Linguagens de Modelagem

Wu Shin-Ting

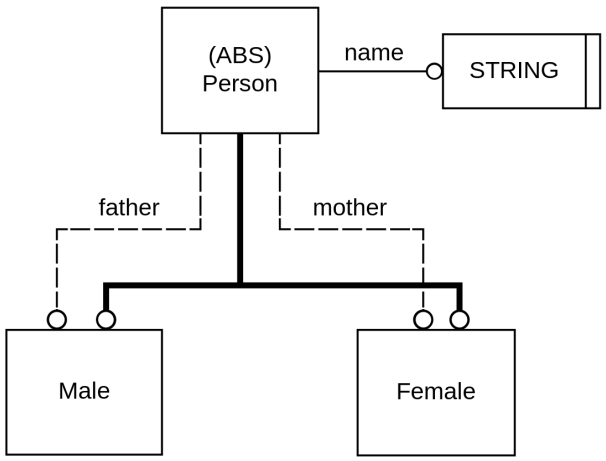
15 de Julho de 2022

1 Introdução

Uma linguagem de modelagem é um meio sistemático de comunicar ideias, conceitos, informações e/ou conhecimento através de notações padroniadas, de maneira que todas as partes interessadas tenham o mesmo entendimento do objeto de interesse. Tais notações podem ser textuais, gráficas ou ambas, desde que as suas regras de interpretação sejam claras, objetivas e precisas. Nas linguagens do tipo gráfico são usados elementos gráficos para representar os conceitos e as relações entre eles, enquanto nas linguagens do tipo textual são definidas uma série de símbolos léxicos, conhecidos como **tokens**, e de palavras reservadas para construir expressões interpretáveis. EXPRESS é um exemplo de linguagem de modelagem de dados que suporta ambas as notações. Fig. 1 mostra a modelagem da relação dos membros de uma família usando o tipo textual e gráfico de EXPRESS.

```
SCHEMA Family;

ENTITY Person
   ABSTRACT SUPERTYPE OF (ONEOF (Male,
Female));
     name: STRING;
     mother: OPTIONAL Female;
     father: OPTIONAL Male;
END_ENTITY;

ENTITY Female
   SUBTYPE OF (Person);
END_ENTITY;

ENTITY Male
   SUBTYPE of (Person);
END_ENTITY;

END_SCHEMA;
```

(a) Tipo textual



(b) Tipo gráfico

Figura 1: EXPRESS: uma linguagem de modelagem de dados (Fonte: [1]).

As linguagens de modelagem são usadas não só para especificar os requisitos de um projeto, como também para representar a estrutura e o comportamento do sistema projetado. Sejam do tipo textual ou gráfico, elas devem ser capazes de representar, de forma simples, legível, e preferencialmente universal, para as partes interessadas, todas as ideias e conhecimentos envolvidos ao longo do desenvolvimento do projeto. Elas devem ser formais, ou seja serem sintaticamente bem formadas obedecendo

um conjunto específico de regras gramaticais pré-definidas, a fim de facilitar uma possível automação na verificação e validação dos modelos desenhados, na simulação das soluções concebidas e na implementação de um sistema físico. Embora seja desejável tal automação, vale ressaltar que é ainda um tema de pesquisa. Uma **linguagem de modelagem não é necessariamente uma linguagem de programação**.

No domínio de *hardware* destacamos a Linguagem de Descrição de *Hardware* de Circuitos Integrados com Altíssima Velocidas (*VHSIC Hardware Description Language* ou VHDL) [2]. Ela é amplamente usada nas indústrias para descrever, no formato textual, a estrutura, o fluxo de dados e o comportamento de um sistema digital a diferentes níveis de abstração. Foi originalmente desenvolvida pelo Departamento de Defesa Americana em 1981, no contexto do programa VHSIC, e padronizada pela IEEE em 1987. Hoje em dia é aplicada na programação de tecnologias programáveis, como Arranjos de Porta Programável em Campo (*Field Programmable Gate Array* ou FPGA) ou Circuitos Integrados para Aplicação Específica (*Application-Specific Integrated Circuit* ou ASIC). Diversas ferramentas de automatização de projetos de circuitos integrados (*Electronic Design Automation* ou EDA) suportam a simulação das funcionalidades dos circuitos descritos por VHDL e conseguem compilar e mapear uma grande parte das entidades descritas em componentes físicos [3].

A linguagem de modelagem mais popular em Engenharia de *Software* e na Gerência de Informação e Gestão de Processos de Negócios é a Linguagem de Modelagem Unificada (*Unified Modeling Language* ou UML). Almejando uma padronização na linguagem de modelagem de *software* de sistemas concorrentes e distribuídos complexos, o Grupo de Gerenciamento de Objetos (*Object Management Group* ou OMG), um consórcio internacional de empresas, abriu uma solicitação de proposta (*Request for Proposal* ou RFP) em 1996. Tal padronização propiciaria a automatização da produção de *software*. A resposta foi a versão UML 0.9. Ela sucedeu ao trabalho em *Rational Software Corporation* da unificação das notações propostas predominantemente por Grady Booch para programação orientada a objetos usando a linguagem de programação ADA, originalmente concebida para sistemas em tempo real, James Rumbaughh para técnica de modelagem e projeto orientado a objetos (*Object-Modeling Technique* ou OMT) e Ivar Jacobson para Casos de Uso (*Object-Oriented Software Engineering* ou OOSE). Em 2000, a UML foi aprovada como padrão pelo OMG. Hoje em dia é gerenciada como um padrão de fato da indústria pelo OMG.

A linguagem expressa os requisitos, as estruturas e os comportamentos de um sistema através de notações diagramáticas. A versão atual da UML (versão 2.5.1) dispõe de 21 diagramas para representar as mais diversas visões (*views*) de um mesmo sistema, atendendo uma grande variedade de perfis das partes interessadas. Além disso, UML suporta mecanismos de extensibilidade e de especialização para ampliar os conceitos básicos. Vale ressaltar, porém, que o princípio de Pareto, ou regra 80/20, se aplica no uso dos diagramas de UML. De acordo com Grady Booch, "*For 80% of all software only 20% of UML is needed.*" Fig. 2 apresenta uma visão geral de UML 2.2 com 14 diagramas.

O uso de microcontroladores altamente programáveis na implementação de sistemas embarcados cada vez mais complexos e a inclusão do diagrama de perfil (*p*rofile diagram) a partir da versão 2.0 [5] têm atraído atenção da comunidade de sistemas embarcados para o potencial da UML na modelagem, validação, simulação e síntese do *software* embarcado em tempo real [6, 7, 8]. Acredita-se que a combinação do uso de plataformas programáveis com o uso de UML para desenvolvimento de *software* para tais plataformas ampliaria os ganhos efetivos em implementação e produtividade. Essa linha de pesquisa iniciou com a aplicação das ferramentas de Engenharia de *Software* Assistida por Computador (*Computer Aided Software Engineering* ou CASE) na geração de códigos para microcontroladores de sistemas embarcados a partir de modelos em UML [9].

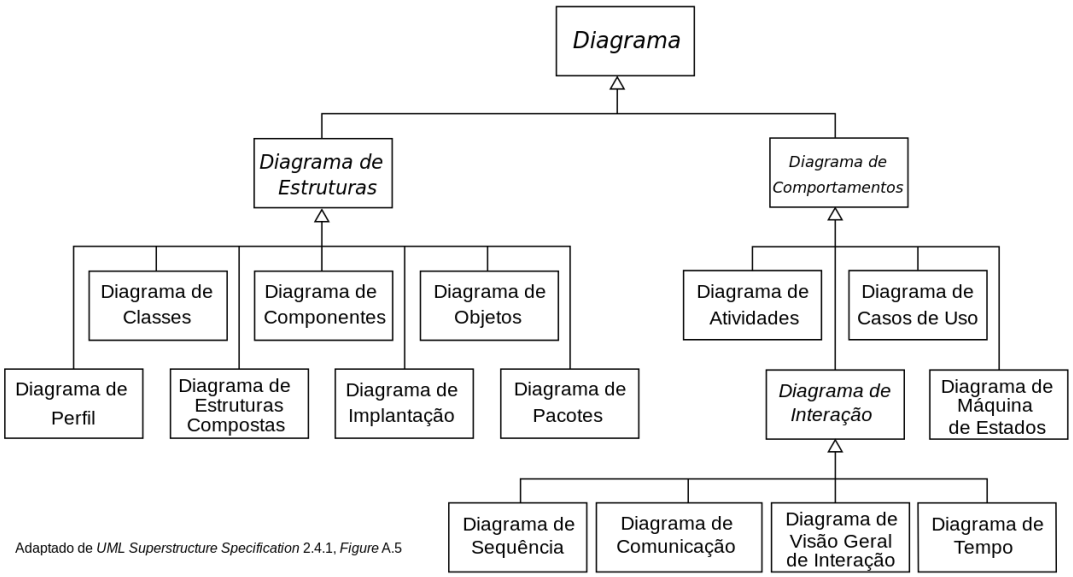


Figura 2: UML 2.2: 14 diagramas para representar a estrutura e o comportamento de um sistema (Fonte: [4]).

Apesar dos problemas identificados na UML para representar heterogeneidade e complexidade dos constituintes de um sistema embarcado, principalmente as restrições relacionadas com *hardware* (tempo de execução, memória e energia) [9, 10], os diagramas de classes, de implantação, de componentes, de sequência, de máquina de estados, de atividades e de casos de uso têm sido satisfatórios para descrever a estrutura e o comportamento de uma grande parte dos projetos de sistemas embarcados. Este documento tem como objetivo dar uma breve introdução aos 14 diagramas de UML, apresentar as notações e representações gráficas dos 7 diagramas mais usados e ilustrar o uso desses 7 diagramas na modelagem de um problema real.

2 UML 2.2

Nesta seção faremos uma breve descrição dos 14 tipos de diagramas definidos na versão UML 2.2 [11]. Embora a versão atual de UML seja 2.5 [12], optamos por apresentar um subconjunto dos seus diagramas presentes na versão 2.2 pela frequência de uso. Os 14 tipos de diagramas provêem uma ampla paleta de notações que cobrem diferentes necessidades para a maioria dos projetos de *software*. Remetemos os interessados em conhecer todos os 21 diagramas da versão 2.5 à apostila em português do Prof. Gudwin [13].

Os 14 diagramas especificados na versão 2.2 são divididos em 2 categorias como mostra a Fig. 2:

- 7 tipos de **diagramas de estruturas** para representar informação estrutural, e
- 7 tipos de **diagramas de comportamento** para modelar o comportamento do sistema sob o ponto de vista tanto de interações entre seus elementos quanto de transformações dos dados retidos no sistema durante a execução do programa.

2.1 Diagramas de Estruturas

Os **diagramas de classes** (*class diagrams*) descrevem as estruturas de um sistema, mostrando as classes do sistema e as relações estáticas entre elas. As classes são caracterizadas pelos seus atributos (dados) e comportamentos (operações, funções ou métodos). Distinguem-se 6 tipos de relações: **associação**, **herança**, **realização**, **dependência**. **agregação** e **composição**. Quando as classes correspondem aos componentes de implementação de um sistema, os diagramas passam a ser chamados de **diagramas de componentes** (*component diagrams*).

As instâncias de uma classe são denominadas **objetos**. Os detalhes da estrutura e dos dados de uma instância, como também as suas relações estáticas com outras instâncias, num ponto especfico de tempo, podem ser modelados através dos **diagramas de objetos** (*object diagrams*). UML dispõe também de diagramas de pacotes e de estruturas compostas para modelar os elementos e as suas relações estáticas em projetos de média e larga escala. Os **diagramas de pacotes** (*package diagrams*) simplificam a visão dos elementos e suas relações através de agrupamentos hierárquicos. Os módulos atômicos são usualmente os diagramas de classe. E os **diagramas de estruturas compostas** (*composite structure diagrams*) permitem detalhar a estrutura interna das partes de uma classe e as suas interações, proporcionando um entendimento melhor do comportamento da classe.

