João Filipe da Costa Ralo
Licenciado em Ciências da Engenharia
Eletrotécnica e de Computadores

Representação da Linguagem nesC Usando Técnicas Baseadas em Modelos

Dissertação para obtenção do Grau de Mestre em
Engenharia Eletrotécnica e de Computadores, pela
Universidade Nova de Lisboa, Faculdade de Ciências e Tecnologias

Orientador: Professor Doutor Pedro Miguel Negrão Maló,
Professor Auxiliar, FCT-UNL

Coorientador: Mestre Edgar Miguel Felício Oliveira da Silva,
UNINOVA-CTS

Júri:

Presidente: Professor Doutor Luís Manuel Camarinha de Matos

Arguente: Professor Doutor André Teixeira Bento Damas Mora;

Vogal: Professor Doutor Pedro Miguel Negrão Maló,

FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Janeiro, 2017

Representação da linguagem nesC usando técnicas baseadas em Modelos

Agradecimentos

Em primeiro lugar quero agradecer ao meu orientador, o Professor Doutor Pedro Maló, e ao meu coorientador, Edgar Silva, pela oportunidade de trabalhar com eles, por toda a disponibilidade demonstrada desde o primeiro dia e por tudo o que aprendi na realização desta dissertação. Quero agradecer também pelas infraestruturas cedidas para a realização desta dissertação.

Quero deixar um grande agradecimento à minha família, pelo apoio incondicional dado ao longo da minha vida pessoal e académica e por me terem dado condições para poder completar a minha formação superior. Quero também deixar uma palavra de agradecimento especial à minha irmã que neste momento está literalmente do outro lado do mundo. Obrigado pela motivação e apoio ao longo do curso, tenho saudades tuas.

Por falar em curso, não posso deixar de agradecer aos meus colegas de curso João Lourenço, Nuno Ramos, Joaquim Pereira, Renato Paulino, Gonçalo Oliveira, João Soares, André Coelho, Alexandra Videira, Flávio Silva e José Ferreira pela sua amizade e companheirismo nesta fase da minha vida. Quero deixar um agradecimento especial ao meu colega Nuno Ramos, pela ajuda com a formatação das imagens neste documento. Foi um prazer partilhar convosco esta fase do meu percurso académico. Ajudaram a tornar estes anos numa altura memorável da minha vida. Desejo-vos tudo de bom no vosso futuro e que continuemos a fazer as nossas jantaradas.

Deixo também um agradecimento muito especial à Maria Inês Lopes. Foste a minha companhia nos últimos sete anos e espero ter muito mais anos pela frente a teu lado. Sem ti não teria feito PEBio.

Para finalizar quero agradecer a todas as pessoas que direta, ou indiretamente, influenciaram a minha vida permitindo-me chegar a este ponto.

Resumo

A Internet-das-Coisas (IdC) é, essencialmente, um sistema de máquinas ou objetos equipados com tecnologias de recolha de dados tais que estes possam comunicar entre si sem intervenção humana. Este paradigma de tecnologia possibilita um vasto número de implementações, tais como Cidades Inteligentes, Transportes Inteligentes, etc. Essas implementações são formadas por dispositivos autónomos e de diferentes plataformas de hardware, capazes de monitorar condições ambientais, tais como temperatura, som e humidade. Devido à heterogeneidade destes dispositivos, o desenvolvimento de sistemas dedicados à IdC é complexo visto que, atualmente, não existem métodos para auxiliar a criação e gestão destes sistemas.

Tendo em conta as suas limitações (processamento, memória, etc), estes dispositivos são classificados como Dispositivos de Recursos Limitados (DRL). Possuem um sistema operativo próprio, sendo o TinyOS o mais utilizado. Este recorre à linguagem de programação *Network Embedded System C* (nesC). É então essencial a existência de um formalismo que habilite sistemas/ferramentas na geração automática de código, consequentemente, implementações IdC. A falta de tal especificação dificulta o desenvolvimento de aplicações, pois é necessário que os engenheiros possuam total conhecimento (detalhes técnicos) da linguagem de programação.

