i(I, actEntryI);
285
286     a.addTransition(EV2, b);
287     a.addTransition(EV17, r);
288     a.addTransition(EV1, m);
289     b.addTransition(EV3, c);
290     b.addTransition(EV18, a);
291     b.addTransition(EV2, l);
292     c.addTransition(EV4, d);
293     c.addTransition(EV1, b);
294     c.addTransition(EV3, k);
295     d.addTransition(EV5, e);
296     d.addTransition(EV2, c);
297     d.addTransition(EV4, j);
298     r.addTransition(EV1, a);
299     r.addTransition(EV16, q);
300     r.addTransition(EV5, e, FSMSkipAction::instance, cond1);
301     q.addTransition(EV18, r);
302     q.addTransition(EV15, p);
303     q.addTransition(EV6, f, FSMSkipAction::instance, cond3);
304     p.addTransition(EV17, q);
305     p.addTransition(EV14, o);
306     p.addTransition(EV7, g, FSMSkipAction::instance, cond5);
307     o.addTransition(EV16, p);
308     o.addTransition(EV13, n);
```

```
309     o.addTransition(EV8, h, FSMSkipAction::instance, cond7);
310     e.addTransition(EV6, f);
311     e.addTransition(EV3, d);
312     e.addTransition(EV10, r, FSMSkipAction::instance, cond2);
313     f.addTransition(EV7, g);
314     f.addTransition(EV4, e);
315     f.addTransition(EV11, q, FSMSkipAction::instance, cond4);
316     g.addTransition(EV8, h);
317     g.addTransition(EV5, f);
318     g.addTransition(EV12, p, FSMSkipAction::instance, cond6);
319     h.addTransition(EV9, i);
320     h.addTransition(EV6, g);
321     h.addTransition(EV13, o, FSMSkipAction::instance, cond8);
322     n.addTransition(EV15, o);
323     n.addTransition(EV12, m);
324     n.addTransition(EV9, i, FSMSkipAction::instance, cond9);
325     m.addTransition(EV14, n);
326     m.addTransition(EV11, l);
327     l.addTransition(EV13, m);
328     l.addTransition(EV10, k);
329     k.addTransition(EV12, l);
330     k.addTransition(EV9, j);
331     j.addTransition(EV11, k);
332     j.addTransition(EV8, i);
333     i.addTransition(EV10, j);
334     i.addTransition(EV7, h);
335     i.addTransition(EV14, n, FSMSkipAction::instance, cond10);
336
337     SharedData sd;
338
339     FSMContext c(a, FSMSkipAction::instance, &sd);
340
341     //ADD FSM FEED CODE HERE
342
343     return 0;
344 }
345
346
```