Os **diagramas de implantação** (*deployment diagrams*) mostram a configuração de nós (*nodes*) de processamento em tempo de execução e dos componentes-participante. É um diagrama estrutural usado na modelagem dos aspectos físicos de um sistema, como a topologia de *hardware* e a implantação física dos componentes. Finalmente, os **diagramas de perfil** (*profile diagrams*) fornecem um mecanismo de extensão genérico para customizar modelos UML para domínios e plataformas específicos. Tem sido investigado pela comunidade de embarcados para modelagem de projetos de *software* embarcado [9, 7, 8].

2.2 Diagramas de Comportamento

A princial forma de especificar os requisitos de um sistema sob a perspectiva de um usuário final é através de **diagrama de casos de uso** (*use case diagrams*). Os casos de uso especificm o comportamento esperado (o quê), e não o método para concretizá-lo (como). Eles nos ajudam a projetar o comportamento de um sistema visível externamente. Os **diagramas de atividades** (*activity diagrams*), por sua vez, descrevem o fluxo de atividades a serem realizadas ao longo do processo de um sistema para proporcionar a visão esperada por um usuário. Esses diagramas são análogos aos fluxogramas. Na execução das atividades, um sistema pode demandar diferentes interações entre os elementos de um sistema e passar por diferentes estados, retendo ou processando diferentes tipos de dados. Os **diagramas de interação** (*interaction diagrams*) capturam o comportamento interativo (estímulo e resposta) entre os objetos de um sistema durante a execução do sistema, enquanto a representação dos estados pelos quais o sistema pode passar e a representação das transições entre esses estados é contemplada pelos **diagramas de máquina de estados** (*state machine diagrams*).

Entre os diagramas de interações, distinguem-se ainda quatro subtipos: diagrama de comunicação, diagrama de sequência, diagrama de tempo e diagrama de visão geral de interação. Os **diagramas de comunicação** (*communication diagrams*), ou de colaboração, mostram as informações (ou mensagens) transferidas entre os elementos do diagrama de objetos de um sistema. Os **diagramas de sequência** (*sequence diagrams*) detalham as interações entre os objetos, sequenciando ao longo de uma linha de tempo as trocas de mensagens. Os instantes de tempo em que os eventos ocorrem e os intervalos de duração dos estados de um sistema são mostrados em **diagramas de tempo** (*timing diagrams*). E,

por fim, os **diagramas de visão geral de interação** (*interaction overview diagrams*) combinam os diagramas de atividades e os diagramas de interação de forma a detalhar as interações entre os elementos do sistema na execução de cada atividade.

2.3 Grau de Popularidade

A popularidade de UML aumentou muito com o desenvolvimento de uma série de ferramentas de suporte ao desenho desses diagramas [14, 15] e/ou a verificação e geração de códigos-fonte a partir deles [16, 17]. Algumas dessas ferramentas são abertamente disponíveis na *internet*. Reggio et al. fizeram um levantamento da popularidade dos 14 tipos de diagramas em função da frequência de ocorrência desses diagramas numa vasta busca pela *internet* [17]. O gráfico de barras na Fig. 3 sintetiza o resultado desse levantamento. Aplicando a classificação proposta por eles de considerar diagramas de amplo uso aqueles que aparecem em mais de 60% das fontes de UML consultadas e de pouco uso aqueles que aparecem em menos de 40% das fontes visitadas, podemos considerar como de maior uso os diagramas de classes, de atividades, de sequência, de casos de uso, de máquina de estados, de componentes e de implantação.

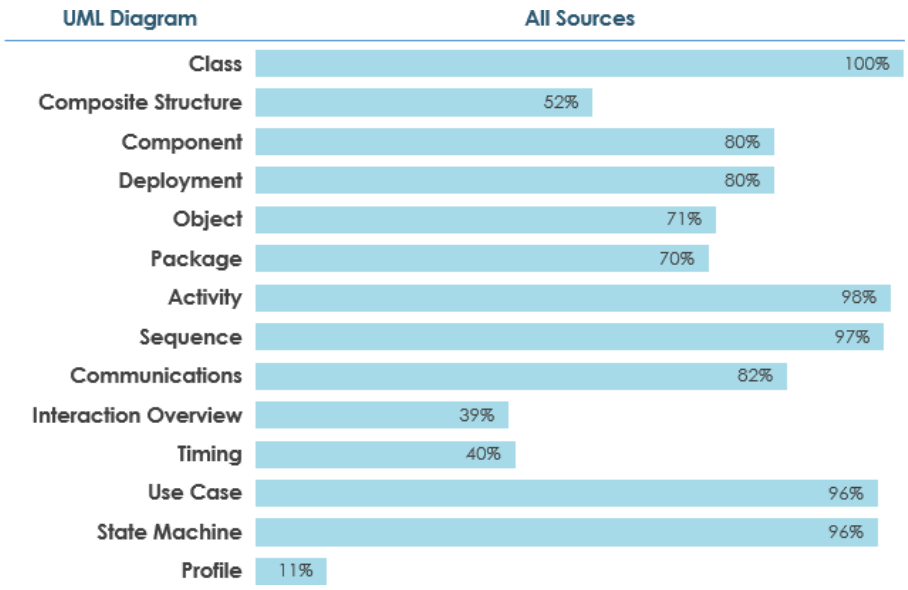


Figura 3: UML 2.2: popularidade dos 14 tipos de diagramas (Fonte: [18]).

3 Notações em UML

UML é uma linguagem de modelagem gráfica. Ela se tornou popular por suas notações diagramáticas. O uso correto dessas notações é muito importante para transformarmos as descrições de um problema e nossas ideias de solução num modelo completo e significativo. O entendimento dessas notações nos ajuda também a interpretar uma grande variedade de modelos baseados em UML disponíveis. Para facilitar a diferenciação dos elementos com propriedades comuns em diagramas de UML, UML dispõe do mecanismo de extensibilidade denominado **estereótipo**, através dele cria-se novos tipos de elementos de modelagem cujos nomes ficam entre colchetes de setas duplas. Pode-se também assosciar

as descrições de texto que complementam a informação contida num diagrama com uso de **notas**, representadas por retângulos com o canto direito superior dobrado.

Nesta seção apresentamos sucintamente as notações diagramáticas dos 7 tipos de diagramas mais populares que podem nos ajudar a descrever um sistema embarcado: os diagramas de classes, de componentes, de implantação, de sequência, de máquina de estados, de atividades e de casos de uso.

3.1 Diagramas de Classes

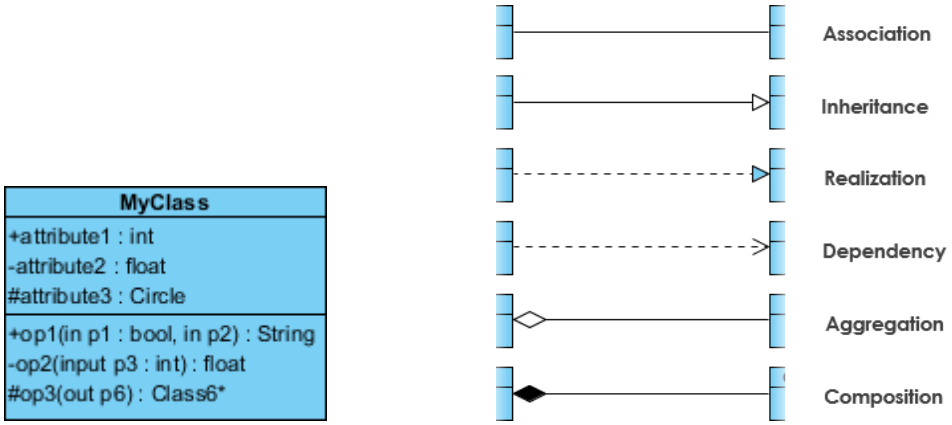
Graficamente, os diagramas de classes são constituídos de notações de classes e de relações entre elas. A notação completa de uma classe contém 3 partes (Fig. 4a): nome, atributos e operações, ou métodos, da classe. A parte mandatória é o nome da classe. Os atributos, na segunda parte, são usualmente mapeados em variáveis de uma linguagem de programação. Portanto, é comum que os atributos sejam seguidos de tipos de dados. Os métodos, na terceira parte, correspondem às funções ou rotinas que a classe disponibiliza. É comum que essas rotinas sejam seguidas de tipos de dados que elas retornam. Opcionalmente, pode-se especificar a visibilidade dos atributos e dos métodos através de 4 símbolos pré-fixados:

+: pública (global),

-: privada (local),

#: protegida (visibilidade extensível às sub-classes),

∼: pacote (visibilidade extensível às classes do mesmo pacote).



(a) Classes (b) Ralações

Figura 4: Representações gráficas de notações em diagramas de classes (Fonte: [18]).

As classes podem se relacionar através de 5 tipos de relações: (Fig. 4b):

- generalização (herança): indica a especialização de uma classe em relação a uma outra.
- associação simples: indica uma conexão lógica entre as instâncias de duas classes.
- agregação: indica a constituição de uma classe a partir de outras classes sem dependência em existência.

- composição: indica a constituição de uma classe a partir de outras classes com dependência em existência.
- dependência: indica a dependência entre os estados de duas classes.

Fig. 5 ilustra a modelagem da descrição de uma classe de janelas Window por um diagrama de classes. Ela é capaz de processar eventos e de renderizar diferentes formas geométricas. É uma especialização da classe Frame e, ao mesmo tempo, derivam-se dela duas outras classes Console Window e Dialog Box. Ficando Console Window na interface entre o sistema e o seu ambiente, ela é classificada como **classe de fronteira** (*boundary class*). Observe ainda uma **nota** no canto superior direito, contendo um comentário a respeito da classe de janela.

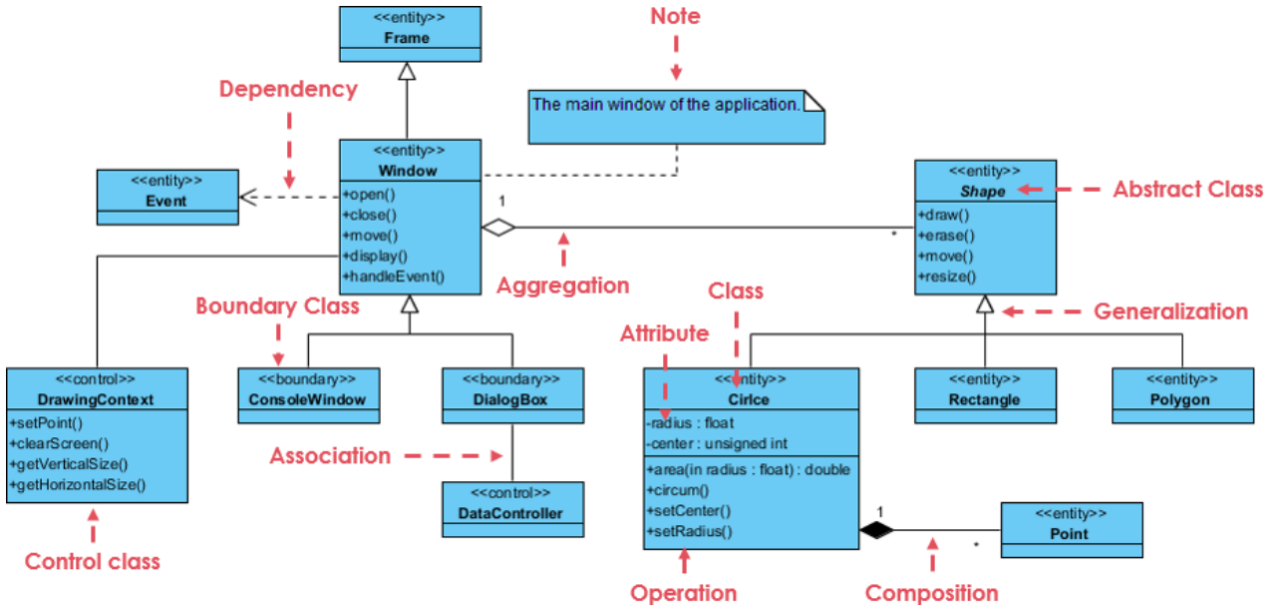


Figura 5: Modelagem de uma janela de exibição por um diagrama de classes (Fonte: [19]).

