Rogério Alexandre Botelho Campos Rebelo

Mestre em Engenharia Electrotécnica e de Computadores

Modelização de Eventos: aplicação a modelos de interação do sistema com o ambiente

Dissertação para obtenção do Grau de Doutor em Engenharia Electrotécnica e de Computadores

Orientador: Luís Filipe dos Santos Gomes, Professor Associado, Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa

Co-orientador: Anikó Katalin Horváth da Costa, Professora Auxiliar, Faculdade de Ciências e Tecnologia, Universidade Nova de Lisboa

Júri:

Presidente: Doutor Luís Manuel Camarinha de Matos

Arguentes: Doutor Alberto Jorge Lebre Cardoso

Doutor Paulo Jorge Pinto Leitão

Vogais: Doutor Adriano da Silva Carvalho

Doutor Manuel Martins Barata

Doutor Luís Filipe dos Santos Gomes

Doutor Luís Filipe Figueira de Brito Palma

FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Março, 2016

Modelização de Eventos: aplicação a modelos de interação do sistema com o ambiente

Para a minha família e amigos

Agradecimentos

Quero agradecer ao meu orientador, Prof. Luís Filipe dos Santos Gomes, e co-orientadora Profª Anikó Katalin Horváth da Costa pelo seu apoio, orientação, disponibilização do seu conhecimento e também pela amizade.

Gostava de agradecer também ao Prof. João Paulo Barros e Prof. Luís Brito Palma que, em conjunto com o meu orientador e co-orientadora compõem a comissão de acompanhamento de tese (CAT), pelos valiosos conselhos e recomendações.

Um agradecimento também à Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa (FCT-UNL) e à UNINOVA – Instituto de Desenvolvimento de Novas Tecnologias.

Um grande obrigado também para todos os meus companheiros de trabalho na FCT-UNL e UNINOVA, em especial para o Filipe Moutinho, Fernando Pereira, José Ribeiro, José Rocha, José Pedro Lucas, Rui Pais e Duarte Guerreiro.

Finalmente agradeço a toda a minha família e amigos pela sua amizade, em especial à Rita pelo apoio e paciência que teve comigo durante estes anos.

Resumo

Neste trabalho propõe-se um conjunto de técnicas de modelização de eventos potencialmente utilizáveis em complemento a formalismos de modelação (Diagramas de Estado e redes de Petri, entre outros) de sistemas a eventos discretos. São definidos eventos de entrada e eventos de saída do sistema. Os eventos de entrada são originados pela análise da evolução de sinais e (outros) eventos de entrada e permitem representar comportamentos complexos das entradas do sistema, enquanto os eventos de saída permitem gerar sinais e (outros) eventos de saída.

São propostos eventos e condições elementares que detetam comportamentos elementares dos sinais e também composições de eventos e condições, incluindo relações de ordem temporal, de forma a obter caracterizações mais abstratas e melhorias na legibilidade e compactação do modelo.

Para representar as técnicas de modelização propostas são definidas duas sintaxes diferentes (gráfica ou textual). A definição destes dois tipos de sintaxe permite a sua integração em diferentes ambientes de desenvolvimento de sistemas.

Desta forma, a principal contribuição deste trabalho foca-se na modelização da interação do sistema com o ambiente, partindo da decomposição do modelo do sistema em partes de interface e de execução (que comunicam através de eventos), e da adição de uma caracterização separada das dependências e dos comportamentos dos sinais de entrada e de saída com o ambiente resultando numa modelação estruturada do sistema, dando, assim, origem a modelos mais compactos e mais simples de interpretar e implementar.

Palavras-chave: Sistemas a eventos discretos, interpretação de sinais, afetação de sinais, formalismos de modelação, interação humano-sistema.

Abstract

This work proposes a set of technics for event modeling with potential application complementing modelling formalisms (including State diagrams and Petri nets) for discrete events systems. System input events and system output events are defined. The Input events are generated by the analysis of the evolution of input signals and (other) events and allows the representation of complex behaviors of the system inputs, while the output events generate output signals and (other) events.

Elementary events and conditions are proposed, which detect elementary behaviors in signal evolution. Composition of events and conditions are also proposed, including time relations, leading to more abstract characterizations and improving the legibility and compactness of the model.

Two different syntaxes are presented in order to represent the proposed modeling techniques (graphical or textual). This definition of two syntax types allows their integration in different system development environments.

Thus, the main contribution of this work focuses on modeling system's interaction with the environment, based on the decomposition of the model of the

system into interface and execution parts (communicating through events), and the addition of a separate characterization of dependencies and behaviors of input and output signals to the environment resulting in a structured modeling of the system, thus giving rise to more compact models that are simpler to interpret and implement.

Keywords: Discrete events systems, signal interpretation, signal assignment, modeling formalisms, human-system interaction.

Conteúdo

Lista de Figuras

Lista de Tabelas

Lista de Acrónimos

CEP - *Complex Event Processing*

DSL - *Domain-Specific Language*

EDA - *Event-Driven Architectures*

FPGA - *Field Programmable Gate Array*

IOPT - *Input-Output Place-Transition*

MAS - *Modelo de Afetação de Sinais*

MIS - *Modelo de Interpretação de Sinais*

MVA - *Model–View–Adapter*

MVC - *Model – View - Controller*

MVP - *Model-View-Presenter*

MVVM - *model View ViewModel*

PAC - *Presentation–Abstraction–Control*

RdP - *Redes de Petri*

UML - *Unified Modeling Language*

1

Introdução

Neste primeiro capítulo é apresentado o contexto deste trabalho, identificado um problema e apresentadas as motivações para a sua resolução. São apresentadas as questões de investigação e respetivas teses. Finalmente é descrita a estrutura do documento, onde são apresentadas as publicações associadas a cada uma das contribuições.

1.1. Contexto

Por interação humano-sistema [1], [2], entende-se a forma como o utilizador comunica com um determinado sistema, introduzindo nele informação que o sistema trata, devolvendo ao utilizador informação com os resultados da sua execução.

Esta interação pode ser suportada por variadíssimos tipos de dispositivos, dependendo do contexto da interação, que incluem dispositivos simples, como contactos e interruptores, vulgares em sistemas de automação e controlo, bem como dispositivos bem mais complexos, como interfaces gráficas ou dispositivos de aquisição de sinais biométricos.

Com a evolução dos sistemas computacionais, também os dispositivos que garantem a interface, têm evoluído, passando de simples subsistemas físicos, como ligações, contactos ou detetores, para subsistemas complexos com capacidade de processamento, como os utilizados no controlo de acessos com base em caraterísticas biométricas, ou em computação vestível (*wearable computing*).

Desde o aparecimento dos primeiros computadores eletrónicos [3] [4], nas décadas de 1940 e 1950, a ciência da computação e a tecnologia de computadores evoluíram de uma forma bastante rápida, passando de máquinas grandes que permitiam a automatização de operações não muito complexas, para os atuais computadores e dispositivos eletrónicos de utilização diária, de tamanho reduzido e que permitem a execução de tarefas bastante complexas.

Como previsto, em 1965, pelo então presidente da Intel, Gordon E. Moore, nas suas declarações que acabaram por ficar conhecidas como lei de Moore [5], tem-se verificado uma evolução sustentada e continua nas capacidades do *hardware*, em termos da densidade de transístores por unidade de área dos circuitos integrados, com impacto direto em termos de capacidades de armazenamento, bem como de processamento. Esse desenvolvimento tem permitido que os sistemas computacionais se tornem mais eficientes, mas também bastante mais complexos.

Este aumento na complexidade dos sistemas levou também a um aumento da complexidade no seu desenvolvimento. Tendo isso em conta, têm vindo a ser propostas metodologias de desenvolvimento específicas, suportadas por formalismos de modelação, por linguagens de programação, complementadas, nalguns casos, por linguagens de descrição de *hardware*, para facilitar a sua implementação. Hoje em dia existe uma grande variedade de linguagens de programação disponíveis, desde as linguagens de baixo nível (como por exemplo o assembly) até às linguagens de alto nível (Java, C#, etc…).

Por outro lado, ao aumentar a complexidade do desenvolvimento dos sistemas, a possibilidade de aparecimento de erros aumenta (principalmente os erros humanos, cometidos pelo projetista). Tendo isso em conta têm sido procuradas abordagens de desenvolvimento que permitam antecipar a deteção dos erros, reduzindo os custos associados à sua ocorrência e à sua correção.

Uma das abordagens comuns consiste na estruturação deste desenvolvimento, permitindo fazê-lo por etapas. Garantindo que certos requisitos são cumpridos ao passar de etapa para etapa, é possível gerir a complexidade do desenvolvimento, permitindo a resolução de problemas específicos em cada etapa do desenvolvimento. Desta forma, é possível em cada uma dessas etapas adotar níveis de abstração diferenciados e ter em atenção apenas certos tipos de erros, garantindo que os modelos de desenvolvimento na etapa seguinte estão livres de erros dos tipos analisados na etapa anterior, reduzindo as variáveis de deteção desses mesmos erros.

Tendo em conta esta ideia, surgiram os desenvolvimentos baseados em modelos ou orientados por modelos ("*model driven engineering*" MDE) dos quais o "*Model-Driven Architecture*" [6], definido pelo *Object Management Group* (OMG) é um dos mais conhecidos.

A adoção de um desenvolvimento baseado em modelos permite fazer uma divisão das várias etapas da elaboração de um sistema, tal como foi dito anteriormente. Algumas das etapas que se podem definir na elaboração de um sistema são: modelação, validação, implementação e teste [7].

A abordagem MDE permite descrever as funcionalidades do sistema utilizando um modelo independente da plataforma onde o sistema será implementado. Ou seja, um modelo que representa o sistema, mas que não está elaborado para uma plataforma específica (como um computador, um microcontrolador específico ou um dispositivo reconfigurável, como uma FPGA – *Field Program-*

mable Gate Array). Este modelo está num nível mais abstrato. Para esse efeito é utilizada uma linguagem de domínio específico apropriada (*domain-specific language"* (DSL)).

Este modelo permite validar as funcionalidades do sistema, verificando se a abordagem de modelação garante as especificações requeridas do sistema ou se pelo contrário é necessário proceder a alterações ou refinamentos do modelo para as garantir.

Nesta fase é possível fazer transformações do modelo sem entrar nas especificidades da plataforma, permitindo uma adequação ao nível de abstração pretendido, podendo haver transformações para outros formalismos de modelação.

Quando o modelo estiver adequadamente validado será traduzido para um ou mais modelos específicos para a(s) plataforma(s) onde irá(irão) ser implementado(s), tendo em conta as suas características específicas.

Uma vantagem da utilização desta abordagem é a possibilidade de utilizar ferramentas de automatização de projeto (*design automation tools*) que realizem esta tradução automaticamente, partindo dos modelos independentes da plataforma e permitindo obter o código para implementação.

Desta forma, a seleção de qual o formalismo/linguagem a utilizar para produzir o modelo independente da plataforma é uma das primeiras decisões a tomar quando se adota uma abordagem orientada por modelos.

Para esse efeito existem muitos formalismos, linguagens e notações, das quais se destaca o UML (*Unified Modeling Language*) [8] [9], que é uma linguagem de modelação que engloba um conjunto de notações gráficas baseadas num único meta modelo. Possui na sua composição um conjunto de diagramas que permitem a modelação do sistema considerando diferentes aspetos (rele-

vantes em diferentes fases do processo de desenvolvimento), entre os quais podem ser evidenciados o diagrama de casos de uso, o diagrama de classes ou o diagrama de sequência. De referir também as linguagens de representação gráfica do comportamento dos sistemas, como os fluxogramas [10] [11], as máquinas de estados [12] [13], estadogramas [14] [15], as redes de Petri (RdP) [16] [17], os Grafcet [18] [19] [20], entre outras. Estas linguagens permitem ao modelador uma análise visual do modelo, facilitando a análise das propriedades associadas através de métodos formais.

Em particular, as redes de Petri possuem uma semântica e sintaxe exatas, representação gráfica bem definida, que lhes permitem modelar explicitamente problemas complexos como o paralelismo, a sincronização ou a gestão de recursos. Os modelos produzidos em redes de Petri são ainda de boa legibilidade e suportam as abordagens associadas à composição de modelos (estruturação ascendente) (*bottom-up*) e à decomposição (estruturação descendente) (*top-down*). Todas estas características tornam este formalismo numa opção a ter em conta na modelação de sistemas.

Para esse efeito torna-se de interesse analisar o impacto de considerar novas características que ao serem introduzidas num determinado formalismo de modelação permitem melhorias na modelação da interação do sistema com o ambiente, nomeadamente na sua expressividade e consequente melhoria na legibilidade e redução da dimensão do modelo definido nessa linguagem específica.

Devido a diferentes especificidades e complexidades têm vindo a ser propostas na literatura várias arquiteturas para o desenvolvimento de sistemas de interface. O exemplo mais conhecido é o MVC (*Model – View - Controller*) [21] onde o sistema é dividido em três partes, uma que recebe a informação do utilizador, outra que executa o sistema e outra que apresenta os resultados ao utili-

zador. Para além desta existem outras arquiteturas como o *Model-View-Adapter* [22], o *Model-view-presenter* [23], o *Model View ViewModel* [24] ou o *Presentation – Abstraction – Control* [25].

Considerando o nível de complexidade que a modelação da interface de um sistema introduz, é de interesse dar relevância a esta parte do sistema, analisando que características deve um formalismo de modelação ter para lidar de forma mais eficiente com a interface do sistema. Essa introdução de características específicas para lidar com a modelação da interface deve permitir criar modelos mais simples e compactos, bem como com mais capacidade de modelação.

A metodologia utilizada neste trabalho é adaptada do método científico tradicional [26].

1.2. Introdução ao Problema

A modelação de sistemas com uma interface com o ambiente complexo, leva à criação de modelos muito grandes e difíceis de entender. Esta complexidade é particularmente crítica em formalismos de modelação gráficos, onde o recurso a uma descrição gráfica é utilizada para uma mais fácil interpretação por um utilizador humano. Nestes casos, o aumento da dimensão dos modelos torna-os ilegíveis retirando-lhes a principal vantagem da utilização de uma descrição gráfica.

Esta complexidade provém do facto de os sistemas de interface possuírem características específicas com as quais os formalismos de modelação não estão preparados para lidar, sendo necessárias adaptações para as utilizar.

Mesmo assim, devido às vantagens já apresentadas, a utilização de uma metodologia de desenvolvimento baseada em modelos é uma mais-valia na modelação de sistemas complexos.

O desenvolvimento baseado em modelos permite uma estruturação deste desenvolvimento, tanto em termos de fases como na partição dessas mesmas fases, de forma a permitir o desenvolvimento em paralelo.

No caso de sistemas com interfaces gráficas, como por exemplo os "*Serious Games*", desenvolvidos para várias plataformas, a interface tem uma complexidade de tal maneira relevante que pode ser vista como um sistema ela própria.

Nestes casos, o sistema pode ser dividido em vários subsistemas que podem ser desenvolvidos separadamente deixando a integração como um problema a tratar no fim, ou alternativamente desenvolver todo o sistema com todos os seus subsistemas apenas num modelo. Para isso podem ser utilizadas abordagens associadas à composição ascendente ("*bottom-up*") e à decomposição descendente ("*top-down*") de modelos.

Esta segunda abordagem permite que toda a análise do sistema continue a ser feita a cada passo da modelação, podendo validar através do modelo todo o sistema.

Alguns formalismos de modelação, como por exemplo as redes de Petri, possuem algumas das características necessárias ao desenvolvimento de um sistema através de uma abordagem como esta última.

Neste trabalho pretende-se determinar que características, deve ter um formalismo de modelação, para que permita uma modelação mais simples e compacta da interface do sistema modelado com o exterior do sistema.

Para isso, pretende-se tirar proveito das características de interesse dos formalismos de modelação existentes, como são exemplo as redes de Petri, bem

como das ferramentas de validação e geração de código que possam ser desenvolvidas, bem como das já existentes. Pretende-se ainda garantir a possibilidade de transformação de modelos desenvolvidos com as contribuições propostas para modelos de outros formalismos, tais como classes mais elementares de redes de Petri, permitindo a reutilização do corpo de teoria e ferramentas disponíveis para essas classes.

Será utilizada a classe de redes de Petri IOPT (*Input-Output Place-Transition*) como formalismo de referência, pois este formalismo já incorpora algumas características dedicadas à modelação de interface, como sinais, eventos e condições. Apesar da utilização desta classe específica como ponto de partida, pretende-se garantir a aplicabilidade dos conceitos desenvolvidos, analisados para as redes de Petri, a outros formalismos de modelação.

1.3. Questões de Investigação

Do problema acima descrito e depois de levantado o trabalho existente à sua volta, surgiu a seguinte questão de investigação:

Considerando os formalismos existentes de modelação de sistemas a eventos discretos, que características a ser utilizadas no modelo poderão ser introduzidas de forma a tornar mais simples e compacta a modelação da interface de um sistema com o ambiente?

Da sua análise resulta ainda a seguinte sub-questão:

Como poderemos representar o comportamento das características não-autónomas, em termos de dependências de sinais, eventos e suas semânticas de execução, de forma a permitir melhorar o suporte à modelação da interface de um sistema com o ambiente?

1.4. Tese

De forma a responder às questões de investigação, a tese defendida nesta dissertação é a seguinte:

É possível realizar uma modelação de um sistema, de forma mais simples e compacta, através da decomposição do modelo em partes de interface e de execução do sistema e da adição de uma caracterização separada das dependências e dos comportamentos dos sinais de entrada e de saída com o ambiente.

Para isso, foi necessário elaborar as seguintes análises à questão e sub-questão de investigação apresentadas anteriormente.

Ao modelar a interação de um sistema com o ambiente, é necessário ter em conta a divisão do sistema em duas partes distintas: a parte de processa-

mento interno, onde o sistema é executado, e a parte de interface com o utilizador.

Esta segunda parte pode também ser dividida em duas partes. Uma parte de *input* onde são recebidos os valores introduzidos pelo utilizador e outra de *output* onde os resultados são apresentados ao utilizador.

Tendo em conta as necessidades específicas de um sistema deste tipo, e a necessidade de separação das partes do sistema referidas anteriormente, por motivações várias (tais como a execução distribuída, a facilidade de substituição de uma delas ou apenas por maior facilidade de modelação), várias arquiteturas específicas têm vindo a ser propostas para suportar um melhor desenvolvimento dos sistemas.

Alguns exemplos destas arquiteturas, são o *Model-View-Controller*, o *Model-View-Adapter*, o *Model View ViewModel*, o *Model-view-presenter* ou o *Presentation–abstraction–control*, que serão apresentadas mais à frente neste documento.

Da mesma forma, ao usar um formalismo de modelação para modelar o sistema, o formalismo a ser utilizado deve garantir as características definidas nestas arquiteturas, logo a estrutura do modelo a usar quando se modela um sistema deste tipo deve:

- Permitir a modelação da estrutura de cada arquitetura, possibilitando a divisão do sistema na forma expressa em cada arquitetura;
- Garantir que o bloqueio de uma das partes não bloqueia todo o sistema.

A estrutura do formalismo deve permitir também uma divisão eficaz da parte de execução da parte do tratamento da interface, de forma a permitira substituição dessa interface por outro, ou o acesso à parte de execução por várias interfaces diferentes.

1.4.1. Sinais

O Sinal é o principal constituinte da parte não autónoma do modelo de um sistema.

Para permitir a modelação de sistemas complexos, o formalismo de modelação definido deve ter sinais com um conjunto de características que permitam:

- A definição de sinais de entrada do sistema, sinais de saída do sistema e ainda sinais internos ao sistema.
- Definir o sinal a partir de um conjunto de tipos de variável (inteiro, booleano, real) que permita a definição do maior número de sinais físicos possível.
- A definição tanto de sinais externos vindos do exterior como de sinais internos resultantes do processamento dos sinais externos.
- A composição de sinais através de estruturas, de forma a poder associar sinais que tenham uma relação entre si, facilitando a identificação da sua relação e a possibilidade de usar estes sinais compostos de forma mais eficiente na construção de funções e outras ferramentas do sistema.

Com as atualizações que venham a ser feitas aos sinais, será necessário redefinir as outras características do modelo onde os sinais sejam utilizados, para as habilitar a usarem os novos sinais, tais como:

- O controlo do disparo das transições,
- As funções de saída do modelo.

Será necessário estudar também a necessidade de adicionar funções associadas a estes sinais de forma a permitir a introdução de valores relativos aos sinais de entrada.

1.4.2. Eventos

Os eventos são associados a sinais e representam variações nestes. Os eventos são criados quando existe um determinado comportamento esperado no sinal associado.

Em alguns casos, os eventos são definidos sem ser apresentado um sinal associado. Nestes casos o evento está associado a um sinal externo do qual o conhecimento do seu valor não tem relevância para o sistema, mas apenas os eventos associados a ele.

A utilização de eventos permite, normalmente, uma modelação mais natural, que leva à criação de modelos mais compactos, libertando o modelo do processamento do sinal para obter esse evento.

De forma a permitir a modelação da interação com o sistema, o formalismo de modelação escolhido deve ter eventos com um conjunto de características que permitam:

- A definição de eventos de entrada do sistema, eventos de saída do sistema e ainda eventos internos do sistema.

- Definir eventos de variados tipos, que permitam uma experiência de modelação mais simples, tais como, a passagem por um valor determinado ou a alterações na variação do sinal.

- Definir eventos gerados a partir de diferentes janelas temporais de análise do sinal.

- Permitir a composição de sequências de eventos, de forma a encontrar comportamentos mais complexos do sinal associado.

- Permitir a representação de não eventos, ou seja, a não existência de evento.

Será necessário estudar também a possibilidade e viabilidade da introdução no sistema de mensagens associadas ao evento, como por exemplo poder introduzir no sistema o valor de certos sinais de interesse para a análise do evento, no momento em que o evento ocorreu.

1.5. Estrutura da dissertação e contribuições

Nesta secção é apresentada a estrutura do documento. São apresentados brevemente cada capítulo e secção. Em cada secção é apresentada uma lista dos artigos dos quais o autor é co-autor e que representam uma contribuição para o tema apresentado na secção.

Este documento está dividido em seis capítulos: Introdução, Revisão da Literatura, Contribuições, Validação, Discussão e Conclusões e Trabalho Futuro

No **Capítulo 1 – Introdução**, são apresentados o contexto do trabalho e feita uma introdução ao problema. Em seguida são apresentadas as questões de investigação e a tese que será defendida neste documento. Finalmente é apresentada a estrutura do documento bem como os artigos relacionados.

No **Capítulo 2 – Revisão da Literatura**, é feita uma introdução aos conceitos relacionados com o trabalho apresentado. Em seguida são apresentados trabalhos realizados no âmbito da interação com o sistema. Neste âmbito são apresentados de forma mais completa os *Complex Event Processing* e o *Quantum Event Programming*. Outros trabalhos são ainda apresentados de forma mais sucinta. Em seguida são apresentados três tipos de formalismos de modelação (fluxogramas, máquinas de estado e redes de Petri) considerados de interesse. Finalmente são apresentadas as principais arquiteturas de interação humano-sistema (*Model-View-Controller*, *Model-View-Adapter*, *Model-View-Presenter*, *Model-View-ViewModel* e *Presentation-Absteraction-Control*).

No **Capítulo 3 – Modelização da interface com o Ambiente**, são apresentadas as contribuições deste trabalho. O capítulo está dividido em três secções organizadas da seguinte forma:

Na **secção 3.1 - Introdução**, é feita uma introdução aos conceitos apresentados neste documento.

Na **secção 3.2 - Estrutura e fluxo da Interface**, são apresentados os conceitos de Modelo de Interpretação de Sinais e Modelo de Afetação de Sinais, definidos os conceitos que os constituem e o seu fluxo de análise. A secção está dividida em oito subsecções, cada uma apresentando um conceito relativo aos Modelo de Interpretação e afetação de Sinais.

A **subsecção 3.2.1 – Sinais Externos**, apresenta os conceitos de Sinais Externos de entrada e de saída. Define também a estrutura dos dados que o sinal deve conhecer. Estes Sinais foram propostos em [27].

[27] R. Campos-Rebelo, A. Costa, and L. Gomes, "Analysis and Generation of Logical Signals for Discrete Events Behavioral Modeling," in *Technological Innovation for Cloud-Based Engineering Systems SE - 16*, vol. 450, L. M. Camarinha-Matos, T. A. Baldissera, G. Di Orio, and F. Marques, Eds. Springer International Publishing, 2015, pp. 147–156.

Na **subsecção 3.2.2 – Sinais Internos** define-se os conceitos de sinal Interno de entrada e de saída. Define-se como ele é gerado e os dados que deve conhecer. Estes sinais foram também apresentados em [27].

Na **subsecção 3.2.3 – Condições Primitivas** define-se o conceito de Condição Primitiva e apresentam-se os diferentes tipos de condições primitivas propostos bem como a sua definição.

Na **subsecção 3.2.4 – Eventos Primitivos** são definidos os eventos primitivos propostos. Estes conceitos foram introduzidos em [28] e foram consecuti-

vamente atualizados em [29] e [30], bem como num capitulo publicado em livro [31]. É também definido o conceito de evento atrasado.

[28] R. Campos-Rebelo, A. Costa, and L. Gomes, "On Structuring Events for IOPT Net Models," in *Technological Innovation for the Internet of Things SE - 25*, vol. 394, L. Camarinha-Matos, S. Tomic, and P. Graça, Eds. Springer Berlin Heidelberg, 2013, pp. 229–238.

[29] R. Campos-Rebelo, A. Costa, and L. Gomes, "Events for Human-System Interaction modeling with IOPT Petri nets," in *Human System Interactions (HSI), 2013 6th International Conference on*, 2013, pp. 56–61.

[30] R. Campos-Rebelo, A. Costa, and L. Gomes, "Elementary Events for Modeling of Human-System Interactions with Petri Net Models," vol. 423, L. Camarinha-Matos, N. Barrento, and R. Mendonça, Eds. Springer Berlin Heidelberg, 2014, pp. 219–226.

[31] R. Campos-Rebelo, A. Costa, and L. Gomes, "Enhanced Event Modeling for Human-System Interactions Using IOPT Petri Nets," in *Human-Computer Systems Interaction: Backgrounds and Applications 3*, Springer, 2014, pp. 39–50.

A **subsecção 3.2.5 – Macro-Eventos** apresenta o conceito de macro-evento e as regras de composição para a sua definição. O conceito de composição de eventos primitivos foi inicialmente apresentado em [28] e em seguida utilizado na definição de macro-evento em [32]. O conceito de macro-evento é ainda atualizado neste documento, fundindo os conceitos já apresentados anteriormente com um conjunto de novos conceitos. São ainda apresentados os seguintes conceitos relacionados com a composição: não evento, tempo de vida de um evento e exclusividade de um evento.

[32] R. Campos-Rebelo, A. Costa, and L. Gomes, "Event Life Time in Detection of Sequences of Events," *2015 IEEE Int. Conf. Ind. Technol.*, 2015.

A **subsecção 3.2.6 – Macro-Condições** apresenta o conceito de macro-Condição, bem como as regras para as definir por composição de condições primitivas.

A **subsecção 3.2.7 – Ações de Saída** apresenta o conceito de Ação de saída, bem como as regras para as definir através da afetação de um sinal de saída. Estes conceitos foram inicialmente propostos em [33].

[33] R. Campos-Rebelo, A. Costa, and L. Gomes, "Output events for human-system interaction modeling," *Human System Interactions (HSI), 2014 7th International Conference on*. pp. 261–266, 2014.

A **subsecção 3.2.8 – Eventos de Saída** apresenta o conceito de evento de saída. São apresentados três conceitos principais que levam a definição de três tipos de eventos de saída. Estes conceitos foram também propostos em [33]. São ainda apresentadas características adicionais dos eventos de saída como o atraso e a ampliação.

Na **secção 3.3 – Sintaxe e Gramática** são apresentadas as duas sintaxes definidas, para além dos meta-modelos e representações em XML (*Extensible Markup Language*) dos conceitos definidos.

A **subsecção 3.3.1 – Sintaxe** apresenta duas sintaxes possíveis para a representação dos modelos de interpretação de sinais. Uma das sintaxes é textual, mais compacta, porém menos fácil de interpretar. A outra sintaxe é gráfica, tornando-se menos compacta, mas mais intuitiva para a interpretação por um utilizador humano.

A **subsecção 3.3.2 – Meta-Modelo e XML** apresenta o meta-modelo do modelo de interpretação dos sinais. São ainda apresentados exemplos XML de representação dos conceitos definidos.

No **Capítulo 4 – Validação** são apresentados exemplos que permitem validar os conceitos apresentados e a eficácia da sua utilização. Os exemplos apresentados são os seguintes:

Na **secção 4.1 – Ronda de um segurança por um bairro** é apresentado um exemplo onde se pretende modelar os resultados de uma ronda de um guarda por um bairro residencial. Neste exemplo pretende-se detetar se a ronda foi terminada corretamente, e caso não seja, onde houve o problema.

Na **secção 4.2 – Controlo de acessos a zona histórica de uma cidade** é apresentado um exemplo onde se pretende modelar o acesso à zona histórica de uma cidade, com uma entrada e uma saída e onde o número de carros no interior da zona é limitado.

