Em Direção a uma Linguagem de Modelagem para Sistema Multiagentes Embarcados Apoiada por Metamodelos

Fabian Cesar Pereira Brandão Manoel[1], **Carlos Eduardo Pantoja**[1,2], **Myrna Cecilia Martins dos Santos Amorim**[1]

[1]Centro Federal de Educação Tecnológica Celso Suckow da Fonseca (CEFET/RJ)
Rio de Janeiro – RJ – Brasil

[2]Universidade Federal Fluminense (UFF)
Niterói, RJ – Brasil

fabiancpbm@gmail.com, pantoja@cefet-rj.br, myrna.amorim@cefet-rj.br

Abstract. *In recent years, Industry 4.0 and cyber physical systems have engaged revolutions that demand greater cognition and autonomy of hardware devices. In this sense, Multiagent Systems (MAS) have been increasingly used for controlling and monitoring the hardware of embedded systems. In design time, as hardware complexity increases, so does the complexity of building embedded systems in the Multiagent context. In cases like these, previous project modeling helps understand the solution and avoids problems such as rework, bad decisions, or inconsistencies. However, although there are modeling solutions for SMA, there is still a gap when the objective is to represent the entire embedded system in a consistent and standardized way because each project uses its modeling approach. This work aims to propose a modeling language for embedded SMA that allows representing the hardware structure, the behavior of the microcontroller, and the SMA. For this, three modeling dimensions were created: structural, behavioral, and Multiagent, the latter being supported by FAML. As a contribution, this work brings a metamodel that extends the FAML to build a modeling language that supports a complete, consistent, and standardized representation of an embedded SMA.*

Resumo. *Nos últimos anos, a indústria 4.0 e os sistemas ciberfísicos têm engajado revoluções que demandam maior cognição e autonomia de dispositivos de hardware. Neste sentido, Sistemas Multiagentes (SMA) vêm sendo cada vez mais utilizados como entidades computacionais de controle e monitoramento do hardware de sistemas embarcados. Em tempo de design, à medida que a complexidade do hardware aumenta, a complexidade de construir sistemas embarcados no contexto Multiagente também aumenta. Em casos como estes, a modelagem prévia do projeto auxilia no entendimento da solução e evita problemas como retrabalho, más decisões ou inconsistências. Contudo, apesar de existirem soluções de modelagem para SMA, ainda há uma lacuna quando o objetivo é representar todo o sistema embarcado de maneira consistente e padronizada porque cada projeto utiliza a própria abordagem de modelagem. O objetivo deste trabalho é propor uma linguagem de modelagem para SMA embarcados que permita representar a estrutura do hardware, o comportamento do microcontrolador e o SMA. Para isso, três dimensões de modelagem foram*

criadas: estrutural, comportamental e Multiagente, sendo esta última apoiada pelo FAML. Como contribuição, este trabalho traz um metamodelo que estende o FAML para construir uma linguagem de modelagem que suporte uma representação completa, consistente e padronizada de um SMA embarcado.

1. Introdução

A indústria 4.0 e outras revoluções tecnológicas do mundo moderno têm demandado soluções que sejam capazes de realizar atividades cada vez mais autônomas. Neste contexto, os sistemas ciberfísicos (*Cyber-Physical Systems* — CPS), surgem como soluções capazes de combinar *software* e *hardware* em ambientes físicos para que realizem as atividades do contexto com mínima intervenção humana [Alur 2015]. Os sistemas embarcados são CPS distribuídos no ambiente, dedicados e embutidos ao *hardware* de um dispositivo físico [Marwedel 2021]. Para atender as demandas de cognição e autonomia da indústria 4.0, algumas soluções e conceitos da inteligência artificial podem ser utilizadas em CPS. Neste contexto, Sistemas Multiagentes (SMA) podem ser utilizados para controlar e monitorar dispositivos físicos voltados ao controle e monitoramento de ambientes físicos. Alguns trabalhos como [Manoel et al. 2017, Sakurada et al. 2019, Brandão et al. 2021] já utilizam SMA embarcados em dispositivos físicos como uma maneira de trazer cognição nos ambientes.

A primeira parte do processo de construção de um SMA está na modelagem da aplicação, que dispõe de ferramentas de linguagens de modelagem como o Prometeus [Padgham and Winikoff 2004], Adelfe [Bernon et al. 2003], PASSI [Cossentino 2005], Gaia [Zambonelli et al. 2003], INGENIAS [Gascuena and Fernández-Caballero 2007], Tropos [Bresciani et al. 2003], entre outras. Este processo de modelagem é importante porque constitui a fase de planejamento do projeto levando em conta apenas as regras de negócio e também porque se torna a documentação de referência para tomadas de decisões ou consultas futuras. Contudo, quando SMA são projetados para o contexto de sistemas embarcados, não há um padrão de representação estabelecido que combine *hardware* e *SMA* levando em consideração alguns detalhes importantes de interfaceamento de dispositivos, comportamentos de recursos físicos e identificação de middleware para comunicação, por exemplo. Além disso, a comunidade tende a criar modelos sem padronização, gerando representações que não são consistentes entre si. À medida que a complexidade do *hardware* aumenta, cresce também a complexidade de integrar sistemas de controle e monitoramento, principalmente se este for um SMA Embarcado.