Este trabalho teve como principal objetivo utilizar metodologias orientadas a modelos para descrever e formalizar a linguagem de programação nesC. O metamodelo nesC é apresentado utilizando classes UML (*Unified Modelling Language*) e linguagem Ecore. A partir do metamodelo foi possível efetuar a transformação para outras linguagens (nomeadamente XML - *eXtensible Markup Language*) e a geração automática de código nesC. Com esta formalização é então possível a integração com ferramentas que abstraiam o utilizador dos pormenores técnicos da linguagem ou, por exemplo, a integração com sistemas de simulação de dispositivos IdC.

Palavras-chave: Internet-das-Coisas, nesC, Modelação, MDA, UML, Ecore.

Abstract

The Internet of Things (IoT) is essentially a system of machines or objects outfitted with data-collecting technologies so that those objects can communicate with one another. This technology paradigm enables a vast number of implementations, i.e. Smart Cities, Smart Transports, etc. These applications are based on autonomous devices that record environmental and physical variables, such as temperature, sound and humidity. Due to the devices heterogeneity, the development of dedicated IoT systems is complex since there are no methods to assist the creation and management of these systems.

Considering their hardware limitations (processing, memory, etc.), devices are classified as Resource Constrained Devices (RCD). They have their own Operating Systems (OS), being TinyOS the most used. This OS uses the Network Embedded System C (nesC) programming language. Therefore, it is essential the existence of a formalism that enables systems/tools to automatically generate application code, consequently, IoT deployments. The lack of such specification hinders the development of applications, since it is necessary that engineers have complete knowledge (technical details) of the programming language.

This work aimed at using model-driven methodologies to describe and formalize the nesC programming language. The nesC metamodel is presented using UML classes (Unified Modelling Language) and Ecore language. From the metamodel it was possible to implement a transformation to other languages (in this case XML - eXtensible Markup Language) and automatically generate nesC code. This formalization enables the integration with tools that abstract the user from the technical details or, for example, the integration with IoT device simulation systems.

Palavras-chave: Internet-of-Things, nesC, Modeling, MDA, UML, Ecore.