Na **secção 4.3 – Sistema de recuperação para segurança de um *drone*** é apresentado um exemplo onde se pretende modelar o um controlador que, em caso de falha do *drone*, toma controle deste e estabiliza-o até que este seja recuperado.

Na **secção 4.4 – Discussão** são discutidas as implementações dos vários exemplos e analisadas as vantagens da utilização de modelos de interpretação e afetação de sinais na sua implementação.

No **Capítulo 5 – Conclusões e Trabalho Futuro** são apresentadas as conclusões e sugeridos alguns temas para trabalhos futuros associados a esta dissertação.

Neste capítulo foi apresentado o problema que se pretende resolver, bem como a tese que se defende para o resolver. Foi ainda apresentada e estrutura do documento. No capítulo seguinte será apresentada uma revisão da literatura existente.

2

Revisão da Literatura

Neste segundo capítulo é apresentada a Revisão da Literatura. Após uma introdução, são apresentados conceitos que se consideram relevantes para a compreensão das contribuições propostas. São ainda apresentados outros trabalhos desenvolvidos na área, bem como uma comparação com as contribuições propostas.

2.1. Introdução

A forma de interagir com os sistemas sempre foi um dos grandes desafios no desenvolvimento de sistemas computacionais. Muitas foram as formas criadas para introduzir informação no sistema. Um exemplo é o cartão perfurado, inventado nos Estados Unidos da América em 1880 por Herman Hollerith [34] e utilizado pela primeira vez em máquinas de calcular em 1940. Com o passar dos tempos tem-se trabalhado em formas mais simples e intuitivas de introduzir informação nestes sistemas.

A generalização da utilização dos sistemas computacionais trouxe também, uma heterogeneidade de utilizadores, deixando de ser a sua utilização exclusivo de quem tem conhecimentos do sistema ou mesmo de sistemas compu-

tacionais. Por outro lado, o aumento da complexidade destes sistemas, de forma a permitir cada vez mais utilizações, dificulta a sua operabilidade.

Por este motivo, tem sido feito um grande investimento no desenvolvimento de interfaces mais eficientes, fáceis de usar e também mais agradáveis, de forma a acomodar o maior número de utilizadores.

Esta evolução na forma de interagir com o sistema levou desta forma a uma maior complexidade no desenvolvimento do sistema. O que leva à necessidade de a própria interface ter não apenas uma parte de comunicação, mas também partes de processamento e execução.

Tendo em conta esta especificidade, foram propostas arquiteturas para o desenvolvimento de sistemas de interface humano-sistema. Algumas destas arquiteturas serão apresentadas neste capítulo.

2.2. *Interação com o Sistema*

Nas formas de interação com um sistema, em particular nos métodos para introduzir os dados no sistema, existem vários trabalhos que apresentam propostas para definir como a informação a ser introduzida no sistema deve ser adquirida, analisada e tratada. Alguns trabalhos, como o desenvolvimento do processamento de eventos complexos (CEP – *Complex Event Processing*) ou o *Quantum Event Programming*, são exemplos dessa abordagem.

2.2.1. Complex Event Processing

Os CEP, que estudam o processamento de eventos que combinam dados provenientes de várias fontes [35][36], foram apresentados pela primeira vez por David Luckham no seu livro "*The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*" [37]. Um exemplo da sua

aplicação é apresentado em [38], onde os CEP são utilizados numa aplicação com base em RFID. No entanto, eles são utilizados numa ampla gama de áreas, tais como simulação discreta de eventos, bases de dados ativas, administração de redes e raciocínio temporal.

Nos CEP, um evento é definido como um objeto onde é registada uma determinada atividade num sistema. Esses eventos podem relacionar-se entre si por relações de tempo, causalidade e agregação.

Os CEP são uma forma de tratar a composição de eventos provenientes de uma nuvem de eventos, o que permite a implementação de Arquiteturas orientadas por eventos (*Event-Driven Architectures*) (EDA) [39].

Isto é, os CEP são uma tecnologia direcionada para a extração de informação de sistemas distribuídos com base em mensagens. Esta tecnologia permite aos utilizadores especificar informações que são de interesse para o sistema [40].

Em arquiteturas orientadas por eventos, quando algum evento de interesse ocorre dentro ou fora da arquitetura do sistema, essa informação é imediatamente divulgada a todos os interessados. Cada uma dessas partes avalia cada caso e decide se deve ou não agir. As ações orientadas a eventos podem incluir a invocação de um serviço [41], o desencadeamento de um processo de negócio, ou a publicação de informações, como no trabalho apresentado em [42].

Juntamente com os CEP também foi desenvolvida uma linguagem que permite a sua implementação. Esta linguagem denomina-se RAPIDE [43][44].

O RAPIDE é uma linguagem de computador para a definição e execução de modelos de arquiteturas de sistema. O resultado da execução de um modelo RAPIDE é um conjunto de eventos que ocorreram durante essa execução, para além da relação temporal e causal entre esses eventos [45].

Os CEP são utilizados em diferentes trabalhos mais recentes. Um exemplo da sua utilização pode encontrar-se em [46] onde é apresentada uma arquitetura de CEP para a Internet das Coisas, baseado em Gráfos acíclicos dirigidos [47].

2.2.2. Quantum Event Programming

Na computação clássica, é necessário compreender a utilização dos recursos disponíveis, como memória ou capacidade de processamento, mas não é necessário compreender os fenómenos físicos associados. Cada bit assume o valor zero ou um.

Na computação quântica, um *qubit* (*quantum bit*) é um sistema de mecânica quântica de dois estados, que possui os dois valores ao mesmo tempo. Por essa razão, um *qubit* não pode ser medido durante uma operação, mas apenas no final de cada processo.

Em 1982, Paul Benioff [48] e Richard Feynman [49], separadamente observaram que um sistema de mecânica quântica pode ser utilizado para fazer computação, introduzindo assim o conceito de computação quântica (*Quantum Computing*).

Foi mostrado por Paul Benioff que a miniaturização de circuitos integrados apresentava efeitos relacionados com a mecânica quântica que afetavam o comportamento dos circuitos. Por sua vez, Richard Feynman, que focou mais o seu trabalho na simulação de sistemas quânticos, verificou que essa simulação, quando feita em computadores clássicos é computacionalmente dispendiosa. Dessa forma definiu que um sistema quântico deve ser simulado mais eficientemente num computador quântico.

Em 1985, David Deutsch [50] dá os primeiros passos para tornar possível a ideia de um computador quântico, quando define a versão quântica de uma máquina de Turing. Dessa forma, e apesar de os exemplos que apresentou na

altura, terem aplicação limitada, demonstraram que um computador quântico seria capaz de realizar algumas tarefas de maneira mais eficiente que o computador clássico.

Apesar disso, o mais conhecido exemplo da eficiência do computador quântico é apresentado em 1994 por Peter Shor [51]. Nesse trabalho descreve-se um conjunto de algoritmos para logaritmos discretos e cálculo de fatoriais, que é muito utilizado na computação de algoritmos criptográficos, demostrando que estes seriam realizados de forma muito mais eficiente em computadores quânticos do que em computadores clássicos, tirando partido do paralelismo quântico.

Vários trabalhos têm sido desenvolvidos em computação quântica ao longo dos anos, tanto na própria definição da computação quântica [52], como no intuito de desenvolver linguagens de programação quântica [53].

Quantum plataforma é uma plataforma que engloba um conjunto estruturado de *software open source*, incluindo uma ferramenta de modelação de mecânica quântica, criado para o desenvolvimento de sistemas cooperativos em tempo real, com base em máquinas de estado, como uma arquiteturas baseadas em eventos [54]. Esta plataforma permite ainda geração automática de código.

2.2.3. Outros Trabalhos

Existem ainda outros trabalhos de interesse nas áreas de interface com o ambiente. Um exemplo é o trabalho apresentado em [55], na área da composição de eventos em bases de dados ativas. Outro exemplo de interesse é o trabalho apresentado em [56] onde são utilizadas redes de Petri para modelar os vários tipos de composição apresentados [56].

Existem também variados trabalhos na área da implementação de interfaces do sistema com o ambiente, utilizando uma abordagem baseada em mode-

los. Alguns exemplos são: Um trabalho onde são utilizadas RdP reativas hierárquicas para modelar o *controller* na arquitetura MVC e onde são apresentados exemplos de implementações de interface gráfica [57], outro que estuda como um modelo formal de interação pode ser utilizado para classificar e esclarecer várias propriedades de comportamento interativo do sistema [58] ou ainda um gerador automático da interface gráfica do utilizador a partir de uma RdP [59].

Finalmente, na implementação dessas arquiteturas usando as abordagens baseadas em modelos, existe, por exemplo, o trabalho apresentado em [60] que introduz o conceito de Partição flexível de aplicações web (*Flexible Web-Application Partitioning*), um modelo de programação e infraestrutura de implementação, que permitem ao desenvolvedor aplicar o padrão MVC numa aplicação de uma forma independente de partição.

2.3. Formalismos de Modelação

A utilização de métodos formais no desenvolvimento de sistemas vem crescendo não apenas na comunidade científica, mas também na indústria. Alguns exemplos podem ser encontrados em [61] [62] [63]. A ênfase recente no desenvolvimento baseado em modelos, frequentemente baseada em *UML* [8] [64] [7] tenta responder a vários problemas distintos, com impacto direto na qualidade e robustez dos sistemas desenvolvidos, bem como sobre o tempo associado ao mercado [65] [66] [67].

Utilizando ferramentas computacionais adequadas, o desenvolvimento baseado em modelos permite uma transição por etapas, entre modelos e código executável para plataformas específicas. Uma das iniciativas mais significativas nessa área é a *Model Driven Architecture* [6] desenvolvida pelo *Object Management Group* (http://www.omg.org). Esta baseia-se na separação entre a especifi-

cação do sistema e a aplicação respetiva para plataformas específicas. Vários modelos de computação têm sido utilizados no desenvolvimento de sistemas para especificar comportamentos e funcionalidades [63] [68] [69] [70]. Entre eles, os formalismos gráficos são úteis quando fornecem uma semântica precisa e ferramentas computacionais que permitem troca de modelo, ou seja, editores, ferramentas de validação e verificação, e ferramentas de geração de código. Entre os formalismos de modelação mais comuns, sublinham-se os diagramas de estado, estadogramas (*Statecharts*) [14] [71] e outros formalismos baseados em máquinas de estado hierárquicas e concorrentes, diagramas de sequência, diagramas de atividades, bem como as redes de Petri [61] [63] [72] [73].

A importância da adoção de atitudes de desenvolvimento baseado em modelos, bem como a utilização de ferramentas de automação de projeto que permitam automatizar tarefas como a geração de código, leva a considerar a possibilidade de associar as características comportamentais dos modelos, utilizadas para a descrição dos componentes de controlo, com as características de ambientes gráficos para uma interação ou visualização [59].

A utilização de uma abordagem baseada em modelos permite também, utilizando o mesmo modelo, a implementação do mesmo sistema em várias plataformas, bem como a possibilidade do sistema ser desenvolvido por várias pessoas e até utilizando ferramentas diferentes [69].

2.3.1. Fluxogramas

Os fluxogramas são um dos formalismos de modelação mais conhecido e mais utilizado na modelação de sistemas. Eles são utilizados nas mais diversas áreas, para representar e esquematizar sistemas. São um formalismo simples, o que traz grandes vantagens em termos de interpretação, mas também o torna mais limitado para modelar sistemas mais complexos.

Apesar de existirem vários trabalhos desenvolvidos na definição de fluxogramas desde a década de 1920 até à década de 1940 só em 1985 foi definida uma norma (ISO 5807) [74].

Os fluxogramas têm uma notação gráfica clara, representada por um conjunto de símbolos apresentados na Figura 2.1:

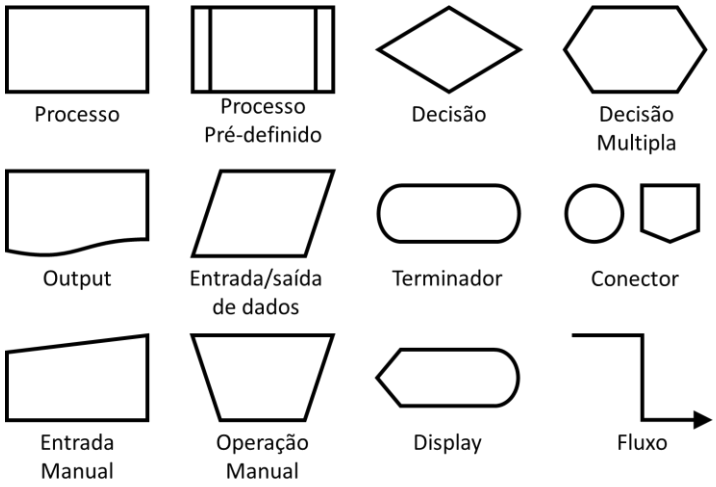


Figura 2.1: Principais Símbolos utilizados num fluxograma (norma ISO 5807: 1985).

O fluxo dentro do fluxograma é representado por uma linha e caracteriza as relações existentes entre os vários símbolos do diagrama. Normalmente tem a ponta em forma de seta indicando a direção do fluxo de ação [75].

2.3.2. Máquinas de Estado

As máquinas de estados são também um formalismo de modelação bastante utilizado em diversas áreas.

Apesar de terem tido origem no desenvolvimento de modelos matemáticos e teoria computacional para as áreas das ciências biológicas, as máquinas de estados têm, hoje em dia, uma utilização relevante nas áreas da engenharia e na

informática. Essa relevância deve-se aos trabalhos desenvolvidos nos Bell Labs e na IBM na década de 1960 por George H. Mealy e Edward F. Moore.

Uma máquina de estados finita, normalmente representada através de um diagrama de estados, define-se como um conjunto finito de estados (um deles definido como estado inicial), um conjunto de transições entre estados um conjunto de entradas e um conjunto de saídas. É ainda necessário garantir que apenas um estado está ativo em cada instante.

As máquinas de estado finitas podem ser divididas em dois tipos: Máquinas de Moore [76] e Máquinas de Mealy [77].

A principal diferença entre uma máquina de Moore e uma máquina de Mealy está na forma como as saídas são afetadas.

Uma máquina de estado finita pode ser definida pelo seguinte tuplo (E, E0, I, O, T, F)

Onde:

E é um conjunto finito de estados

E0 é um estado do conjunto E que representa o estado inicial da máquina.

I é um conjunto de Entradas;

O é um conjunto de Saídas;

T é um conjunto de funções de transição que analisam o estado atual e as entradas de forma a obter o próximo estado.

No caso de uma máquina de Moore:

F é um conjunto de funções de saída que analisam o estado atual e alteram os valores das saídas.

No caso de uma máquina de Mealy:

F é um conjunto de funções de saída que analisam o estado atual e o valor das entradas, e alteram os valores das saídas.

Numa máquina de Moore, os valores das saídas são determinados apenas pelo estado atual.

Numa máquina de Mealy, os valores das saídas são determinados pela análise do estado atual e pelo valor das entradas.

2.3.3. Redes de Petri

As RdP [78] foram propostas por Carl Adam Petri em 1962. Foram apresentadas pela primeira vez na sua dissertação de Doutoramento na Faculdade de Matemática e Física da Universidade Técnica de Darmstadt, Alemanha [16].

Nesta dissertação foram apresentadas uma variedade de ideias e propostas acerca dos fundamentos da concorrência.

As redes de Petri como são conhecidas atualmente foram apresentadas pelo próprio Petri, num colóquio em Hannover que ocorreu em 1966 [79].

Entre as características que normalmente são referidas em sua defesa, incluem-se a semântica e sintaxe exatas, a modelação explícita de características como a sincronização, a concorrência e a gestão de recursos, bem como uma boa legibilidade dos modelos produzidos e o suporte para as abordagens associadas à composição ascendente de modelos (“bottom-up”) e à decomposição descendente (“top-down”) [80].

Estas permitem ainda a visualização simultânea da estrutura e do comportamento do sistema [81], tendo ainda a capacidade de modelar problemas como o sincronismo, concorrência, conflitos e partilha de recursos [82].

A representação gráfica das RdP é feita através de um grafo com dois tipos de nós: lugares e transições ligados através de arcos. Estes são os três elementos constituintes de uma RdP [83].

As RdP podem classificar-se segundo dois critérios: em relação ao tipo de marcas, e em relação à dependência das características físicas.

Em relação ao tipo de marcas, se as marcas são indiferenciadas, a rede é classificada como de Baixo Nível; se existem vários tipos de marcas a rede é classificada de Alto Nível [84].

Em relação à dependência das características físicas, classifica-se como autónoma se as regras de disparo das transições não dependerem destas e como não-autónomas se as regras de disparo das transições dependerem de características físicas [85].

As RdP não-autónomas são de especial interesse para o trabalho apresentado neste documento, pois permitem uma representação do ambiente dentro do sistema.

Existem vários exemplos de RdP não autónomas como por exemplo as RdP interpretada [86] [87] [88], as RdP Sincronizadas [89] ou as RdP temporizadas [90] todas de baixo nível, ou as RdP reativas [91] [92] de alto nível.

As RdP IOPT, apresentadas em [93], são uma classe não-autónoma de baixo nível que estende as RdP Lugar-Transição com características que permitem a modelação da interação com o ambiente através de sinais e eventos, tanto de entrada como de saída.

Devido às suas características não autónomas que, apesar de, de uma forma elementar, permitem a modelação da interação com o ambiente, esta classe serve como ponto de partida para a solução proposta neste documento.

2.4. *Arquiteturas de Interação Humano– Sistema*

Em aplicações informáticas complexas que necessitam de apresentar grandes quantidades de dados ao utilizador, o desenvolvedor geralmente deseja separar os dados da interface de utilizador para que as alterações na interface do utilizador não afetem o tratamento dos dados e para os dados poderem ser reorganizados sem alterar a interface do utilizador.

Na literatura podem-se encontrar vários padrões de arquiteturas que, de formas distintas, se propõem a resolver este problema.

Neste capítulo serão apresentadas as arquiteturas de maior relevância e maior incidência na literatura.

2.4.1. Model - View - Controller

O padrão MVC (*Model – View – Controller*), é um padrão de arquitetura de *software* que separa a representação de informação a partir da interação do utilizador com ele, foi apresentado pela primeira vez em 1979 no artigo " *Applications Programming in Smalltalk-80: How to use Model–View–Controller*" [21] e é atualmente a arquitetura mais conhecida. A sua definição, proposta nesse documento, é apresentada na Figura 2.2.

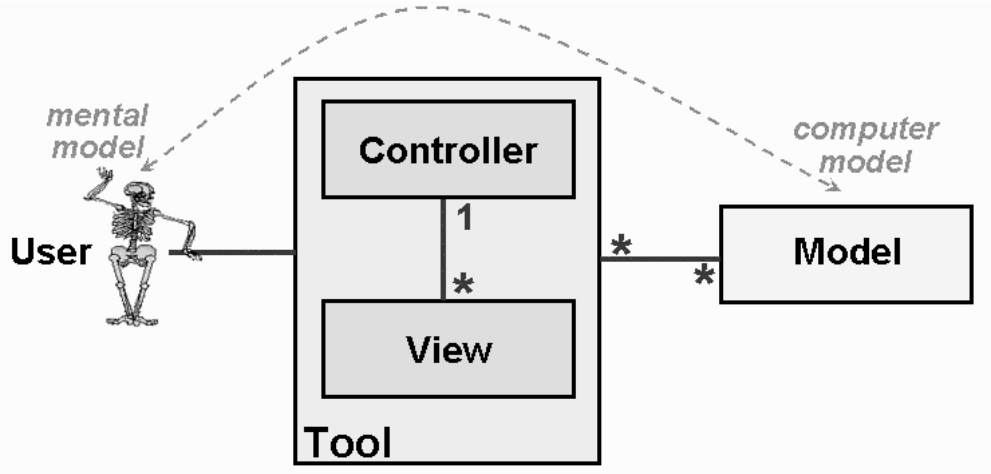


Figura 2.2: MVC como apresentado em [21].

Na arquitetura MVC, os dados introduzidos pelo utilizador, a modelação do sistema, e a apresentação visual para o utilizador dos resultados são separados de forma robusta. Estes são implementados em três partes distintas do modelo, cada parte especializada na sua tarefa: o modelo (*Model*), a visualização (*View*) e o controlador (*Controller*).

O modelo (***model***) é utilizado para definir e gerir o domínio da informação e notificar observadores sobre mudanças nos dados. É uma representação detalhada da informação que a aplicação opera. A lógica de negócio adiciona valor semântico aos dados, e quando há mudanças no estado do modelo, ele notifica os observadores.

A componente visão (***View***) apresenta o modelo num formato apropriado para ser visualizado pelo utilizador. Podem existir várias visões (*views*) diferentes para o mesmo modelo.

O controlador (***controller***) recebe os dados de entrada, introduzidos pelo utilizador. É responsável por validar e filtrar esses dados para depois serem aplicados ao modelo (***model***).

A colaboração típica entre os três componentes desta arquitetura é descrita em seguida.

O utilizador introduz os dados de entrada no *controller*. Esta informação vai ser entregue ao *model* que a vai manipular. Esta manipulação gera uma alteração dos dados de saída que irão ser utilizados para atualizar o *view*. No *view* a informação será apresentada de forma que o utilizador a possa entender. Este fluxo é apresentado na Figura 2.3.

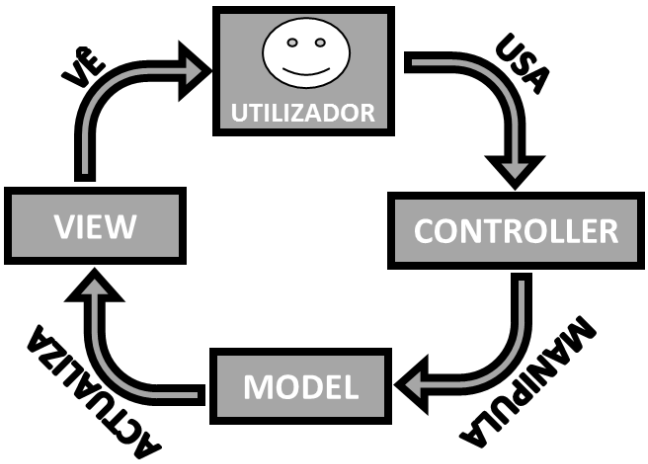


Figura 2.3: Diagrama do fluxo de dados no MVC

Em [21] são apresentadas duas variações do MVC: O *model* passivo e o *model* ativo.

O *model* passivo é utilizado quando o *model* é manipulado por apenas um *controller*. Nesta variação, o *controller* modifica o modelo e depois informa a componente visão que esta deve ser atualizada, pois o modelo foi alterado. Neste cenário, o *model* é completamente independente do *view* e do *controller*, o que significa que o *model* não tem forma de comunicar que houve alterações do seu estado. O protocolo HTTP é um exemplo desta variante. Não há nenhuma maneira simples no navegador de obter atualizações assíncronas do servidor. O navegador exibe a informação e responde às entradas do utilizador, mas não deteta alterações nos dados no servidor. Apenas quando o utilizador solicita expressamente uma atualização é que o servidor é questionado se há alterações. O seu comportamento é apresentado no diagrama da Figura 2.4.

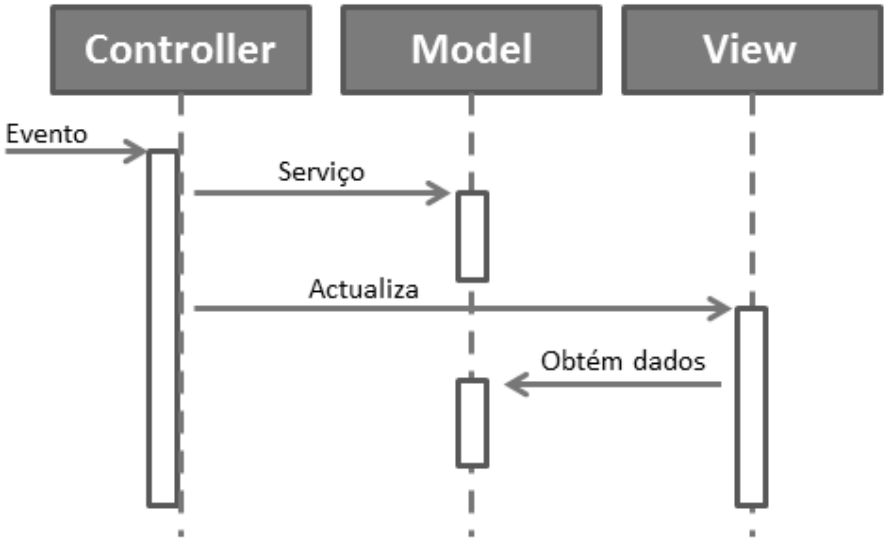


Figura 2.4: Comportamento do Model Passivo. Adaptado de [94]

O *model* ativo, por seu lado, é utilizado quando o *model* pode mudar de estado sem ser por intervenção do *controller*. Por exemplo quando existe mais que um *controller* a alterar o *model*. Neste caso apenas o *model* tem conhecimento de todas as vezes que é alterado, logo deve ser ele a alterar os *views*. Esta necessidade torna-se um problema, pois uma das motivações para usar o Padrão MVC é tornar o *model* e o *view* independentes. Com a necessidade do *model* notificar o *view* está a reintroduzir-se dependência entre eles. Para resolver este problema pode usar-se o padrão de *Observer* [95] que fornece um mecanismo para alertar outros objetos de mudanças de estado, sem introduzir dependência entre eles.

Como referido anteriormente o MVC é o padrão mais aceite atualmente, sendo inclusivamente referido nos documentos de desenvolvimento das principais tecnologias [96] [97] [94]. É também a referência para todos os outros padrões, sendo utilizado como ponto de comparação nas suas definições.

2.4.2. *Model–view–adapter*

O *model–view–adapter* (MVA), também conhecido como *mediating-controller* MVC, é um padrão de arquitetura de múltiplas camadas. O MVA tenta resolver os mesmos problemas que o MVC, apresentado anteriormente, mas com soluções diferentes. O MVC dispõe o *model*, o *view* e o *controller* nos vértices de um triângulo, onde cada um tem comunicação com os outros dois, o que significa que pode haver informação a passar entre o *model* e o *view*, sem conhecimento do *controller*. O MVA dispõe os três constituintes em linha, com o *adapter* ou *mediating controller* colocado entre o *model* e o *view*. Desta forma não há comunicação direta entre o *model* e o *view*. As formas de comunicar das duas arquiteturas são apresentadas na Figura 2.5.

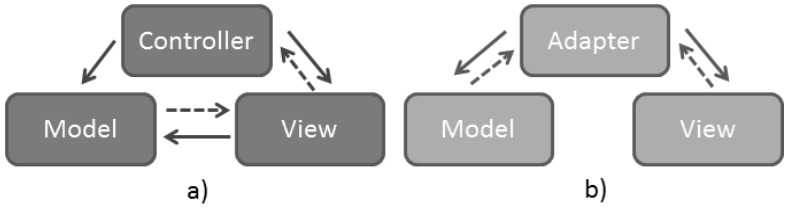


Figura 2.5: Diagrama de arquitetura. a) *Model-View-Controller*. b) *Model-View-Adapter*

O *model* e o *View* estão completamente dissociados um do outro, podendo apenas comunicar por intermédio do *adapter*. Dessa forma, o *model* e o *view* são intencionalmente dissociados um do outro, não tendo qualquer conhecimento direto entre si, apenas o *adapter* tem conhecimento de todo o sistema.

Esta separação permite que a informação contida no *model* e no *view* sejam completamente diferentes uma da outra. Dessa forma, é possível que vários *view*, mesmo que diferentes, possam aceder ao mesmo *model* através do *adapter* que trata os dados para serem consumidos pelo *model*.

Podem ser criados ainda vários *adapters* para o mesmo *model* que tratam os dados vindos de diferentes *views*.

Existem alguns exemplos da utilização deste padrão tais como um trabalho de restruturação de um *framework* orientado a objetos [98].

2.4.3. Model–view–presenter

O *model-view-presenter* (MVP), é um padrão de arquitetura criado no início dos anos de 1990 no projeto "*Taligent*", uma *joint venture* da Apple, IBM e HP. Este foi mais tarde migrado do "*Taligent*" para o Java e apresentado por Mike Potel [23]. Depois do fim do "*Taligent*" em 1997, Andy Bower e Blair McGlashan da *Dolphin Smalltalk* adaptaram o padrão para ser a base da interface de utilizador do projeto "*Smalltalk*" [99]. Em 2006, a Microsoft começou a incorporar o MVP na sua documentação e em exemplos de interfaces de utilizador no .NET *framework* [100]. Também o *Google developers* apresenta informação de desenvolvimento com este padrão [101]. As evoluções e múltiplas variações do padrão MVP, incluindo a relação com outros padrões de *design* de interface como o MVC são analisadas em detalhe em vários artigos [102][103].

O MVP é constituído por 3 componentes.

O ***model*** é uma interface que define os dados a ser apresentados ou executados na interface de utilizador.

O ***view*** é uma interface passiva que apresenta os dados vindos do *model* e direciona os comandos do utilizador para o *presenter* para executar essa informação.