Sendo assim, o objetivo deste trabalho é propor uma linguagem de modelagem para SMA Embarcados que permita representar a estrutura e o comportamento do *hardware* e do SMA de um dispositivo físico. A linguagem de modelagem proposta está dividida em três dimensões: Multiagente, estrutural e comportamental. A camada Multiagente é responsável pela modelagem do SMA utilizando o *FAME Agent-oriented Modeling Language* (FAML) [Beydoun et al. 2009] como base, uma vez que o FAML agrupa conceitos e técnicas das principais linguagens de modelagem de SMA. Já a camada estrutural é responsável pela modelagem dos recursos e componentes de *hardware*, suas conexões físicas e funções no ambiente. Por fim, a camada comportamental é responsável pela modelagem algorítmica das funções físicas que cada recurso ou componente exerce no ambiente.

Este artigo está estruturado da seguinte forma: na seção 2 será apresentado o referencial teórico e alguns trabalhos relacionados serão destacados; na seção 3, as noções gerais do modelo, bem como as três linguagens de modelagem serão detalhadas; e por fim, a seção 4 traz as considerações finais do trabalho.

2. Referencial Teórico e Trabalhos Relacionados

A evolução da computação promoveu mudanças disruptivas na maneira que a informação é utilizada para alterar o ambiente e automatizar processos. Neste cenário nasce a Computação Ubíqua ou Pervasiva [Krumm 2018], responsável pelos sistemas computacionais capazes de realizar atividades contínuas de maneira transparente, ou seja, sem que o usuário perceba a sua presença no ambiente. Naturalmente, estas características trazem para Computação Ubíqua as noções de ambientes inteligentes, ou AmI [Weber and Rabaey 2005]. AmI é uma sub-área da computação ubíqua focada em ambientes compostos de dispositivos sensoreados e com capacidade de atuação. Os dispositivos neste ambiente cooperam entre si para realização de tarefas que auxiliam o usuário de maneira pervasiva.

Em ambientes inteligentes, os dispositivos presentes são Sistema Ciberfísico (CPS), que são estruturas de *hardware* combinadas a uma camada computacional para realizarem tarefas e processos bem definidos [Alur 2015]. Quando o *software* está embutido e dedicado ao controle e monitoramento do *hardware* no qual está instalado, ele passa a ser conhecido como um sistema embarcado [Marwedel 2021]. Dependendo do domínio e do objetivo do CPS, o seu funcionamento pode envolver processos cognitivos, demandando soluções capazes de deliberar sobre o *hardware* de maneira autônoma e proativa. Neste sentido, SMA podem ser utilizados com o objetivo de oferecer a estrutura necessária para realizar os processos cognitivos demandados.

Um SMA é um conjunto de agentes que cooperam entre si para realizar tarefas complexas [Wooldridge 2009]. Estes agentes são entidades computacionais autônomas capazes de interagir no ambiente que estão inseridos, tomar decisões para alcançar determinados objetivos e realizar interações com outros agentes [Bordini et al. 2007]. O ambiente de um SMA pode ser endógeno ou exógeno [Ricci et al. 2009]: um ambiente endógeno é aquele contido dentro do SMA e que modela as ferramentas que serão exploradas pelos agentes; já um ambiente exógeno pode ser entendido como aquele que é percebido e afetado pelo agente no contexto real. Na prática, o ambiente exógeno representa o ambiente real (físico ou virtual) que impacta e é impactado pelo agente, enquanto que o ambiente endógeno é aquele que modela o ambiente exógeno dentro do contexto multiagente.

Ao desenvolver SMA Embarcados em dispositivos no contexto ciberfísico, as dificuldades existentes estão no seu fator multidisciplinar, uma vez que existe demanda para as áreas de eletrônica, computação e automação. Consequentemente, ou uma equipe extensa deve ser montada ou haverá sobrecarga de atividades sobre um desenvolvedor que não possui domínio em todas as disciplinas envolvidas. Para auxiliar neste processo, uma das abordagens possíveis está na modelagem prévia do sistema. Esta etapa tem como vantagem oferecer uma visão clara sobre o domínio do projeto, sem que seja necessário lidar com questões tecnológicas precocemente. Além disso, a etapa de modelagem auxilia na documentação do projeto, garantindo que times multidisciplinares e com alta rotatividade

de pessoal se mantenham alinhados sobre a estrutura de domínio do projeto.

Esta modalidade de desenvolvimento de projeto é conhecida como desenvolvimento dirigido a modelo (ou *Model-Driven Development* — MDD), que utiliza ferramentas baseadas na arquitetura dirigida a modelos (ou *Mopdel-Driven Architecture* — MDA). Neste contexto, já existem diversas ferramentas de modelagem como o Adelfe, PASSI, Gaia, INGENIAS e Tropos. Contudo, elas são utilizadas apenas para representar SMA e não possuem uma representação padronizada entre elas. Com isso, surge o FAML como um metamodelo que unifica essas e outras ferramentas de modelagem existentes.