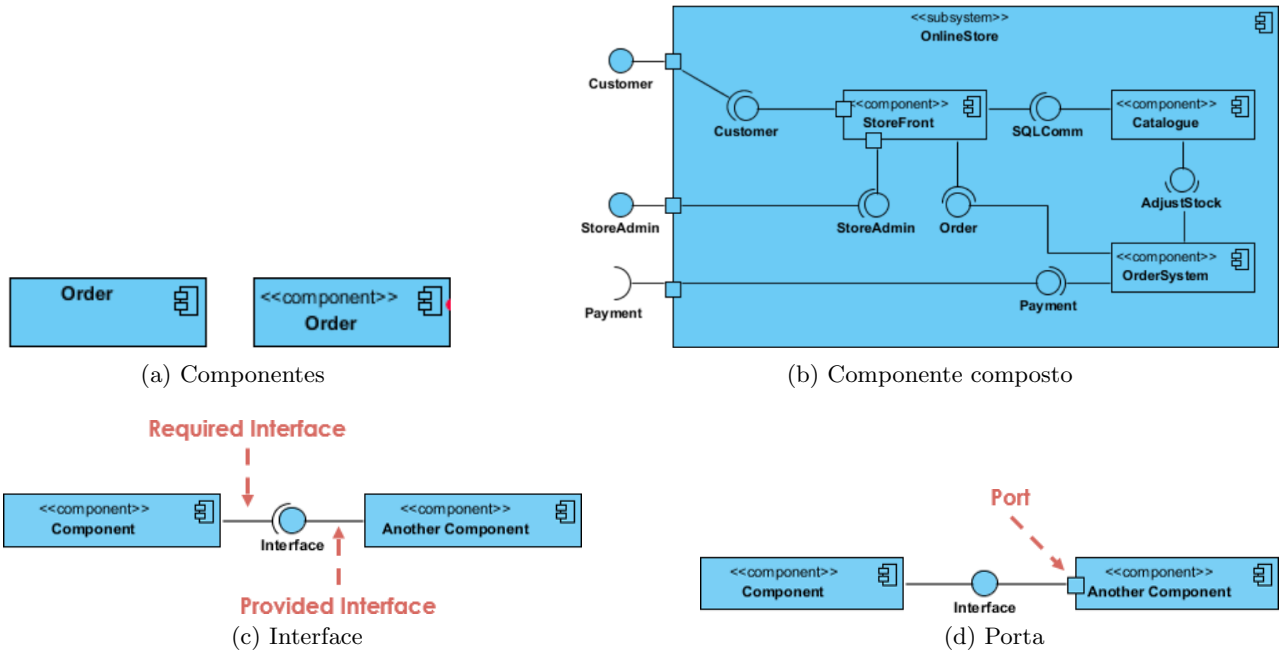
3.2 Diagramas de Componentes

Os diagramas de componentes nos permitem visualizar a organização dos elementos de *software* (os dados, as bibliotecas dinâmicas e os códigos executáveis) de um sistema sob o ponto de vista funcional. Todos os detalhes de implementação de um componente são encapsulados e as suas funções são visíveis por meio de uma série de **interfaces**, através das quais ele se comunica com outros componentes durante a execução do sistema. Assim, fazem parte dos elementos de um diagrama de componentes:

- componentes que modelam a implementação física (usualmente, por *software*) de uma ou mais classes. São representados por retângulos acompanhados de um ícone de componente (Fig. 6a). A estrutura interna de um componente pode conter diversos componentes como mostrados na Fig. 6b.
- interfaces entre os componentes correspondem aos dados e os serviços trocados entre eles. São graficamente representadas as **interfaces fornecidas** (*provided interface*, o que um componente

fornece) por "pirulitos" (*lollipops*) e as **necessárias/requeridas** (*required interface*, o que um componente recebe) por "soquetes" (*sockets*) (Fig. 6c).

- portas definem pontos específicos de interação de um componente com os seus arredores (ambiente) e são representadas por pequenos quadrados na borda do retângulo do componente indicando a sua visibilidade pública (Fig. 6d).



(a) Componentes

(b) Componente composto

(c) Interface

(d) Porta

Figura 6: Notações em diagramas de componentes (Fonte: [20]).

Além dos 5 tipos de relações classes, os componentes podem ter **relações de restrições** ou **condições** expressas com um texto, em linguagem natural ou numa linguagem processável por uma máquina, dentro de uma nota (retângulo com o canto direito superior dobrado). Fig. 7 ilustra o uso de um diagrama de componentes na descrição da dependência do arquivo código-fonte em C++, device.cpp (componente), de um outro arquivo em C++, interp.cpp (componente), que por sua vez, depende diretamente de dois arquivos-cabeçalho, irq.h e signal.h.

3.3 Diagramas de Implantação

Os diagramas de implantação são, graficamente, uma coleção de nós (dispositivos físicos) conectados por arcos (relações), modelando a topologia do *hardware* de um sistema onde *software* manifestados através de artefatos são implantados. Os **artefatos** são modelos de uma informação concreta que é usada/produzida ou por um processo de desenvolvimento de *software*, ou por implantação ou por operação do sistema. Usualmente, eles são as **manifestações** dos componentes.

As notações frequentes em diagramas de implantação são ilustradas na Fig. 8:

- um componente implantado no nó é um retângulo com duas abas.

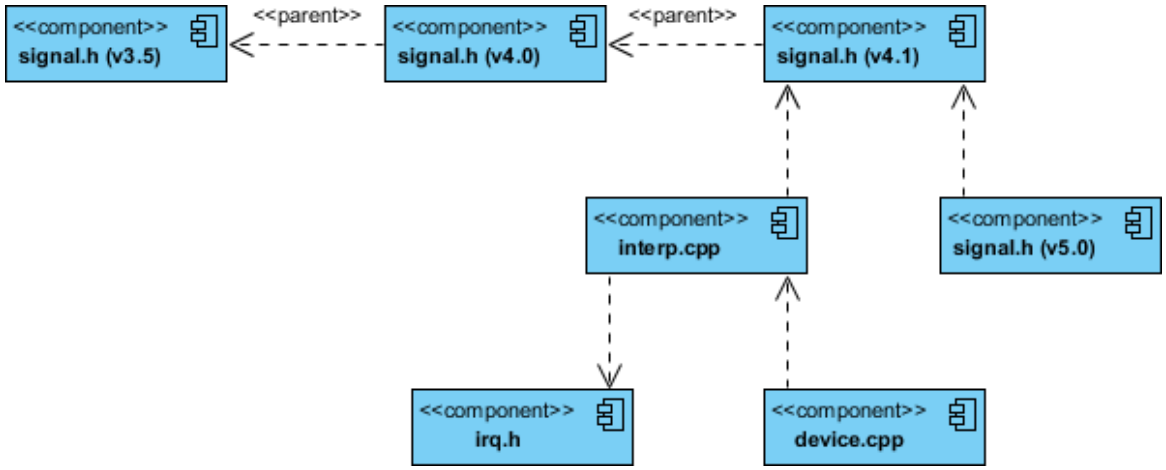


Figura 7: Modelagem da dependência dos códigos-fonte versionados em C++ por um diagrama de componentes (Fonte: [20]).

- artefatos são manifestações dos componentes. São representados graficamente por um retângulo com a palavra-reservada "<<artifact>>". Opcionalmente, pode-se acrescentar no canto superior direito de retângulo um ícone de retângulo com o canto superior direito dobrado.
- interface do *software* com o ambiente é uma círculo.
- nó é uma caixa 3D, usualmente com o nome do estereótipo ao qual ele pertence. Dois estereótipos mais comuns são "<<device>>"para designar os equipamentos físicos capazes de processamento e "<<execution environment>>" para designar o ambiente em que os programas são executados.

Cabe chamar atenção à representação gráfica de um componente implantado num equipamento físico, mostrada no lado mais esquerdo da Fig. 8.

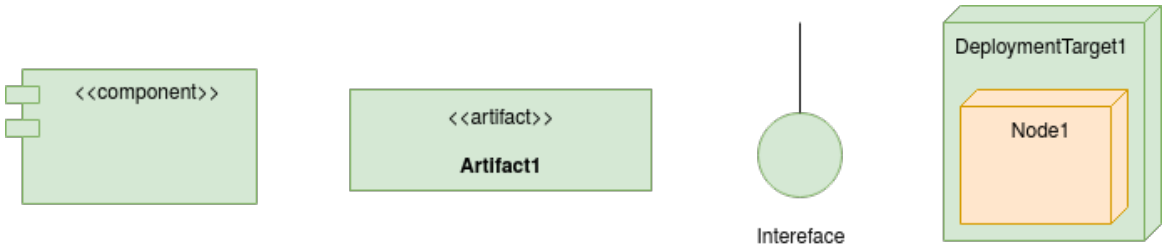


Figura 8: Notações em diagramas de implantação.

As relações entre os nós são predominantemente do tipo de dependência ou do tipo associação, indicando mensagens ou tipo de conexão entre os nós. Fig. 9 ilustra um diagrama de implantação que modela a relação entre a função *HTML5 player* do navegador *Web Browser* implantada num computador *User PC* e um servidor *Web Server* que faz uso de um servidor de base de dados *Database Server* que se manifesta através da interface de *SQL database*.

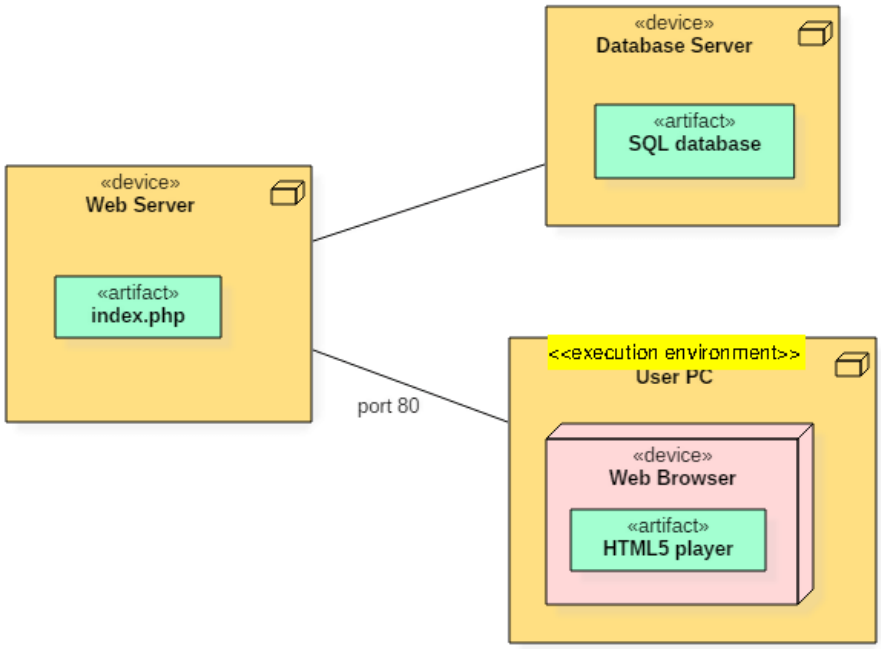


Figura 9: Modelagem da implantação de um *player* de vídeo HTML5 em um Web Browser instalado em User PC por um diagrama de implantação (Fonte: [21]).