O ***presenter*** atua sobre o *model* e o *view*, obtendo dados do *model* e formatando-os para serem apresentados no *view*.

Quando o *model* é atualizado, o *view* também tem de ser atualizado para refletir as alterações. A atualização do *view* pode ser tratada de várias maneiras. As variantes do *Model-View-Presenter*, "*Passive View*" e "*Supervising Controller*", especificam diferentes abordagens para a implementação da atualização do *view*.

No "*Passive View*", o *presenter* atualiza o *view* para refletir as mudanças no modelo. A interação com o *model* é tratada exclusivamente pelo *presenter*, o *view* não tem conhecimento de alterações no modelo.

No "*Supervising Controller*", o *view* interage diretamente com o modelo para realizar ligações de dados simples que podem ser definidas de forma declarativa, sem a intervenção do *presenter*. O *presenter* atualiza o modelo, que manipula o estado do *view* apenas nos casos que não possam ser especificados. A Figura 2.6 ilustra a visão lógica do "*Passive View*" e do "*Supervising Controller*" [104].

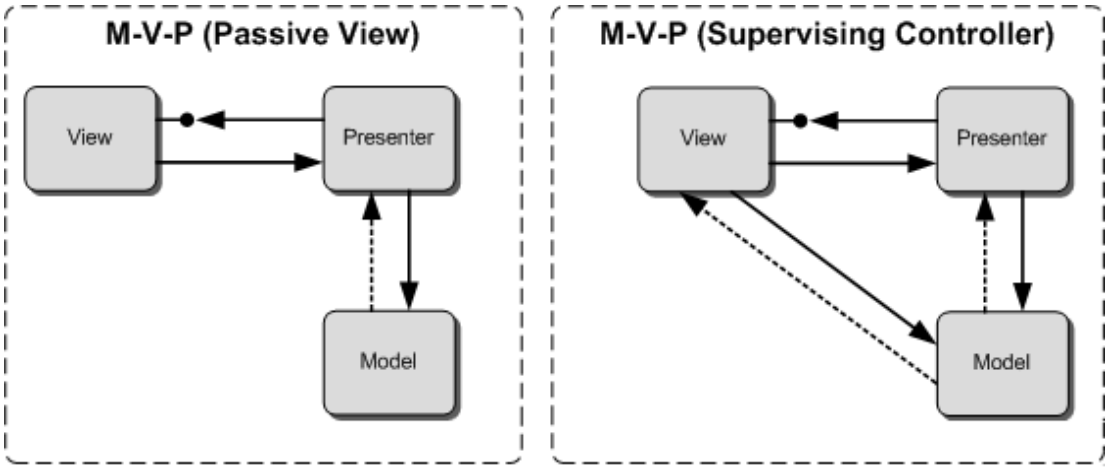


Figura 2.6: Diagrama de arquitetura do *Model-View-Adapter*. a) *Passive View*. b) *Supervising Controller*.

2.4.4. Model View ViewModel

O *Model View ViewModel* (MVVM) [24] [105] é um padrão de arquitetura de *software* da *Microsoft*, desenvolvida por John Gossman [106], que teve origem numa especialização do padrão *model-view-presenter*. Em grande parte baseada no padrão MVC, o MVVM é direcionado a plataformas de desenvolvimento de interface do utilizador que suportam programação orientada a eventos, como o HTML5, *Windows Presentation Foundation* (WPF) [107] ou o *Silverlight* [108].

O MVVM é constituído por 3 elementos:

Model: Tal como no padrão MVC, o *model* refere-se a um modelo de domínio que representa o conteúdo do estado, ou à camada de acesso de dados que representa o conteúdo.

View: Da mesma forma que no padrão MVC, o *view* refere-se a todos os elementos apresentados pela interface gráfica (GUI).

ViewModel: O *ViewModel* é um "modelo do *view*" o que significa que é uma abstração do *view* que serve de mediador entre o *view* e o *model*, podendo ser visto como um tipo especializado do que seria um controlador no padrão MVC que atua como um conversor que transforma a informação do *model* em informação para o *view* e passa comandos do *view* para o *model*. O *viewModel* pode ser visto como um estado conceptual dos dados, em oposição ao estado real dos dados no *model*. O termo "*ViewModel*" é uma das principais causas de confusão na compreensão do padrão em comparação com os padrões mais implementados MVC ou MVP. O papel do *controller* ou do presenter dos outros padrões tem sido substituído por um *framework binder* (por exemplo, *Extensible Application Markup Language* (XAML)) e pelo *viewModel* como mediador e conversor do *model* para o *binder*.

Na Figura 2.7 pode ver-se a relação entre o *View*, o *ViewModel* e o *Model*, como apresentada em [105].

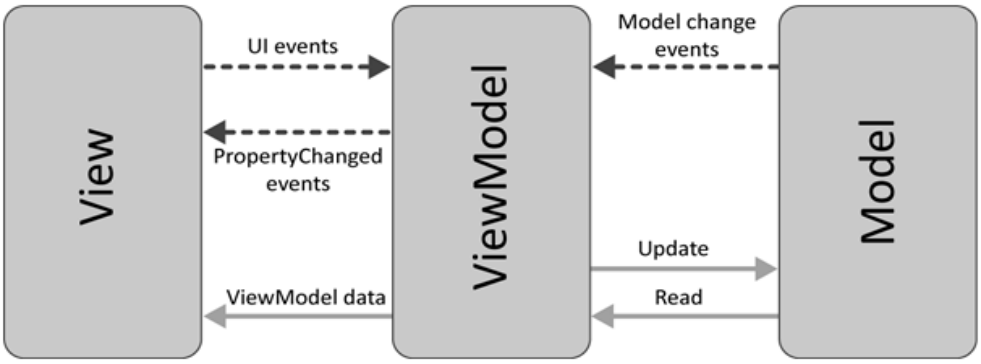


Figura 2.7: Relação entre os três constituintes da arquitetura MVVM

O MVVM facilita uma separação clara entre o desenvolvimento da interface gráfica e o desenvolvimento do resto do sistema, como mostra a Figura 2.8. O modelo "*view*" do MVVM é um conversor de valor o que significa que o modelo "*view*" é responsável por apresentar os objetos de dados ao modelo de maneira a que esses sejam mais facilmente geridos e consumidos. O *viewModel* lida com a maioria, se não toda a lógica do *view* de exibição. Ele é na realidade mais *model* do que *view*. A demarcação exata entre as funções tratadas por cada um dos elementos é uma discussão não finalizada na comunidade científica.

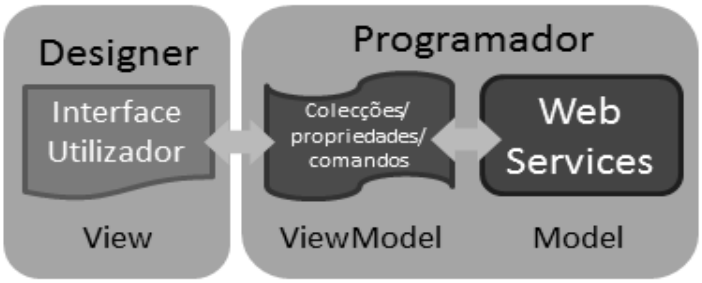


Figura 2.8: Separação do MVVM

MVVM foi concebido para utilizar funções de dados de ligação em WPF para facilitar a separação o desenvolvimento do *view* do resto do sistema, removendo virtualmente todo o código GUI do *view* [107]. Em vez disso, a GUI é desenvolvida separadamente utilizando uma linguagem padrão como por exemplo o XAML e criando ligações ao *viewModel*, que é escrito e mantido por quem desenvolve as aplicações. Esta separação permite que as várias partes de uma aplicação sejam desenvolvidas em vários fluxos de trabalho para maior produtividade.

2.4.5. Presentation–abstraction–control

Presentation–abstraction–control (PAC) é outro padrão de arquitetura de software, apresentado em 1987 em [25]. É uma arquitetura de *software* orientada à interação, e é um pouco semelhante ao model-view-controller (MVC), na medida em que separa um sistema interativo em três tipos de componentes responsáveis por aspetos específicos da funcionalidade da aplicação. O componente *Abstraction* recupera e processa os dados, o componente *Presentation* que formata a apresentação visual e de áudio dos dados, e o componente *control* que lida com o fluxo de controlo e a comunicação entre os outros dois componentes [109].

O PAC consiste numa hierarquia de agentes [110]. Cada agente é feito a partir de um conjunto de três partes (*Presentation Abstraction e Control*). O agente *control* atua como um mediador, através do qual os agentes *presentation* e *abstraction* comunicam. A forma como estes agentes se ligam pode ser vista na Figura 2.9. Devido ao isolamento entre a *presentation* e a *Abstraction*, é possível fazer *multi-tasking* entre estes dois agentes para dar ao utilizador tempos de exe-

cução muito curtos. Esta é uma diferença para o MVC, onde estes conceitos não estão totalmente isolados, podendo haver alguma ligação entre eles.

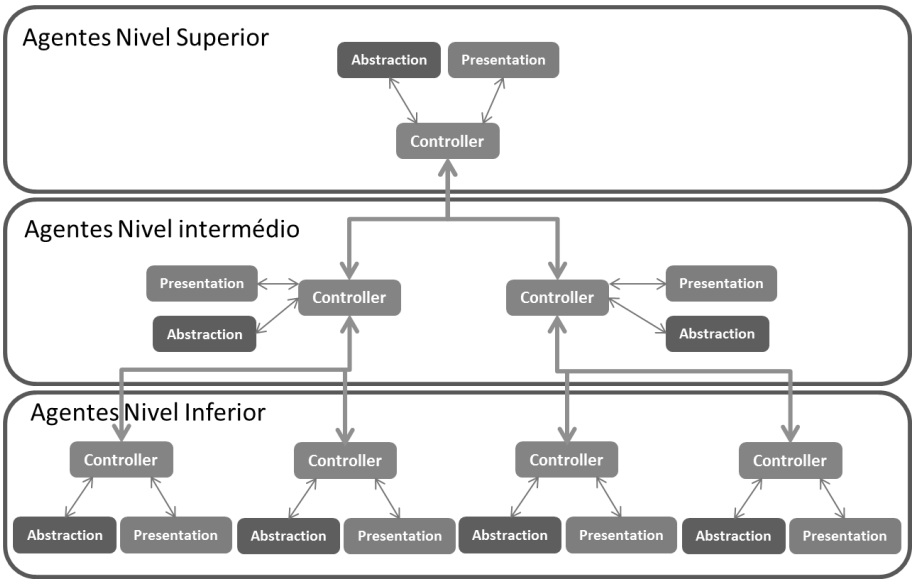


Figura 2.9: Comunicação entre agentes na arquitetura PAC

Em contraste com o MVC, o PAC é utilizado como uma estrutura hierárquica de agentes, cada um consistindo de uma tríade de peças de abstração, apresentação e controlo. Os agentes (ou tríades) comunicam uns com os outros apenas através da parte de controlo de cada tríade. Outra diferença para o MVC está no facto de cada tríade isolar totalmente a *presentation* (view no MVC) da *abstraction* (model no MVC), que permite, por exemplo, que a interface (*presentation*) possa ser apresentada antes do *abstraction* estar totalmente inicializada.

Neste capítulo foi apresentada a revisão da literatura associada, dando ênfase às técnicas de interação com o sistema, aos formalismos de modelação existentes e às arquiteturas de Interação Humano-Sistema mais utilizadas. No capítulo seguinte serão apresentadas as contribuições deste trabalho para a modelização de interfaces com o ambiente.

3

Modelização da interface com o Ambiente

Neste capítulo são apresentadas as contribuições propostas com o intuito de melhorar a modelação da interface do ambiente com um sistema, permitindo uma modelação mais simples e compacta. São apresentadas a estrutura e o fluxo de evolução da visão integrada dos vários modelos que compõem o sistema e ainda os conceitos definidos, de forma individual.

3.1. Introdução

A solução proposta neste documento divide o modelo em três partes, de forma a permitir a modelação da interpretação dos sinais de entrada, do fluxo de execução e da afetação dos sinais de saída em partes distintas, podendo assim usar formalismos específicos para cada um deles.

Dessa forma, o Modelo global é definido pela divisão em três partes distintas (*INPUT*, *OUTPUT* e MODELO DE EXECUÇÃO), como mostra a Figura 3.1.

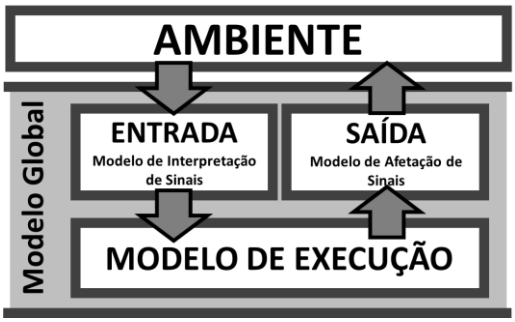


Figura 3.1: Composição e estrutura do modelo global

O sistema recebe informação do ambiente na *entrada* através de sinais de entrada. Este interpreta os valores recebidos e gera informação para o modelo de execução. Este, por sua vez, gera informação de saída que é passada ao *output*. O *output* interpreta a informação recebida do modelo de execução de forma a afetar os sinais de saída, a serem disponibilizados para o ambiente.

O modelo de execução é modelado recorrendo a formalismos de modelação existentes.

A contribuição deste trabalho prende-se com a modelação dos módulos *input* e *output* e com a interligação entre os três módulos.

De forma a permitir uma modelação simplificada da interação do ambiente com o sistema, são definidos dois formalismos de modelação específicos para a modelação de interpretação de sinais de entrada (Modelo de Interpretação de Sinais) e para afetação dos sinais de saída (Modelo de Afetação de Sinais). Estes dois modelos juntos definem-se como o modelo de interface.

A comunicação do sistema global com o ambiente é feita através de sinais, ou seja, a cada passo de execução o conjunto de sinais de entrada adquire os valores das variáveis de ambiente associadas, e um conjunto de sinais de saída são afetados com valores que ficam disponíveis para o ambiente.

A comunicação entre os três módulos é feita através de eventos. Estes eventos são gerados por cada módulo e recebidos no outro módulo.

São utilizados eventos para esta comunicação por duas razões principais. Primeiro, pretende-se que esta comunicação seja o mais simples possível, facilitando a interligação entre os módulos. Os eventos, tendo uma definição simples e apenas dois estados, acontecer ou não acontecer, são uma forma eficiente de partilhar informação de forma simples. Por outro lado, muitos formalismos de modelação com interface aceitam eventos como entrada, e no caso dos formalismos que não têm interface, devido à sua simplicidade já referida, os eventos são a forma mais simples de introduzir essa interface num formalismo. Como se pretende modelar a execução com formalismos existentes, os eventos são a forma mais compatível de interagir com eles.

Os três módulos definidos podem ser vistos como subsistemas de um sistema global ou como sistemas independentes que comunicam entre si. Podem também ser implementados de forma centralizada (todos os módulos na mesma plataforma e com o mesmo relógio), ou de forma distribuída (com diferentes relógios, em diferentes plataformas ou mesmo geograficamente distribuídos).

No caso de uma implementação distribuída, os métodos de comunicação a utilizar necessitam de garantir a semântica pretendida.

No caso de uma implementação centralizada, o sistema global tem uma semântica definida por ciclos de execução. Por cada ciclo de execução, primeiro são adquiridos e interpretados os sinais do ambiente, gerando os eventos de comunicação, em seguida é efetuado um passo de execução que gera eventos posteriormente utilizados para afetar os sinais de saída para o ambiente.

Dessa forma, pode definir-se um ciclo de execução do modelo global, implementado de forma centralizada, como apresentado na Figura 3.2.

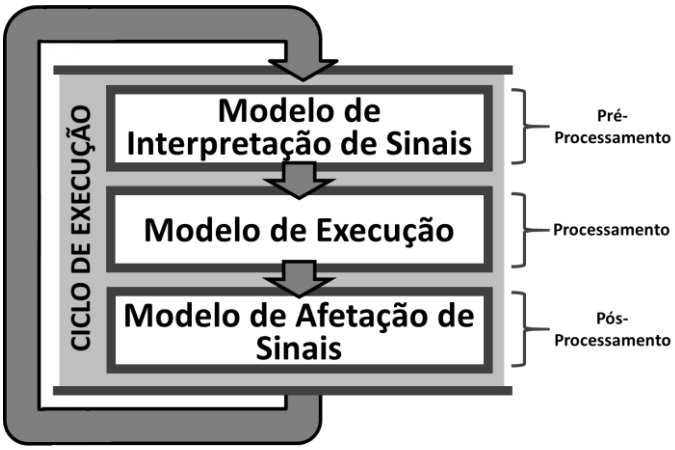


Figura 3.2: Fluxo de execução do modelo global com uma implementação centralizada.

Para uma implementação centralizada onde existe apenas um único relógio igual para todos os módulos, o modelo de interpretação de sinais pode ser definido como pré-processamento do sistema e o modelo de afetação de sinais pode ser definido como pós-processamento do sistema.

Nas seguintes secções são apresentados os modelos de interpretação e afetação de sinais, bem como os conceitos que os permitem definir.

3.2. Estrutura e fluxo da Interface

O modelo de interface é constituído por duas partes: O modelo de interpretação de sinais (MIS) e o modelo de afetação de sinais (MAS).

O Modelo de Interpretação de Sinais (MIS) (em inglês, *Signal Interpretation Model, SIM*) tem como objetivo modelar a aquisição e interpretação da informação que entra no sistema. Para isso recebe sinais do ambiente externo e gera eventos, em relação aos quais o modelo de execução está recetivo.

O Modelo de Afetação de Sinais (MAS) (em inglês, Signal Assignment Model, SAM) tem como objetivo a afetação dos sinais de saída com os comportamentos obtidos da análise da informação recebida do modelo. Para esse efeito, recebe eventos do modelo de execução e gera os novos valores dos sinais para o ambiente externo.

A Figura 3.3 mostra o fluxo da informação no MIS entre o ambiente e o modelo, bem como no MAS entre o modelo e o ambiente.

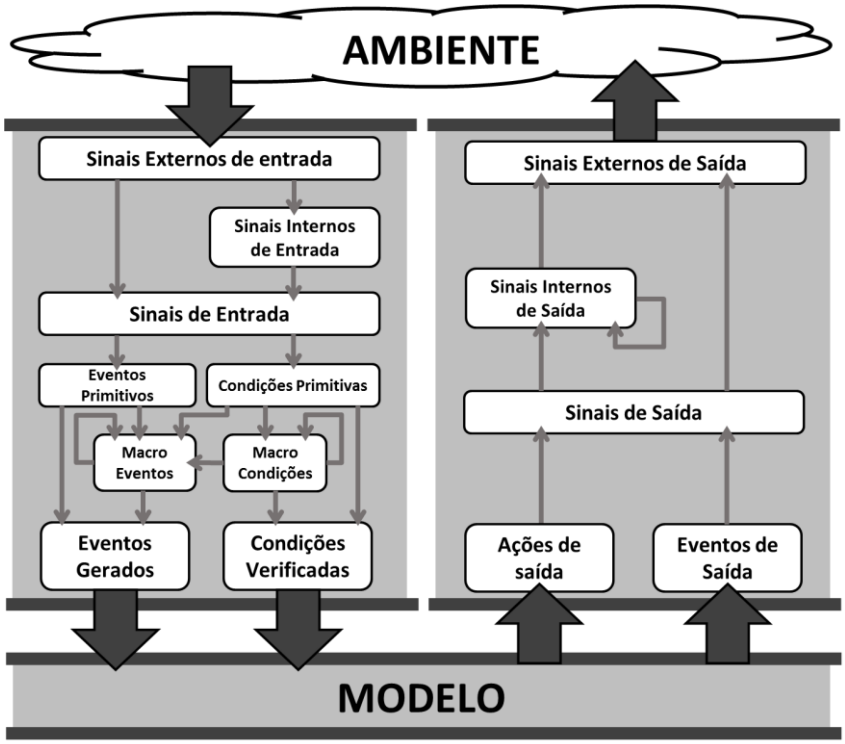


Figura 3.3: Fluxo do Modelo de Interface.

No fluxo de entrada, definido como modelo de interpretação de sinais (MIS), a informação é recebida através de sinais Externos que fazem a ligação com o ambiente. Em seguida, sinais Internos podem ser gerados utilizando a informação proveniente do ambiente para gerar nova informação. O conjunto de toda esta informação (proveniente do ambiente e gerada internamente) é em

seguida analisado de forma a encontrar comportamentos elementares de interesse previamente definidos, tanto através de condições como de eventos. Finalmente esses comportamentos mais simples são analisados conjuntamente de forma a determinar comportamentos mais complexos. Quando os comportamentos desejados são encontrados, o modelo é informado de forma a responder a esse comportamento.

No fluxo de saída, definido como modelo de afetação de sinais (MAS), a informação gerada pelo modelo é recebida e analisada de forma a encontrar comportamentos de interesse. Esses comportamentos permitem a geração de informação que é disponibilizada ao ambiente.

Nas subsecções seguintes serão apresentados e definidos os conceitos constantes no diagrama de fluxo do Modelo de Interface da Figura 3.3, tanto para o MIS como para o MAS.

3.2.1. Sinais Externos

Um sinal é definido em [111] como uma função de valor único de uma ou mais variáveis independentes que contêm alguma informação. Um sinal também pode ser definido como uma quantidade física que varia com o tempo, espaço ou outra qualquer variável independente.

Neste documento, o sinal, além do referido, também é a forma como o sistema guarda a informação proveniente do exterior desse sistema bem como a disponibiliza ao exterior.

Um sinal externo é um sinal que tem comunicação direta com o ambiente e o seu valor está ligado a uma variável do ambiente.

Os sinais externos estão presentes tanto no MIS como no MAS.

Um sinal externo de entrada é um sinal que recebe e armazena a informação proveniente do exterior do sistema. Ele guarda a informação, adquirida ao longo do tempo, necessária para fazer a interpretação do comportamento do sinal do ambiente associado.

Um sinal externo de saída é um sinal que é afetado de forma a disponibilizar para o ambiente a informação proveniente do modelo do sistema. Ele guarda a informação, adquirida ao longo do tempo, gerada através da interpretação do comportamento do modelo.

A representação interna de um sinal externo de entrada é composta pelo valor recebido do ambiente no passo de execução atual e um conjunto de valores previamente recebidos durante uma determinada janela de tempo. Associado a esta janela, vários níveis de diferenças entre os valores são também calculados. Esta lista de valores, como mostrado na Figura 3.4, guarda um histórico de valores do sinal nos últimos passos de execução e a diferença entre eles. Esses valores são definidos por uma estrutura de valores que guardam a informação necessária para a interpretação do sinal (Figura 3.4).

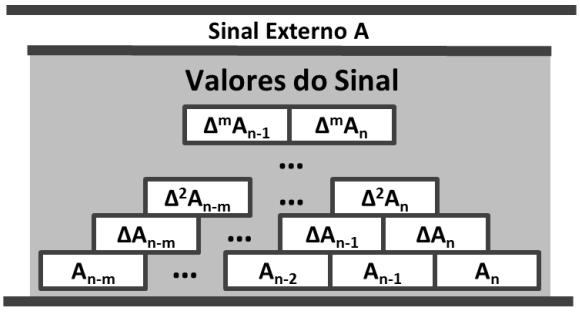


Figura 3.4: Estrutura de representação de um Sinal Externo

Cada Sinal Externo guarda um histórico de valores do sinal. A dimensão deste histórico depende da análise que se pretende realizar. Na Figura 3.4 pode ver-se que é possível armazenar ainda a informação das diferenças delta (Δ) entre valores do sinal. A definição de delta para Δ=1 é dado por:

$$\Delta X_n = \Delta^1 X_n = X_{n-1} - X_n$$

E genericamente para $\Delta > 1$:

$$\Delta^m X_n = \Delta^{m-1} X_{n-1} - \Delta^{m-1} X_n$$

O número de valores a armazenar das diferenças para cada sinal externo depende da análise que se pretenda fazer deste.

A informação do sinal é guardada numa estrutura de duas dimensões (histórico e diferenças do sinal). As dimensões dessa estrutura são dadas pela análise tanto dos sinais internos como dos eventos que o utilizam na sua análise.

O número de deltas a armazenar é dado pelo máximo dos deltas máximos analisados em casa sinal ou evento.

3.2.2. Sinais Internos

Um sinal interno pode representar tipos diferentes de informação. Como exemplos de informação que um sinal interno pode guardar referem-se uma conversão de medidas (ex. metros para polegadas), o valor médio do sinal numa janela temporal, ou valores resultantes de funções mais complexas como as definidas por troços ou o cálculo da FFT de um sinal.

Um sinal interno tem todas as características de um sinal externo. A única diferença prende-se com a aquisição dos valores. Enquanto num sinal externo, a informação é adquirida de uma variável externa, o sinal interno é calculado através de uma função associada (ou conjunto de funções), dependente de outros sinais. Em cada passo de execução o valor do sinal é dado pelo resultado dessa função.

O sinal interno pode depender de um ou mais sinais. Estes sinais de que o sinal interno depende podem ser externos ou outros sinais internos.

Dessa forma, um sinal interno tem a estrutura apresentada na Figura 3.5

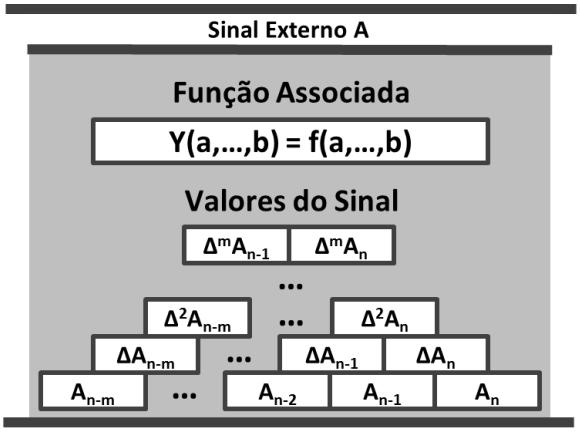


Figura 3.5: Estrutura de representação de um Sinal Interno

Os valores de ordem dos valores definidos de *n* até *n-m*, no caso dos sinais internos, podem não representar o número de passos de execução. O cálculo da FFT de um sinal referido anteriormente é um exemplo desse caso, onde estes valores representam valores de frequências. Embora a representação seja equivalente, cabe ao modelador modelar coerentemente os passos seguintes do modelo de interpretação de sinais, tendo em conta o significado dos valores do sinal.

Os sinais internos estão presentes tanto no MIS (definidos como Sinais internos de entrada) como no MAS (definidos como Sinais Interno de Saída). Os sinais internos de entrada são analisados no passo de execução imediatamente após a aquisição dos sinais externos de entrada.

Tendo em conta que um sinal interno pode depender de outros sinais internos, a análise destes sinais tem de ser feita depois da aquisição dos sinais externos por uma ordem específica, de acordo com a sua estrutura de dependências.

Na Figura 3.6 é apresentado o processo de análise dos sinais internos de entrada, em que o sinal D é uma função dos sinais A e B, enquanto o sinal E é uma função dos sinais B, C e D.

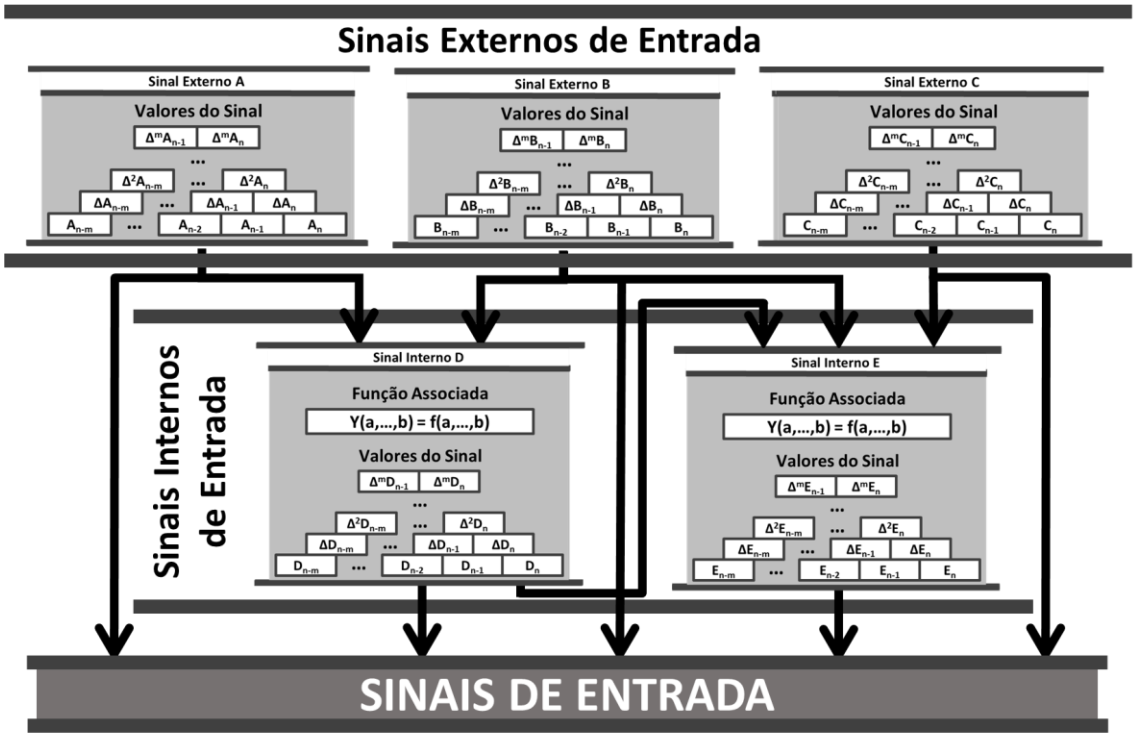


Figura 3.6: Análise de Sinais Internos de entrada

Uma vez recebidos os valores dos sinais externos de entrada, os primeiros sinais internos a ser gerados são os que dependem apenas de sinais externos de entrada. Em seguida são gerados os valores dos sinais que dependam de sinais já analisados. Este processo repete-se até que o valor de todos os sinais esteja gerado. Não são permitidas dependências circulares entre sinais internos.