O *FAME Agent-oriented Modeling Language* (FAML) [Beydoun et al. 2009] é um metamodelo genérico para desenvolvimento de SMA que combina metamodelos anteriores, como TAO e Islander, com outras abordagens de modelagem como o Adelfe, PASSI, Gaia, INGENIAS e Tropos. A partir dessa combinação, o FAML propõe um metamodelo capaz de abranger os conceitos das outras linguagens de modelagem. Para isso, foram propostos quatro níveis diferentes de modelagem: de sistema, de definição de agentes, de ambiente e de agente. Os dois primeiros focalizam a modelagem para representar o ambiente e os agentes do SMA em tempo de design, enquanto que os dois últimos focam no tempo de execução. Apesar do FAML abranger grande parte dos conceitos relativos a um SMA, o metamodelo não está preparado para definir com um nível de detalhe maior os componentes de *hardware*. Uma vez que o metamodelo engloba as principais linguagens de modelagem para agentes, não há necessidade de discuti-las individualmente. A abordagem proposta nesse trabalho busca definir em um metamodelo os conceitos e regras necessárias para a criação de um SMA Embarcado.

3. Proposta da Linguagem de Modelagem para SMA Embarcado

Este trabalho vai em direção a uma linguagem de modelagem que deve cobrir o processo de desenvolvimento de SMA Embarcados desde a criação de diagramas independentes de plataforma até a geração do código específico de plataforma como o JaCaMo e arduino. Esta transição deverá acontecer da seguinte maneira: o diagrama será instanciado em um metamodelo independente de plataforma; a partir da instância do metamodelo, será gerada uma estrutura específica de plataforma; essa estrutura será então utilizada num processo de tradução que irá gerar os códigos do SMA Embarcado. Para ir em direção à linguagem de modelagem descrita, este trabalho propõe um metamodelo independe de plataforma que comporta as definições e regras deste tipo de um SMA Embarcado, dividindo-o em três dimensões: Multiagente, Estrutural e Comportamental.

Cada uma das dimensões está focalizada em uma competência específica do projeto e portanto, pode ser modelada por profissionais de diferentes competências. Por exemplo, a dimensão Estrutural pode ser elaborada por um profissional da área de eletrônica, a dimensão Comportamental por um profissional de automação e a dimensão Multiagente por um profissional da inteligência artificial.

Por fim, cada dimensão do metamodelo possui uma ponte de conexão com a camada vizinha, o que permite que toda a modelagem do SMA Embarcado esteja interconectada entre as diferentes partes: SMA, ambiente exógeno e programação comportamental. As seções a seguir apresentam de forma detalhada cada um dos Modelos, suas regras e os metamodelos construídos.

3.1. Dimensão Multiagente

A dimensão Multiagente é do metamodelo permite representar o SMA com base no metamodelo proposto no FAML [Beydoun et al. 2009]. Neste sentido, o objetivo é preservar as contribuições do FAML ao máximo para garantir a maior aderência possível às linguagens de modelagem já existentes na área de SMA. No entanto, a dimensão Multiagente proposta neste trabalho possui algumas diferenças em relação ao FAML a fim de possibilitar maior aderência ao domínio de SMA embarcados.

Por exemplo, neste trabalho, o FAML passou por um processo de fusão entre os seus quatro níveis para permitir que a modelagem do sistema adote conceitos de tempo de *design* e tempo de execução. Assim, será possível modelar os planos dos agentes (*Plan*) e também seu estado mental inicial (*MentalState*), caso faça sentido no projeto modelado. Além disso, os conceitos de *FacetDefinition* e *Resource* foram combinados de modo que cada recurso do ambiente seja responsável por uma ou mais facetas que os agentes podem interagir. Na dimensão estrutural, a ideia de *FacetDefinition* e *Resource* serão reaproveitadas para representar o *hardware*.

Para criar os elementos da linguagem de modelagem Multiagente, o metamodelo da figura 1 foi construído inspirado no FAML, utilizando os seguintes conceitos:

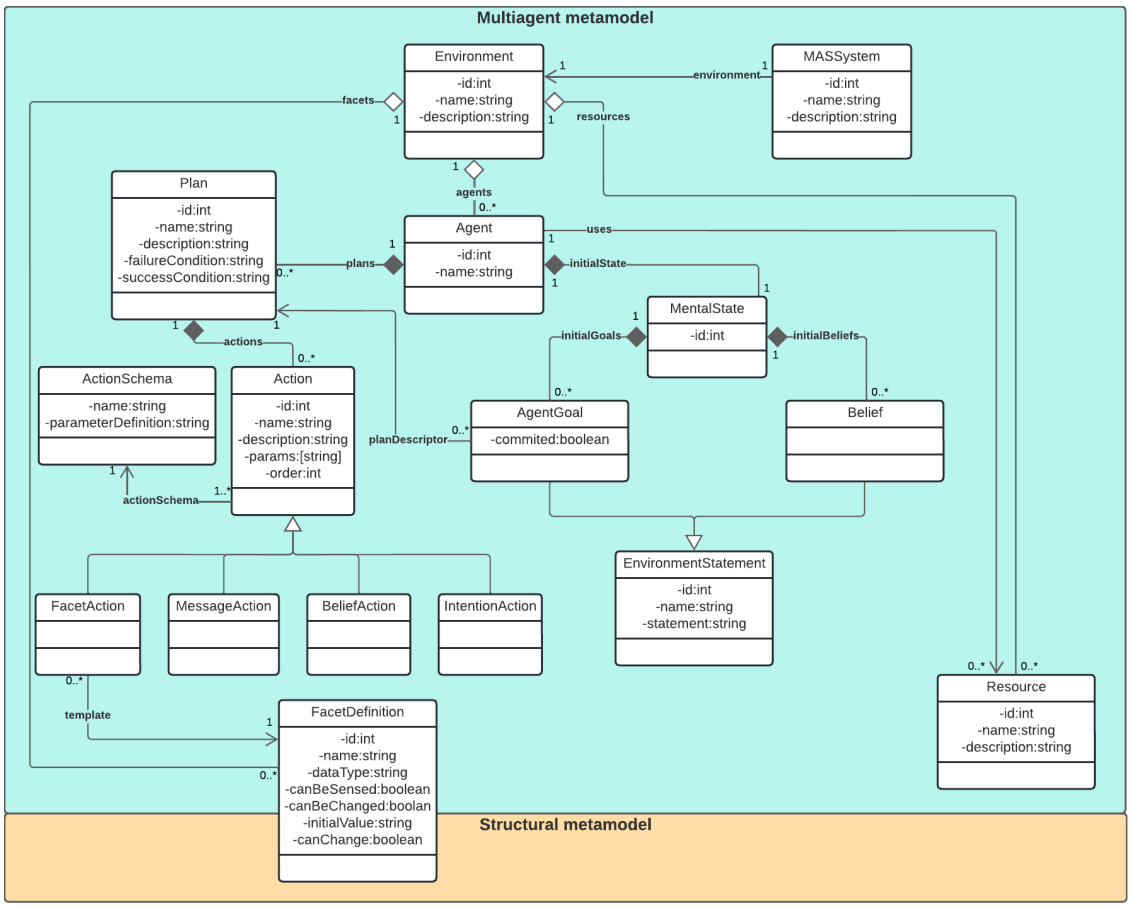

Figura 1. Dimensão Multiagent.

- *MASSystem*: representa o produto final do projeto de software orientado a agentes.

Cada projeto de modelo possui apenas um *MASSystem*, que é a entidade que acopla todos os elementos criados do metamodelo Multiagente.

- *Environment*: é o ambiente endógenos no qual os agentes estão situados.
- *Agent*: é a entidade autônoma e racional do ambiente. Enquanto o FAML separou a ideia de agentes entre tempo de design (*Agent Definition*) e tempo de execução (*Agent*), a dimensão Multiagente agrupou estes dois conceitos em *Agent*. Sendo assim, a representação de *Agent* durante a modelagem pode ser convertida para ambos os tipos de agentes do FAML.
- *MentalState*: é o estado mental onde são guardadas as crenças e objetivos do agente. No FAML, o *MentalState* armazena as crenças e objetivos durante o ciclo de execução do agente, mas na dimensão Multiagente, o objetivo é utilizar o *MentalState* para representar o estado inicial deste agente.
- *EnvironmentStatement*: é a declaração feita sobre o ambiente. *Beliefs* e *AgentGoals* são considerados como *EnvironmentStatement*.
- *Belief*: é a crença do agente.
- *AgentGoal*: é o obejetivo do agente. Este objetivo deve estar assiciado a um *Plan*.
- *Plan*: é uma coleção de ações que podem ser executadas para que o agente atinja um objetivo em particular.
- *Action*: é a unidade fundamental do comportamento do agente. Uma *Action* pode ser uma *MessageAction* ou *FacetAction*.
- *Action*: é a unidade fundamental do comportamento do agente. Uma *Action* é composta por parâmetros, cuja regra de parametrização segue o que é descrito no *ActionSchema* instanciado na *Action*. Além disso, uma *Action* pode ser especializada como uma *MessageAction*, *FacetAction*, *BeliefAction* ou *IntentionAction*. Estes dois últimos conceitos não estão previstos no FAML, mas foram adicionados neste tranalho para dar maior suporte aos agentes BDI.
- *MessageAction*: é uma *Action* que resulta no envio de uma mensagem.
- *FacetAction*: é uma *Action* que resulta na alteração de uma interface de acesso ao ambiente exógeno.
- *FacetDefinition*: é a estrutura de uma interface de acesso ao ambiente exógeno, incluindo nome, tipo de dado, modo de acesso e valor inicial.
- *BeliefAction*: é uma *Action* que manipula as crenças que o agente terá. Ou seja, esta *Action* adiciona ou remove um plano da base de crença do agente.
- *IntentionAction*: é uma *Action* que declara a intenção do agente de realizar um plano.
- *Resource*: é definido como alguma coisa que possui nome, representação razoável e pode ser adquirido, compartilhado ou produzido.

Na dimensão apresentada, *FacedDefinition* é um conceito utilizado pelo FAML para definir uma interface de acesso ao ambiente exógeno, conhecido também como *Facet*. Na linguagem de modelagem proposta neste trabalho, *FacetDefinition* é o elemento da dimensão Multiagente que deve ser interligado a outro elemento da dimensão Estrutural para viabilizar a troca de informações entre o ambiente exógeno e o SMA. Em [Boissier et al. 2013], a ideia de *Facets* representa o mesmo conceito, porém renomeada como *Artifacts*.

Outra definição de destaque no metamodelo é *Resource*, que é definido de maneira genérica como algo que possui um nome, uma representação bem definida e que pode ser manipulado. Um *Resource* pode ainda ser utilizado por um agente em tempo de execução.

Na dimensão Multiagente, um *Resource* ganha uma definição mais orientada ao ambiente exógeno e é utilizado também para mapear um *PhysicalResource* da dimensão Estrutural. Apesar de *FacetDefinirion* e *Resource* serem conceitos independentes na dimensão Multiagente, estes dois conceitos se combinam no Modelo Estrutural.