3.4 Diagramas de Sequência

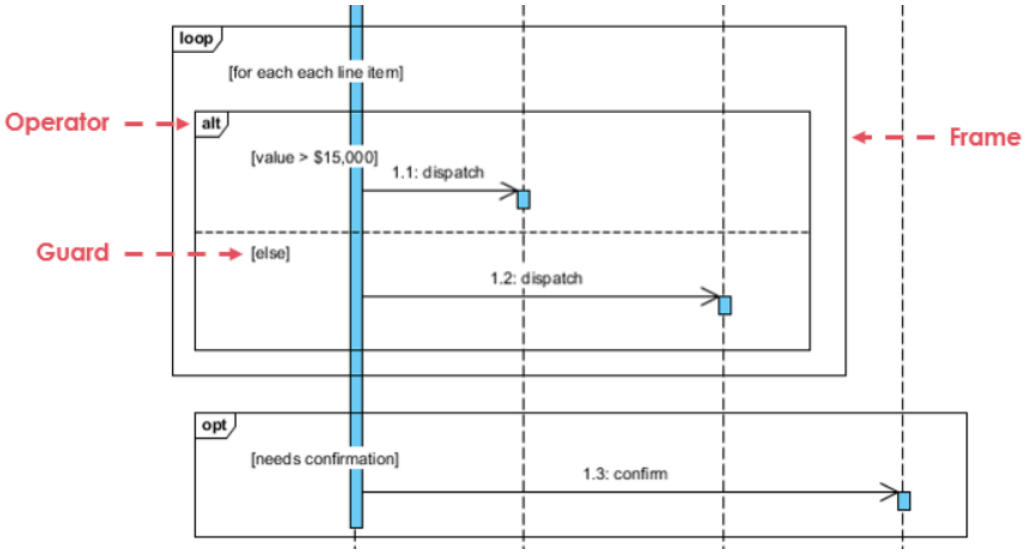
Diagramas de interação em UML são usados para modelar a comunicação entre os elementos de um sistema, denominados *lifelines* no diagrama, com foco no sequenciamento da dinâmica de passagens de mensagens entre eles na execução de uma ou mais funções.

As notações frequentemente encontradas em diagramas de sequência são (Fig. 10):

- fragmentos de sequências representados por retângulos com um pentágono no canto esquerdo superior (*frame*) para diferenciar tipos de interações mostrados pelos operadores no pentágono. Exemplos desses operadores são **alt**, quando somente um dos múltiplos fragmentos é executado em cada instante; **par**, quando os fragmentos são executados em paralelo; **opt**, quando o fragmento só é executado se a opção for satisfeita; e **loop**, quando o fragmento é executado repetidamente num laço de iteração. Observe na Fig. 10a que foi usado o símbolo *guard* para explicitar o tipo de alternativas.

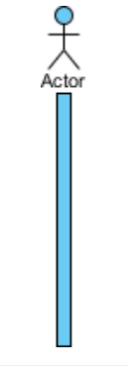
- *lifelines* representados por linhas pontilhadas verticais com o nome do participante num retângulo (Fig. 10b). Em especial, quando o participante é um usuário, usa-se a notação de **ator** cuja representação gráfica é um *stick man* (Fig. 10c).

- período de ativação de um elemento numa operação representado por uma barra vertical sobre a linha pontilhada. Os dois extremos da barra correspondem ao início e o término de uma ativação (Fig. 10d).

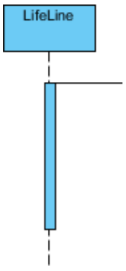


(a) Fragmentos de sequência

(b) *Lifelines*



(c) Ator



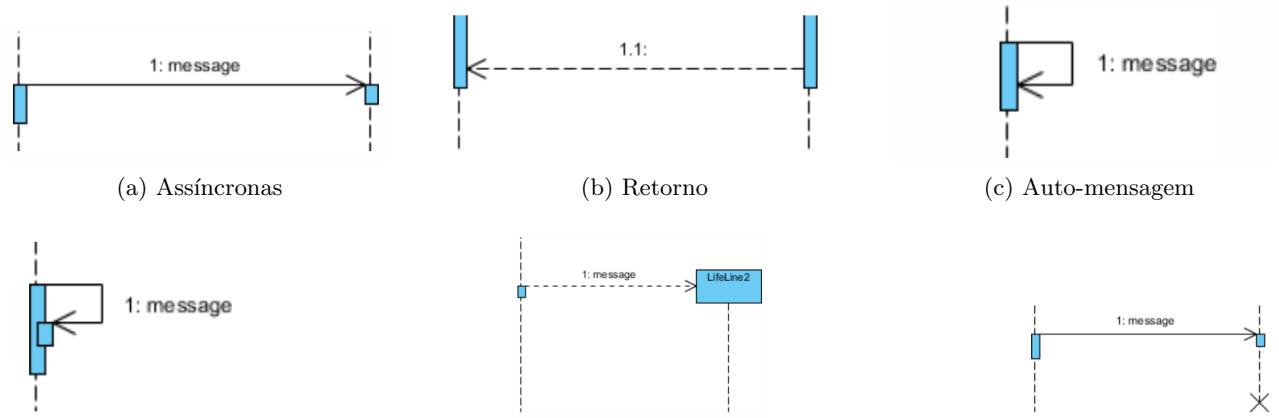
(d) Ativação

Figura 10: Notações gráficas em diagramas de sequência (Fonte: [22]).

UML permite diferenciar os seguintes tipos de mensagens que fluem entre dois elementos durante a execução de um sistema, cujas representações gráficas são mostradas na Fig. 11:

- mensagens sícronas representadas por setas cheias,
- mensagens assíncronas representadas por setas vazadas (Fig. 11a),
- mensagens de retorno representadas por setas vazadas opostas (Fig. 11b),
- mensagens enviadas para o próprio *lifeline* representadas por setas vazadas que saem e voltam no mesmo *lifeline* (Fig. 11c),
- mensagens enviadas para o próprio *lifeline* representadas por setas vazadas que saem e voltam no mesmo *lifeline* mas em outro trecho de ativação (Fig. 11d),

- mensagens enviadas para criação de uma nova instância de um objeto (um novo *lifeline*) (Fig. 11e), e
- mensagens enviadas para destruição de uma instância de um objeto (Fig. 11f).



(a) Assíncronas (b) Retorno (c) Auto-mensagem

(d) Recursiva (e) Criação de objeto (f) Destruição de objeto

Figura 11: Notações gráficas das mensagens entre *lifelines* em diagramas de sequência (Fonte: [22]).

Fig. 12 ilustra como integrar as notações apresentadas num diagrama de sequência para modelar a comunicação entre dois elementos LifeLine1 e LifeLine2 na execução dos seus trechos de códigos. Observe ainda o uso de notas para acrescentar textos explicativos.

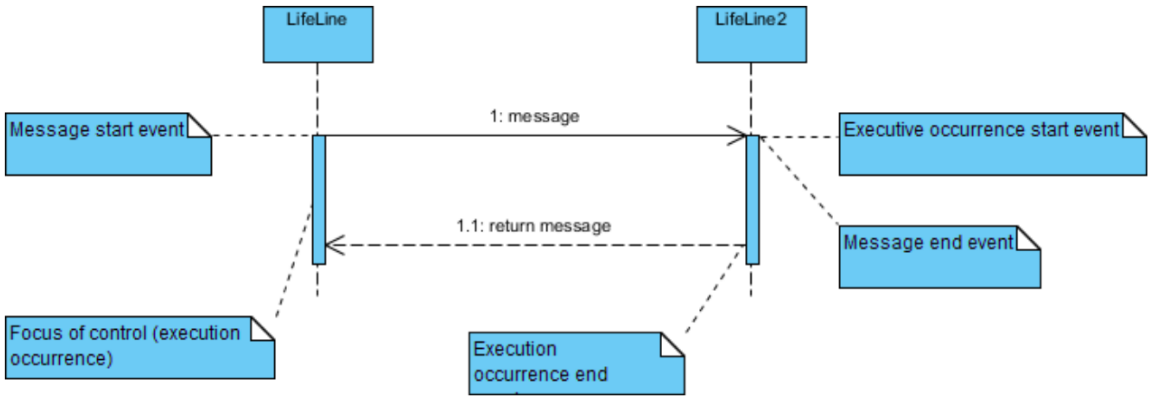
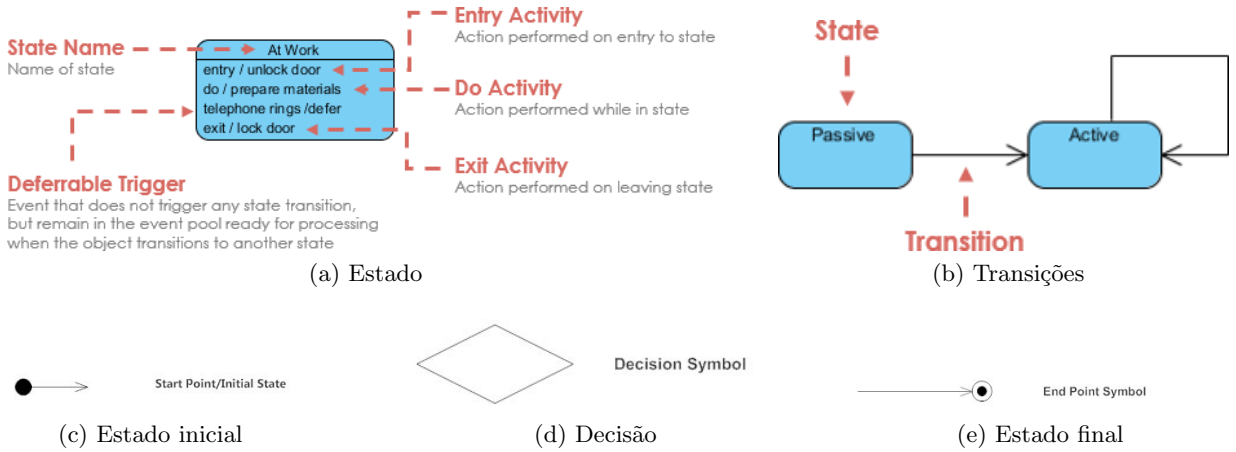


Figura 12: Modelagem de interação entre dois objetos por um diagrama de sequência (Fonte: [22]).

3.5 Diagramas de Máquina de Estados

Ao longo da sua vida útil, os dados de um sistema podem passar por uma série de transformações, em resposta às operações internas e/ou eventos externos. Podemos modelar o resultado de cada estágio como um estado do sistema. Diagramas de máquina de estados permitem visualizar as **transições**

entre esses **estados**, desde o **estado inicial** até o **estado final**. É permitido adicionar pontos de **decisão** com as condições de guarda (*guard*) explicitadas nas transições. As representações gráficas dessas notações são sintetizadas na Fig. 13. Os **eventos** são representados pelos seus nomes, podendo aparecer dentro dos estados ou nas transições dos diagramas. Os estados podem ser simples ou compostos de vários **sub-estados**. Dentro dos estados podem ocorrer ações ou atividades. Uma **ação** é uma execução atômica não interrompível, enquanto uma **atividade** é uma coleção de ações que podem ser interrompidas por eventos, podendo um objeto passar para um outro estado antes de finalizá-la.



(a) Estado (b) Transições

(c) Estado inicial (d) Decisão (e) Estado final

Figura 13: Representações gráficas de elementos em diagramas de máquina de estados (Fonte: [23]).

Fig. 14 ilustra a modelagem, por um diagrama de máquina de estados, de um processo de leilão em que o processamento de lance (*bid*) e o processamento de autorização do limite de pagamento ocorrem em paralelo.