No final da análise, os sinais internos e externos são disponibilizados para análise e tratamento posteriores de forma indiferenciada. Isto significa que, para as partes seguintes do fluxo não importa se estes sinais vêm do ambiente (externos) ou foram gerados (internos).

A análise dos sinais internos de saída é semelhante à análise de sinais internos de entrada. O fluxo é equivalente ao apresentado na Figura 3.6, onde os sinais externos de entrada são substituídos por sinais de saída e os sinais de entrada são substituídos por sinais externos de saída.

No MAS, após serem analisados todos os eventos e executadas todas as ações, um conjunto de sinais de saída fica disponível. Estes sinais são fornecidos para o ambiente, mas também são utilizados na análise dos sinais internos de saída.

Assim, a análise dos sinais internos de saída é realizada nos passos finais do MAS imediatamente antes da última etapa na qual os sinais externos de saída estão disponíveis para o ambiente. As etapas de análise entre os sinais de saída e sinais externos de saída são semelhantes àquelas apresentadas anteriormente entre os sinais externos de entrada e sinais de entrada.

A estrutura de representação de um sinal interno é calculada exatamente da mesma forma que a apresentada para o cálculo da estrutura de representação de um sinal externo.

O sinal interno é analisado na definição das dimensões da estrutura de cada sinal presente na sua função. Para cada sinal que constitui a função do sinal interno é necessário analisar o valor mais antigo desse sinal a ser analisado no sinal interno, bem como a maior diferença.

Considere-se como exemplo, um sinal interno Y com a seguinte função associada:

$$\mathbf{Y_n = \Delta^4 X_n + X_{n-5}}$$

O número de linhas (*nl*) da estrutura do sinal X é dado pela adição de um ao valor do delta máximo, neste caso:

$$\mathbf{\Delta^{max} = 4, \text{ logo } nl = 4+1 = \Delta^{max} + 1 = 5}$$

Quanto ao número de colunas (*nc*) é dado pela adição de um ao valor máximo de histórico da equação, neste caso X_{n-5}, logo:

$$\mathbf{n_{Máx} = 5, logo\ nc = n_{Máx} + 1 = 5+1 = 6}$$

3.2.3. Condições Primitivas

Uma condição é uma análise do sinal num determinado instante. Na sua forma mais elementar é definida como uma análise do estado do valor atual do sinal. É dado pela seguinte expressão geral:

$$\mathbf{\Delta^{DELTA}\ SINAL\ OPER\ VALOR}$$

DELTA (Δ) é o nível de diferenças do sinal a analisar.

SINAL é o sinal (externo ou interno) associado à condição. Este é o sinal que se pretende analisar.

OPER define o tipo de análise que se pretende fazer ao sinal associado.

Finalmente, o VALOR define o nível com o qual o sinal vai ser comparado. Na Tabela 3.1 este valor é definido por uma constante k.

Tabela 3.1 – Tipos de Condições Primitivas

	Δ^0	Δ^1		Δ^m
>	$X_n > k$	$\Delta X_n > k$	...	$\Delta^m X_n > k$
≥	$X_n \geq k$	$\Delta X_n \geq k$	...	$\Delta^m X_n \geq k$
<	$X_n < k$	$\Delta X_n < k$	...	$\Delta^m X_n < k$
≤	$X_n \leq k$	$\Delta X_n \leq k$	...	$\Delta^m X_n \leq k$
=	$X_n = k$	$\Delta X_n = k$	...	$\Delta^m X_n = k$
≠	$X_n \neq k$	$\Delta X_n \neq k$	...	$\Delta^m X_n \neq k$

São definidos seis tipos de condições associados a seis tipos de operação. A Tabela 3.1 apresenta os seis tipos de condições primitivas definidos, bem como as equações tipo para os deltas 0, 1 e genérico.

3.2.4. Eventos Primitivos

Um evento é um acontecimento que ocorre quando um determinado comportamento se verifica. Na sua forma mais elementar, um evento ocorre quando o sinal associado se cruza com uma outra função definida com um comportamento especificado (Figura 3.7).

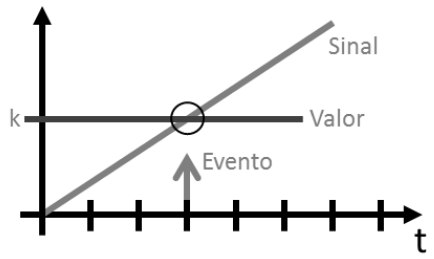


Figura 3.7: Análise de um evento primitivo.

Um evento é analisado em, pelo menos, dois passos de execução onde tem de obedecer a duas condições diferentes e que são disjuntas (pré-condição e pós-condição). Caso ambas as condições se verifiquem no tempo de análise em questão, o evento ocorre. Devido a esta definição, pode dizer-se que um evento não pode ocorrer em dois passos de execução consecutivos pois, depois de ter obedecido uma pré-condição e em seguida uma pós-condição, para que ocorra novo evento, o sinal associado tem de obedecer novamente a uma pré-condição para que depois possa voltar a ocorrer. Como as condições são disjuntas não pode haver sobreposição da primeira pós-condição com a segunda pré-

condição, logo é necessário pelo menos um tempo de análise entre duas ocorrências do mesmo evento sem que o evento ocorra.

Um evento primitivo define-se como um evento que não pode ser definido através de composição de outros eventos do mesmo tipo. São definidos dois tipos de eventos primitivos: Eventos primitivos abertos e eventos primitivos fechados.

A principal diferença entre os dois tipos de eventos primitivos está nos intervalos de valores associados à pré- e à pós-condição.

Num evento primitivo fechado, a união do intervalo de valores que satisfazem a pré-condição com o intervalo de valores que satisfazem a pós-condição resulta sempre no intervalo]-∞,+∞[, ou seja, abrange todos os valores que o sinal pode ter.

Por exemplo, para um evento num sinal X com a pré-condição:

$$\mathbf{X_{n-1} > 5}$$

Que é válida no intervalo:

$$\mathbf{]5,+\infty[}$$

E com a pós-condição:

$$\mathbf{X_n \leq 5}$$

Que é válida no intervalo:

$$\mathbf{]-\infty,5]}$$

Tem-se que:

$$\mathbf{]5,+\infty[\cup]-\infty,5] =]-\infty,+\infty[}$$

Tendo isso em conta, verifica-se que em qualquer evento primitivo fechado, se um sinal não satisfaz uma das condições, obrigatoriamente satisfaz a ou-

tra condição. Dessa forma é possível analisar este tipo de eventos analisando apenas uma das condições. Isto é, é necessário analisar dois passos de execução do sistema, mas só é necessário verificar uma das condições (exemplos: a pré-condição é verdadeira em n-1 e não é verdadeira em n, ou a pós-condição não é verdadeira em n-1 e passa a ser verdadeira em n).

Por outro lado, nos eventos primitivos abertos, a união dos intervalos que satisfazem as duas condições nunca resulta no intervalo]-∞,+∞[. Isto significa que existem valores que o sinal pode adquirir que não satisfazem nenhuma das condições.

Por exemplo, para um evento num sinal X com a pré-condição:

$$\mathbf{X_{n-1} > 5}$$

Que é válida no intervalo:

$$\mathbf{]5,+\infty[}$$

E com a pós-condição:

$$\mathbf{X_n = 5}$$

Que é válida apenas para o valor:

$$\mathbf{X = 5}$$

Tem-se que:

$$\mathbf{]5,+\infty[\cup \{5\} = [5,+\infty[}$$

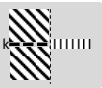
Isto leva a outra diferença. Ao contrário do que acontece nos eventos primitivos fechados, nos eventos primitivos abertos, um evento pode deixar de estar habilitado a disparar sem ter disparado. Isto é, existe um conjunto de valo-

res que não satisfaz a pré-condição, logo o evento não está habilitado, mas também não satisfaz a pós-condição, logo o evento não pode ter ocorrido.

Um evento primitivo aberto não pode ser obtido por composição de nenhum outro evento e um evento primitivo fechado não pode ser composto por qualquer outro evento exceto por eventos primitivos abertos.

São definidos doze tipos de eventos primitivos (6 abertos e 6 fechados). Estes doze tipos de eventos, tal como as suas condições para os diferentes deltas e os intervalos válidos para as condições são apresentados na Tabela 3.2.

Tabela 3.2 – Tipos de eventos primitivos

EVENTOS PRIMITIVOS ABERTOS					EVENTOS PRIMITIVOS FECHADOS				
	Δ^0	Δ^1	Δ^m			Δ^0	Δ^1	Δ^m	
$<$ to $>$	$X_{n-1} < k \wedge X_n > k$	$\Delta X_{n-1} < k \wedge \Delta X_n > k$	... $\Delta^m X_{n-1} < k \wedge \Delta^m X_n > k$		$\leq$ to $>$	$X_{n-1} \leq k \wedge X_n > k$	$\Delta X_{n-1} \leq k \wedge \Delta X_n > k$	... $\Delta^m X_{n-1} \leq k \wedge \Delta^m X_n > k$	
$>$ to $<$	$X_{n-1} > k \wedge X_n < k$	$\Delta X_{n-1} > k \wedge \Delta X_n < k$	... $\Delta^m X_{n-1} > k \wedge \Delta^m X_n < k$		$<$ to $\geq$	$X_{n-1} < k \wedge X_n \geq k$	$\Delta X_{n-1} < k \wedge \Delta X_n \geq k$	... $\Delta^m X_{n-1} < k \wedge \Delta^m X_n \geq k$	
$>$ to $=$	$X_{n-1} > k \wedge X_n = k$	$\Delta X_{n-1} > k \wedge \Delta X_n = k$	... $\Delta^m X_{n-1} > k \wedge \Delta^m X_n = k$		$\geq$ to $<$	$X_{n-1} \geq k \wedge X_n < k$	$\Delta X_{n-1} \geq k \wedge \Delta X_n < k$	... $\Delta^m X_{n-1} \geq k \wedge \Delta^m X_n < k$	
$<$ to $=$	$X_{n-1} < k \wedge X_n = k$	$\Delta X_{n-1} < k \wedge \Delta X_n = k$	... $\Delta^m X_{n-1} < k \wedge \Delta^m X_n = k$		$>$ to $\leq$	$X_{n-1} > k \wedge X_n \leq k$	$\Delta X_{n-1} > k \wedge \Delta X_n \leq k$	... $\Delta^m X_{n-1} > k \wedge \Delta^m X_n \leq k$	
$=$ to $>$	$X_{n-1} = k \wedge X_n > k$	$\Delta X_{n-1} = k \wedge \Delta X_n > k$	... $\Delta^m X_{n-1} = k \wedge \Delta^m X_n > k$		$\neq$ to $=$	$X_{n-1} \neq k \wedge X_n = k$	$\Delta X_{n-1} \neq k \wedge \Delta X_n = k$	... $\Delta^m X_{n-1} \neq k \wedge \Delta^m X_n = k$	
$=$ to $<$	$X_{n-1} = k \wedge X_n < k$	$\Delta X_{n-1} = k \wedge \Delta X_n < k$	... $\Delta^m X_{n-1} = k \wedge \Delta^m X_n < k$		$=$ to $\neq$	$X_{n-1} = k \wedge X_n \neq k$	$\Delta X_{n-1} = k \wedge \Delta X_n \neq k$	... $\Delta^m X_{n-1} = k \wedge \Delta^m X_n \neq k$	

Os eventos analisados em diferentes deltas analisam mais que dois valores do sinal. Por exemplo, pela definição anterior temos que, $\Delta X_n = X_{n-1} - X_n$ e $\Delta X_{n-1} = X_{n-2} - X_{n-1}$. Logo um evento em $\Delta^2 X$ analisa os valores do sinal em três instantes (X_{n-2}, X_{n-1} e X_n). Da mesma forma, analisando as condições de um evento em $\Delta^3 X$ tem-se que $\Delta^2 X_n = \Delta X_{n-1} - \Delta X_n = (X_{n-2} - X_{n-1}) - (X_{n-1} - X_n)$ e $\Delta^2 X_{n-1} = \Delta X_{n-2} - \Delta X_{n-1} = (X_{n-3} - X_{n-2}) - (X_{n-2} - X_{n-1})$. Então, para $\Delta^3 X$ são analisados quatro valores do sinal (X_{n-3}, X_{n-2}, X_{n-1} e X_n).

Como referido na apresentação dos sinais (externos e internos), é necessário analisar todos os eventos primitivos associados a um determinado sinal para poder calcular as dimensões da estrutura de representação desse sinal.

3.2.4.1. Eventos Atrasados

Até agora, todos os eventos foram apresentados como análise de variações no sinal entre dois passos consecutivos de análise (X_n e X_{n-1}). No entanto, é possível definir todos estes acontecimentos através de uma análise baseada em dois instantes distintos (não consecutivos) separados por *p* etapas de análise (em vez de uma).

Neste sentido, os eventos atrasados (*delayed*) não são um novo tipo de eventos, mas sim uma característica que pode ser alterada em qualquer tipo de eventos primitivos. Com a adição desta característica, a análise do evento é feita tendo em consideração os valores (X_n e X_{n-p}). A análise realizada anteriormente é feita considerando-se dois passos consecutivos como um caso particular em que o valor de *p* é *p = 1*.

Na Figura 3.8 é representada a forma de analisar um evento com atraso.

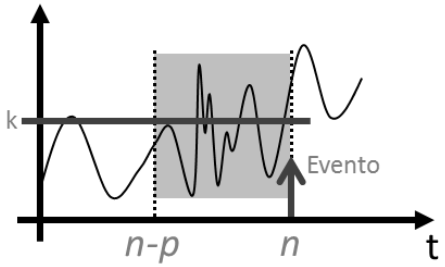


Figura 3.8: Análise de um evento atrasado (*delayed*).

Para analisar este evento é criada uma janela de tamanho p e o sinal dentro desta janela não é utilizada para a análise do evento.

Para cada evento primitivo, o número máximo de valores que deve ser armazenado na linha da estrutura que guarda os valores referentes a delta zero ($d_{Máx}$) do sinal associado, depende de duas variáveis: de delta (Δ) e do atraso do evento (p), que será definido mais à frente neste documento. O valor máximo é dado pelo resultado da equação:

$$d_{Máx} = \Delta + p + 1$$

Por exemplo, para um evento que analisa o segundo delta do sinal (Δ^2) e tem um atraso de três passos de execução, o sinal necessita de armazenar:

$$d_{Máx} = 2 + 3 + 1 = 6$$

Ou seja, são armazenados os valores de A_n a A_{n-5}.

3.2.5. Macro-Eventos

Os eventos primitivos apresentados permitem definir os eventos que são gerados quando um conjunto de comportamentos elementares ocorre. Utilizados em conjunto com um formalismo de modelação, esses eventos ainda permitem definir composição de comportamentos mais complexos usando vários eventos primitivos e modelação multi-estado. No entanto, esta abordagem aumenta a complexidade dos modelos gerados.

Apesar disso, esta análise permite perceber que é possível definir comportamentos mais elaborados e complexos por composição de comportamentos elementares.

Em [29], é apresentada uma composição de eventos definidos através de sequências de eventos primitivos. Estes são denominados de macro-eventos.

Para esta composição é necessário definir os eventos primitivos que compõem a sequência e a sua ordem.

O macro-evento ocorre no mesmo passo de execução do último evento primitivo da sequência.

Considerando-se o macro-evento *EvX*, o qual ocorre quando o evento *EvC* ocorre precedido pelo evento *EvB* e este precedido pelo evento de *EvA*, ele pode ser definido da seguinte forma:

$$\boldsymbol{EvX = EvA\ ;\ EvB\ ;\ EvC}$$

Esta definição encapsula a sequência de eventos ilustrada no diagrama temporal da Figura 3.9, onde $p < s$. Neste exemplo, o evento *EvX* ocorre quando se detetar um evento *EvC* precedido por um evento *EvB*, por sua vez precedido por um evento *EvA*.

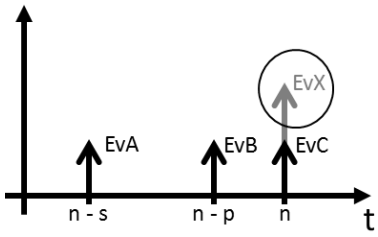


Figura 3.9: Diagrama temporal do macro-evento *EvX*.

Depois de definido, um macro-evento pode ser utilizado como parte da sequência de outros macro-eventos. Esta introdução de uma estruturação hierárquica de eventos permite uma maior compressão da definição dos eventos.

Em seguida os macro-eventos são estendidos para que permitam a análise de mais tipos de comportamento.

Garantindo a propriedade associativa na composição, é possível definir a composição de um macro-evento composto por *r* eventos através da composição entre dois eventos. Isto é, um macro-evento composto por três eventos primitivos pode ser definido por uma composição entre um evento primitivo e outro evento resultante da composição dos outros dois eventos primitivos.

Na definição dos tipos de composição entre dois eventos, definem-se três análises de composição diferentes: análise temporal, a análise da relação com o sinal associado, e a análise do tipo de operação de álgebra booleana.

Em termos de álgebra booleana, é possível definir 16 operações diferentes envolvendo dois operandos de eventos. Estas 16 operações são apresentadas na Tabela 3.3. No entanto, no caso da composição de eventos definida neste trabalho, nem todos eles são válidos.

Tabela 3.3 – Operações Booleanas

		X								X	X	X	X	X	X	X	X
Ev A	**Ev B**	**0**	AND	$\overline{A}$ B	A	A $\overline{B}$	B	XOR	OR	NOR	XNOR	$\overline{B}$	→	$\overline{A}$	$\overline{A}$+B	NAND	1
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

Define-se que, para um macro-evento ocorrer, pelo menos um dos eventos primitivos que a define tem de ocorrer. Esta regra permite prevenir a definição de eventos com ocorrências sucessivas sem que o valor do sinal associado se altere.

Assim, todas as operações que resultam num macro-evento quando nenhum dos dois eventos que a compõem ocorre podem ser eliminadas. Estas operações são as oito no lado direito da Tabela 3.3. Estão assinaladas com um **X**.

Além disso, a operação que resulta num macro-evento quando nenhum dos eventos ocorre pode ser eliminada, esta operação está representada por "0" na Tabela 3.3. As operações que resultam num macro-evento quando há ocorrência de um único evento primitivo podem ainda ser eliminadas, pois não representam uma composição. Estas são representadas por "A" e "B" na Tabela 3.3.

Quanto às operações $\overline{A}\,B$ e $A\,\overline{B}$, definindo o não evento como ocorrendo em todos os passos de execução em que o evento afetado não ocorre, é possível defini-las tendo em conta a Operação AND e o não evento.

Resumindo estas considerações, conclui-se que, para representar todos os tipos de composição válidos de eventos só é necessário considerar três operações booleanas. Estas operações são AND, OR e XOR,

Em termos de análise temporal, são definidos dois tipos de composição: quando os eventos ocorrem no mesmo passo de execução, e quando os eventos ocorrem em diferentes intervalos de tempo de análise, ou seja, sequencialmente.

Na Figura 3.10 são apresentadas as diferentes formas como dois eventos podem ser relacionados entre si. No eixo de tempo (*t*) estão marcados os diferentes passos de execução, que correspondem aos instantes em que os eventos ocorrerem (a cinzento) e os instantes em que são observados pelo sistema (a preto). Esta abordagem está em linha com a semântica de execução *cycle accurate*.

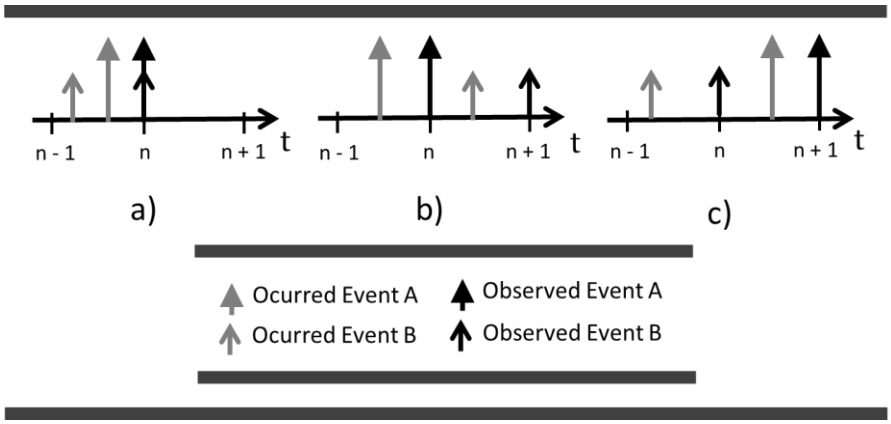


Figura 3.10: Formas como os eventos se podem relacionar

Como apresentado na Figura 3.10 a), dois eventos que ocorrem durante o mesmo intervalo de tempo, ou seja, o tempo entre dois passos de execução, são detetados ao mesmo tempo, como se tivessem ocorrido no mesmo instante. Neste caso, a ordem pela qual ocorrem estes eventos não é relevante e não é possível determinar.

Dois eventos (A e B) que ocorrem em diferentes intervalos de tempo podem ser definidos de duas formas distintas: primeiro ocorre o evento A e, em seguida, o evento B (Figura 3.10 b)) ou primeiro ocorre o evento B e, em seguida, o evento A (Figura 3.10 c)).

Pretende-se também, gerar um evento que ocorre quando os dois eventos ocorrem sequencialmente (em diferentes intervalos de tempo), mas a ordem na qual eles ocorrem não é relevante. Este caso pode ser definido por um OR dos dois casos anteriores.

A ocorrência de dois eventos, independentemente de tempo ou ordem em que eles acontecem também é interessante para definir. Isso significa que têm de ocorrer os dois eventos independentemente de quando ocorre cada um. Este caso pode ser definido por uma composição OR dos três casos da Figura 3.10.

Em termos de análise do sinal associado, os dois eventos podem estar associados ao mesmo sinal, ou associados a sinais diferentes.

Dois eventos associados ao mesmo sinal analisam um comportamento composto de um único sinal. Neste caso, os eventos podem analisar a mesma variação (Δ) do sinal ou diferentes variações do sinal, considerando-se ou não os atrasos, tal como apresentado anteriormente. Quando os dois eventos estão associados ao mesmo sinal na mesma variação (Δ), compondo dois eventos com condições prévias disjuntos irá conduzir a um evento que nunca ocorre (este representa um caso de má modelação).

A combinação da análise destas três variáveis produz doze tipos de combinações possíveis entre dois eventos.

Na Figura 3.11 são apresentadas em forma de um cubo todas as combinações possíveis das três variáveis definidas.

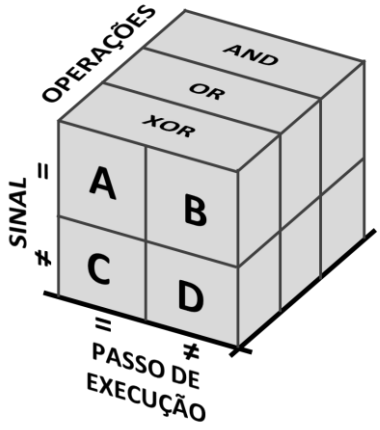


Figura 3.11: Tipos de composição entre dois eventos

Para cada operação, estão definidos quatro tipos de composição, tendo em conta o sinal e o passo de execução associado. Essas Composições estão repre-

sentadas na Figura 3.11 como Composição A, B, C e D e têm o seguinte significado:

Na **composição A** é definida a composição entre dois eventos associados ao mesmo sinal no mesmo passo de execução. Isto significa que se estão a compor dois comportamentos que ocorreram no mesmo sinal detetados ao mesmo tempo.

Um exemplo simples deste tipo de composição é a deteção de um evento que ocorre quando o sinal passa de menor que um valor k1 para maior que outro valor k2. Neste caso tem de se compor um evento que ocorre quando o sinal passa a ser maior que k1 com um evento que ocorre quando o sinal passa a ser maior que k2.

Na **composição B** é definida a composição entre dois eventos associados ao mesmo sinal em passo de execução diferentes. Isto significa que se estão a compor dois comportamentos do mesmo sinal que são detetados um a seguir ao outro (sequencialmente).

Um exemplo simples deste tipo de composição é a deteção da mudança de direção da evolução de um sinal, ou seja, detetar que um sinal estava a crescer e passou a decrescer. Para isso tem de se compor um evento que ocorre quando as diferenças do sinal passam a ser maior que 1, com outro que ocorre quando as diferenças do sinal passam a ser menor que 1.

Na **composição C** é definida a composição entre dois eventos associados a sinais diferentes no mesmo passo de execução. Isto significa que se está a compor um comportamento de um sinal com um comportamento de outro sinal, onde ambos são detetados ao mesmo tempo.

Um exemplo simples deste tipo de composição é a deteção de um comportamento similar em dois sinais. Por exemplo, para dois sinais booleanos detetar que esses dois sinais passaram de zero para um no mesmo passo de execução.

Na **composição D** é definida a composição entre dois eventos associados a sinais diferentes em passos de execução diferentes. Isto significa que se está a compor um comportamento de um sinal com um comportamento de outro sinal, e estes são detetados um a seguir ao outro (sequencialmente).

Um exemplo simples deste tipo de composição é a deteção do sentido de um veículo numa estrada através de dois sensores. Se o evento de ativação do sensor A ocorre primeiro e em seguida o evento do sensor B o veículo vai numa direção, no caso do evento do sensor B ocorrer primeiro e depois o evento do sensor A, o veículo vai na direção oposta.

Além da definição destes tipos de composição, a adição de características aos eventos permite também melhorar a composição e permitem a definição de comportamentos mais complexos de uma forma mais simples.

Estas características não são apenas aplicáveis à composição de dois eventos (primitivos ou macro). Elas também são válidas para o caso da composição de um evento com uma condição, e para a composição de duas condições.

A composição de um evento com uma condição, tal como a composição de dois eventos, resulta na geração de um macro-evento. A composição de duas condições resulta na geração de uma macro-condição.

No caso da composição de um evento com uma condição, apenas as composições A e C são válidas. Isto deve-se ao facto de uma composição não ter um tempo de ocorrência como o evento. Dessa forma, num macro-evento, cada condição tem de estar associada ao tempo de análise de um dos eventos que compõe o macro-evento.

As características de interesse adicionadas aos eventos que são definidas são o não evento, o tempo de vida de um evento e a exclusividade de um evento. Estas características são apresentadas nas subsecções seguintes.

3.2.5.1. Não Evento

Um **NÃO evento** (já referido anteriormente) é definido como a negação de um evento e ocorre em cada passo de execução em que o evento, afetado com a característica NÃO, não ocorre. Na Figura 3.12 é representado um diagrama temporal com a ocorrência de dois eventos *EvX* e o valor de $\overline{EvX}$ para todos os passos de execução.

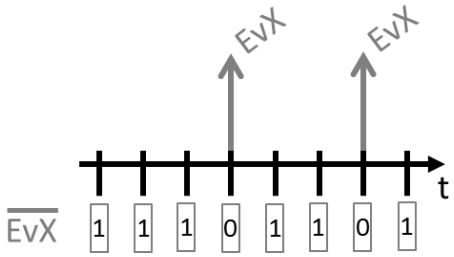


Figura 3.12: Valor de um não evento.

Um NÃO evento, não pode ser definido como um evento primitivo porque não reflete as características definidas para esse tipo de eventos, mais especificamente, a característica que diz que um evento não pode ocorrer em intervalos de tempo consecutivos. Um Não evento é definido como uma característica adicionada a um dos eventos primitivos definidos no macro-evento.

Um Não evento, apesar de estar associado a um evento, possui todas as características de uma condição. Dessa forma pode ser definido como uma condição, logo deve ser utilizado na composição como uma condição.

3.2.5.2. *Tempo de Vida de um Evento*

A representação de um macro-evento através da composição apresentada permite definir a ordem pela qual os eventos ocorrem, sequencialmente ou no mesmo passo de execução. Esta composição permite definir comportamentos onde existe uma ordem entre os constituintes do macro-evento. Esta composição assume que quando um evento da composição ocorre, a análise passa para um estado seguinte do qual não pode retornar.

Esta definição não permite representar o tempo ou distância temporal entre os eventos da composição. A Figura 3.13 mostra o diagrama temporal da Figura 3.9 com a explicitação das distâncias entre eventos.