3.2. Dimensão Estrutural

A dimensão Estrutural permite representar a estrutura do ambiente exógeno e a relação dele com o SMA. Portanto, o foco desta dimensão não estará concentrado nas crenças, comportamentos ou organizações sociais dos agentes, assim como os detalhes sobre o funcionamento algorítmico do *hardware* também será desprezado. Nesta dimensão, o objetivo é estruturar recursos físicos (*Physical Resources*) e os componentes que os formam (*Component*), descrever a conexão que estes componentes fazem com o microcontrolador (*Microcontroller*), definir as funções (*Function*) que estes componentes terão e por fim, definir quais informações que os recursos poderão trocar com o SMA. Esta dimensão foi inspirada na representação de hardware do trabalho que propõe abordagens de engenharia para SMA embarcados em uma arquitetura de gerenciamento de recursos para IoT [Brandão et al. 2021]. Com isso, a dimensão Estrutural segue uma abordagem *Sensor-as-a-Service* (SaaS) e fica mais próxima de uma futura modelagem envolvendo IoT.

Para esta dimensão, foi construído o metamodelo da figura 2 com o objetivo de definir todos os elementos do modelo, a forma que eles devem se relacionar entre si e a cardinalidade deste relacionamento. Além disso, cada elemento deste metamodelo foi formalizado e descrito abaixo:

- *PhysicalResource*: é um resurso de *hardware* do ambiente exógeno que possui escopo bem definido de funcionamento e é reconhecido como elemento específico do domínio do projeto. Além disso, é capaz de atuar no ambiente realizando *Commands* e disponibilizar informações sobre este ambiente através de *Topics*. Por fim, um *PhysicalResource* ainda pode ser combinado com outros para compor um recurso mais complexo que unirá todas as características que antes eram vistas isoladamente. Ou seja, os recursos teclado, mouse, gabinete e monitor, que já possuem escopos bem definidos, podem compor o recurso notebook, que também estará com escopo bem definido.
- *Function*: é o elemento que representa um comportamento algorítmico de *Components* ou *PhysicalResources*. Na dimensão Estrutural, é possível apenas indicar a existência das *Functions* sem descrever o comportamento por trás delas. Isso porque o detalhamento comportamental de uma *Function* é mais amplo e por isso requer um metamodelo específico para defini-lo completamente. A melhor maneira de pensar em uma *Function* neste metamodelo é associá-la às funções da linguagens de programação que serão usadas para definir o comportamento físico dos *Components* e *PhysicalResources*.
- *Topic*: é um elemento que disponibiliza informações que um *PhysicalResource* pode coletar do ambiente. Portanto, *Topics* serão interfaces que o SMA poderá acessar para receber informações do ambiente exógeno.
- *Command*: é um elemento que representa uma atuação que o *PhysicalResource* pode fazer no ambiente. Portanto, *Commands* serão ações que o SMA poderá realizar para afetar o ambiente exógeno.