Tabela de Conteúdos

Tabela de Figuras

Índice de Tabelas

Lista de Acrónimos

AOM – Arquitetura Orientada a Modelos

ATL – ATLAS Transformation Language

CIM – *Computation Independent Modeling*

COOJA – *Contiki Os JAva*

CPU – *Central Processing Unit*

DRL – Dispositivos de Recursos Limitados

DSL – *Domain-Specific Language*

EMF – *Eclipse Modeling Framework*

EMP – *Eclipse Modeling Project*

HTML – *HyperText Markup Language*

IdC – *Internet*-das-Coisas

IEEE – *Institute of Electrical and Electronics Engineers*

IoT – *Internet-of-Things*

IPv4 – *Internet Protocol version* 4

IPv6 – *Internet Protocol version* 6

MDA – *Model-Driven Architecture*

MDE – *Model-Driven Engineering*

MIC – Modelo Independente de Computação

MIP – Modelo Independente de Plataforma

MOF – *Meta-Object Facility*

MEP – Modelo Especifico de Plataforma

nesC – *Network Embedded Systems* C

OMG – *Object Management Group*

PIM – *Platform Independent Model*

PSM – *Platform Specific Model*

QVT – *Query/View/Transformation*

RAM – *Random Access Memory*

ROM – *Read-Only Memory*

RSF – Redes de Sensores sem Fios

SGML – *Standard Generalised Markup Language*

SO – Sistema Operativo

TOSSIM – *TinyOS SIMulator*

UML – *Unified Modeling Language*

WSN – *Wireless Sensor Networks*

XMI – *XML Metadata Interchange*

XML – *eXtensible Markup Language*

1

Introdução

O constante desenvolvimento da infraestrutura e aplicações da *Internet* e o aparecimento de novas tecnologias, nomeadamente nas áreas da microeletrónica e dos protocolos de telecomunicação, criou condições para existirem implementações de redes segundo o paradigma da Internet-das-Coisas (IdC), também referida como IoT – *Internet-of-Things* em inglês. Este consiste em interligar objetos do quotidiano tornando a tecnologia omnipresente, de modo a proporcionar uma integração perfeita da tecnologia/computação com o ambiente do ponto de vista do utilizador. Possibilitando assim a implementação de um vasto número de áreas de aplicação, como por exemplo cidades inteligentes (*Smart Cities*), redes de energia inteligentes (*Smart Grids*), transportes inteligentes (*Smart Transport*) e domótica (*Smart Buildings*) (Vermesan e Friess, 2014).

A IdC é considerada a terceira revolução das tecnologias de informação, sendo a primeira a implementação de paginas *web* estáticas (www), seguida pela introdução de páginas interativas com o utilizador (*web2.0*), como por exemplo redes sociais (Gubbi et al., 2013).

A introdução destes "objetos inteligentes" na *Internet* foi uma das motivações para a criação do protocolo IPv6 (Savolainen et al., 2013), devido à incapacidade do protocolo IPv4, com capacidade para 4 294 967 296 (2^{32}) endereços, responder ao elevado número de dispositivos que se tornaram acessíveis pela *Internet* nas ultimas duas décadas.

Tipicamente, um "objeto inteligente" consiste num sensor capaz de medir variáveis do ambiente em que se encontra, por exemplo luz, som, movimento, humidade entre outros, ao qual se acrescenta capacidade de processar, armazenar e comunicar com outros dispositivos os dados recolhidos, ainda que, devido a restrições energéticas, estas capacidades sejam severamente limitadas (Kortuem et al., 2010). Consequentemente, estes sensores são categorizados como Dispositivos de Recursos Limitados (DRL) e habitualmente são implementados em grupo como uma

Rede de Sensores sem Fios (RSF), designada em inglês como WSN – *Wireless Sensor Network*, onde comunicam entre eles e para a *Internet*.

A implementação de RSF pode ser feita em qualquer ambiente (industrial, selvagem, doméstico, etc.) e como estas redes são concebidas para aplicações extensas em termos de tempo, tipicamente tarefas de monitorização, os vários dispositivos da rede são dimensionados e construídos para requerem pouca manutenção, mas por outro lado é necessário garantir alta eficiência energética dos dispositivos.

A IdC inclui serviços, redes e qualquer dispositivo com capacidades de comunicação com outros. Estas características permitem uma conectividade a uma escala muito maior do que a que existia há uma década atrás. Com o número de dispositivos que se tornam acessíveis pela *Internet* a crescer a uma velocidade exponencial, a IdC aos poucos vai-se estabelecendo na sociedade e no modo de vida dos utilizadores.

1.1. Identificação do Problema

O desenvolvimento de *software* para DRLs é feito com recurso a linguagens de programação de baixo nível (como por exemplo Linguagem C), mas também existem sistemas operativos dedicados a estes dispositivos dados os seus recursos limitados (memória, processamento, bateria, etc.) (Lajara et al., 2010).

Como o desenvolvimento de software IdC envolve programação de muito baixo nível, isto traduz-se numa elevada complexidade na geração de aplicações funcionais para DRLs. Este facto implica que programadores tenham que possuir um conhecimento profundo de como desenvolver *software* neste tipo de programação. Dada esta falta de formalismo na descrição de linguagens IdC, as ferramentas de desenvolvimento de *software* existentes não possuem o grau de abstração necessário para que alguém, que não conheça a linguagem a fundo, possa facilmente criar as suas aplicações.