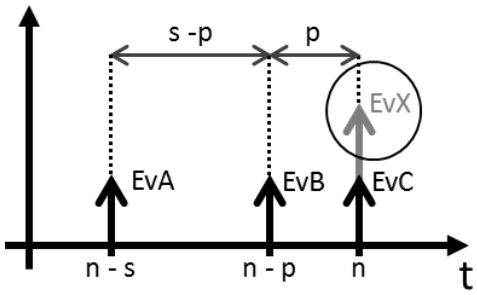


Figura 3.13: Diagrama temporal do macro-evento *EvX* com distâncias entre eventos.

Qualquer valor de s e p representados na Figura 3.13 no intervalo [0,+∞[, com $s > p$, resulta na geração do evento *EvX*.

No entanto, esta caracterização não permite definir todos os comportamentos de interesse, nomeadamente aqueles onde a especificação do tempo entre os dois eventos é importante.

A aquisição do duplo clique de um rato é um exemplo desse tipo de comportamentos. O evento de duplo clique só é gerado se o segundo evento de cli-

que ocorrer dentro de um tempo limitado depois de o primeiro evento ter acontecido. Caso o segundo evento não ocorra nesse tempo, o primeiro evento deixa de ser válido para essa sequência.

A introdução do conceito de tempo de vida associado a um evento vem suprimir esta lacuna, permitindo definir o intervalo de tempo onde um determinado evento é válido.

O tempo de vida de um evento define-se como a duração de um intervalo de tempo durante o qual o efeito de um evento é válido.

Este conceito pode ser relacionado com o conceito de macro-evento. A mesma ocorrência de um evento pode ter tempos de vida diferentes na definição de diferentes macro-eventos. Uma ocorrência do evento pode já não ser válida num determinado macro-evento e ainda estar válida para outro ou outros macro-eventos.

De forma a permitir uma definição mais alargada de comportamentos, são definidos dois tipos de Tempo de Vida: Tempo de Vida Absoluto e Tempo de Vida Relativo.

Tempo de Vida Relativo

O **Tempo de Vida Relativo** define o tempo de vida de um evento constituinte do macro-evento. Está associado a cada evento da composição e define o tempo máximo e mínimo a partir da sua ocorrência entre os quais o próximo evento tem de ocorrer para que a sua ocorrência seja válida para o macro-evento analisado.

Este tipo de tempo de vida é definido por dois valores inteiros, que definem os valores máximo e mínimo no intervalo de tempo entre os quais o evento é válido, ou seja, entre os quais o seguinte evento na composição do macro-

evento deve ocorrer. Se o evento seguinte não ocorre entre os valores mínimo e máximo de tempo de vida relativo do evento, a análise do macro-evento volta para o último estado onde exista ainda um evento válido. Este estado pode estar associado a um evento de tempo de vida ilimitado ou a um evento de tempo de vida limitado que ainda se encontra válido; caso contrário, a análise do macro-evento deve regressar ao estado inicial.

Considere-se um macro-evento *EvX* que deve ser gerado quando o evento *EvB* ocorre entre 3 e 7 passos de execução depois do evento *EvA*.

A Figura 3.14 representa três exemplos onde o evento *EvA* e *EvB* ocorrem com distâncias diferentes entre eles.

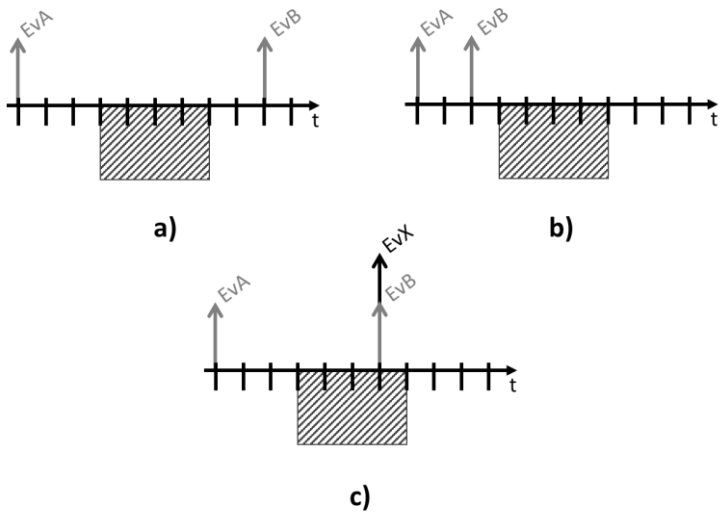


Figura 3.14: Exemplos de Tempo de Vida Relativo. a) Não ocorrência por exceder o tempo de vida. b) Não ocorrência por não atingir o tempo de vida. c) Ocorrência do evento.

Na Figura 3.14 a) o evento *EvX* não ocorre porque o evento *EvB* ocorre depois do tempo de vida relativo do evento *EvA* ter terminado, pois o evento *EvB* ocorre nove passos de execução depois do evento *EvA* e o tempo de vida do

evento *EvA* está definido entre três e sete passos de execução depois da ocorrência do evento.

Na Figura 3.14 b) o evento *EvX* não ocorre porque o evento *EvB* ocorre antes do tempo de vida relativo do evento *EvA* ter começado, pois o evento *EvB* ocorre dois passos de execução depois do evento *EvA*, logo antes do valor mínimo do tempo de vida relativo que é três.

Na Figura 3.14 c) o evento *EvX* ocorre porque o evento *EvB* ocorre dentro do tempo de vida relativo do evento *EvA*, pois o evento *EvB* ocorre seis passos de execução depois do evento *EvA*, logo dentro do seu intervalo de tempo de vida relativo que está definido entre três e sete.

Definindo um macro-evento para os casos em que ocorrem vários eventos no mesmo passo de execução pode pensar-se no macro-evento como uma sequência de análises de eventos.

Dessa forma, a análise de um macro-evento pode ser dividida em estados de análise, sendo casa estado a ocorrência de um evento (ou conjunto de eventos no mesmo tempo) da sequência.

Como já foi referido anteriormente, quando um evento expira, o macro-evento passa para o estado anterior mais avançado que se encontra ainda válido.

A definição deste comportamento permite o aparecimento de diferentes comportamentos quando um evento expira, sendo necessário analisar o histórico de eventos regressivamente até encontrar um evento que ainda se encontre válido.

Na Figura 3.15 são apresentados alguns exemplos que explicam o funcionamento da regra que define em que estado fica um macro-evento quando um evento expira.

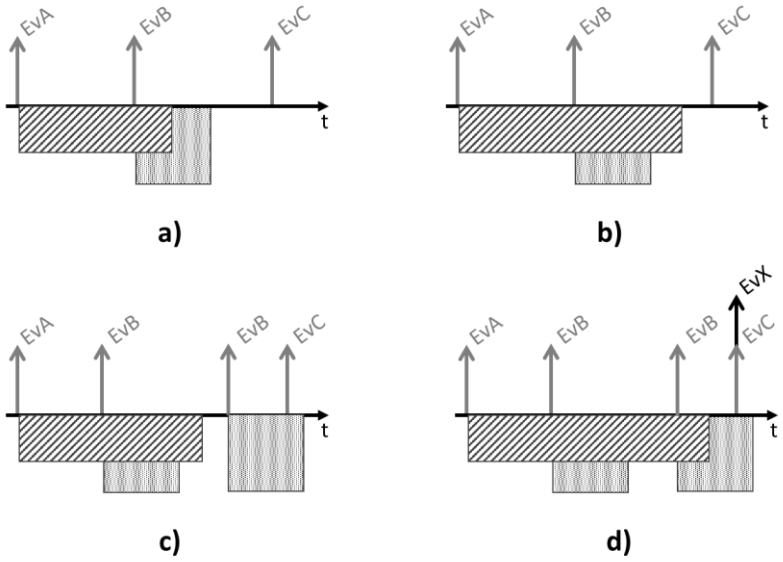


Figura 3.15: Regra de expiração de um evento relativo. a) Evento *EvB* expira quando evento *EvA* já expirou. b) Evento *EvB* expira quando evento *EvA* ainda não expirou. c) Evento *EvB* expira e ocorre novo evento *EvB* depois de *EvA* ter expirado. d) Evento *EvB* expira, mas ocorre segundo evento *EvB* ainda antes do evento *EvA* expirar e sequência continua.

Na Figura 3.15 a), o tempo de vida do evento *EvB* termina sem o evento seguinte *EvC* ocorrer. Nesse momento o tempo de vida do evento *EvA* já não é válido, portanto, a análise do macro-evento deve voltar ao estado de avaliação inicial.

Na Figura 3.15 b), o tempo de vida do evento *EvB* também termina sem a ocorrência do evento seguinte. Neste caso, o tempo de vida do evento *EvA* ainda é válido. Isto significa que o evento pode retornar à etapa onde *EvA* ocorreu e, se houver um novo evento *EvB* dentro do tempo de vida de *EvA*, a sequência continua neste passo. Isto é o que acontece na Figura 3.15 d), em que um segundo evento *EvB* ocorre antes do tempo de vida de *EvA* expirar. Neste caso, a sequência continua e o macro-evento ocorre. Do mesmo modo, na Figura 3.15 c)

um segundo evento *EvB* ocorre já depois do tempo de vida do evento *EvA* ter expirado. Neste caso, o segundo evento *EvB* já não é valido para o macro-evento, logo apesar de ocorrer um evento *EvC* no tempo do último *EvB*, o evento *EvX* não ocorre.

Tempo de Vida Absoluto

O tempo de vida absoluto, da mesma forma que o tempo de vida relativo, define uma janela de ocorrência com dois valores de tempo. Estes valores representam o valor máximo e mínimo de tempo durante o qual a sequência completa do macro-evento deve ocorrer, isto é, o tempo máximo (ou mínimo) entre a ocorrência do primeiro e do último evento do macro-evento. Ele é definido por um par de valores associados ao macro-evento.

Para a análise do tempo de vida absoluto, o tempo que decorre entre dois eventos do macro-evento não tem relevância. Para esta análise, todos os eventos apenas têm de ocorrer dentro do tempo definido, apenas interessando a duração total do macro-evento, calculada desde a ocorrência do primeiro evento até à ocorrência do último evento.

Como exemplo, define-se o evento *EvX* definido anteriormente, que ocorre quando se verifica a sequência *EvA*, em seguida *EvB* e depois *EvC*, com um intervalo de tempo de vida absoluto entre zero e oito passos de execução.

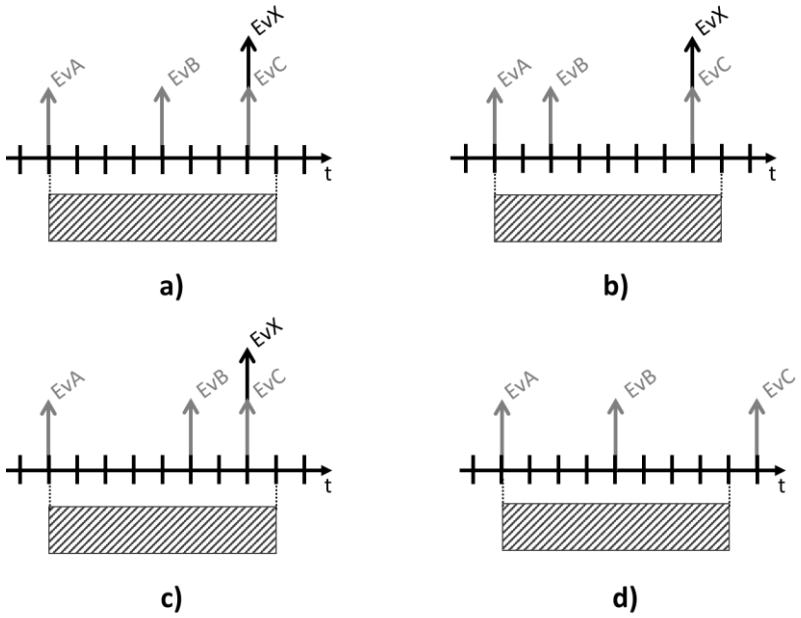


Figura 3.16: Exemplos de Tempo de Vida Absoluto. a) Os três eventos ocorrem dentro do tempo de vida absoluto com espaçamentos semelhantes entre eles. b) Os três eventos ocorrem dentro do tempo de vida absoluto com um espaçamento menor entre os dois primeiros eventos. c) Os três eventos ocorrem dentro do tempo de vida absoluto com um espaçamento menor entre os dois últimos eventos. d) Os três eventos não ocorrem todos dentro do tempo de vida absoluto.

Apesar de ocorrerem com distâncias diferentes entre si, a Figura 3.16 a), b) e c), todos os eventos ocorrem dentro do tempo de vida absoluto definido. Nestes três casos ocorre o macro-evento *EvX*. No caso apresentado na Figura 3.16 d), o evento *EvC* já ocorre fora do tempo de vida absoluto definido, logo, o macro-evento *EvX* não ocorre.

Um exemplo de utilização do conceito de tempo de vida absoluto é a deteção da colocação do código PIN numa caixa multibanco. Neste caso, é definido um tempo especifico para que o utilizador coloque os quatro dígitos do PIN. Se o utilizador não colocar os quatro dígitos dentro do tempo definido, um evento de expiração ocorre. Não interessa se o utilizador coloca os primeiros dígitos

mais rapidamente e os últimos mais lentamente ou vice-versa. Apenas os quatro dígitos têm de ser colocados dentro do tempo especificado.

3.2.5.3. Evento Exclusivo

A exclusividade de um evento define-se como uma característica a adicionar ao evento que relaciona a sua ocorrência com os outros eventos definidos para o mesmo sistema.

Um evento exclusivo é um evento que para ocorrer não necessita apenas que as suas condições sejam satisfeitas. Caso essas condições sejam satisfeitas, para que um evento exclusivo ocorra é necessário verificar que os outros eventos não ocorreram.

São definidos três níveis diferentes de exclusividade, associados ao tipo de eventos a que a exclusividade se refere: Exclusividade Absoluta, Exclusividade no Sinal e Exclusividade em Delta.

Um **Evento Exclusivo Absoluto** ocorre apenas se nenhum dos outros eventos definidos no sistema ocorrerem. Isto significa que no passo de execução da sua ocorrência, nenhum outro evento pode ocorrer para que ele seja válido.

Um **Evento Exclusivo no Sinal** é um evento que ocorre se nenhum outro evento associado ao mesmo sinal ocorrer. Isto significa que no mesmo tempo de análise podem ocorrer outros eventos desde que estejam associados com sinais diferentes. O evento com exclusividade de sinal tem de ser o único a ocorrer dos associados ao seu sinal. Esta exclusividade modela a composição do tipo AND no mesmo tempo de análise do evento exclusivo com Não eventos de todos os outros eventos definidos.

Finalmente, um **Evento Exclusivo em Delta** ocorre apenas quando ele é o único evento associado ao seu sinal no delta que está a analisar. Qualquer even-

to de outro sinal ou de outro delta do seu sinal pode ocorrer sem interferir com a sua exclusividade.

A utilização da exclusividade de eventos permite uma definição mais simples dos casos em que o acontecimento de outros eventos impede a ocorrência desse evento. Sem a utilização da exclusividade ter-se-ia de criar um macro-evento com uma composição entre o evento exclusivo e a negação de todos os outros eventos existentes.

3.2.6. Macro-Condições

Uma macro-condição representa uma condição mais complexa. Esta é definida por composição de condições primitivas e pode estar associada a um ou mais sinais do sistema.

Uma macro-condição segue as mesmas regras definidas para os macro-eventos, apenas com uma restrição. Uma macro-condição tem de ser analisada num tempo definido, dessa forma, não podem ser definidas sequências.

A representação dos tipos de composição de uma macro-condição é dada pela Figura 3.17, adaptada do cubo da Figura 3.11.

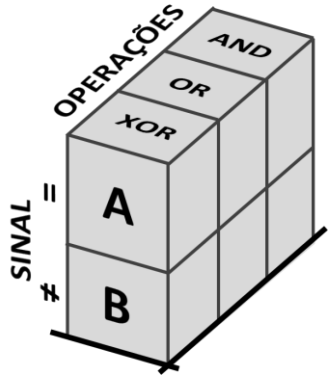


Figura 3.17: Tipos de Composição entre duas Condições

Da mesma forma que para os macro-eventos, adicionando o conceito de **condição negada** (não condição), apenas são necessárias três operações para representar todos os tipos booleanos de composição, mantendo-se que as duas condições a serem compostas podem analisar o mesmo sinal ou podem ser relativas a sinais diferentes.

3.2.7. Ações de Saída

Uma Ação de saída afeta um sinal de saída com um determinado valor k definido.

A sua representação tem a seguinte forma.

$$Sinal = Valor$$

Quando o MAS recebe o evento discreto que representa a ação de saída afeta o sinal associado com o valor definido pela ação.

3.2.8. Eventos de Saída

Eventos de saída são eventos que contribuem para o valor do sinal de saída com um determinado valor resultante de um comportamento específico definido durante uma determinada janela de tempo

Este comportamento é definido através de uma função de evolução que determina o valor da contribuição do evento para o valor do sinal associado em cada passo de execução, durante a janela de tempo definida.

Esta abordagem baseia-se em quatro conceitos principais, associados à definição de um evento de saída. Estes conceitos são a Janela, a Função de Evolução, o Valor Final e o Tipo de Ação.

A **janela (*w*)** é um valor inteiro que define o número de passos após a ocorrência do evento em que o evento contribui para o valor do sinal de saída

associado. Por exemplo, se o valor da janela for dois, o evento irá contribuir para o valor de sinal associado nos passos de execução $n + 1$ e $n + 2$, considerando n o passo de execução em que o evento ocorreu. Genericamente, o evento contribui para os valores do sinal associados entre o passo $n + 1$ e o passo $n + w$ onde w é o valor da janela. Se a janela tem o valor um, o sinal vai ser afetado apenas num único passo de execução ($n + 1$).

A **função de evolução ($f(n)$)** pode ser definida como qualquer função $f(n)$. Esta função vai orientar a evolução do sinal, indicando, para cada passo de execução, o valor da contribuição do evento para o sinal associado. Isto significa que para o passo $n + 1$, a contribuição do evento é $f(1)$ e no passo $n + 2$, a contribuição do evento para o valor do sinal é dada por $f(2)$. Genericamente a contribuição do evento para o sinal associado em cada passo $n + p$ é igual ao valor de $f(p)$. O valor da função pode ser aplicado como um valor absoluto ou um valor relativo ao valor do sinal na ocorrência de um evento, dependendo do tipo de ação definido.

O **valor final (vf)** é o valor que o sinal associado deve ter no final da contribuição do evento, ou seja, no valor final da janela. Este valor pode ser definido como absoluto ou relativo ao valor do sinal no passo de execução em que o evento ocorreu, dependendo do tipo de ação.

Finalmente, o **tipo de ação** pode ser definido de duas formas: absoluto ou relativo. No caso do tipo absoluto, o evento contribui, para o valor do sinal, exatamente com os valores definidos pela função. No caso do tipo relativo, a contribuição do evento para o sinal de saída é dada pela adição da contribuição do evento ao valor que o sinal tinha no passo de execução em que este ocorreu.

As diferenças entre os dois tipos de ação são representadas na Figura 3.18. Nos dois exemplos apresentados, o evento é definido com o mesmo valor de $w = 3$ e com o mesmo $vf = 3$. Neste caso, o valor do sinal associado no passo de

execução em que o sinal ocorre é igual a dois. A única diferença entre os dois eventos de saída definidos é o tipo de ação. No evento à esquerda o tipo de ação é absoluto e no evento da direita é relativo.

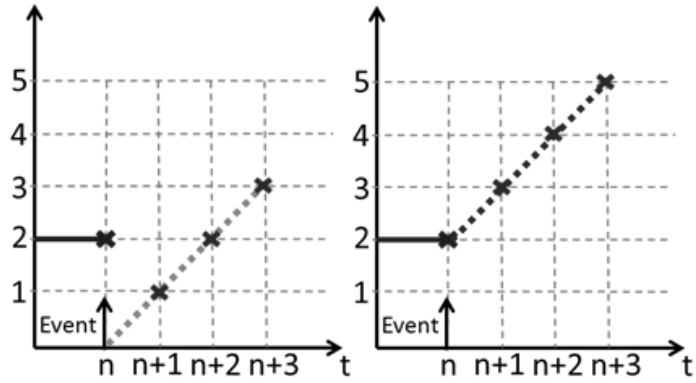


Figura 3.18: Comparação entre o tipo de ação Absoluto (à esquerda) e Relativo (à direita)

Pela observação da Figura 3.18 é clara a diferença entre o tipo de ação absoluto e relativo. No lado esquerdo, onde o valor do sinal de saída evolui com o tipo de ação absoluto, o valor do sinal é o valor exato da função. À direita, o valor do sinal de saída evolui com o tipo de ação relativo. Neste caso, o ponto de partida é igual ao valor do sinal quando ocorre o evento.

Com estas definições, considerando X como o sinal associado, X0 o valor inicial(*vi*) do sinal, e Y a contribuição do evento para o sinal associado, a evolução da contribuição do evento para o valor do sinal associado pode ser definida como:

$$\begin{cases} Y_p = at + f(p), \ \forall p \in [0,w] \\ f(w) = fv \end{cases}$$

Onde *at* é zero, se o tipo de ação for absoluto e se o tipo de ação for relativo *at* é X0.

Tendo em conta esta abordagem, são definidos três tipos de eventos de saída: Eventos de Atribuição Temporizada, Eventos de Atribuição Guiada e Eventos de Evolução.

3.2.8.1. Evento de Atribuição Temporizada

Os **Eventos de Atribuição Temporizada** baseiam-se na definição do valor da janela e do valor do sinal no último passo da janela. Neste caso, conhecendo previamente o valor final do sinal, a função precisa de fazer a aproximação a este valor no número exato de passos definidos na janela.

Tendo esta restrição em mente, nem todas as funções são válidas para um determinado evento. Apenas as funções que permitem chegar ao ponto final definido no número de passos definidos pela janela são válidas.

Para determinar se uma função é válida, é necessário calcular o valor da função no ponto final da janela.

Existem alguns tipos de funções que podem ser analisadas individualmente devido às suas características particulares. Neste grupo, as seguintes funções serão analisadas: funções constantes e funções lineares.

Nas funções constantes, para cada evento de atribuição temporizada, apenas uma é válida. Essa função é a função que toma o valor constante igual ao valor final definido. No entanto, o valor final depende também do tipo de ação, podendo ser exatamente o *vf* definido caso seja definido tipo de ação absoluto ou *vf* mais o valor inicial do sinal caso seja definido tipo de ação relativo. Este tipo de evento com esta função de evolução associada permite uma evolução

em que a contribuição para o sinal é igual ao valor final em todos os valores da janela, tal como apresentado na Figura 3.19.

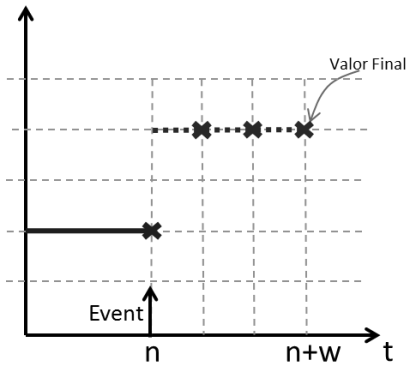


Figura 3.19: Evolução da contribuição de um evento de atribuição temporizada com função de evolução constante

Outro tipo de funções com características especiais é o das funções lineares, apresentado na Figura 3.20.

Uma função linear é uma função que possui as duas seguintes propriedades: Aditividade e Homogeneidade:

- ✓ $f(x + y) = f(x) + f(y)$
- ✓ $f(ax) = a\, f(x)$

Estas funções atravessam a origem e têm uma equação do tipo $f(x) = ax$ onde $a \neq 0$. No caso particular dos eventos de atribuição temporizada, como no caso das funções constantes, apenas uma função linear permite que se atinja o valor final esperado.

Por outro lado, tendo o valor final e o valor inicial é possível calcular a inclinação *a*, que gera uma função que satisfaça os dois valores dos extremos. A inclinação é calculada com a equação:

$$a = \frac{fv - iv}{w}$$

Na equação anterior, *iv* é o valor inicial da função. Se considerarmos o tipo de ação absoluto, é o valor da função em *x=0*, que para qualquer função linear é zero. Para um tipo de ação relativo, *iv* é igual ao valor do sinal quando o evento ocorre.

O *fv* representa o valor final que o sinal associado deve ter no final da contribuição do evento de saída. Como referido anteriormente, se se considerar o tipo de ação absoluto, o valor final é o valor definido no evento, no caso de se considerar o tipo de ação relativo, o valor final é o valor definido no evento adicionado ao valor inicial.

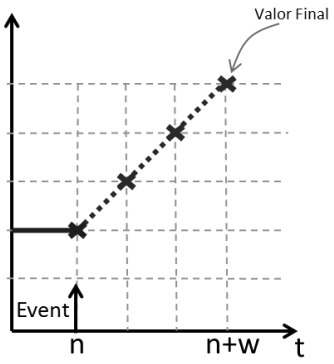


Figura 3.20: Evolução da contribuição de um evento de atribuição temporizada com função de evolução linear

Para todos os outros tipos de funções, só um pequeno número de cada tipo é válido, de modo a satisfazer os valores limite, ou seja, valores iniciais e finais.

Neste sentido, para permitir a utilização de qualquer função arbitrária para descrever o comportamento pretendido do sinal de saída, a adoção de uma das seguintes duas abordagens é considerada como permitindo a satisfação dos valores iniciais e finais.

Uma destas abordagens permite considerar uma função arbitrária e realizar uma transformação, de forma a permitir à função cruzar os valores limite. Neste caso, é dada a função arbitrária f(n) pelo evento, mas a função g(n) é a que realmente define a evolução e é dada pela expressão:

$$g(n) = K1.f(n) + K2$$

Nesta equação, *K1* é o fator de escala e *K2* é o valor da translação que permite alterar o *offset* do valor inicial. Para aplicar esta abordagem, é necessário que os valores, inicial e final, da função sejam diferentes.

Por outra abordagem é possível definir uma função arbitrária contendo um parâmetro livre, representada pela função *f (n, a)*, onde a variável *a* pode ser ajustada de modo a satisfazer os valores dos limites. Este parâmetro *a* é semelhante ao já utilizado quando a função linear foi apresentada.

Em termos do tipo de ação, para além dos dois tipos diferentes definidos anteriormente em que todos os valores da contribuição são absolutos ou relativos, é possível definir dois outros casos mistos.

Estes dois casos são apresentados na Figura 3.21, no lado esquerdo, mostra-se como evoluir a partir de um valor inicial relativo até um valor final absoluto. Neste caso seja qual for o valor de partida relativo ao valor inicial, o valor final é sempre o mesmo, pois o tipo de ação é absoluto. Na Figura 3.21, no lado direito, evolui-se a partir de um valor inicial absoluto para um valor final relativo. Neste caso, o valor inicial é sempre o valor da função, pois o tipo de ação é absoluto, enquanto o valor final é relativo, logo depende do valor que o sinal tinha no início da contribuição.

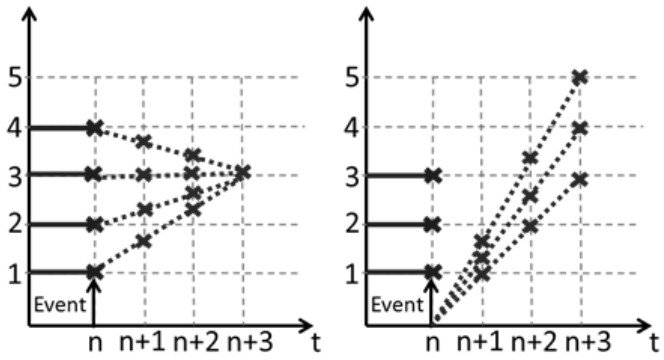


Figura 3.21: Evolução da contribuição de um evento de atribuição temporizado com tipos de ação diferentes. Relativo para Absoluto (à esquerda) e Absoluto para Relativo (à direita)

Nos dois casos, a evolução da função é afetada pelo limite relativo.

3.2.8.2. Evento de Atribuição Guiada

O **Evento de Atribuição Guiada**, como o evento de atribuição anterior, define o valor do sinal no último passo da contribuição. A função de evolução é definida, bem como com o valor final do sinal. Neste caso, a janela não é defini-

da *a priori*. O valor da janela é obtido através do número de passos necessários para atingir o valor final definido, através de uma evolução guiada pela função definida.

No exemplo da Figura 3.22, é apresentado um evento que tem o objetivo de atingir o valor 5, utilizando a função $f(n) = n$. No exemplo A, o sinal tem o valor inicial $vi = 3$. Isto significa que o evento irá contribuir para o valor do sinal durante duas etapas, que são necessárias para atingir o valor de 5. No exemplo B, o mesmo evento começa quando o valor de sinal é 2. Neste caso, o número de contribuições será de três passos para permitir atingir 5.