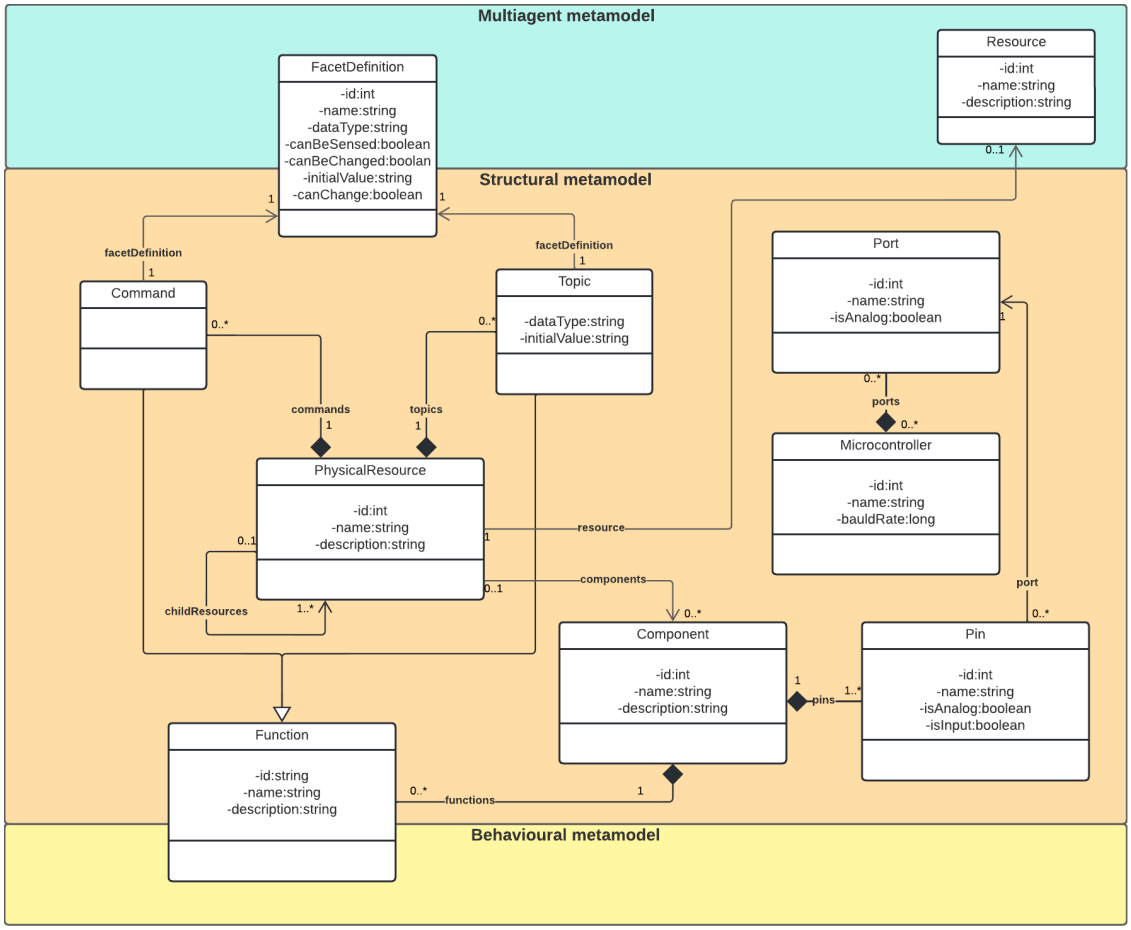

Figura 2. Dimensão Structural.

- *Component*: é o elemento que será usado como componente eletrônico do *hardware* de *PysicalResources*. Enquanto um *PhysicalResource* pode ser entendido como uma unidade funcional e específica para o contexto do projeto, um *Component* pode ser entendido como uma ferramenta livre do contexto do projeto e que possui conexão física com o microcontrolador do hardware. Em outras palavras, um *Component* é a ponte que une o microcontrolador aos recursos. Na prática, *Components* podem ser utilizados para representar eletrônicos como motores, válvulas, LEDs, etc. Além disso, os *Components* são extensíveis o suficiente para suportar a modelagem de eletrônicos menos comuns. Com isso, a modelagem de componentes na dimensão estrutural tem potencial para customizações, o que significa que se um novo dispositivo eletrônico for criado, sua representação já poderia ser modelada como um novo componente.
- *Pin*: é o elemento que representa o pino de um *Component*. Este pino pode ser classificado como analógico ou digital e também como entrada ou saída.
- *Microcontroller*: é a representação do microcontrolador que irá alocar o programa (*firmware*) com todas as *Functions* e outras diretrizes necessárias para o funcionamento do código (como a função *void setup()* do arduino por exemplo). O *Microcontroller* é composto de portas de entrada ou saída e estas podem ser conectadas aos pinos dos componentes eletrônicos.

- *Port*: é a representação da porta digital ou analógica de um *Microcontroller*. Todos os *Components* que estão conectados ao *Microcontroller* realizam esta conexão através da ligação dos seus *Pins* com os *Ports*. Além disso, uma *Port* pode receber conexão de mais de um *Pin*, o que neste caso significaria que estes *Pins* leem ou escrevem dados no mesmo local do *Microcontroller*.

Na dimensão Estrutural, *Commands* e *Topics* realizam integração com o *Facet-Definition* e por conta disso, permite que o SMA opere e monitore *PhysicalResources* do ambiente exógeno. Além disso, a integração entre *Resources* do Modelo Multiagente e *PhysicalResources* do Modelo Estrutural permite que os agentes utilizem/conheça, de maneira indireta, o *hardware* do sistema embarcado.

3.3. Dimensão Comportamental

A dimensão Comportamental deve ser utilizada para descrever o comportamento de uma função (*Function*), que por sua vez está atrelada a um *Component* ou a um *PhysicalResource* (através de *Commands* ou *Topics*). Neste sentido, em um Modelo Comportamental, não deve haver preocupação acerca de como funções como *void loop()* devem funcionar, por exemplo. O objetivo neste Modelo é representar o comportamento lógico e dedicado de um *Component*/*PhysicalResource*, pois durante uma futura conversão deste modelo para uma linguagem de programação, todas as dependências e adaptações que foram abstraídas serão aplicadas para que o comportamento físico seja igual ao representado na modelagem.

A representação de comportamento no diagrama de atividades da UML, uma vez que este diagrama é utilizado para representar atividades em uma sequência lógica. A figura 3 apresenta a dimensão Comportamental do metamodelo. Além disso, os conceitos desta dimensão do metamodelo são detalhados e formalizados abaixo:

- *Parameter*: é um valor que a *Function* recebe para realizar sua atividade. Uma *Function* pode ter nenhum ou muitos parâmetros e cada um desses parâmetros representam valores que a função receberá no início da execução.
- *Block*: é a unidade de comportamento de uma *Function*, onde todos os *Blocks* são distribuído sequencialmente. *Blocks* podem representar especificamente o início ou o final da funcão, uma decisão, uma paralelização ou junção ou uma chamada de função. Por exemplo, a função responsável por piscar um LED a cada segundo terá 6 blocos: um de início, outro de final e os demais serão chamadas de funções para ligar o LED, contar um segundo, desligar o LED e contar msis um segundo.
- *Start*: é o bloco que indica o início da funcão.
- *End*: é o bloco que indica o final da funcão.
- *Fork*: é o bloco no qual a partir dele, o comportamento da função será paralelizado. Por exemplo, se um *PhysicalResource* precisa monitorar ou controlar mais de uma parte do *hardware* simultaneamente, o bloco *Fork* representa este comportamento.
- *Join*: é o bloco no qual a partir dele, o comportamento da função unirá paralelismos existentes. Por exemplo, se um *PhysicalResource* está monitorando ou controlando mais de uma parte do *hardware* simultaneamente, o bloco *Join* irá unificar estes comportamentos ema única Thread.
- *Decision*: é o bloco que indica uma decisão a ser tomada no algorítmo. Para isso, a *Decision* deve conter uma expressão *boolean*. Dependendo da maneira que é