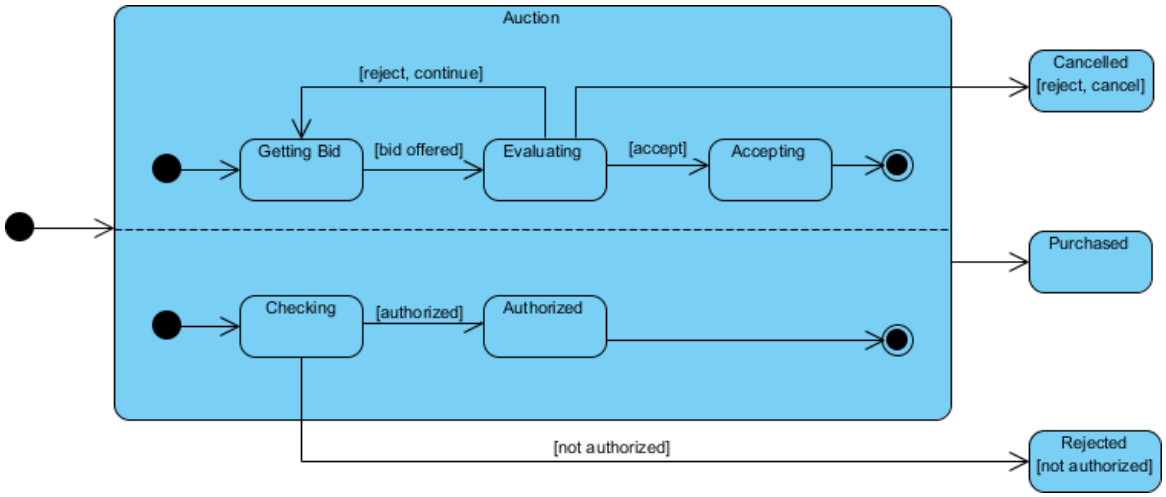
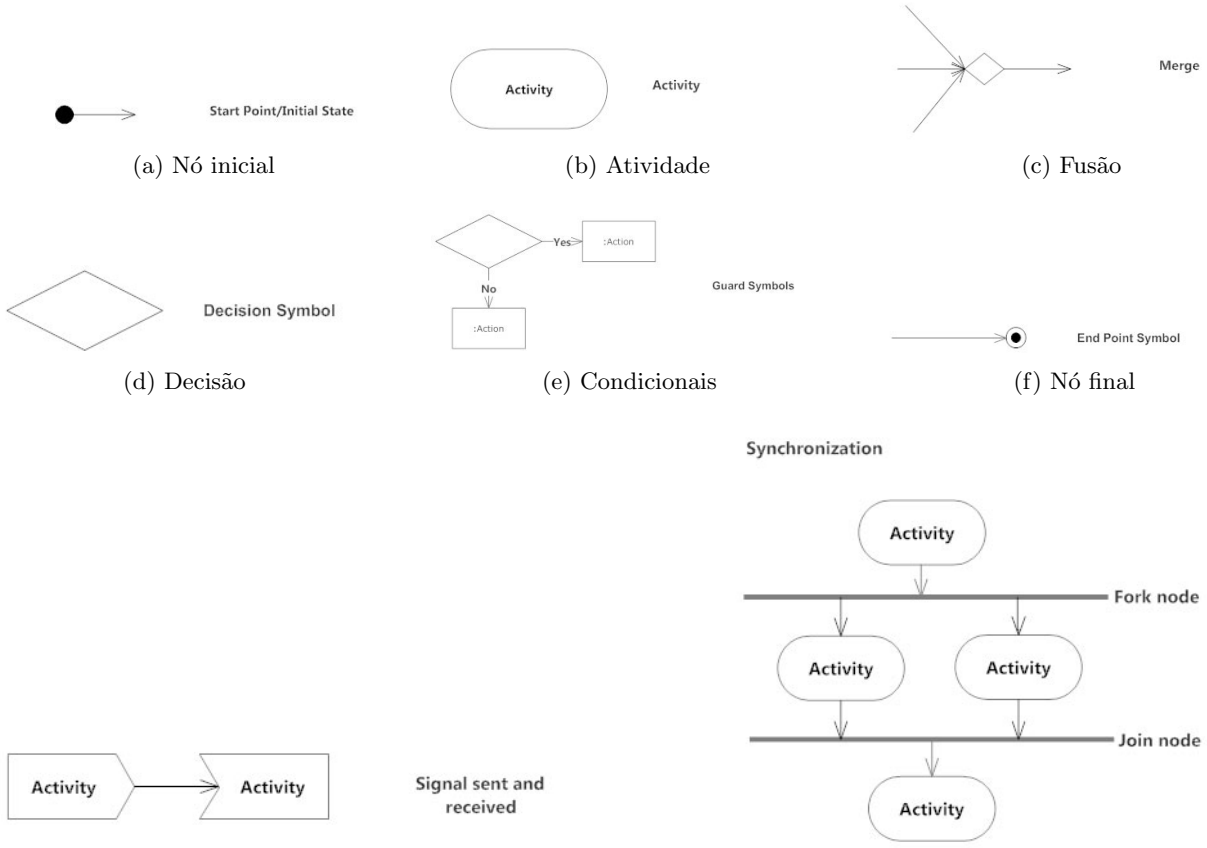


Figura 14: Modelagem de um processo de leilão por um diagrama de máquina de estados (Fonte: [23]).

3.6 Diagramas de Atividades

Os diagramas de atividades modelam o fluxo de execução de uma coleção de atividades para atender uma funcionalidade específica. Eles permitem visualizar as restrições, o sequenciamento e a concorrência na execução as atividades de um sistema.

As notações adotadas em UML para a descrição de um fluxo síncrono são mostradas na Fig. 15. A maioria das notações são similares às encontradas nos fluxogramas, a menos a barreira de sincronismo com os nós de bifurcação e os de junção (Fig. 15h). Nos **nós de bifurcação** um fluxo é dividido em múltiplos fluxos concorrentes e nos **nós de junção** múltiplos fluxos concorrentes se juntam num único fluxo.



(a) Nó inicial (b) Atividade (c) Fusão

(d) Decisão (e) Condicionais (f) Nó final

(g) Envio e Recepção de Sinais (h) Nós de bifurcação e junção (i)

Figura 15: Representações gráficas de elementos em diagramas de atividades (Fonte: [24]).

Em diagramas de atividades podemos representar tanto o fluxo de controle quanto o fluxo de objeto (de dado) (Fig. 16). Trata-se de um fluxo de controle quando uma seta de fluxo conecta duas atividades. Dizemos que é um fluxo de objeto quando a atividade está atuando sobre um objeto (usamos a seta apontando da atividade para o objeto) ou quando o objeto está sendo usado pela atividade (invertemos o sentido da seta). Por exemplo, na Fig. 16b temos um fluxo de objeto. A atividade pode estar criando ou alterando uma instância do objeto.

O fluxo de atividades pode ser interrompido por eventos ou sinais externos. A notação de **aresta**

(a) Fluxo de controle (b) Fluxo de objeto

Figura 16: Representações gráficas de fluxos em diagramas de atividades (Fonte: [24]).

de interrupção (*interrupting edge*) é aplicada para descrever essa interrupção assincrona (Fig. 17a). Pode-se também condicionar a execução de uma atividade a um instante de tempo. A notação de **evento de tempo** é usada para representar esse condicionamento (Fig. 17b).

(a) Aresta de Interrupção (b) Evento de Tempo

Figura 17: Representações gráficas de eventos assíncronos em diagramas de atividades (Fonte: [24]).

Fig. 18 ilustra a modelagem do despacho de uma encomenda em que um evento (*Order Cancel Request*) pode ocorrer assincronamente ao processo de despacho e interromper o processo, levando ao cancelamento do processo e o término da sequência de atividades referentes ao "Despacho de Encomenda". Note no diagrama o uso de nós de bifurcação e de junção para representar atividades paralelas. Vale ainda ressaltar que faltam no diagrama as condições de guarda no nó de decisão (*Receive Order*) com duas setas de saída. Elas são [Yes] e [No] para as setas horizontal e vertical, respectivamente. O segundo nó de decisão pode ser removido, conectando diretamente a linha [No] com o nó de junção e a saída do nó de junção em *Close Order*.

3.7 Diagramas de Casos de Uso

Os diagramas de casos de uso mostram as relações entre os **atores** (usuários) e os **casos de uso**. **Casos de uso** consistem de descrições de interações entre os usuários e o **sistema** na realização de uma tarefa. A **participação** de um ator num caso de uso é representado por uma linha sólida. As representações gráficas dessas notações num diagrama de casos de uso são apresentadas na Fig. 19

Os casos de uso podem guardar entre si a relação de **extensão** quando um caso de uso inclui o comportamento do outro, a relação de **inclusão** quando um caso de uso usa as funcionalidades do outro caso de uso, e a relação de **generalização** quando um caso de uso é um caso específico do outro. Fig. 20 mostram as representações gráficas desses três tipos de notações entre os casos de uso.

Fig. 21 ilustra a modelagem, por um diagrama de casos de uso, das diferentes formas de participação de pessoas de perfis distintos num sistema de venda de veículos.

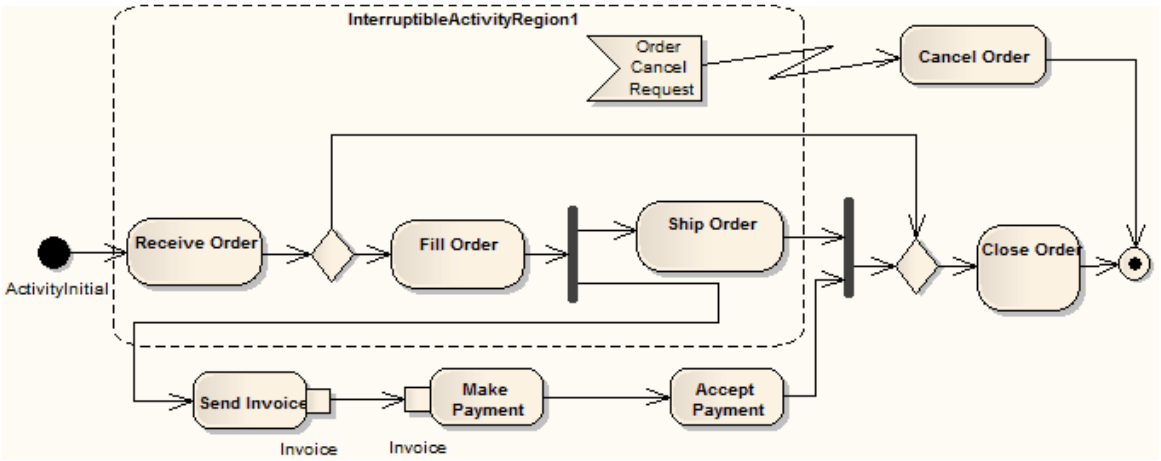
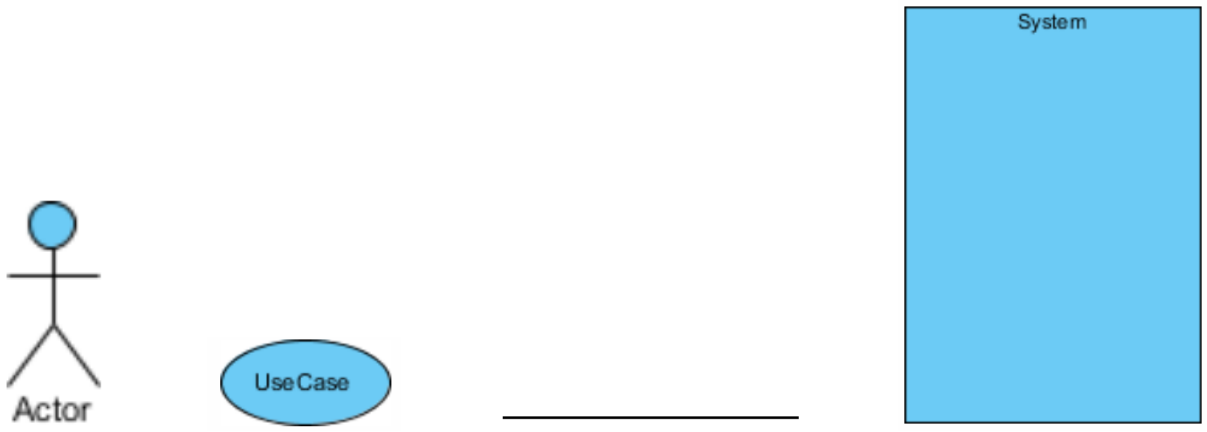


Figura 18: Modelagem de um despacho de encomenda por um diagrama de atividades (Fonte: [25]).



(a) Ator (b) Caso de Uso (c) Conexão (d) Sistema

Figura 19: Representações gráficas de elementos em diagramas de casos de uso (Fonte: [26]).

(a) Extensão (b) Inclusão (c) Generalização

Figura 20: Representações gráficas de relações entre casos de uso em diagramas de casos de uso (Fonte: [26]).