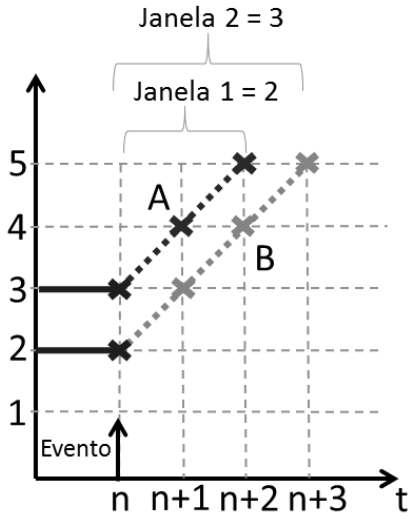


Figura 3.22: Exemplo da contribuição de um evento de atribuição guiada para o valor do sinal associado.

Dessa forma, um evento de atribuição guiado com a mesma função e valor final, contribui para o valor do sinal durante diferentes janelas de tempo, dependendo do valor inicial do sinal.

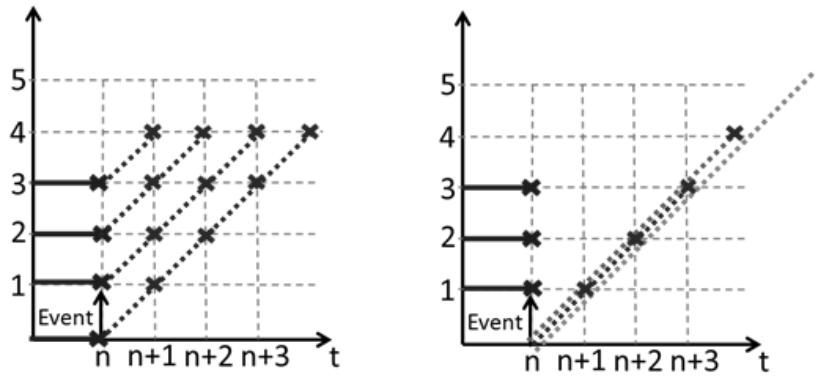


Figura 3.23: Evolução da contribuição de um evento de evolução com tipos de ação diferentes. Relativo para Absoluto (à esquerda) e Absoluto para Relativo (à direita)

Para os eventos de atribuição guiada, de forma semelhante aos eventos de atribuição temporizada, é possível definir outros dois tipos de ação, onde a evolução ocorre a partir de um valor relativo para um valor absoluto ou a partir de um valor absoluto para um valor relativo. Estes dois casos podem ser vistos na Figura 3.23. Para ambos os casos é definida a função $f(n) = n$ como função de evolução. No lado esquerdo, é considerada uma evolução de um valor relativo para um valor absoluto, de forma a obter um valor final igual a quatro. São considerados vários valores iniciais diferentes. A janela de tempo será diferente para os diferentes valores iniciais. No lado direito, é considerada uma evolução de um valor absoluto para um valor relativo. É definido o valor final igual a 2, e são considerados três valores diferentes para o valor inicial (1, 2, e 3), que conduzem a três possíveis valores finais (3, 4, e 5) (pois o tipo de ação no valor final é relativo) e três janelas de tempo associadas diferentes.

3.2.8.3. Evento de Evolução

Num **Evento de Evolução**, ao contrário dos eventos de atribuição, o valor final do sinal associado não é conhecido *a priori*. O valor do sinal evolui dependendo da função associada ao evento durante a janela de tempo definida. O va-

lor final é obtido em conformidade, a partir da aplicação da função de evolução durante a janela de tempo do evento.

Como os eventos de evolução não têm restrições ao valor final, todas as funções de evolução são válidas. Na verdade, neste caso, a função tem uma importância central na contribuição para o sinal, pois define o comportamento e o caminho da evolução desde o início até ao fim do valor da janela.

Na Figura 3.24, dois eventos de evolução diferentes são apresentados com a mesma janela de tempo (*w = 3* neste caso), e o mesmo valor inicial do sinal associado. Define-se *f1(n)* e *f2(n)* como duas funções diferentes (*f1(n) = N e f2(n) = N/3*). Como se pode verificar, o valor final é diferente para diferentes funções de evolução.

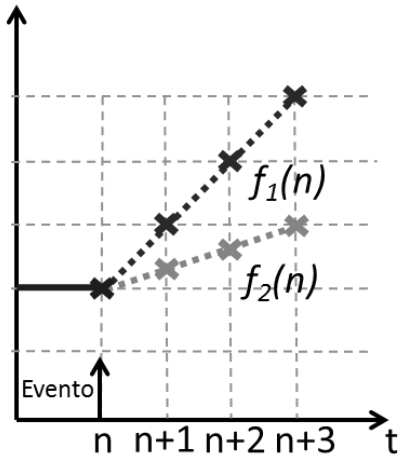


Figura 3.24: Exemplo da contribuição de um evento de evolução para o valor do sinal associado.

É interessante notar que, para as funções constantes, os eventos de evolução têm o mesmo comportamento que os eventos de atribuição temporizada. Embora o valor final não está diretamente definido, como a função é constante, este valor constante, também define o valor final da evolução.

3.2.8.4. Evento de Saída Atrasado

Um **Evento de Saída Atrasado** não é um novo tipo de evento. É um evento de qualquer um dos tipos apresentados anteriormente, que possui a característica de atraso.

Nas seções anteriores os eventos são definidos tendo em conta que a contribuição do evento para o sinal começa no passo seguinte à ocorrência do evento. Esta é a abordagem mais utilizada a fim de aplicar a contribuição do evento no menor tempo possível. No entanto, em alguns casos, adiar a aplicação da contribuição do evento com um atraso de uma série de passos definidos, pode trazer vantagens.

Um evento atrasado tem as mesmas características que um evento não atrasado, mas em vez de iniciar a sua contribuição para o sinal na etapa imediatamente seguinte à ocorrência do evento, ele começa, essa contribuição, um número definido de passos depois.

Na Figura 3.25, é definido um evento sem atraso (A), e exatamente o mesmo evento, mas com um atraso de valor 2 (B).

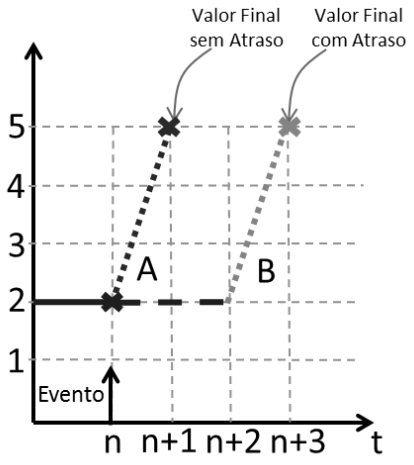


Figura 3.25: Evento de saída atrasado.

Durante o tempo do atraso, o evento não tem nenhuma contribuição para a evolução do sinal associado.

3.2.8.5. Evento de Saída Ampliado

Como nos eventos de saída com atraso, os eventos de saída ampliados são uma característica que pode ser associada a qualquer dos tipos de eventos de saída definidos.

Na definição inicial dos eventos de saída é definido que estes contribuem para o valor do sinal desde o início da contribuição até ao fim da sua contribuição num conjunto de passos consecutivos. Num evento de saída ampliado a contribuição é ampliada ou comprimida no tempo por um valor definido.

Na Figura 3.26, são apresentados, um evento sem ampliação, e exatamente o mesmo evento com ampliação igual a dois.

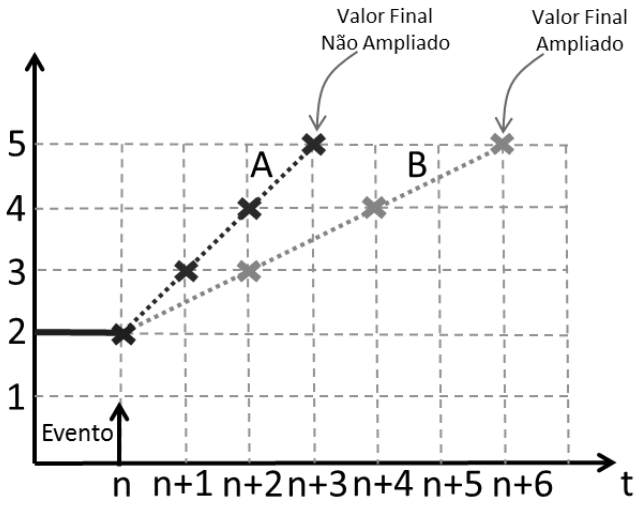


Figura 3.26: Evento de saída ampliado.

O evento tem exatamente o mesmo número de contribuições para o sinal e essas contribuições têm o mesmo valor, mas a janela é ampliada para um valor igual a *n* * *janela*, em que o *n* é o valor da ampliação. Pode ver-se também que a distância entre as contribuições passa a ser maior que um, havendo uma contribuição apenas de *n* em *n* passos.

3.3. Sintaxe e Gramática

Para se poder representar e definir os conceitos apresentados, são definidas duas sintaxes. Uma destas sintaxes é gráfica e permite uma interpretação mais fácil por um utilizador humano. A outra sintaxe é textual e permite uma representação mais compacta dos mesmos conceitos.

Nesta secção, estas duas sintaxes serão apresentadas simultaneamente.

3.3.1. Sintaxe

Um evento primitivo é definido, textualmente dentro de parênteses curvos. Dentro dos parênteses a variação (delta) é definida na forma de "d (delta)", o sinal associado (representado pelo nome), o tipo de evento (representado pelo tipo definido na Tabela 3.2, por exemplo, "< to ≥ "), e o nível de comparação. Graficamente é representado por um retângulo de vértices retos, dividido verticalmente em duas partes. Em cima é representado o Nome do evento e em baixo a equação textual sem parênteses. Na Figura 3.27 a) são apresentadas as representações, textual e gráfica, de um evento primitivo. De forma semelhante, uma condição primitiva é definida, textualmente, dentro de parênteses retos. Dentro dos parênteses a variação (delta) é definida na forma de "d (delta)", o sinal associado (representado pelo nome), o tipo de condição (representado pelo tipo definido na Tabela 3.1, por exemplo, "<"), e o nível de comparação. Gra-

ficamente, uma condição primitiva é representada por um retângulo de vértices curvos dividido verticalmente em duas partes. Em cima representa-se o nome da condição, em baixo a equação textual sem parênteses. A sua representação é apresentada na Figura 3.27 b)

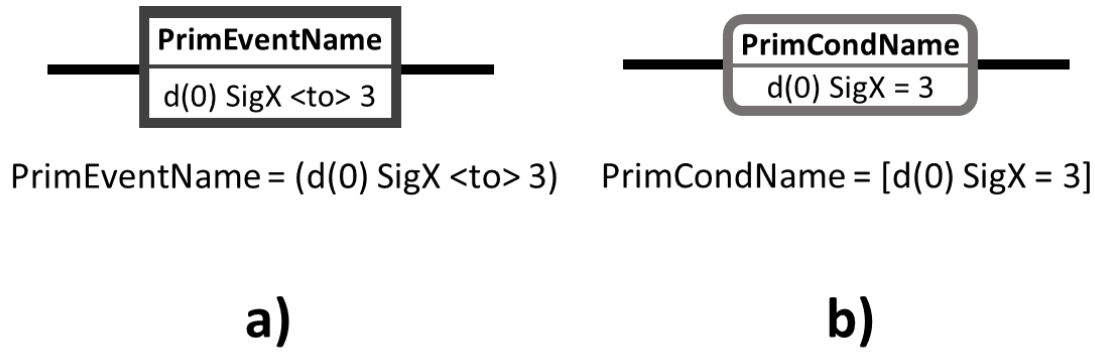


Figura 3.27: Representação de: a) Evento Primitivo; b) Condição Primitiva.

A negação, tanto no caso de um não evento como numa não condição é representada textualmente por um ponto de exclamação antes e outro depois dos parênteses e graficamente por um círculo depois do retângulo. Na Figura 3.28 são representados um não evento (Figura 3.28 a)) e uma não condição (Figura 3.28 b)).

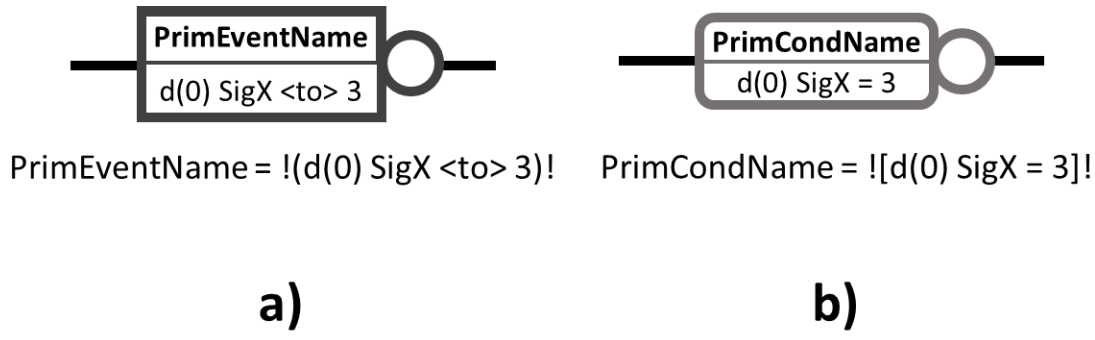


Figura 3.28: Representação de: a) Não Evento; b) Não Condição.

Com esta sintaxe consegue-se representar todos os conceitos primitivos.

Em termos de composição de duas condições, os tipos de operação definidos (AND, OR e XOR) são representadas na Figura 3.29. É ainda apresentada a composição AND entre uma condição e uma não condição ($A\ \overline{B}$).

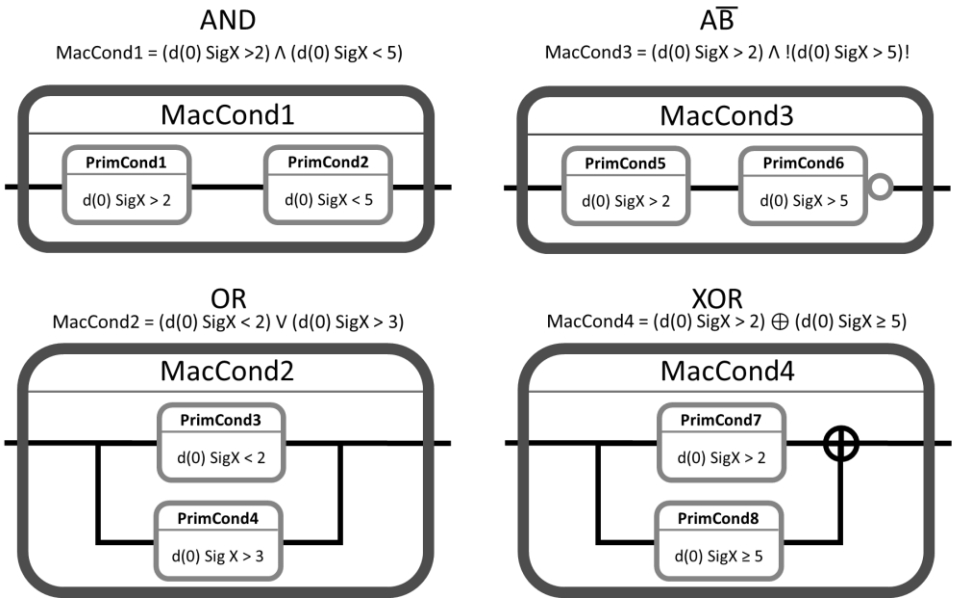


Figura 3.29: Representação dos vários tipos de composição de condições.

Na Figura 3.30, são representados os tipos de composição (AND, OR e XOR) para dois eventos. Nesta figura é também apresentada a composição de $A\ \overline{B}$. Como definido anteriormente, um não evento é uma condição, logo esta representação é válida também para a composição de um evento com uma qualquer condição.

A condição só pode ser analisada tendo em conta um determinado tempo, logo só pode aparecer num macro-evento, associada ao tempo de ocorrência de um determinado evento. Na representação gráfica, a condição é definida dentro do evento a que pertence o tempo em que tem de ser analisada.

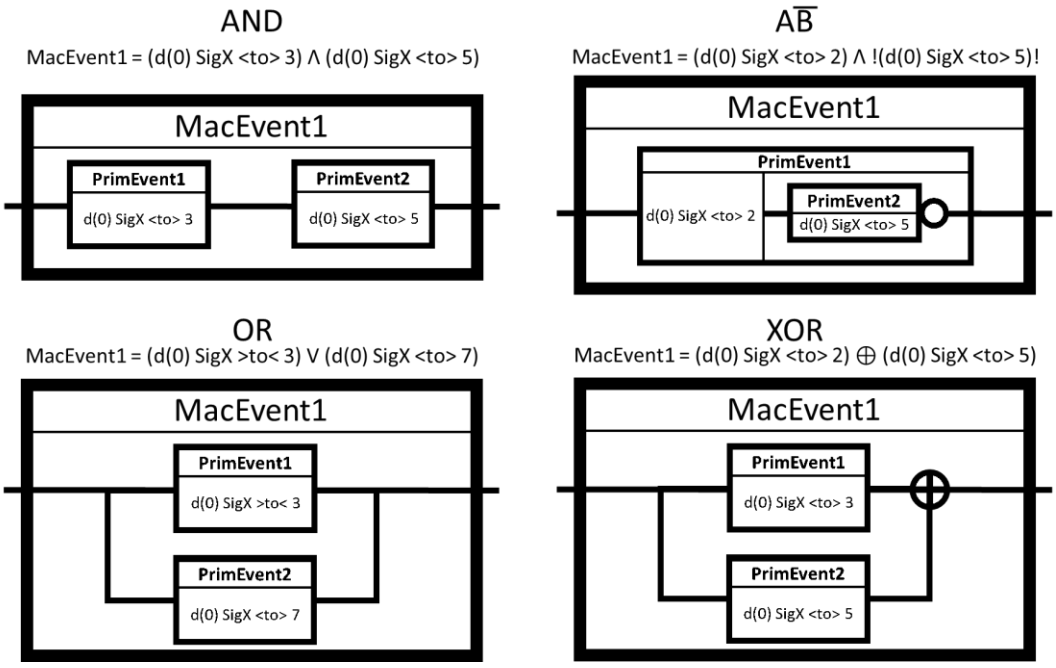


Figura 3.30: Representação dos vários tipos de composição de eventos.

Para ser possível representar a composição é ainda necessário representar a ordem pela qual os eventos acontecem. Textualmente a ordem dos eventos é representada por uma ordem numérica apresentada, inferior à linha, depois do evento. A ordem zero (0) representa o tempo em que o macro-evento deve ocorrer, ou seja, o último evento da composição a ocorrer. Quanto maior o valor da ordem, mais cedo o evento tem de ocorrer. Na Figura 3.31 são apresentadas as várias ordens para ocorrência de dois eventos. Na Figura 3.31 a) é representada a ocorrência de dois eventos no mesmo passo de execução. É representada textualmente pelo mesmo valor de ordem. Graficamente os eventos são representados um em cima do outro em cima da linha de ocorrência. Na Figura 3.31 b) é representada a ocorrência dos dois eventos sem relevância da ordem. Textualmente é representado sem ordem, textualmente por uma caixa em cima da linha do tempo com os dois eventos representados dentro dela divididos por duas setas a apontar para sentidos opostos. Na Figura 3.31 c) e d) é representada uma

sequência. Em c) ocorrendo primeiro PrimEvent1 e em seguida PrimEvent2 e em d) PrimEvent2 e depois PrimEvent1.

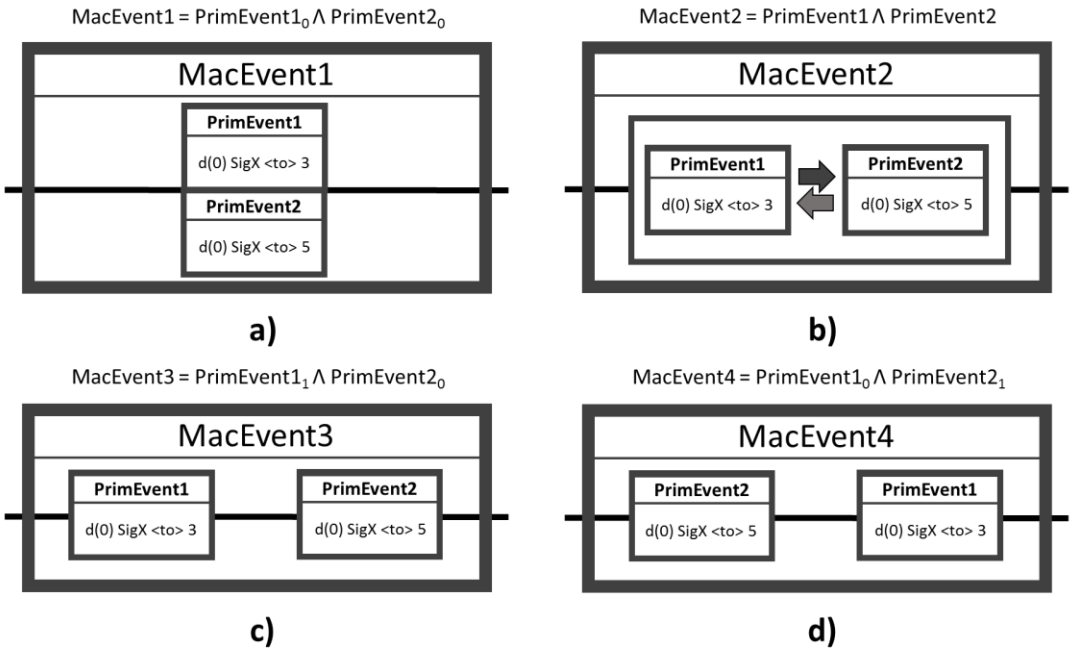


Figura 3.31: Representação da ordem de dois eventos para uma composição do tipo AND. a) Eventos *Ev1* e *Ev2* no mesmo passo de execução; b) Ordem indiferente; c) Primeiro *Ev1* e depois *Ev2*; d) Primeiro *Ev2* e depois *Ev1*.

Finalmente, o tempo de vida é apresentado na Figura 3.32. Na Figura 3.32 a) é apresentado o tempo de vida absoluto. Textualmente é apresentado inferior à linha depois do nome do macro-evento a que se refere, imediatamente antes do sinal de igualdade. Graficamente é representado por uma linha terminada nas duas pontas por setas e tracejada. Em cima dessa seta é colocado o tempo de vida absoluto dentro dos parênteses. Na Figura 3.32 b), o tempo de vida relativo é apresentado. Textualmente é representado entre parênteses retos, inferior à linha, depois da ordem do evento. Graficamente é representado entre parênteses curvos entre os dois eventos a que se refere. Sempre que um dos tempos de vida não é apresentado, significa que não existe relação temporal definida.

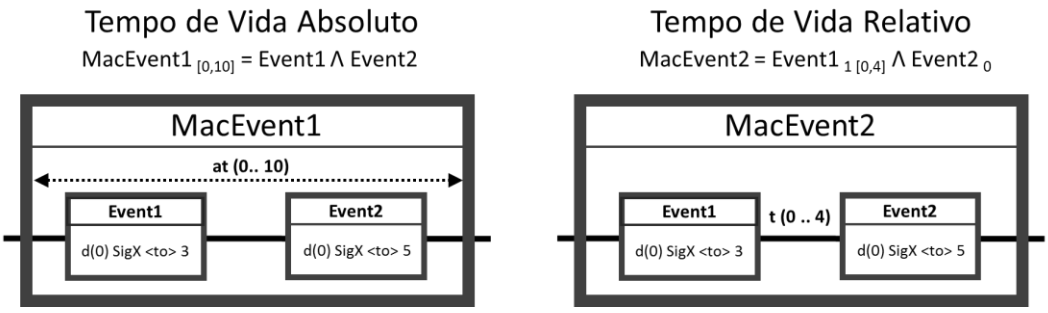


Figura 3.32: Representação do tempo de vida absoluto e relativo.

Por exemplo, o evento X que ocorre quando se verifica A, seguido de B, de 3 a 7 passos depois, e em seguida C e a condição N de 0 a 3 passos depois ou D (sem tempo de vida relativo associado entre B e D), tendo de ocorrer todo no máximo entre 2 e 10 passos no total, representa-se como na Figura 3.33.

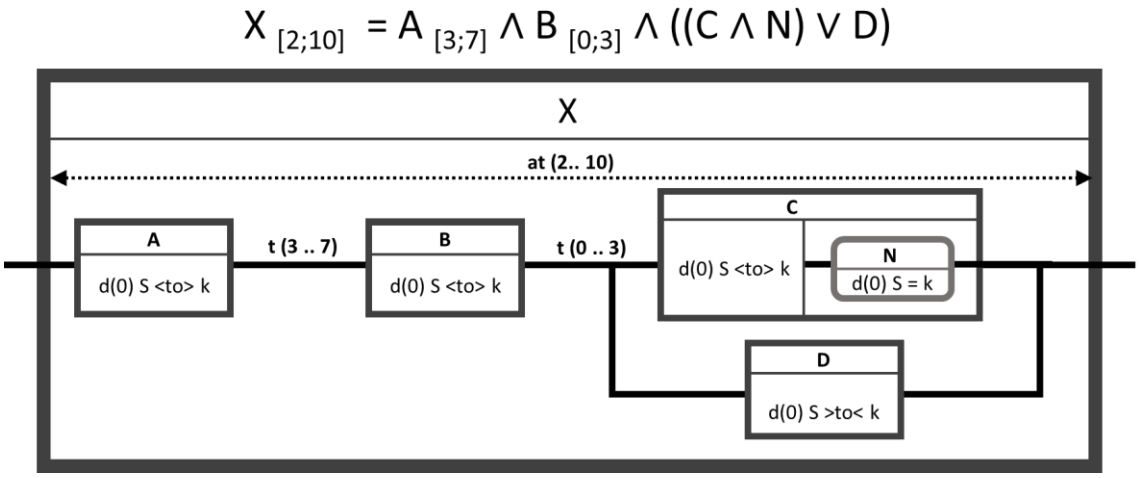


Figura 3.33: Representação do macro-evento X.

3.3.2. Meta-Modelo e XML

Para a implementação das ferramentas de edição e geração de código foi desenvolvido um meta-modelo para representação do modelo. Esse meta-modelo é apresentado na Figura 3.34

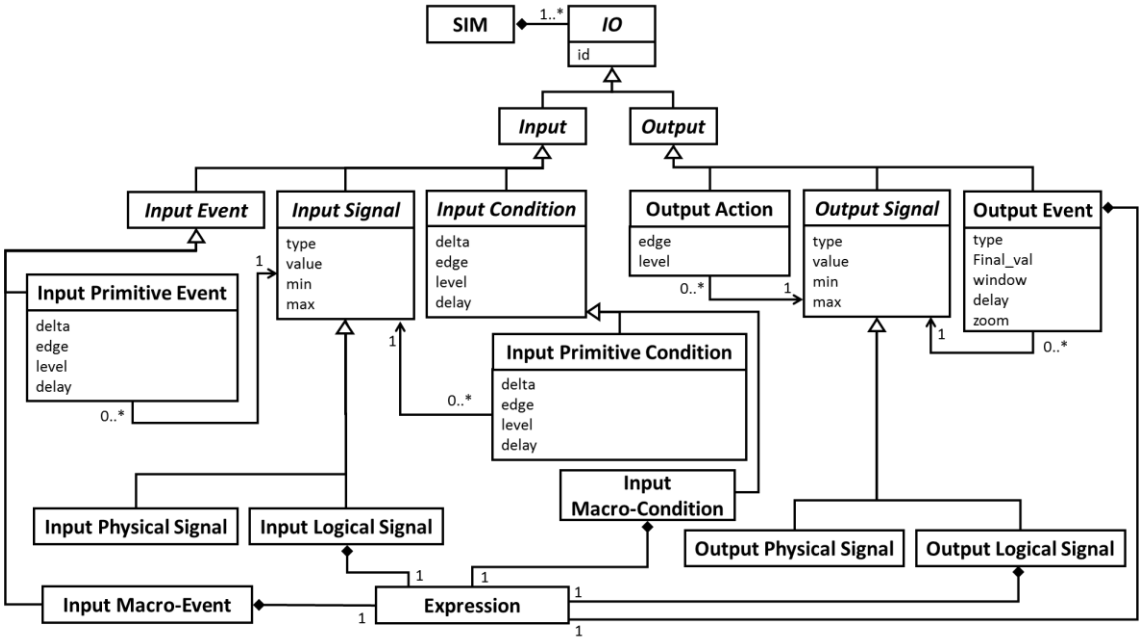


Figura 3.34: Meta-Modelo de um Modelo de Interpretação ou Afetação de Sinais.

O Meta-Modelo apresenta os elementos do Input e do Output e a sua relação, bem como os atributos associados a cada um.

Na Figura 3.34 é definida a *Expression*. O seu Meta-Modelo é apresentado na Figura 3.35.