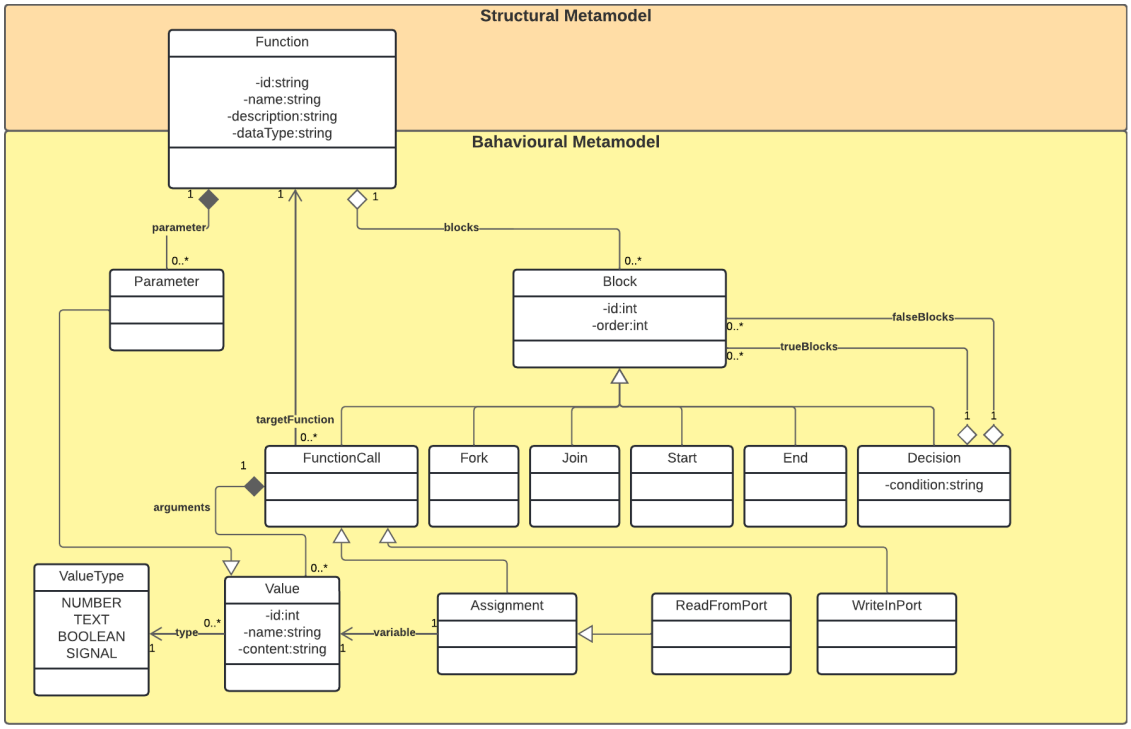

Figura 3. Dimensão *Behavioural*.

utilizada, uma *Decision* pode se tornar uma estrutura de decisão (como um *if*) ou de repetição (como um *while*) no código final. Em particular, uma *Decision* pode se tornar uma estrutura de repetição quando um outro bloco futuro está apontando a direção do próximo comportamento para esta *Decision*. Por fim, um *Block é composto de um conjunto de Blocos para representar as atividades no caso da condição da decisão ser verdadeira e um conjunto de Blocks para representar atividades de condição falsa.*

- *FunctionCall*: é o bloco que faz referência a outras *Functions* criadas. Uma *FunctionCall* pode fazer referência a uma função que foi criada para o projeto específico ou a uma função padrão já existente por uma biblioteca de *Components* padrão.
- *Value*: é a representação de um valor/variável no Modelo Comportamental. Possui um nome, tipo e conteúdo
- *ValueType*: é o tipo que um valor pode assumir (numérico, textual, *boolean* e sinal eletrônico.
- *Assignment*: É uma *FunctionCall* que atribui o seu valor de retorno a uma variável.
- *ReadFromPort*: é um *Assignment* cuja variável recebe como valor o que foi lido de uma porta física do microcontrolador.
- *WriteInPort*: é uma *FunctionCall* que escreve um valor numa porta física do microcontrolador.

4. Considerações Finais

Este trabalho apresentou uma proposta de uma linguagem de modelagem para auxiliar no projeto e especificação de SMA Embarcados. A linguagem de modelagem proposta está dividida em três camadas de especificação: do SMA, a Estrutural e a Comportamental. Um metamodelo foi apresentado integrando todas três dimensões propostas. Na

dimensão Multiagente, foi utilizado como base o FAML, que reúne todos os conceitos das tradicionais linguagens de modelagens orientadas a agentes.

Atualmente, os SMA Embarcados são especificados utilizando apenas as linguagens de modelagem orientada a agentes tradicionais, que nem sempre levam em consideração todos os detalhes que um SMA precisa ter para gerenciamento de recursos físicos. Com isso, muitas vezes o SMA é modelado e uma arquitetura física é apresentada sem a utilização de um padrão ou interconexão de conceitos, dificultando o entendimento do projeto em tempo de design. O metamodelo proposto permite que esses novos conceitos necessários para a criação de um SMA Embarcado seja integrado às linguagens tradicionais orientadas a agentes que já possuem suporte do FAML. Com essa abordagem, será possível que um SMA Embarcado possa ser devidamente documentado permitindo um gerenciamento de atividades tradicionais da engenharia de software como manutenção do software/dispositivo e testes, por exemplo.