Desenvolver estes formalismos e descrições de linguagens existentes possibilitaria um avanço na conceção de ferramentas de desenvolvimento de *software* IdC que proporcionassem um nível de abstração superior ao estado atual, ocultando a complexidade inerente a linguagens de baixo nível. Desta forma seria possível, por exemplo, desenvolver um sistema de geração de código compilável automaticamente.

Por outro lado, o crescimento da utilização de DRL nas mais variadas áreas de aplicação, levanta questões sobre o seu desempenho e integridade em implementações a médio/longo prazo, pois há que garantir o mínimo de padrões de funcionamento para se obter resultados fidedignos e ser rentável a sua aplicação.

1.2. Objetivos da Dissertação

Esta dissertação tem como objetivo desenvolver um formalismo de discrição para uma linguagem de programação utilizada na IdC, nomeadamente, um metamodelo da linguagem nesC (*network embedded system* C), uma variação da linguagem C desenvolvida estaticamente para DRL. É pretendido também desenvolvimento de um mecanismo de transformação de modelos entre nesC e XML (*eXtensible Markup Language*), uma vez que o XML é um formalismo padrão na descrição de dados. Para fins de validação, será concebido um processo de geração de ficheiros de código nesC a partir do metamodelo, de forma a gerar automaticamente código compilável.

1.3. Metodologia de Investigação

A metodologia adotada para o desenvolvimento da investigação proposta para esta dissertação segue os passos do método científico, ver Figura 1.1.

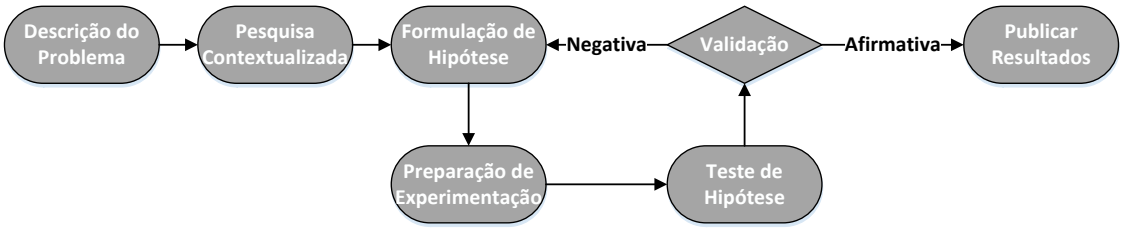

Figura 1.1 – Método Científico.

Os passos do método cientifico passam por inicialmente introduzir o problema e contextualizar a área de estudo. De seguida elabora-se uma pesquisa sobre a tecnologia existente relacionada com o domínio do problema. Em terceiro lugar, é formulada uma hipótese de resolução do problema, elaborando depois uma experiência de modo a validar a hipótese. Se os resultados forem afirmativos, o problema pode ser considerado resolvido. Caso contrário, é necessário ajustar/reformular a hipótese consoante os resultados e voltar a testar a sua validade.

1.4. Organização do Documento

Do primeiro ao sexto capitulo encontra-se o corpo principal do documento, iniciando com o capítulo atual que fornece uma breve introdução e contextualização do domínio de investigação, terminando com a identificação do problema. O restante documento está organizado de acordo com a seguinte estrutura:

- **Capítulo 2 – Estado de Arte:** Contém um estudo de investigação sobre os temas principais para estudo do domínio do problema. É dada uma visão geral sobre alguns conceitos específicos do problema, tecnologias existentes mais utlizadas pelos investigadores, baseado em documentos existentes, como por exemplo artigos de jornais científicos ou dissertações para obtenção de graus de educação superior. Este capítulo finaliza com a definição da pergunta de investigação.

- **Capítulo 3 – Solução Proposta:** Definição da hipótese de investigação com base na pesquisa efetuada e na pergunta de investigação definida no capítulo anterior. Descrição da solução proposta para a implementação da hipótese.