4 Um Exemplo

Nesta seção vamos ilustrar a modelagem de um sistema a ser implementado na plataforma de desenvolvimento FRDM-KL25Z [28] com o *shield* FEEC871 [29] acoplado. Os diagramas de UML apresentados na Seção 3 são usados para descrever os requisitos, a estrutura e o comportamento do sistema visado

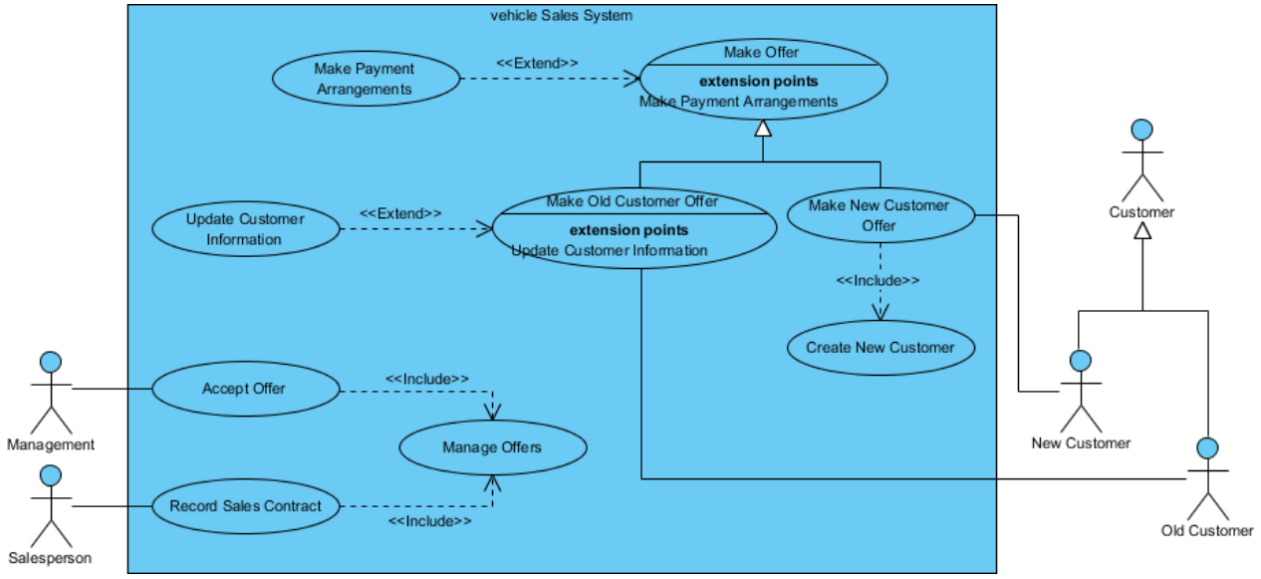


Figura 21: Modelagem por um diagrama de casos de uso das interações de um sistema de venda] de veículos com pessoas de diferentes interesses (Fonte: [26]).

a partir de uma descrição do problema. O aplicativo livre *online diagrams.net* [30] é usado para o desenho de todos os diagramas apresentados nesta seção.

Segue-se a descrição do problema:

> Um cliente encomendou um **relógio digital** com as seguintes características. Há um visor **LCD 16×2** para **mostrar o horário** e **3 botoeiras, verde (G), vermelho (R), e azul (B),** para **ajustar o horário**. O horário é mostrado no meio da primeira linha do visor do LCD no formato padrão HH:MM:SS (24 horas). O ajuste do horário é por unidade de tempo, HH, MM e SS. A **botoeira verde ativa o modo de ajuste de horário do relógio e seleciona cicilicamente as unidades** HH $\longrightarrow$ MM $\longrightarrow$ SS $\longrightarrow$ HH. Como **realimentação visual** da unidade selecionada, o cursor do LCD é posicionado na célula do dígito mais significativo de cada unidade e fica piscando. **A unidade de tempo deve ser individualmente in- ou decrementada ao acionarmos a botoeira vermelha e azul, respectivamente.** Como realimentação visual de um novo valor, o novo valor deve ser mostrado nas células correspondentes. Se **dentro de um intervalo de tempo correspondente a timeout=3s nenhuma botoeira for acionada,** o relógio sai do modo de ajuste e **volta a operar com um relógio** que atualiza o visor com um novo horário a cada 1s.

4.1 Modelagem de Requisitos

A primeira fase de um projeto é a captura dos requisitos funcionais esperados pelos potenciais usuários. O comportamento esperado do sistema sob o ponto de vista de usuários (atores) é modelado em UML pelo diagrama de casos de uso. Fig. 22 ilustra a modelagem das funcionalidades especificadas do relógio por um diagrama de casos de uso contendo 7 casos de uso. Note que o caso de uso "Ajustar horário" (do relógio) é "estendido" para o caso de uso "Ajustar por botoeiras" quando ocorrem interações entre o usuário e o sistema (relógio). Esse caso de uso inclui os casos de uso de ajuste dos valores em

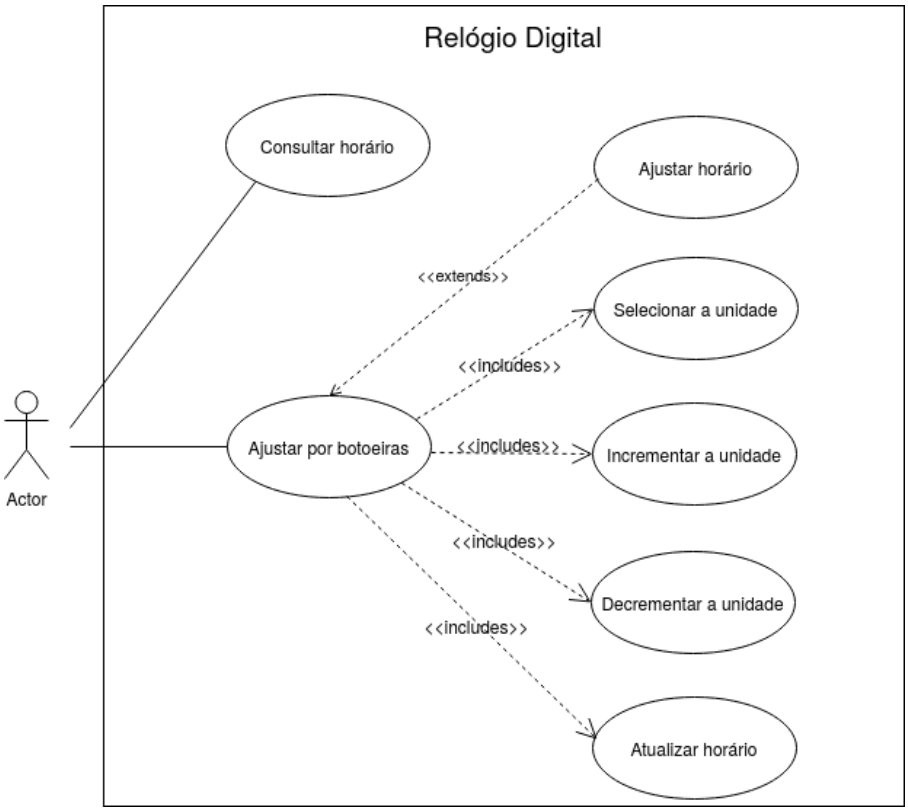


Figura 22: Modelagem do relógio digital por um diagrama de casos de uso.

cada unidade de tempo, horas (HH), minutos (MM) e segundos (SS) por interações que selecionam a unidade de tempo e fazem incrementos ou decrementos na unidade selecionada.

4.2 Modelagem Estrutural

A segunda fase de um projeto é a extração dos objetos relevantes e suas relações a partir da descrição do problema. Esses elementos são destacados em negrito no texto da descrição do problema. Veja na Fig. 23 o mapeamento dos conceitos "relógio digital", "consulta do horário", "ativação do modo de ajuste", "ajuste do horário por unidade de tempo" e "LCD" em diferentes classes de um diagrama de classes. Note ainda que foram adicionadas as classea "RTC" e "Evento" porque o microcontrolador Kinetis KL25Z128 [27] integrado na placa de desenvolvimento FRDM-KL25Z contém um circuito RTC (*Real Time Clock*) dedicado para contagem de segundos e um circuito NVIC (*Nested Vectored Interrupt Control*) para processamento de interrupções.

Tanto a classe "Ajuste" quanto a classe "Mostra" dependem dos dados (segundos) gerados pelo "RTC". As duas classes são associadas à classe "LCD" para prover realimentação visual das ações de um usuário em três botoeiras abstraídas pela classe "Botoeira:Evento". Essa classa é associada à classe "Timeout:Evento", porque toda vez que uma botoeira for acionada é iniciada a contagem de tempo nela. Se não acontecer um segundo acionamento num intervalo de 3 segundos, é disparado automaticamente um evento para retornar à classe "Mostra".

Uma implementação das classes e relações identificadas na Fig. 23 é sintetizada no diagrama de componentes apresentado na Fig. 24. Uma característica peculiar dos projetos de sistemas embarcados

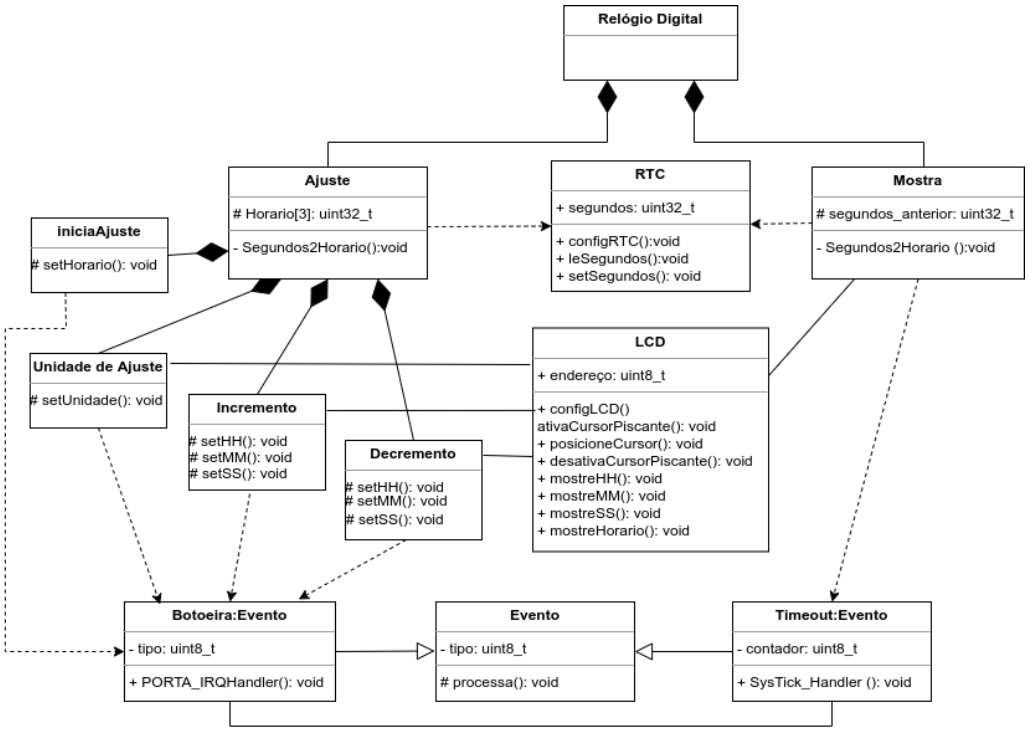


Figura 23: Modelagem do relógio digital por um diagrama de classes.

baseados em microcontroladores é a customização dos seus módulos no *setup* para execução das tarefas especificadas.

Para o projeto de relógio digital, decidimos usar os módulos (*hardware*, em vermelho) disponíveis no microcontrolador Kinetis KL25Z128 [27] para controle dos eventos por interrupção. Mais especificamente, usamos os módulos PORTx (*Port*) e GPIO (*General Purpose IO*) para controle dos sinais digitais de entrada (botoeiras) e de saída (LCD), os temporizadores RTC e SysTick (*System Tick Timer*) para controle de tempo, o controlador de interrupções NVIC (*Nested Vectored Interrupt Control*) para controle de eventos gerados assincronamente e o o módulo SIM (*System Integration Module*) para ativar sinais de relógios dos módulos selecionados para uso na implementação do projeto. Por *software* (main.c, em preto), são configurados os registradores de controle desses módulos.