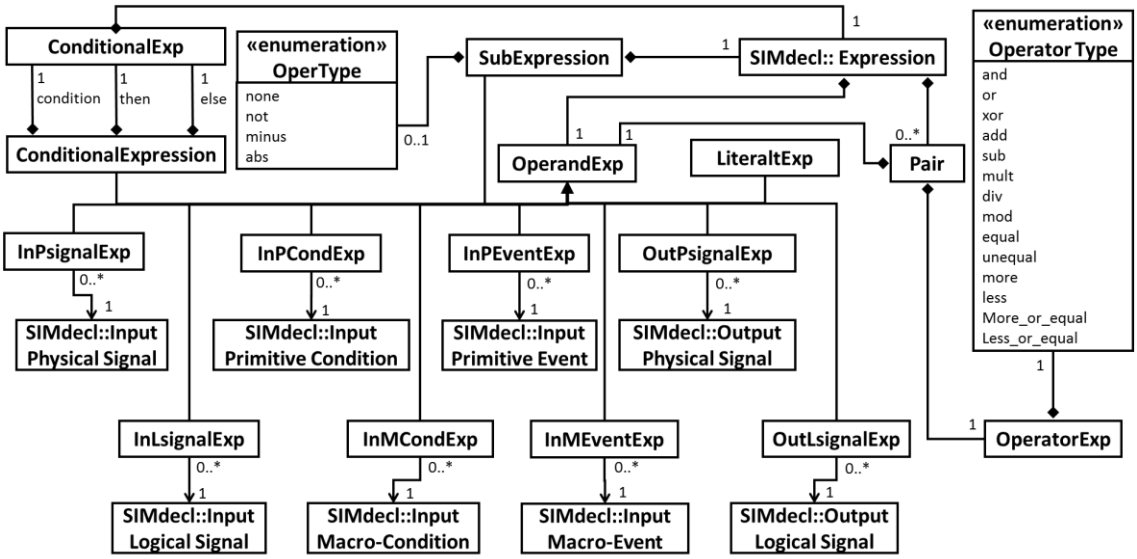


Figura 3.35: Meta-Modelo de uma equação.

Os elementos e seus atributos são definidos em XML como será apresentado em seguida.

Todos os elementos têm definida a representação gráfica onde é dada a posição do elemento na imagem. É representada em *graphics*.

O XML que representa um sinal externo está apresentado na Figura 3.36. É definido como *signal* e tem um ID que identifica o sinal, o *type* que define o tipo de sinal (booleano ou range), o *value* que define o valor inicial do sinal e um max e um min que definem o valor máximo e mínimo que o sinal pode tomar.

```
<signal id="Temp1" type="range" value="21" min="0" max="100">
  <graphics>
    <position page="1" x="100" y="60"/>
  </graphics>
</signal>
```

Figura 3.36: Representação XML de um Sinal Externo.

O XML que representa um sinal interno está apresentado na Figura 3.37. É definido como *lsignal* e tem um ID que identifica o sinal, o *type* que define o tipo de sinal (booleano ou range), o *value* que define o valor inicial do sinal e um max e um min que definem o valor máximo e mínimo que o sinal pode tomar.

Tem também uma função associada, definida como *lfunction*.

```
<lsignal id="dif_temp12" type="range" value="0" min="0" max="100">
  <graphics>
    <position page="1" x="230" y="130"/>
  </graphics>
  <lfunction>
    <concreteSyntax language="sim">
      <text>ABS( Temp1(0) - Temp2(0) )</text>
      <expression>
        <operand type="subexpression" absolute="yes" seq="3">
          <operand type="input-signal" idRef="Temp1" seq="4" time="0"/>
          <operation operator="subtraction" seq="5">
            <operand type="input-signal" idRef="Temp2" seq="6" time="0"/>
          </operation>
        </operand>
      </expression>
    </concreteSyntax>
  </lfunction>
</lsignal>
```

Figura 3.37: Representação XML de um Sinal Interno.

Na Figura 3.38 é representado o XML de uma condição primitiva. Define-se por um elemento *condition* e tem como atributos um ID, o delta, o sinal associado, o tipo de evento definido como *edge*, o *level* e o *delay* associados.

```
<condition id="M25_1" delta="0" signal="Temp1" edge="LT" level="20" delay="1">
  <graphics>
    <position page="1" x="110" y="400"/>
  </graphics>
</condition>
```

Figura 3.38: Representação XML de uma Condição Primitiva.

Na Figura 3.39 é representado o XML de um evento primitivo. Define-se por um elemento *event* e tem como atributos um ID, o delta, o sinal associado, o tipo de condição definido como *edge*, o *level* e o *delay* associados.

```
<event id="maior4" delta="0" signal="dif_temp12" edge="LTtoGE" level="4" delay="1">
  <graphics>
    <position page="1" x="100" y="200"/>
  </graphics>
</event>
```

Figura 3.39: Representação XML de um evento primitivo.

O XML que representa uma macro-condição está apresentado na Figura 3.40. É definido como *mcondition* e tem um ID que identifica a condição

Tem também uma função associada, definida como *mequation*.

```
<mcondition id="M25">
  <graphics>
    <position page="1" x="210" y="500"/>
  </graphics>
  <mequation>
    <concreteSyntax language="sim">
      <text>M25_1 OR M25_2</text>
      <expression>
        <operand type="input-cond" idRef="M25_1" seq="1"/>
        <operation operator="or" seq="2">
          <operand type="input-cond" idRef="M25_2" seq="3"/>
        </operation>
      </expression>
    </concreteSyntax>
  </mequation>
</mcondition>
```

Figura 3.40: Representação XML de uma macro-condição.

Finalmente, o XML que representa um macro-evento está apresentado na Figura 3.41. É definido como *mevent* e tem um ID que identifica a condição, bem como o *minlt* que define o tempo de vida absoluto mínimo e o *maxlt* que define o tempo de vida absoluto máximo

Tem também uma função associada, definida como *mequation*.

```
<mevent id="inbetween" minlt="" maxlt="">
  <graphics>
    <position page="1" x="200" y="300"/>
  </graphics>
  <mequation>
    <concreteSyntax language="sim">
      <text>maior4(0) AND menor6(0)</text>
      <expression>
        <operand type="input-event" idRef="maior4" seq="1" order="0" minlt="" maxlt=""/>
        <operation operator="and" seq="2">
          <operand type="input-event" idRef="menor6" seq="3" order="0" minlt="" maxlt=""/>
        </operation>
      </expression>
    </concreteSyntax>
  </mequation>
</mevent>
```

Figura 3.41: Representação XML de um macro-evento.

Neste capítulo foram apresentadas as estruturas e fluxos do MIS e do MAS bem como os conceitos que os constituem. Foram ainda apresentadas sintaxes e gramática que permitem a sua representação. No capítulo seguinte são apresentados três exemplos da utilização dos conceitos apresentados.

4

Validação

Neste capítulo são apresentados três exemplos onde os conceitos definidos são aplicados, de forma a ilustrar a sua utilização e como a modelação recorrendo a esses conceitos difere da modelação usando os formalismos de modelação existentes.

Para a validação dos conceitos propostos, foi desenvolvido um conjunto de ferramentas computacionais, que se podem encontrar em (http://gres.uninova.pt/~rcr/events). Este conjunto de ferramentas permite editar modelos de interpretação de sinais e gerar automaticamente código para esses modelos.

4.1. Ronda de Um Segurança por um Bairro

Pretende-se definir um sistema que monitoriza uma ronda feita por um segurança num bairro habitacional (ou em instalações industriais ou outras).

O segurança deve percorrer o bairro de forma a passar em cinco pontos definidos, seguindo uma determinada ordem, e com um intervalo de tempo específico entre eles.

Em cada ponto o guarda tem de introduzir um código (e/ou apresentar um identificador eletrónico), de forma a registar o momento em que atingiu o ponto.

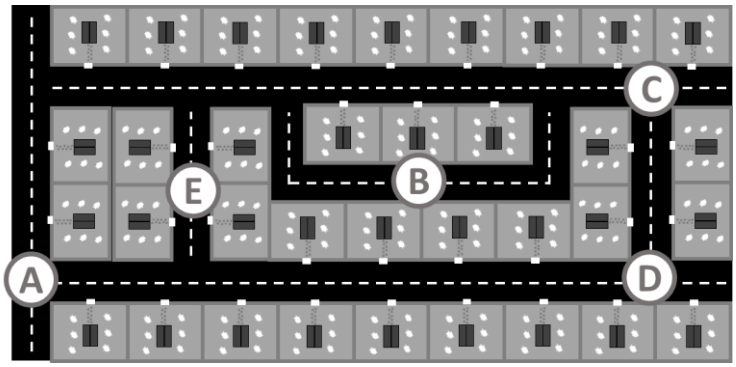


Figura 4.1: Ronda com cinco pontos de controlo.

Na Figura 4.1 apresenta-se um exemplo de um bairro onde o segurança deve fazer a ronda começando no ponto A e terminando no ponto E. A ronda deve seguir as seguintes regras:

- O guarda deve começar a ronda no Ponto A;
- Em seguida, deve chegar ao ponto B no espaço temporal entre 10 e 15 minutos depois de sair do ponto A;
- Deve atingir o ponto C entre 5 e 10 minutos depois de sair do ponto B;
- Após um tempo entre 5 e 10 minutos após o registo no ponto C, deve chegar ao ponto D;
- Finalmente deve terminar a ronda no ponto E, chegando a este num intervalo temporal entre 20 e 30 minutos depois do ponto D.

- A ronda tem de ser feita, desde a partida do ponto A, até à chegada ao ponto E, num tempo total compreendido entre um mínimo de 45 minutos e um máximo de 60 minutos.

O sistema tem apenas cinco sinais booleanos, cada um representando a ativação de cada um dos postos de controlo (A, B, C, D e E).

Pretende-se detetar um conjunto de comportamentos que serão utilizados num modelo de gestão da ronda. Apenas usando os cinco sinais de entrada definidos pretende-se detetar os seguintes comportamentos:

- Ronda terminada com sucesso.
- Ponto B, C, D ou E ativados cedo demais.

Para facilitar a representação, os tempos de vida dos eventos serão definidos em minutos.

O evento que ocorre quando a ronda é completada com sucesso está definido na Figura 4.2:

$$EvP_{[45,60]} = EvA_{4\ [10,15]} \wedge EvB_{3\ [5,10]} \wedge EvC_{2\ [5,10]} \wedge EvD_{1\ [20,30]} \wedge EvE_{0}$$

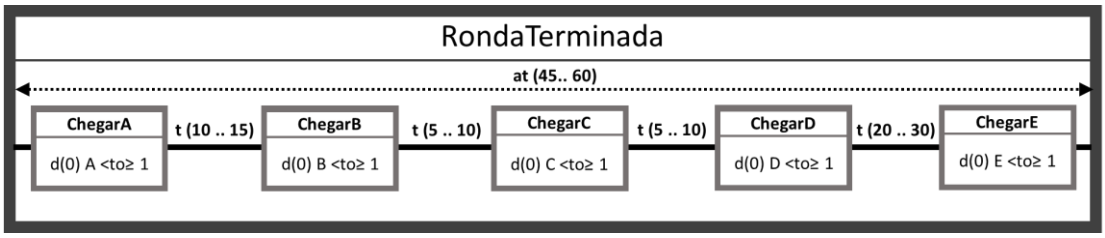


Figura 4.2: Evento gerado quando ocorre corretamente a ronda com cinco pontos de controlo.

Os eventos gerados quando o segurança chega mais cedo que o previsto aos pontos de registo são definidos como apresenta a Figura 4.3, onde o tempo

para atingir o ponto a verificar é definido entre zero e o mínimo para fazer a ronda corretamente.

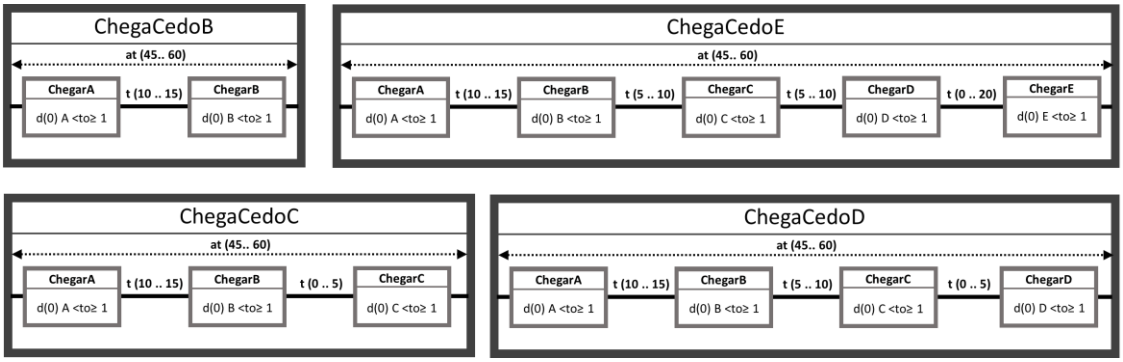


Figura 4.3: Eventos gerados quando o segurança chega cedo demais, aos pontos B, C, D e E.

É gerado um evento discreto cada vez que um destes eventos ocorre. Estes eventos podem ser utilizados como entrada de um modelo de um sistema de gestão da ronda definida expresso num formalismo de modelação a eventos discretos.

4.2. Controlo de Acesso a Zona Histórica de Uma Cidade

Neste exemplo pretende-se modelar um sistema de controlo de tráfego para o centro histórico de uma cidade.

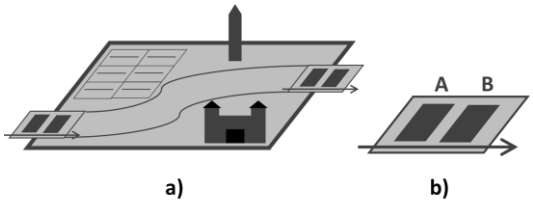


Figura 4.4: Exemplo de controlo de centro histórico a) Exemplo de centro histórico b) Representação do sensor.

O centro histórico da cidade (Figura 4.4 a)) tem uma estrada com apenas um sentido de tráfego. Existe uma entrada e uma saída para automóveis e existe uma condicionante que apenas permite 5 carros dentro do centro histórico de cada vez.

Para controlar o acesso dos veículos, a entrada e a saída têm um conjunto de sensores, apresentados na Figura 4.4 b). Cada um destes sensores consiste num par de sensores A e B, tão próximos que o veículo pode pisa-los ao mesmo tempo. Para que seja detetado um carro a atravessar o sensor, este deve acionar A, em seguida B, em seguida soltar A e finalmente soltar B.

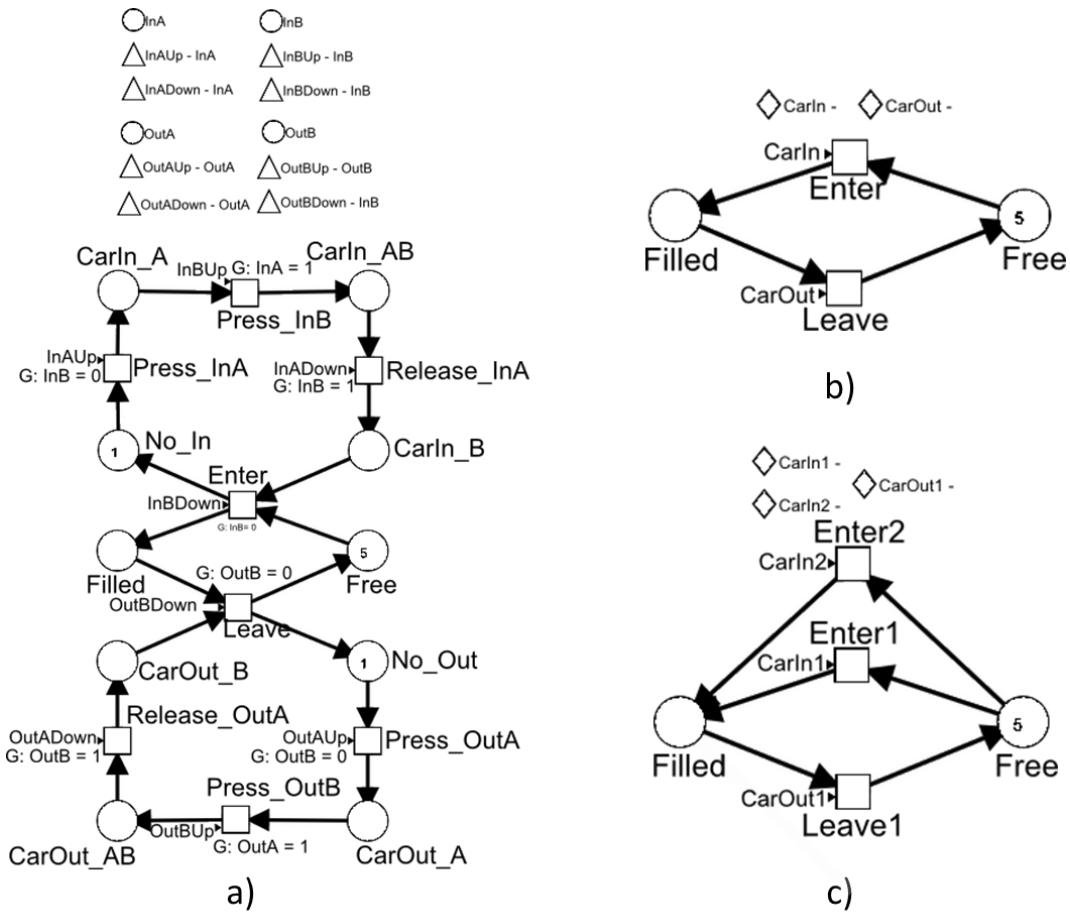


Figura 4.5: Modelo de controlo do centro histórico da cidade; a) sem modelos de interface; b) com modelos de interface; c) com modelos de interface e duas entradas.

O modelo do controlador de acessos à zona histórica da cidade, modelado com redes de Petri IOPT, é apresentado na Figura 4.5. a)

Usando uma modelação baseada nas propostas apresentadas neste documento, o modelo reduzido em redes de Petri é definido na Figura 4.5 b). Nesse caso, o Modelo de interpretação de sinais é definido na Tabela 4.1.

Tabela 4.1 – Modelo de interpretação de sinais do controlo da zona histórica

Eventos Primitivos	Condições Primitivas
InAUp = (d(0) InA <to≥ 1)	InAOne = [d(0) InA = 1]
InADown = (d(0) InA >to≤ 0)	InAZero = [d(0) InA = 0]
InBUp = (d(0) InA <to≥ 1)	InBOne = [d(0) InB = 1]
InBDown = (d(0) InA >to≤ 0)	InBZero = [d(0) InB = 0]
OutAUp = (d(0) InA <to≥ 1)	OutAOne = [d(0) OutA = 1]
OutADown = (d(0) InA >to≤ 0)	OutAZero = [d(0) OutA = 0]
OutBUp = (d(0) InA <to≥ 1)	OutBOne = [d(0) OutB = 1]
OutBDown = (d(0) InA >to≤ 0)	OutBZero = [d(0) OutB = 0]

Macro Eventos

$$CarIn = (InAUp \wedge InBZero)_3 \wedge (InBUp \wedge InAOne)_2 \wedge (InADown \wedge InBOne)_1 \wedge (InBDown \wedge InAZero)_0$$

$$CarOut = (OutAUp \wedge OutBZero)_3 \wedge (OutBUp \wedge OutAOne)_2 \wedge (OutADown \wedge OutBOne)_1 \wedge (OutBDown \wedge OutAZero)_0$$

A modelação da entrada de um veículo é feita através de um macro-evento definido por composição de quatro macro-eventos, cada um definido por uma composição entre um evento primitivo e uma condição primitiva.

Estes quatro macro-eventos detetam a seguinte sequência de eventos nesta ordem: o carro pisa o sensor A enquanto o sensor B não está ativo; o carro pisa o sensor B, mantendo-se o sensor A ativo; o carro deixa de pisar o sensor A, mantendo-se o sensor B ativo e, finalmente, o sensor B deixa de ser pressionado, enquanto o sensor A se mantém não ativo. O evento *CarIn* ocorre quando este

último evento ocorre, e somente se todos os eventos anteriores já ocorreram na sequência predefinida.

A composição de eventos também pode acomodar a extensão do sistema para lidar com duas entradas. Neste caso, com duas entradas, o evento CarIn pode ser definido pela seguinte sequência:

$$CarIn = CarIn1 \ V \ CarIn2$$

Onde CarIn1 e CarIn2 são dois macro-eventos semelhantes ao CarIn anteriormente apresentados.

Este macro-evento utiliza uma composição OR e permite mostrar que o modelo de compressão pode ser ainda maior, dependendo do que se pretende para modelar com macro-eventos.

No entanto, esta definição de evento, que seria adequada se todas as variáveis do sistema fossem booleanas, tem um problema, uma vez que permite a entrada de dois carros em simultâneo e só conta um. Para resolver este problema, a transição que modela a entrada de carros foi duplicada (uma associada a cada entrada), resultando no modelo apresentado na Figura 4.5 c),

4.3. Sistema de Recuperação para Segurança de um Drone

Neste exemplo pretende-se modelar um Sistema de recuperação de falhas de *software* de um *drone quadcopter*, com a estrutura representada na Figura 4.6 a).

O sistema é constituído por um controlador principal do *drone* e por um controlador alternativo de recuperação, que possui alimentação própria através de uma bateria dedicada.

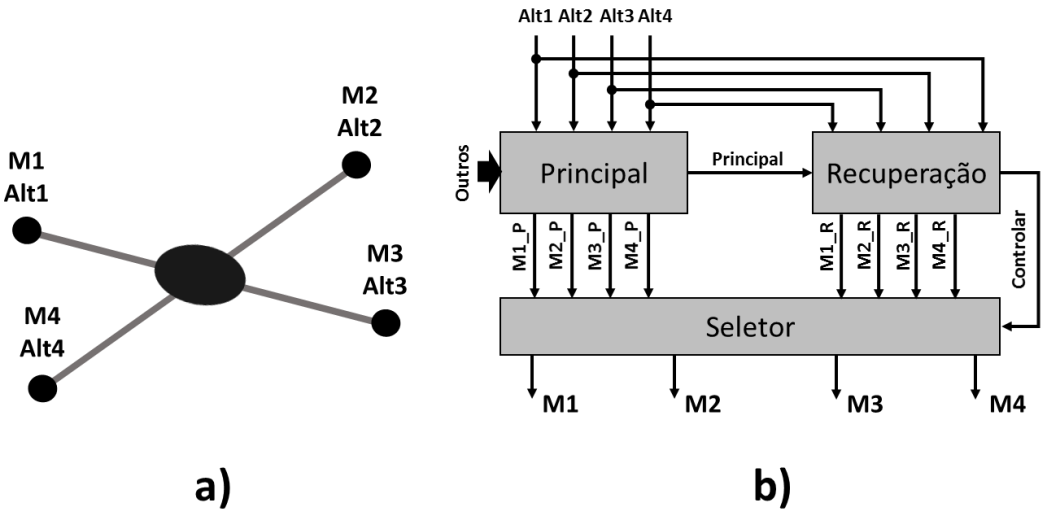


Figura 4.6: Sistema de recuperação de um *drone*. a) Representação do *drone*. b) Diagrama do sistema completo do controlador do *drone*.

O sistema de recuperação (Figura 4.6 b)) recebe como entradas um sinal que informa que o controlador principal está ativo e o valor da altitude de cada um dos motores. Como saídas tem um sinal que permite tomar o controlo dos motores e o valor dos quatro motores. Os quatro motores têm um valor entre 0 e 127, onde o valor zero define o motor parado e o valor 127 define a velocidade máxima.

Quando o valor do sinal que vem do controlador principal passa de um para zero, ou quando a variação da altitude de um dos motores for superior a 100, ou seja, se a velocidade a que um dos motores sobe ou desce for muito rápida, o recuperador toma o controlo dos motores. Quando o recuperador toma o controlo dos motores, aumenta ou diminui a velocidade de cada motor gradualmente até que a altitude de todos eles seja igual à média das altitudes, ou seja, que o *drone* esteja estabilizado. Quando isso acontecer, mantém o *drone* estabilizado à espera de ser resgatado.

O recuperador é uma forma de último recurso para garantir a não destruição do *drone*. Dessa forma, quando o recuperador passa a controlar o *drone*, não existe nenhuma forma de o sistema automaticamente voltar a sair do recuperador. O recuperador tomará conta do controlo até que o *drone* seja recuperado e o sistema verificado e reiniciado.

A Figura 4.7 mostra o modelo em RdP IOPT do controlador do recuperador.

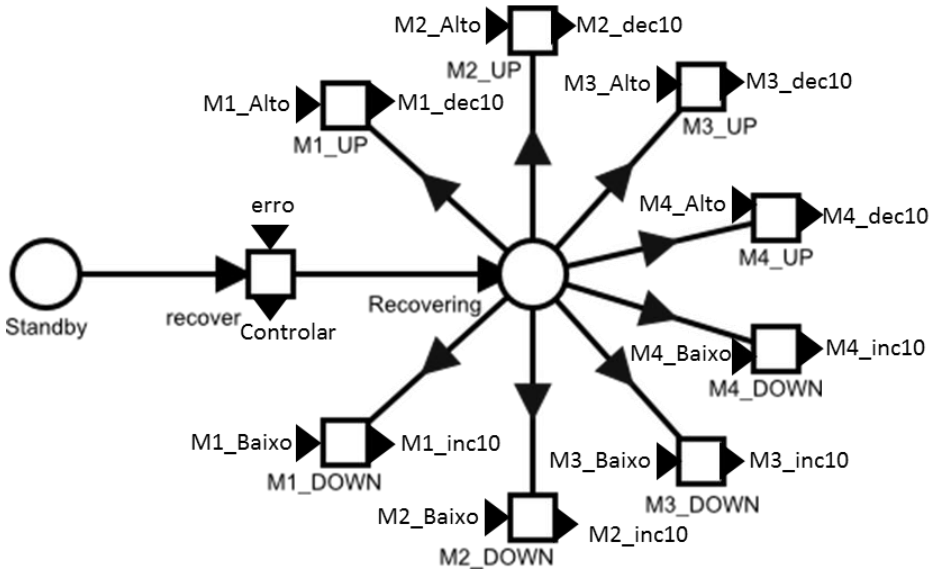


Figura 4.7: Modelo do sistema de recuperação.

O modelo representado na Figura 4.7 mostra o modelo simplificado do sistema, recebendo um conjunto de eventos discretos como entradas e saídas. Estes eventos são definidos nos outros dois blocos do modelo de interpretação de sinais.

Os modelos de interpretação e afetação de sinais são representados, respetivamente, na Tabela 4.2 e na Tabela 4.3

Tabela 4.2 – Modelo de interpretação de sinais do controlador de recuperação de um *drone*.

Entradas
Sinais lógicos M_Med = (M1+M2+M3+M4)/4 M1_dif = (M1 – M_Med) M2_dif = (M2 – M_Med) M3_dif = (M3 – M_Med) M4_dif = (M4 – M_Med)
Condições M1baixo = d(0) M1_dif < -5 M2baixo = d(0) M2_dif < -5 M3baixo = d(0) M3_dif < -5 M4baixo = d(0) M4_dif < -5 M1alto = d(0) M1_dif > 5 M2alto = d(0) M2_dif > 5 M3alto = d(0) M3_dif > 5 M4alto = d(0) M4_dif > 5
Eventos erro = (principal >to≤ 0) ∨ (d(1) M1 ≤ to> 100) ∨ (d(1) M2 ≤ to> 100) ∨ (d(1) M3 ≤ to> 100) ∨ (d(1) M4 ≤ to> 100)

Nas entradas é definido um sinal interno que calcula a altitude média do *drone*. São ainda definidos mais quatro sinais internos que calculam a diferença de altitude de cada motor para o valor médio.

São definidas oito condições (duas por motor) que analisam se cada um dos quatro motores (altitudes) está acima ou abaixo do valor médio.

Tabela 4.3 – Modelo de afetação de sinais do controlador de recuperação de um *drone*

Saídas
Eventos de Saída M1_inc10 => f(Alt1) = 2(Alt1) => W = 5 (rel) M2_inc10 => f(Alt2) = 2(Alt2) => W = 5 (rel) M3_inc10 => f(Alt3) = 2(Alt3) => W = 5 (rel) M4_inc10 => f(Alt4) = 2(Alt4) => W = 5 (rel) M1_dec10 => f(Alt1) = -2(Alt1) => W = 5 (rel) M2_dec10 => f(Alt2) = -2(Alt2) => W = 5 (rel) M3_dec10 => f(Alt3) = -2(Alt3) => W = 5 (rel) M4_dec10 => f(Alt4) = -2(Alt4) => W = 5 (rel)

Como saídas são definidos oito eventos de evolução guiada de saída. Estes eventos fazem o valor do motor aumentar ou diminuir de uma forma progressiva até ao valor inicial adicionado de dez, ou seja, evoluem segundo as funções $f(n)= 2n$ e $f(n)= -2n$ respetivamente durante uma janela de 5 passos de execução.

A solução apresentada é uma solução simplificada para o problema apresentado. Várias soluções podem ser adicionadas para tornar o modelo mais eficiente na recuperação do *drone*. Podem, por exemplo, ser adicionadas análises que permitem detetar variações continuadas, como por exemplo, se uma altitude se mantem maior que um valor N durante vários passos de execução, então tomar outras medidas, como aumentar a intensidade da recuperação. Tirando partido dos eventos de entrada atrasados, é possível adicionar, para cada motor, novas transições no modelo da Figura 4.7 que tenham como objetivo aumentar o incremento/decremento na velocidade do motor para o dobro no caso em que a diferença entre as altitudes dos motores não diminui nos primeiros três ciclos em que se faz a recuperação. Onde x representa o número do motor.