- **Capítulo 4 - Implementação:** Descrição das abordagens utilizadas para implementação da solução proposta no capítulo anterior

- **Capítulo 5 - Validação:** Demonstração e discussão dos resultados da implementação da solução proposta.

- **Capítulo 6 - Conclusões e Trabalho Futuro:** Neste capítulo serão discutidas as principais conclusões a retirar do trabalho realizado e trabalho futuro.

Depois do corpo principal do documento, existe um capítulo com a bibliografia consultada no decorrer do desenvolvimento desta dissertação, seguido de seis capítulos de anexos com o código gerado na implementação e validação deste trabalho.

2

Estado de Arte

Neste capítulo será feita uma pesquisa sobre as tecnologias existentes com relevância para esta dissertação. Em primeiro lugar, nas Secções 2.1 e 2.2, será abordada a camada de *hardware* das RSF (Redes de Sensores sem Fios) e o seu *software*. Finalmente, nas Secções 2.4 e 2.5, será discutida uma abordagem orientada a modelos que, atualmente, é considerada bastante promissora – Arquitetura Orientada a Modelos (AOM), referida em inglês como MDA (*Model-Driven Architecture*).

2.1. Dispositivos Sem Fios

Para qualquer aplicação de uma Rede de Sensores sem Fios (RSF), em inglês WSN – *Wireless Sensor Networ*k, o seu sistema físico consiste num conjunto de sensores aos quais se adiciona capacidade de armazenamento (memória), fonte de energia (bateria autónoma ou não), poder de computação (CPU – *Central Processing Unit*) e capacidade de comunicação com outros sensores/estações base (Gajjar et al., 2014). A Figura 2.1 ilustra um destes sensores.

Estes dispositivos, devido à sua limitação em termos de *hardware*, são categorizados como *resource-constrained devices* (Ishaq et al., 2012), Dispositivos com Recursos Limitados (DRL), e também são designados por *motes* ou *sensor nodes*. Num estudo comparativo dos principais modelos deste tipo de dispositivos realizado em 2014 (Gajjar et al., 2014), é possível ter uma ideia de quão limitados são os seus recursos, como por exemplo, memória reduzida (entre

1kB e 512kB de RAM, *Random Access Memory*, e 32kB até 4MB de memória *FLASH*) e processadores de baixa frequência (entre 8Mhz e 180Mhz).

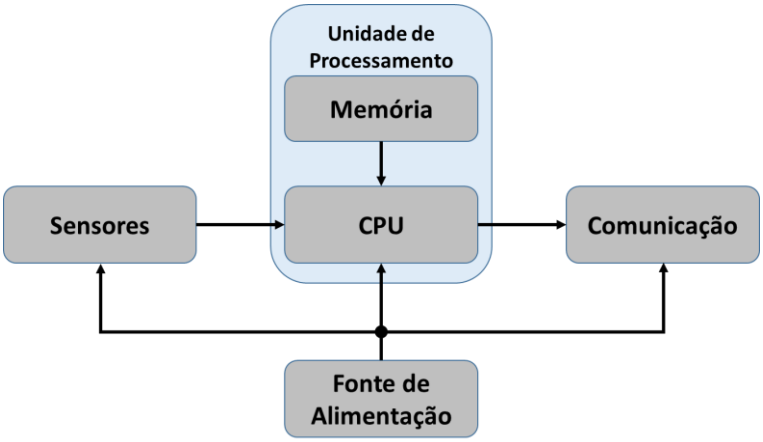

Figura 2.1 – Arquitetura de um sensor sem fios.

Estes aparelhos captam e processam dados, comunicando com outros elementos dentro da rede através de protocolos de rede. Para isto, possuem um sistema operativo desenhado especificamente para estas aplicações (ver Secção 2.2), controladores de sensores e de comunicações. Estes dispositivos têm que ser bastante reduzidos em termos de tamanho e por esta razão são por vezes designados por "*smart dust*".