Após a configuração, o processamento das entradas via as três botoeiras é gerenciado por NVIC usando os dados pré-instalados pelo componente Setup Code. Em resposta aos eventos assíncronos, NVIC desvia o fluxo de controle para as rotinas de serviço no componente ISR.c onde são atualizados os valores das unidades de tempo conforme as interações detectadas. Para proporcionar uma realimentação visual às ações do usuário, esses valores são passados para o componente main.c que os enviam no formato apropriado para serem renderizados no LCD. Em paralelo, o módulo SysTick faz a contagem de tempo de tal forma que quando passa de 3 segundos ele dispara um eventos assíncrono para sair do modo de ajuste.

Como a operação específica de um microcontrolador requer *setup* do seu *hardware*, todos os componentes destacados em vermelho na Fig. 24 são circuitos integrados ao microcontrolador e manifestam artefatos tanto para configuração dos seus registradores de controle como para acessos aos seus registradores de dados e de estado. Note que, antes de qualquer acesso aos registradores dos módulos

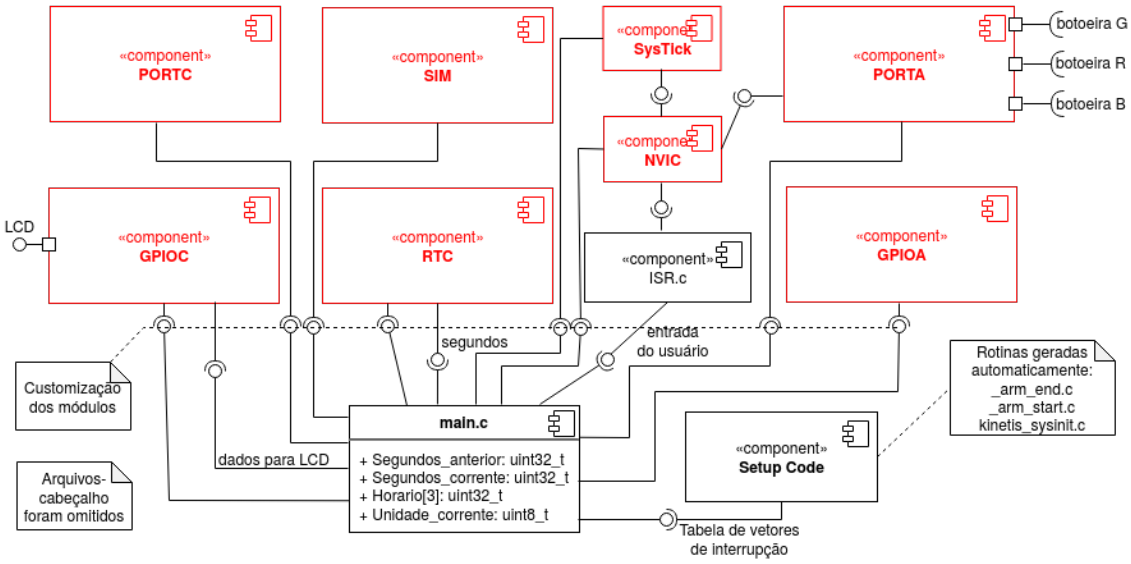


Figura 24: Modelagem do relógio digital especificado por um diagrama de componentes.

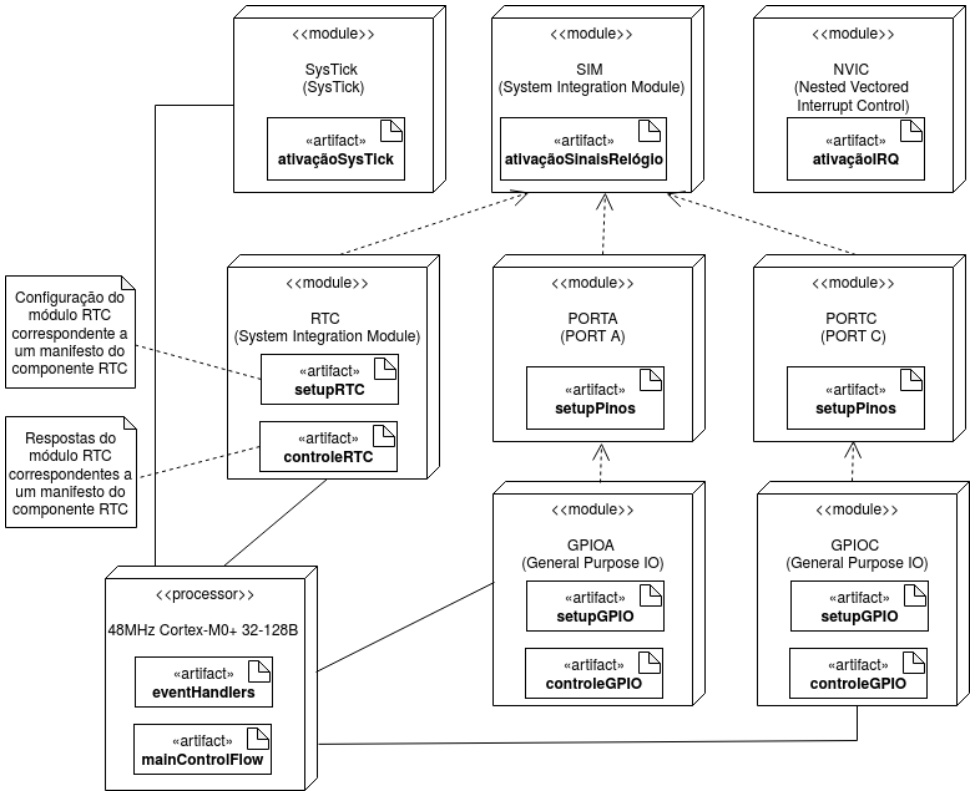


Figura 25: Modelagem do relógio digital especificado por um diagrama de implantação.

RTC, PORTA e PORTC, é necessário ativar a sua fonte de sinais de relógio via o módulo SIM. Somente as instruções dos três componentes de *software*, main.c, ISR.c e Setup Code, são executadas

pelo processador Cortex-M0+. O processador se comunica com todos os módulos ativos, através dos seus registradores, para complementar as ações das instruções programadas. Fig. 25 apresenta um diagrama de implantação que descreve essa visão de implantação dos códigos do projeto de relógio digital para a sua execução.

4.3 Modelagem Comportamental

Vimos na Seção 4.2 que os elementos estruturais de um sistema se interagem para realizar uma tarefa específica. Para descrever essa dinâmica de interações no tempo, fazemos uso de diagramas de comportamento. Através do diagrama de atividades apresentado na Fig. 26, é possível ver como os métodos implementados em cada componente podem ser sequencializados na linha de tempo para o sistema desempenhar as atividades programadas. Note que é possível não só representar nesse diagrama os eventos assíncronos gerados pelas botoeiras e capturados por interrupção, como também explicitar a estratégia adotada de manter a execução da rotina de serviço mais simples e mais rápido possível. Para realimentação visual das respostas do sistema às interações, deslocamos as instruções relacionadas com a renderização no LCD para o fluxo de controle principal, ao invés de executá-las nas rotinas de serviço,

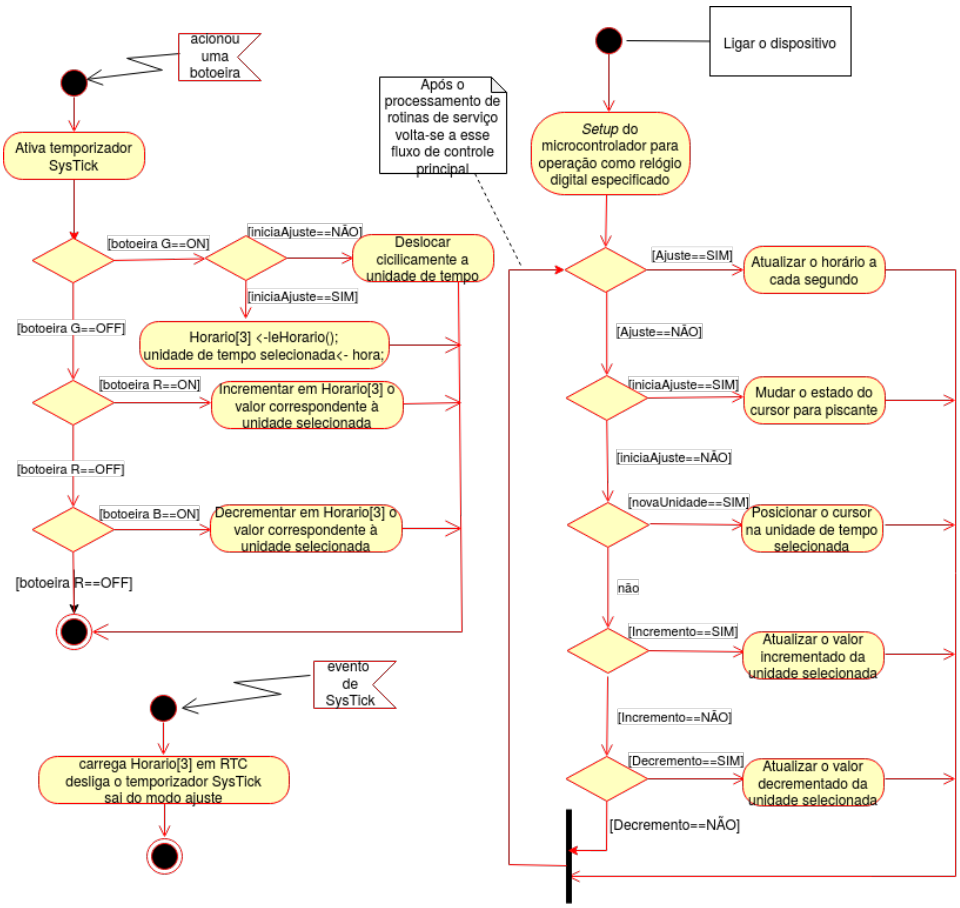


Figura 26: Modelagem do relógio digital especificado por um diagrama de atividades.

O diagrama de atividades não revela, no entanto, as passagens de mensagens entre os elementos

de um sistema durante a execução de uma atividade. Para isso, usamos diagramas de sequência, como ilustra a Fig. 27. Nessa visão fica fácil de perceber as interações entre os sinais de *hardware* (botoeiras, NVIC e LCD) e os componentes de *software* (main e ISR) para completar a sequência de ações requeridas a uma atividade. Note que no diagrama todos os elementos de *hardware* estão em vermelho e os relacionados com *software* são desenhados em preto.