Essas transições teriam, cada uma, um evento de entrada do tipo:

Mxalto3 = d(0) Mx_dif > 5, delay=3

Que ocorre três passos de execução depois de o valor da diferença do motor x ser maior/menor que cinco.

A transição tem também um evento de saída do tipo:

Mx_inc25 => f(Alt1) = 5(Alt1) => W = 5 (rel)

Este evento aumenta o valor do motor nos cinco passos de execução seguintes guiado pela função $f(x) = 5x$ num total final de 25.

4.4. Discussão

Os modelos de interpretação e afetação de sinais apresentados neste trabalho são utilizados para desenvolver sistemas digitais, mais especificamente os que apresentam uma parte de interação do sistema com o ambiente relevante, tendo foco na análise e deteção de comportamentos nos sinais.

Estes modelos acrescentam um conjunto de conceitos que permitem a modelação do comportamento dos sinais de entrada e de saída. Esses conceitos podem ser utilizados individualmente, sem utilização de qualquer outro formalismo de modelação, como se apresenta no exemplo da ronda de um segurança por um bairro, onde um modelo de interpretação de sinais é utilizado sozinho para modelar o problema. Neste caso, as saídas do sistema são os eventos discretos gerados pelo modelo.

Estes conceitos podem ainda ser utilizados em adição a outro formalismo de modelação. Nesse caso, apesar de introdução da necessidade de conhecer outros conceitos de modelação, permitem retirar complexidade do outro modelo, permitindo uma modelação mais simples da parte de interpretação de comportamentos de sinais. Quando utilizado em conjunto com outro formalismo pode ainda ser utilizado apenas um dos modelos de interface (interpretação ou afetação de sinais) ou em conjunto com os dois.

Nos casos do controlo de acessos à zona histórica de uma cidade e ao sistema de recuperação de um *drone*, o modelo é dividido em mais que uma parte, dividindo o modelo representado num formalismo de modelação conhecido, dos modelos de interpretação e afetação de sinais. No controlo de acessos à zona histórica de uma cidade, apenas as entradas são modeladas com um modelo de interpretação de sinais. Na recuperação do *drone*, são modeladas a entrada com um modelo de interpretação de sinais e a saída com um modelo de afeta-

ção de sinais. Estes exemplos mostram uma variedade de formas de implementar um sistema usando os modelos definidos.

Por outro lado, nos exemplos 2 e 3, onde se usam os modelos de interface como parte integrante do modelo global composto pelo modelo representado num formalismo de modelação já conhecido e pelos MIS e MAS, verifica-se uma redução da dimensão do modelo, pois a parte da complexidade referente à análise do comportamento do sinal é modelada separadamente. Desta forma definindo vários modelos com legibilidade melhorada, em vez de um modelo complexo e difícil de interpretar.

No exemplo 1 são utilizados eventos compostos para detetar comportamentos associados a diferentes sinais. Estes comportamentos poderiam ser definidos usando alguns tipos de formalismos de modelação existentes, mas existem algumas características, como o tempo de vida, que não podem ser modeladas por alguns formalismos de modelação.

No exemplo 2, o modelo de interpretação de sinais é utilizado reduzindo a dimensão do modelo inicial com 18 nós (10 lugares + 8 transições) para um modelo com 4 nós (2 lugares + 2 transições).

No caso específico, mesmo modelo sem MIS não sendo muito grande, conseguiu-se uma redução superior a 75%, em número de nós. Em casos de modelos muito complexos, esta característica pode permitir a redução para um modelo que se torna legível por um utilizador humano.

No exemplo 3 é utilizado um conjunto de eventos de saída que permitem reduzir ainda mais o modelo, desta vez passando o tratamento dos sinais de saída para um modelo de afetação de sinais.

No caso do exemplo 3, é implementada a abordagem apresentada nas Figura 3.1, Figura 3.2 e Figura 3.3, dividindo a implementação em três modelos,

um de interpretação de sinais, um de modelação de fluxo e outro de afetação de sinais.

Analisando o exemplo 3 verifica-se ainda uma utilização de sinais internos que permite adicionar informação ao modelo, que não vem diretamente dos sinais recebidos, sem a necessidade de adição de sensores externos ou da adição de sinais de comunicação com o exterior. Por exemplo, apenas com um formalismo de modelação conhecido como por exemplo uma rede de Petri elementar ou uma Máquina de estados finita, caso se pretendesse analisar a velocidade com que o *drone* está a cair, ter-se-ia de adicionar um sensor de velocidade ou pelo menos um operador externo que devolvia a variação da posição. Usando os modelos de interface, que são criados especificamente para modelar a interpretação e afetação de sinais, evita-se essas adições e permite-se a análise de mais variáveis relativas aos sinais de entrada e de saída. Dessa forma permite-se tirar um maior partido da informação recebida e enviada pelo modelo.

Neste capítulo foram apresentados três exemplos de aplicação do MIS e do MAS. Foram ainda discutidas as vantagens e desvantagens da sua utilização. No capítulo seguinte (final) serão apresentadas as conclusões e os trabalhos relacionados a desenvolver futuramente.

5

Conclusões e Trabalho Futuro

Neste capítulo são apresentadas as conclusões a que se chegou com o desenvolvimento do trabalho apresentado neste documento, bem como o trabalho relacionado a desenvolver no futuro.

Este trabalho propõe uma abordagem para a modelação da interface de um sistema com o ambiente. Para isso são definidos conceitos que permitem a modelação da deteção de comportamentos nos sinais de entrada, bem como a geração de comportamentos nos sinais de saída. A estes modelos chamam-se, respetivamente, modelo de interpretação de sinais e modelo de afetação de sinais.

Os formalismos definidos são desenvolvidos especificamente para modelar a interpretação dos sinais de entrada e a afetação dos sinais de saída num sistema, através de análise de comportamentos dos sinais e definições de comportamentos, respetivamente, tornando-se dessa forma mais adequados à modelação destes comportamentos do que um formalismo desenvolvido para modelar o comportamento global de um sistema.

Estes modelos podem ser utilizados de forma independente, modelando um sistema por si só, usando apenas um modelo de entrada ou de saída. Podem

ainda ser utilizados como adição a um modelo representado num formalismo existente. Neste caso podendo ser utilizado apenas um modelo de entrada, um modelo de saída ou ambos. No caso em que se usam os três modelos, a adição dos modelos de interpretação de sinais e de afetação de sinais, permite uma estruturação do sistema de forma semelhante à definida numa arquitetura de interação humano-sistema, como por exemplo o *Model-View-Controller*.

A utilização de uma modelação baseada em três modelos permite a separação das complexidades específicas, resultando em modelos menos complexos e mais fáceis de interpretar. Além de se separar a complexidade em três, cada parte está modelada num formalismo específico para o efeito. Dessa forma permitindo uma modelação mais simples e intuitiva.

Ao adicionar sinais internos ao modelo, permite-se a criação de sinais através de funções específicas, permitindo a criação de conhecimento extra dentro do modelo.

Da mesma forma, a implementação de macro-condições e macro-eventos permite a definição de comportamentos compostos, o que permite a representação de comportamentos mais complexos constituídos por um conjunto de sinais de entrada.

Adicionando os tempos de vida aos macro-eventos é ainda possível adicionar os conceitos de distância temporal aos comportamentos compostos definidos.

Os eventos de saída, da mesma forma que os macro-eventos de entrada, permitem remover a sua representação explícita do modelo do sistema, mas permitem programar evoluções de determinados sinais, independentemente da evolução do restante modelo. Ou seja, é possível definir uma evolução para um

sinal de saída num determinado espaço de tempo que ocorrerá independentemente da evolução do modelo.

A implementação da ferramenta de edição, disponível em http://gres.uninova.pt/~rcr/events, bem como do gerador automático de código C associado, permitiu validar a utilização dos modelos de interpretação de sinais, bem como a sua integração num fluxo de desenvolvimento baseado em modelos.

Desta forma pode concluir-se que a tese inicialmente apresentada é válida. Adicionando os conceitos definidos, é possível melhorar a modelação de interação de um sistema com o ambiente, incluindo com a utilização dos formalismos de modelação conhecidos e com a implementação de arquiteturas especificas para a modelação da interação humano-sistema.

No trabalho desenvolvido, todos os pontos de análise de eventos e condições são definidos com base num valor constante k. Seria de interesse definir eventos e condições primitivas que permitissem analisar o cruzamento entre dois sinais, bem como a comparação do valor do sinal com o valor das suas diferenças.

Neste trabalho é apresentado um desenvolvimento centralizado dos três modelos. No futuro é de interesse desenvolver sobre como implementar o sistema em plataformas heterogenias ou fisicamente distribuídas.

Seria também de interesse desenvolver uma sintaxe gráfica mais fácil de usar na modelação quando se usa a ferramenta de edição, de forma a proporcionar uma modelação mais intuitiva.

A adição dos conceitos aos formalismos de modelação existentes é também uma ação de interesse, estando a ser desenvolvidos esforços no sentido de integrar os conceitos no ambiente de desenvolvimento de redes de Petri IOPT

nets. Esta inclusão dos conceitos no ambiente permitirá tirar proveito das ferramentas (devidamente adaptadas) de validação e verificação dos modelos.

Bibliografia

[1] H. R. Booher, *Handbook of Human Systems Integration*, no. v. 1. John Wiley & Sons, 2003.

[2] C. H. S. D. S. C. Technology, C. H. Factors, N. R. Council, R. W. Pew, and A. S. Mavor, *Human-System Integration in the System Development Process: A New Look*. National Academies Press, 2007.

[3] J. Von Neumann, W. Aspray, and A. W. Burks, *Papers of John von Neumann on computing and computer theory*. MIT Press, 1987.

[4] H. H. Goldstine and A. Goldstine, "The Electronic Numerical Integrator and Computer (ENIAC)," *Math. Tables Other Aids to Comput.*, vol. 2, no. 15, pp. 97–110, 1946.

[5] C. A. Mack, "Fifty Years of Moore's Law," *Semiconductor Manufacturing, IEEE Transactions on*, vol. 24, no. 2. pp. 202–207, 2011.

[6] "MDA. 'The Architecture of Choice for a Changing World,'" 2015. [Online]. Available: http://www.omg.org/mda/.

[7] G. Booch, J. Rumbaugh, and I. Jacobson, *The Unified Modeling Language User Guide*. Addison-Wesley, 2005.

[8] "UML Resource Page," 2015. [Online]. Available: http://www.uml.org/.

[9] J. Rumbaugh, I. Jacobson, and G. Booch, *Unified Modeling Language Reference Manual, The (2Nd Edition)*. Pearson Higher Education, 2004.

[10] F. B. Gilbreth and L. M. Gilbreth, "Process Charts-First Steps in Finding the One Best Way," *Am. Soc. Mech. Eng. (ASME), New York, NY*, 1921.

[11] J. Yang and H. Zhiyong, "An Improved Flowchart for Gabor Order Tracking," *Measuring Technology and Mechatronics Automation (ICMTMA), 2011 Third International Conference on*, vol. 1. pp. 414–419, 2011.

[12] E. Börger and R. F. Stärk, *Abstract State Machines: A Method for High-level System Design and Analysis*, vol. 14. Springer Berlin Heidelberg, 2003.

[13] D. Lee and M. Yannakakis, "Principles and methods of testing finite state machines-a survey," *Proceedings of the IEEE*, vol. 84, no. 8. pp. 1090–1123, 1996.

[14] D. Harel, "Statecharts: A visual formalism for complex systems," *Sci. Comput. Program.*, vol. 8, no. 3, pp. 231–274, 1987.

[15] M. El-Attar, H. Luqman, P. Karpati, G. Sindre, and A. L. Opdahl, "Extending the UML Statecharts Notation to Model Security Aspects," *Software Engineering, IEEE Transactions on*, vol. 41, no. 7. pp. 661–690, 2015.

[16] C. A. Petri, "Kommunikation mit Automaten (Communication with Automata)," Hamburg, 1962.

[17] C. Petri, "Fundamentals of a Theory of Asynchronous Information Flow," in *1st IFIP World Computer Congress*, 1962, pp. 386–390.

[18] R. David, "Grafcet: a powerful tool for specification of logic controllers," *IEEE Transactions on Control Systems Technology*, vol. 3, no. 3. pp. 253–268, 1995.

[19] R. David and H. Alla, *Petri Nets and Grafcet: Tools for Modelling Discrete Event Systems*. Prentice Hall, 1992.

[20] A. Visioli, M. Berenguel, P. B. de M. Oliveira, C. Mendes, L. Iria, E. J. S. Pires, and J. B. Cunha, "IFAC Workshop on Internet Based Control Education IBCE15E-GRAFCET+: An Internet Based Multimedia Tool Refined," *IFAC-PapersOnLine*, vol. 48, no. 29, pp. 111–116, 2015.

[21] S. Burbeck, *Applications Programming in Smalltalk-80: How to Use Model-View- Controller (MVC)*. Softsmarts, Incorporated, 1987.

[22] A. C. Bert, *Model-View-Adapter*. International Book Marketing Service Limited, 2012.

[23] M. Potel, *MVP: Model-View-Presenter, The Taligent Programming Model for C++ and Java*. 1996.

[24] E. Sorensen and M. I. Mihailesc, "Model-View-ViewModel (MVVM) Design Pattern using Windows Presentation Foundation (WPF) Technology," *MegaByte J.*, 2010.

[25] J. Coutaz, "PAC: an Implementation Model for Dialog Design," in *Interact'87 conference*, 1987, pp. 431–436.

[26] B. D. Haig, "Grounded theory as scientific method," *Philos. Educ.*, vol. 28, no. 1, pp. 1–11, 1995.

[27] R. Campos-Rebelo, A. Costa, and L. Gomes, "Analysis and Generation of Logical Signals for Discrete Events Behavioral Modeling," in *Technological*

Innovation for Cloud-Based Engineering Systems SE - 16, vol. 450, L. M. Camarinha-Matos, T. A. Baldissera, G. Di Orio, and F. Marques, Eds. Springer International Publishing, 2015, pp. 147–156.

[28] R. Campos-Rebelo, A. Costa, and L. Gomes, "On Structuring Events for IOPT Net Models," in *Technological Innovation for the Internet of Things SE - 25*, vol. 394, L. Camarinha-Matos, S. Tomic, and P. Graça, Eds. Springer Berlin Heidelberg, 2013, pp. 229–238.

[29] R. Campos-Rebelo, A. Costa, and L. Gomes, "Events for Human-System Interaction modeling with IOPT Petri nets," in *Human System Interactions (HSI), 2013 6th International Conference on*, 2013, pp. 56–61.

[30] R. Campos-Rebelo, A. Costa, and L. Gomes, "Elementary Events for Modeling of Human-System Interactions with Petri Net Models," vol. 423, L. Camarinha-Matos, N. Barrento, and R. Mendonça, Eds. Springer Berlin Heidelberg, 2014, pp. 219–226.

[31] R. Campos-Rebelo, A. Costa, and L. Gomes, "Enhanced Event Modeling for Human-System Interactions Using IOPT Petri Nets," in *Human-Computer Systems Interaction: Backgrounds and Applications 3*, Springer, 2014, pp. 39–50.

[32] R. Campos-Rebelo, A. Costa, and L. Gomes, "Event Life Time in Detection of Sequences of Events," *2015 IEEE Int. Conf. Ind. Technol.*, 2015.

[33] R. Campos-Rebelo, A. Costa, and L. Gomes, "Output events for human-system interaction modeling," *Human System Interactions (HSI), 2014 7th International Conference on*. pp. 261–266, 2014.

[34] L. Heide, "Shaping a technology: American punched card systems 1880-1914," *Ann. Hist. Comput. IEEE*, vol. 19, no. 4, pp. 28–41, 1997.

[35] A. Buchmann and B. Koldehofe, "Complex event processing," *it-Information Technol.*, vol. 51, no. 5, pp. 241–242, 2009.

[36] D. C. Luckham, *Event Processing for Business: Organizing the Real-Time Enterprise*. Wiley, 2011.

[37] D. Luckham, "The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems," in *Rule Representation, Interchange and Reasoning on the Web SE - 2*, vol. 5321, N. Bassiliades, G. Governatori, and A. Paschke, Eds. Springer Berlin Heidelberg, 2008, p. 3.

[38] W. Hu, W. Ye, Y. Huang, and S. Zhang, "Complex Event Processing in RFID Middleware: A Three Layer Perspective," in *Convergence and Hybrid Information Technology, 2008. ICCIT '08. Third International Conference on*, 2008, vol. 1, pp. 1121–1125.

[39] R. Bruns and J. Dunkel, *Event-Driven Architecture*. Springer London, Limited, 2010.

[40] D. C. Luckham and B. Frasca, "Complex event processing in distributed systems," *Comput. Syst. Lab. Tech. Rep. CSL-TR-98-754. Stanford Univ. Stanford*, vol. 28, 1998.

[41] H. Taylor, A. Yochem, L. Phillips, and F. Martinez, *Event-Driven Architecture: How SOA Enables the Real-Time Enterprise*. Pearson Education, 2009.

[42] S. Overbeek, B. Klievink, and M. Janssen, "A Flexible, Event-Driven, Service-Oriented Architecture for Orchestrating Service Delivery," *Intelligent Systems, IEEE*, vol. 24, no. 5. pp. 31–41, 2009.

[43] D. C. Luckham, J. J. Kenney, L. M. Augustin, J. Vera, D. Bryan, and W. Mann, "Specification and analysis of system architecture using Rapide," *Softw. Eng. IEEE Trans.*, vol. 21, no. 4, pp. 336–354, 1995.

[44] D. C. Luckham, *Rapide: A language and toolset for simulation of distributed systems by partial orderings of events*. Computer Systems Laboratory, Stanford University, 1996.

[45] D. C. Luckham and J. Vera, "An event-based architecture definition language," *Softw. Eng. IEEE Trans.*, vol. 21, no. 9, pp. 717–734, 1995.

[46] L. Yuan, D. Xu, G. Ge, and M. Zhu, "Study on distributed complex event processing in Internet of Things based on query plan," *Cyber Technology in Automation, Control, and Intelligent Systems (CYBER), 2015 IEEE International Conference on*. pp. 666–670, 2015.

[47] K. Thulasiraman and M. N. S. Swamy, *Graphs: theory and algorithms*. John Wiley & Sons, 2011.

[48] P. Benioff, "Quantum mechanical models of Turing machines that dissipate no energy," *Phys. Rev. Lett.*, vol. 48, no. 23, p. 1581, 1982.

[49] R. P. Feynman, "Simulating physics with computers," *Int. J. Theor. Phys.*, vol. 21, no. 6/7, pp. 467–488, 1982.

[50] D. Deutsch, "Quantum theory, the Church-Turing principle and the universal quantum computer," in *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 1985, vol. 400, no. 1818, pp. 97–117.

[51] P. W. Shor, "Algorithms for quantum computation: Discrete logarithms and factoring," in *Foundations of Computer Science, 1994 Proceedings., 35th Annual Symposium on*, 1994, pp. 124–134.

[52] M. A. Nielsen and I. L. Chuang, *Quantum computation and quantum information*. Cambridge university press, 2010.

[53] J. J. Grattage, "A functional quantum programming language." University of Nottingham, 2006.

[54] "State Machines & Tools--Quantum Leaps." [Online]. Available: http://www.state-machine.com/. [Accessed: 16-Nov-2015].

[55] S. Chakravarthy, V. Krishnaprasad, E. Anwar, and S.-K. Kim, "Composite events for active databases: Semantics, contexts and detection," in *VLDB*, 1994, vol. 94, pp. 606–617.

[56] S. Gatziu and K. R. Dittrich, "Detecting composite events in active database systems using Petri nets," *Research Issues in Data Engineering, 1994. Active Database Systems. Proceedings Fourth International Workshop on*. pp. 2–9, 1994.

[57] L. Gomes and J. P. Barros, "Automated Code Generation From Petri Nets Based System Specification," in *Proceedings of the 5" World WSESfiEEE Conference on COMPUTERS, Greece*, 2001.

[58] D. J. Duke and M. D. Harrison, "Event model of human-system interaction," *Softw. Eng. J.*, vol. 10, no. 1, pp. 3–12, 1995.

[59] L. Gomes and J. Lourenco, "Rapid Prototyping of Graphical User Interfaces for Petri-Net-Based Controllers," *Ind. Electron. IEEE Trans.*, vol. 57, no. 5, pp. 1806–1813, 2010.

[60] A. Leff and J. T. Rayfield, "Web-application development using the Model/View/Controller design pattern," in *Enterprise Distributed Object Computing Conference, 2001. EDOC '01. Proceedings. Fifth IEEE International*, 2001, pp. 118–127.

[61] C. Girault and R. Valk, *Petri Nets for Systems Engineering: A Guide to Modeling, Verification, and Applications*. Springer, 2002.

[62] F. Balarin, *Hardware-Software Co-Design of Embedded Systems: The Polis Approach*. Kluwer Academic, 1997.

[63] L. Gomes, J. P. Barros, and A. Costa, "Modeling Formalisms for Embedded System Design," in *Embedded Systems Handbook*, CRC Press, 2005, pp. 5–34.

[64] R. Miles and K. Hamilton, *Learning UML 2.0*. O'Reilly Media, 2008.

[65] B. P. Douglass, *Doing Hard Time: Developing Real-Time Systems With Uml, Objects, Frameworks, and Patterns*. Addison-Wesley, 1999.

[66] B. P. Douglass, *Real-time UML: developing efficient objects for embedded systems*. Addison-Wesley, 2000.

[67] B. P. Douglass, *Real Time UML Workshop for Embedded Systems*. Elsevier Science, 2011.

[68] L. Lavagno, A. Sangiovanni-Vincentelli, and E. Sentovich, "Models of Computation for Embedded System Design," in *in System-Level Synthesis*, Kluwer Academic Publishers, 1998, pp. 45–102.

[69] M. Sgroi, L. Lavagno, and A. Sangiovanni-Vincentelli, "Formal Models for Embedded System Design," *IEEE Des. Test*, vol. 17, no. 2, pp. 14–27, 2000.

[70] S. Edwards, L. Lavagno, E. A. Lee, and A. Sangiovanni-Vincentelli, "Design of embedded systems: formal models, validation, and synthesis," *Proc. IEEE*, vol. 85, no. 3, pp. 366–390, 1997.

[71] D. Harel, "On visual formalisms," *Commun. ACM*, vol. 31, no. 5, pp. 514–530, 1988.

[72] J. Desel and W. Reisig, "Place/transition Petri Nets," in *Lectures on Petri Nets I: Basic Models*, vol. 1491, W. Reisig and G. Rozenberg, Eds. Springer Berlin Heidelberg, 1998, pp. 122–173.

[73] K. Jensen, *Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use*. Springer, 2003.

[74] I. S. O. ISO, "5807: 1985 Information processing-Documentation symbols and conventions for data, program and system flowcharts, program network charts and system resources charts," *Geneva ISO*, 1985.

[75] J. N. Manzano, "Revisão e Discussão da Norma ISO 5807-1985 (E)," *São Paulo, São Paulo*, 2004.

[76] E. F. Moore, "Gedanken-experiments on sequential machines," *Autom. Stud.*, vol. 34, pp. 129–153, 1956.

[77] G. H. Mealy, "A method for synthesizing sequential circuits," *Bell Syst. Tech. J.*, vol. 34, no. 5, pp. 1045–1079, 1955.

[78] "Petri Nets World," 2015. [Online]. Available: http://www.informatik.uni-hamburg.de/TGI/PetriNets/.

[79] C. A. Petri, *Grundsätzliches zur Beschreibung diskreter Prozesse (Fundamentals on the description of discrete processes)*. Rheinisch-Westfälisches Institut für, Instrumentelle Matematik, 1967.

[80] L. Gomes and J.-P. J. P. Barros, "Structuring and Composability Issues in Petri Nets Modeling," *IEEE Trans. Ind. Informatics*, vol. 1, no. 2, pp. 112–123, May 2005.

[81] R. Zurawski and Z. MengChu, "Petri nets and industrial applications: A tutorial," *Ind. Electron. IEEE Trans.*, vol. 41, no. 6, pp. 567–583, 1994.

[82] J. L. Peterson, "Petri Nets," *ACM Comput. Surv.*, vol. 9, no. 3, pp. 223–252, 1977.

[83] T. Murata, "Petri nets: Properties, analysis and applications," *Proc. IEEE*, vol. 77, no. 4, pp. 541–580, 1989.

[84] L. Bernardinello and F. De Cindio, "A survey of basic net models and modular net classes," in *Advances in Petri Nets 1992*, Springer, 1992, pp.

304–351.

[85] R. David and H. Alla, “Non-Autonomous Petri Nets,” in *Discrete, Continuous, and Hybrid Petri Nets SE - 3*, Springer Berlin Heidelberg, 2010, pp. 61–116.

[86] M. Adamski, “A rigorous design methodology for reprogrammable logic controllers,” in *International Workshop in Discrete-Event System Design–DESDes*, 2001, vol. 1, pp. 53–60.

[87] M. Adamski and J. L. Monteiro, “From interpreted Petri net specification to reprogrammable logic controller design,” in *Industrial Electronics, 2000. ISIE 2000. Proceedings of the 2000 IEEE International Symposium on*, 2000, vol. 1, pp. 13–19.

[88] M. Silva, *Las redes de Petri en la Autom{á}tica y la Inform{á}tica*. AC, 1985.

[89] M. Moalla, J. Pulou, and J. Sifakis, “Synchronized petri nets: A model for the description of non-autonomous sytems,” *Math. Found. Comput. Sci. 1978*, pp. 374–384, 1978.

[90] C. Ramchandani, “ANALYSIS OF ASYNCHRONOUS CONCURRENT SYSTEMS BY TIMED PETRI NETS,” Massachusetts Institute of Technology, Cambridge, MA, USA, 1974.

[91] R. Eshuis and J. Dehnert, “Reactive petri nets for workflow modeling,” *Appl. Theory Petri Nets 2003*, pp. 296–315, 2003.

[92] L. F. dos S. Gomes, “Redes de Petri reactivas e hierárquicas-integração de formalismos no projecto de sistemas reactivos de tempo-real,” 1997.

[93] L. Gomes, J. P. Barros, A. Costa, and R. Nunes, “The Input-Output Place-Transition Petri Net Class and Associated Tools,” in *Industrial Informatics, 2007 5th IEEE International Conference on*, 2007, vol. 1, pp. 509–514.

[94] “Model-View-Controller,” 2014. [Online]. Available: http://msdn.microsoft.com/en-us/library/ff649643.aspx.

[95] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Pearson Education, 1994.

[96] “Java SE Application Design With MVC,” 2014. [Online]. Available: http://www.oracle.com/technetwork/articles/javase/mvc-136693.html.

[97] “Concepts in Objective-C Programming - Model-View-Controller,” 2014. [Online]. Available: http://developer.apple.com/library/ios/#documentation/general/conceptual/CocoaEncyclopedia/Model-View-Controller/Model-View-Controller.html.

[98] S. A. Zamudio, R. Santaolaya, and O. G. Fragoso, “Restructuring Object-Oriented Frameworks to Model-View-Adapter Architecture,” *Lat. Am.*

Trans. IEEE (Revista IEEE Am. Lat., vol. 10, no. 4, pp. 2010–2016, 2012.

[99] A. Goldberg and D. Robson, *Smalltalk-80: the language and its implementation*. Addison-Wesley Longman Publishing Co., Inc., 1983.

[100] "Model View Presenter," 2014. [Online]. Available: http://msdn.microsoft.com/en-us/magazine/cc188690.aspx.

[101] "Large scale application development and MVP," 2014. [Online]. Available: https://developers.google.com/web-toolkit/articles/mvp-architecture.

[102] M. Fowler, "GUI Architectures," 2006. [Online]. Available: http://www.martinfowler.com/eaaDev/uiArchs.html.

[103] D. Greer, "Interactive Application Architecture Patterns," 2007. [Online]. Available: http://aspiringcraftsman.com/2007/08/25/interactive-application-architecture/.

[104] "Model-View-Presenter Pattern," 2014. [Online]. Available: http://msdn.microsoft.com/en-us/library/ff647543.aspx.

[105] "Implementing the Model-View-ViewModel Pattern," 2014. [Online]. Available: http://msdn.microsoft.com/en-us/library/ff798384.aspx.

[106] J. Gossman, "Introduction to Model/View/ViewModel pattern for building WPF apps," *Viitattu 14.10. 2013. Saatavissa http//blogs. msdn. com/b/johngossman/archive/2005/10/08/478683. aspx*, 2005.

[107] "WPF Apps With The Model-View-ViewModel Design Pattern," 2014. [Online]. Available: http://msdn.microsoft.com/en-us/magazine/dd419663.aspx.

[108] C. Anderson, "The Model-View-ViewModel (MVVM) Design Pattern," in *Pro Business Applications with Silverlight 5*, Apress, 2012, pp. 461–499.

[109] F. Buschmann, *Pattern oriented software architecture: a system of patters*. John Wiley&Sons, 1996.

[110] J. Coutaz, "PAC-ing the architecture of your user interface," in *Proceedings of the 4th Eurographics Workshop on Design, Specification and Verification of Interactive Systems*, 1997, pp. 15–32.

[111] A. A. Kumar, *DIGITAL SIGNAL PROCESSING*. PHI Learning, 2014.