A comunicação numa rede sem fios é feita de acordo com o protocolo IEEE 802.15.4 que utiliza uma frequência de aproximadamente 2.4 GHz. Tipicamente os sensores comunicam a informação entre si até esta chegar a uma estação base, também designada por *sink*. Esta faz a comunicação entre a rede e Internet, funcionando como um *gateway*, fazendo as conversões entre os protocolos de comunicação das duas redes, de modo a que a informação fique disponível remotamente para os utilizadores da rede (Figura 2.2) (Lin et al., 2009).

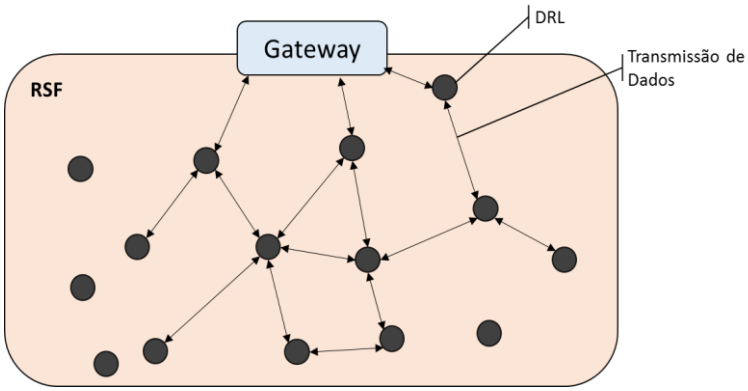

Figura 2.2 – Arquitetura de uma RSF.

Grande parte da investigação elaborada nestes dispositivos foi feita em Berkeley, Universidade da Califórnia, e originou os motes Rene, Mica (Figura 2.3) e Mica2. Os modelos Telosb também são bastante utilizados para aplicações Internet-das-Coisas (IdC).

Figura 2.3 – Dispositivo Mica (Welsh and Lorincz, 2006).

2.2. Sistemas Operativos

Para que um dispositivo execute as aplicações que lhe são destinadas, necessita de uma camada intermédia entre estas e o *hardware*. Com isto em mente, foram desenvolvidos vários Sistemas Operativos (SO) vocacionados especificamente para aplicações e dispositivos IdC. Estes controlam o funcionamento do *hardware*, implementam os protocolos de rede e regulam o consumo energético para maximizar o tempo de vida de um dispositivo.

Do lado do programador é necessário escolher bem o método de desenvolvimento da aplicação, orientado a eventos ou utilizando várias *threads*, de modo a conceber o programa para a máxima eficiência dos recursos disponíveis num dispositivo.

De acordo com um estudo (Lajara et al., 2010) realizado para determinar a percentagem de publicações que se referem a sistemas operativos dedicados a IdC, realizado nas principais bases de dados científicas, concluiu-se que: 81% das publicações referem-se ao uso do *TinyOS*, 9% para o *Contiki*, 8% para o *Mantis* e 2% para outros SO's, Figura 2.4. Consequentemente o estudo dos SO para DRL foca-se no *TinyOS* por este ser o mais utilizado, e no SO *Contiki* por ser baseado em linguagem C, uma linguagem de programação muito utilizada. O *Contiki* acaba por ser o principal concorrente do primeiro.

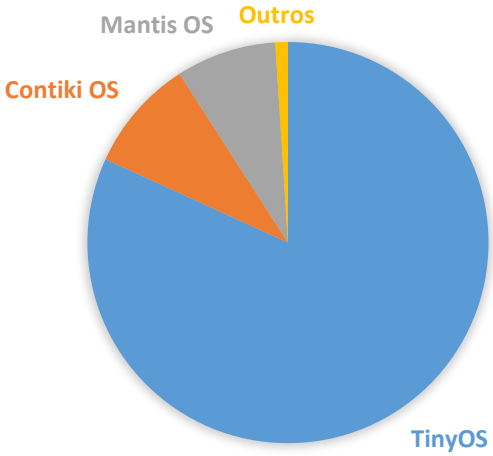

Figura 2.4 – Distribuição de SO's em aplicações IdC (Lajara et al., 2010).