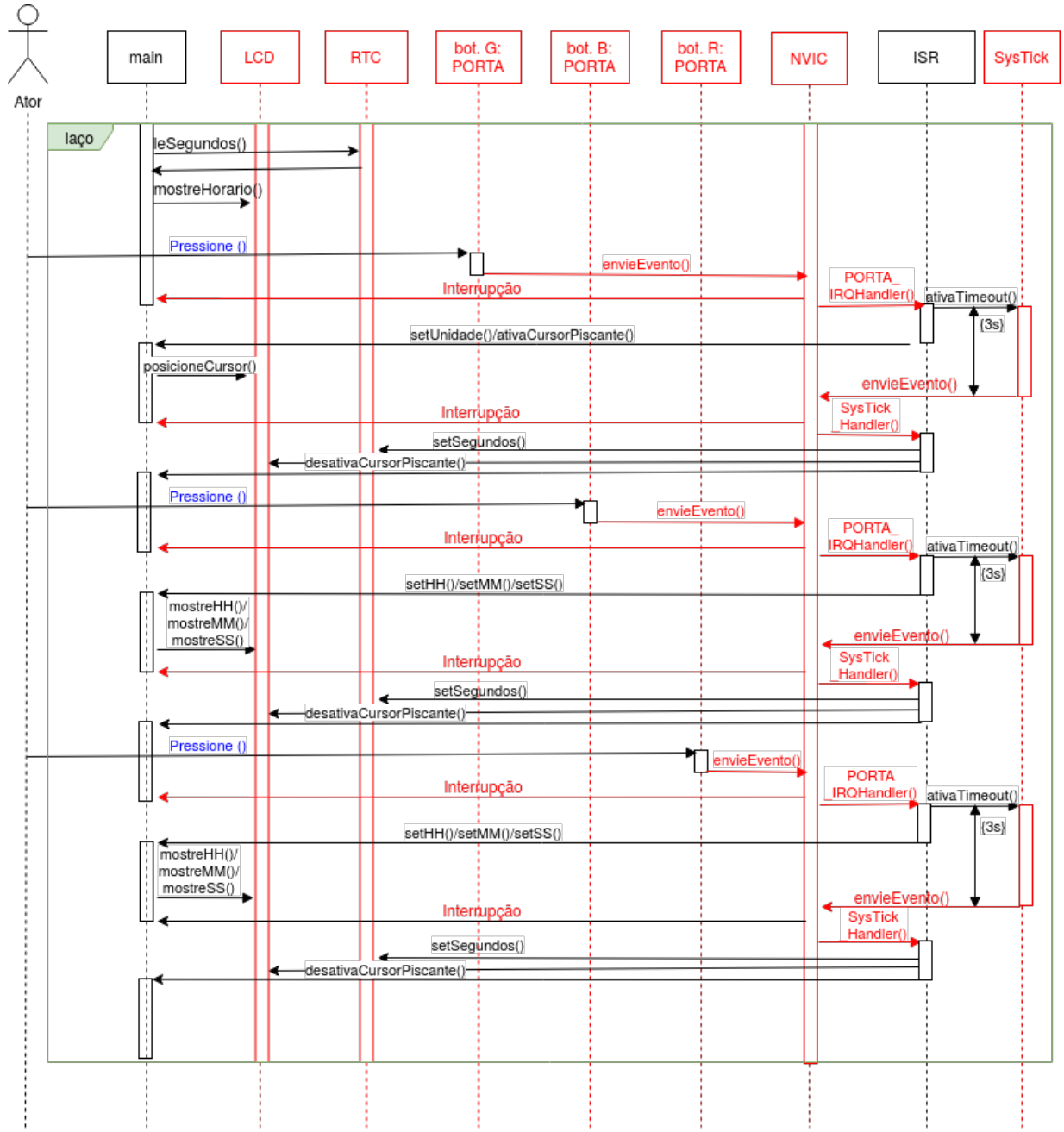


Figura 27: Modelagem do relógio digital especificado por um diagrama de sequência.

Tanto os diagramas de atividades quanto os diagramas de sequência focam na dinâmica de ações

dos elementos e entre os elementos de um sistema. Para visualizarmos a dinâmica da transformação de dados/informações/estados de um sistema sob essas ações, os diagramas de máquina de estados são os mais apropriados. Nesse projeto de relógio digital, definimos os estados em função dos dados renderizados no visor do LCD. Fig. 28 mostra todos os possíveis estados pelos quais o visor do LCD pode passar em resposta aos acionamentos das três botoeiras feitos por um usuário.

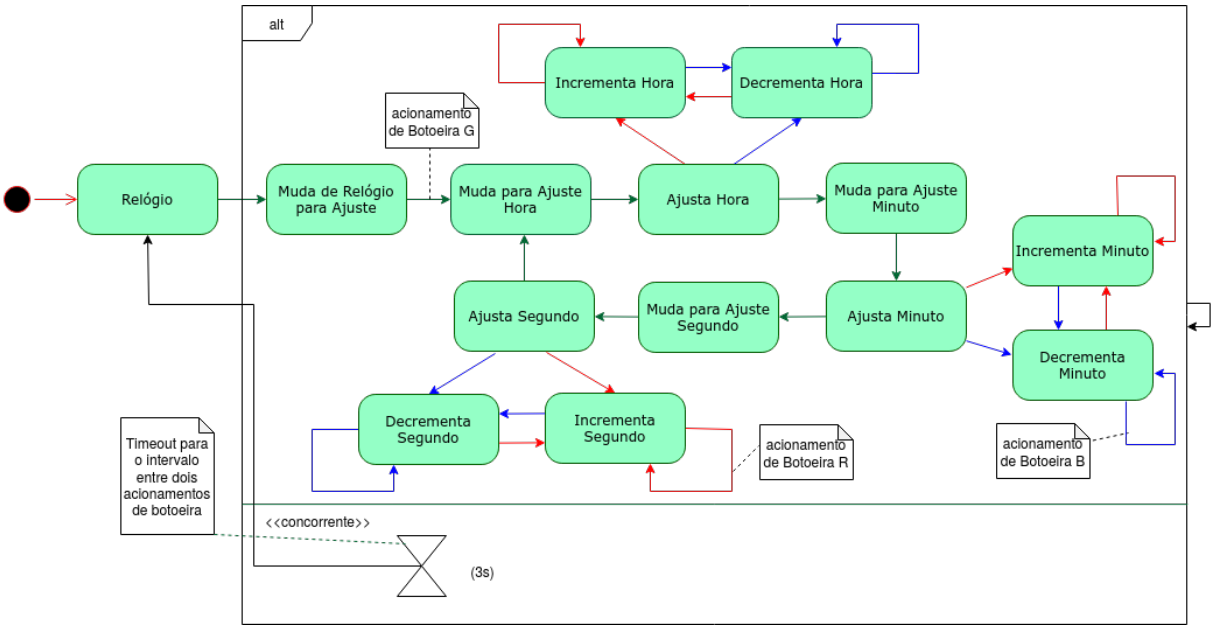


Figura 28: Modelagem do relógio digital especificado por um diagrama de máquina de estados.

5 Conclusões

Nesta nota de aulas foi apresentada uma visão introdutória da linguagem de modelgem unificada, UML (*Unified Modeling Language*), cuja especificação contém quase 800 páginas [12]. Essa linguagem de notação é considerada hoje em dia uma linguagem universal para profissionais de produção de *software* pela padronização das notações simples e intuitivas, favorecendo uma comunicação clara e eficaz das ideias sobre um projeto em seus diferentes estágios de desenvolvimento.

No entanto, mais importante do que ter um meio de comunicação universal e de fácil entendimento, é familiarizar-se com um processo de "sistematização" de um amontoado de possíveis soluções que brotam na cabeça de um projetista ao ler o enunciado de um problema. Toda linguagem de modelagem impõe, através de suas notações, restrições à modelagem pouco convencional e conduz os projetistas a analisar melhor as diferentes alternativas e organizar melhor as relações entre os elementos identificados na sua análise a fim de adequá-las a um padrão de modelagem universalmente aceito.

Vale ressaltar que o processo de "sistematização" das ideias, essencial para o sucesso de qualquer projeto de *software*, não implique no uso da UML. Qualquer recurso de suporte à estruturação das ideias num procedimento com um número finito de passos, tais como fluxogramas e pseudo-códigos, é válido. A preferência pela UML está na sua riqueza de expressão (21 tipos de diagramas para representar um mesmo sistema), na sua representação gráfica universal, na sua padronização e manutenção pele ativo grupo OMG, e na quantidade de ferramentas disponíveis para a sua edição. Especificamente

para projetos de sistemas embarcados em que o *software* se comunica frequente e diretamente com o *hardware*, a UML provê notações apropriadas para descrever tais comunicações, como demonstram o diagrama de componentes na Fig. 24 e o diagrama de sequência na Fig. 27. Além disso, o seu diagrama de implantação nos permite visualizar o ambiente físico em que estão instalados os programas desenvolvidos, como ilustra a Fig. 25.

Referências

[1] Wikipedia. EXPRESS (data modeling language). `https://en.wikipedia.org/wiki/EXPRESS_(data_modeling_language)`, acessado em 06/2022.

[2] David Pellerin. An Introduction to VHDL. `http://www.uco.es/~ff1mumuj/h_intro.htm`, acessado em 06/2022.

[3] CSIE - National Taiwan University List of EDA Tools, July, 2002. `https://sourceforge.net/projects/mingw/`, 2002, acessado em 0/2022.

[4] Wikipedia. UML. `https://pt.wikipedia.org/wiki/UML`, acessado em 06/2022.

[5] Uml-Diagrams. The Unified Modeling Language. `https://www.uml-diagrams.org/`, acessado em 06/2022.

[6] Embedded Staff An Introduction to VHDL. `https://www.embedded.com/uml-statecharts/`, acessado em 06/2022.

[7] Chen, Rong and Sgroi, Marco and Lavagno, Luciano and Martin, Grant and Sangiovanni-Vincentelli, Alberto and Rabaey, Jan. Embedded System Design using UML and Platforms. Em: System Specification & Design Languages: Best of FDL'02., Springer, 2003.

[8] Sébastien Gérard and Huascr Espinoza and François Terrier and Bran Selic. Modeling Languages for Real-Time and Embedded Systems: Requirements and Standard-Based Solutions. Em: Model-based Engineering of Embedded Real-Time Systems, Springer, 2007.

[9] Grant Martin. UML for embedded systems specification and design: motivation and overview. Proceedings 2002 Design, Automation and Test in Europe Conference and Exhibition, 2002

[10] Embedded Staff. Demystifying UML. `https://www.embedded.com/demystifying-uml/`, 2006

[11] OMG ABOUT THE UNIFIED MODELING LANGUAGE SPECIFICATION VERSION 2.2 `https://www.omg.org/spec/UML/2.2/About-UML/`, acessado em 06/2022

[12] OMG ABOUT THE UNIFIED MODELING LANGUAGE SPECIFICATION VERSION 2.5.1 `https://www.omg.org/spec/UML/2.5.1/About-UML/`, acessado em 06/2022

[13] Ricardo R. Gudwin Engenharia de Software: Uma Visão Prática, 2a. edição, 2015. `https://faculty.dca.fee.unicamp.br/gudwin/sites/faculty.dca.fee.unicamp.br.gudwin/files/ea975/ESUVP2.pdf`, acessado em 06/2022.

[14] Greg. Best UML tools to use in 2022. https://www.gleek.io/blog/best-uml-tools.html, 2021, acessado em 06/2022.

[15] Alyssa Walker. UML Diagrams: History, Types, Characteristics, Versions, Tools. https://www.guru99.com/uml-diagrams.html, acessado em 06/2022

[16] Visual Paradigm UML/Code Generation Tool. https://www.visual-paradigm.com/features/code-engineering-tools/.

[17] Gianna Reggio and Maurizio Leott and Filippo Ricca and Diego Clerissi What are the used UML diagrams? http://ceur-ws.org/Vol-1078/paper1.pdf, acessado em 06/2022

[18] Visual Paradigm. Overview of the 14 UML Diagram Types. https://www.visual-paradigm.com/guide/uml-unified-modeling-language/overview-of-the-14-uml-diagram-types/, acessado em 06/2022

[19] Visual Paradigm. What is Class Diagram? https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-class-diagram/, acessado em 06/2022

[20] Visual Paradigm. What is Component Diagram? https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-component-diagram/, acessado em 06/2022

[21] Alyssa Walker. Deploymet Diagram: UML Tutorial with Example. https://www.guru99.com/deployment-diagram-uml-example.html, acessado em 06/2022

[22] Visual Paradigm. What is Sequence Diagram? https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-sequence-diagram/, acessado em 06/2022

[23] Visual Paradigm. What is State Machine Diagram? https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-state-machine-diagram/, acessado em 06/2022

[24] Smartdraw. Activity Diagram. https://www.smartdraw.com/activity-diagram/, acessado em 06/2022

[25] Sparx Systems. Interruptible Activity Region. https://sparxsystems.com/enterprise_architect_user_guide/14.0/model_domains/interruptibleactivityregion.html, acessado em 06/2022

[26] Visual Paradigm. What is Use Case Diagram? https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-use-case-diagram/, acessado em 06/2022

[27] NXP – Freescale Semiconductor, Inc. Kinetis KL25 Sub-Family Data Sheet. https://www.dca.fee.unicamp.br/cursos/EA871/references/ARM/KL25P80M48SF0.pdf, acessado em 06/2022.

[28] Freescale Semiconductor, Inc. FRDM-KL25Z User's Manual. https://www.dca.fee.unicamp.br/cursos/EA871/references/ARM/FRDMKL25Z.pdf, acessado em 06/2022.

[29] FEEC. Esquemático do shield FEEC871. `https://www.dca.fee.unicamp.br/cursos/EA871/references/complementos_ea871/Esquematico_EA871-Rev3.pdf`, acessado em 06/2022.

[30] Diagrams.net. What is Use Case Diagram? `https://app.diagrams.net/?client=1`, acessado em 06/2022.