Como trabalhos futuros, serão desenvolvidos os diagramas que comporão todas as dimensões do desenvolvimento de um SMA Embarcado, como a dimensão Multiagente, Estrutural e Comportamental. Estes diagramas serão convertidos em instâncias do metamodelo proposto e, a partir disso, será possível fazer a tradução de modelo em código específico de plataforma, ou até mesmo fazer a engenharia reversa do código gerado para diagrama. Este mecanismo de desenvolvimento utilizará técnicas de Desenvolvimento Orientado a Modelos (*Model-Driven Architecture* — MDA). Diversas são as abordagens que exploram a MDA para transformações de modelos e geração de códigos [Pantoja and Choren 2012, Pantoja and Choren 2013, Rosa et al. 2013]. Além disso, também é importante permitir a integração da linguagem de modelagem a um ambiente de desenvolvimento para utilização de todas as funcionalidades que a linguagem e o metamodelo podem oferecer.

Referências

Alur, R. (2015). Principles of cyber-physical systems. MIT press.

Bernon, C., Gleizes, M., Peyruqueou, S., and Picard, G. (2003). ADELFE: a methodology for adaptive multi-agent systems engineering. In Proceedings of the 3rd international conference on Engineering societies in the agents world III, ESAW'02, page 156–169, Berlin, Heidelberg. Springer-Verlag.

Beydoun, G., Low, G., Henderson-Sellers, B., Mouratidis, H., Gomez-Sanz, J. J., Pavon, J., and Gonzalez-Perez, C. (2009). Faml: a generic metamodel for mas development. IEEE Transactions on Software Engineering, 35(6):841–863.

Boissier, O., Bordini, R. H., Hübner, J. F., Ricci, A., and Santi, A. (2013). Multi-agent oriented programming with jacamo. Science of Computer Programming, 78(6):747–761.

Bordini, R. H., Hübner, J. F., and Wooldridge, M. (2007). Programming multi-agent systems in AgentSpeak using Jason, volume 8. John Wiley & Sons.

Brandão, F. C., Lima, M. A. T., Pantoja, C. E., Zahn, J., and Viterbo, J. (2021). Engineering approaches for programming agent-based iot objects using the resource management architecture. Sensors, 21(23):8110.

Bresciani, P., Giorgini, P., Giunchiglia, F., Mylopoulos, J., and Perini, A. (2003). Tropos: An Agent-Oriented Software Development Methodology.

Cossentino, M. (2005). From Requirements to Code with the PASSI Methodology. In Sellers, H. B. and Giorgini, P., editors, Agent-Oriented Methodologies, volume 3690 of LNCS, pages 79–106. Idea Group Pub.

Gascuena, J. M. and Fernández-Caballero, A. (2007). The INGENIAS methodology for advanced surveillance systems modelling. In Proceedings of the 2nd international work-conference on Nature Inspired Problem-Solving Methods in Knowledge Engineering: Interplay Between Natural and Artificial Computation, Part II, IWINAC '07, page 541–550, Berlin, Heidelberg. Springer-Verlag.

Krumm, J. (2018). Ubiquitous computing fundamentals. CRC Press.

Manoel, F. C. P. B., Nunes, P. d. S. M., de Jesus, V. S., Pantoja, C. E., and Viterbo, J. (2017). Applying Multi-Agent Systems in Prototyping: Programming Agents For Controlling a Smart Bathroom Model With Hardware Limitations.

Marwedel, P. (2021). Embedded system design: embedded systems foundations of cyber-physical systems, and the internet of things. Springer Nature.

Padgham, L. and Winikoff, M. (2004). Developing Intelligent Agent Systems: A Practical Guide. Wiley Series in Agent Technology. John Wiley.

Pantoja, C. E. and Choren, R. (2012). A MDA Approach for Agent-oriented Development using FAML. In ICEIS (2), pages 415–420.

Pantoja, C. E. and Choren, R. (2013). A MDA Methodology to Support Multi-Agent System Development. In ICAART (1), pages 393–396.

Ricci, A., Piunti, M., Viroli, M., and Omicini, A. (2009). Environment programming in CArtAgO. In Seghrouchni, A., Dix, J., Dastani, M., and Bordini, H. R., editors, Multi-Agent Programming: Languages, Tools and Applications, pages 259–288. Springer US, Boston, MA.

Rosa, A., Gonçalves, I., and Pantoja, C. E. (2013). A MDA Approach for Database Modeling. Lecture Notes on Software Engineering, pages 26–30.

Sakurada, L., Barbosa, J., Leitão, P., Alves, G., Borges, A. P., and Botelho, P. (2019). Development of Agent-Based CPS for Smart Parking Systems. In IECON 2019 - 45th Annual Conference of the IEEE Industrial Electronics Society, volume 1, pages 2964–2969.

Weber, W. and Rabaey, J. (2005). Ambient intelligence. Springer Science & Business Media.

Wooldridge, M. (2009). An introduction to multiagent systems. John Wiley & Sons.

Zambonelli, F., Jennings, N. R., and Wooldridge, M. (2003). Developing multiagent systems: The gaia methodology. ACM Trans. Softw. Eng. Methodol., 12(3):317–370.