2.2.1. TinyOS

Desenvolvido no *campus* de Berkeley da Universidade da Califórnia, em conjunto com a *Intel* e a *Crossbow Technologies*, o *TinyOS* é um sistema operativo *open-source* concebido para dispositivos sem fios de baixa potência (Levis et al., 2005). Atualmente na versão 2.1.2 (agosto de 2012), este SO continua ainda a ser melhorado.

A sua implementação está feita de modo a responder a quatro requisitos fundamentais numa rede de sensores sem fios:

- Baixos requisitos de *hardware*: a grande maioria dos dispositivos sem fios utilizados em aplicações e redes IdC possuem recursos bastante limitados, tais como capacidade de armazenamento de dados e processamento, como descrito na secção 2.1;
- Concorrência reativa: as várias tarefas que um dispositivo tem que executar têm de ser sincronizadas (como por exemplo a amostragem de valores de diferentes sensores e/ou operar sobre atuadores);
- Flexibilidade: para que seja possível responder à variedade de *hardware* e aplicações presentes no mercado e que seja independente do *firmware* dos dispositivos;
- Baixo consumo energético: característica fundamental no desenho de redes de sensores sem fios para operarem com o mínimo de manutenção durante o máximo tempo possível.

  As aplicações são construídas utilizando o *nesC* (*network embedded systems C*), uma variação da linguagem C desenvolvida especialmente para o *TinyOS*.

2.2.1.1. nesC – network embedded systems C

nesC significa linguagem C para redes de sistemas embutidos e é uma variação baseada em componentes da linguagem C tendo sido concebida com foco no desenvolvimento de sistemas de rede embutida. Embora esta linguagem possa ser usada para um elevado número de aplicações, a sua utilização é mais comum na construção de aplicações para redes de sensores sem fios e, atualmente, é a linguagem padrão neste campo (Gay et al., 2003). Em contraste com linguagens convencionais, uma aplicação nesC consiste em operações reativas com pouco tempo de vida, acionadas como resposta a estímulos ambientais (por exemplo, sensores de luz), também referidos como eventos.

A estrutura das aplicações nesC é constituída por conjuntos de módulos que fornecem funcionalidades através de interfaces de *software*. Cada módulo contém estados e comportamentos de uma forma semelhante a classes orientadas a objetos. Ainda assim, existem diferenças fundamentais entre módulos e classes. Os módulos são construídos estaticamente, comportam-se como instâncias únicas e são completamente desacoplados entre si, nunca se referenciando diretamente. Por outro lado, cada módulo especifica um conjunto de funções necessárias para a execução do seu comportamento declarando as interfaces que forem essenciais (Gay et al., 2003). Na Figura 2.5 está representada a definição do módulo *BlinkC* da aplicação *Blink* que é instalada por defeito com a instalação do TinyOS. Esta aplicação acende os leds do *mote* em intervalos regulares.

```
module BlinkC @safe()
{
  uses interface Timer<TMilli> as Timer0;
  uses interface Timer<TMilli> as Timer1;
  uses interface Timer<TMilli> as Timer2;
  uses interface Leds;
  uses interface Boot;
}
```

Figura 2.5 – Módulo BlinkC.

Consequentemente, os módulos nesC definem as interfaces que fornecem, assim como as interfaces que utilizam. Além da declaração de comandos, uma interface pode ainda declarar eventos. Módulos que utilizem uma interface são obrigados a implementar um *handler* de eventos com a assinatura especificada. Este *handler* é chamado como resposta a eventos sinalizados pelo

fornecedor da interface. Logo, uma interface nesC é bidirecional. Outra característica do nesC é o suporte a *tasks*, uma alternativa "leve" às *threads*. Uma *task* é uma função privada de um módulo, declarada para uso futuro pelo planeador de tarefas do *TinyOS*. As *tasks* são usadas tipicamente para operações de longa duração ou operações que têm que ser executadas sem interrupção, possibilitando a sua execução assincronamente com o resto do programa (Gay et al., 2009).

Uma forma comum de codificação nesC envolve a execução de um certo tipo de função de longa duração (transmissão de dados, amostragem sensorial, etc.) iniciado por um certo comando. Este comando inicia o trabalho do *hardware* e imediatamente devolve o controlo à entidade que o invocou. Quando o *hardware* sinaliza o fim da operação, o *handler* de interrupção respetivo disponibiliza uma tarefa, que, quando executada, sinaliza o fim da ação no modulo inicializador (fora do contexto de interrupção). Outro método é a existência de uma biblioteca de um conjunto de *tasks* que dividem a operação do processador em operações mais pequenas.

Para além de módulos, as aplicações nesC são constituídas por componentes especiais designados por configurações. Cada configuração interliga as interfaces utilizadas por um conjunto específico de módulos para outros módulos que fornecem essas mesmas interfaces. Sintaticamente, este "mapa" tem a forma de uma lista de módulos participantes e uma série de afirmações que unem essas interfaces.

Note-se que vários módulos fornecedores podem estar ligados à mesma interface. Uma invocação através desta interface é comunicada a todos os fornecedores ligados.

2.2.2. Contiki

O *Contiki* é um sistema operativo *open-source*, portável, com capacidade de executar várias tarefas em paralelo, para sistemas embutidos e redes de sensores sem fios (Vortler et al., 2015). Foi desenvolvido por um grupo de investigadores liderados por Adam Dunkels do *Swedish Institute of Computer Science* (Paul and Kumar, 2009).

Uma configuração *Contiki* contém, tipicamente, 2 kB de RAM e 50 kB de ROM (*Read Only Memory*). Este sistema operativo consiste num *kernel event-driven*, sobre o qual é possível executar aplicações dinamicamente. O *Contiki* fornece comunicações IP, quer por IPv4 ou IPv6 (Vortler et al., 2015).

Uma das principais vantagens do *Contiki* é a utilização de *protothreads*. Estas consistem em "operações condicionais de espera e podem ser utilizadas para reduzir o número de máquinas

de estado explicitas em programas *event-driven* para sistemas embutidos com memória limitada" (Dunkels et al., 2006). A utilização de memória pelas *protothreads* é muito baixa porque partilham a mesma pilha de memória e a troca entre *threads* requere apenas um pequeno retrocesso nas posições da pilha. Os programas para este SO são escritos na linguagem C.

2.2.3. Simulação de DRL e RSF

A conceção de uma Rede de Sensores sem Fios (RSF) é uma tarefa que requer uma análise cuidada caso-a-caso, pois as características de ambientes de implementação de RSF são fundamentalmente diferentes entre cada ambiente que se considere. Logo, é necessário proceder a testes meticulosos previamente à fase de implementação da rede em hardware, quer através de testes concebidos para o efeito ou utilizando simuladores fidedignos. Uma RSF tem que ser submetida a uma vasta gama de condições dinâmicas, de modo a simular a resposta da infraestrutura da rede, nomeadamente um conjunto de Dispositivos de Recursos Limitados (DRL), em relação a possíveis alterações repentinas no ambiente em que será inserida.

Um simulador de DRL (Dispositivos de Recursos Limitados) consiste em vários módulos distintos, nomeadamente eventos, ambiente, nós, protocolos e aplicações. Um simulador deve ser possuir capacidade de escalamento de redes, suporte para linguagens para definir experiências, suporte gráfico e ferramentas de auxílio à deteção de erros (Sundani et al., 2011).

Apesar destas características, os simuladores disponíveis possuem as suas limitações. Alguns pecam pela falta de disponibilidade de modelos de protocolos, o que aumenta o tempo de desenvolvimento, alguns são limitados em termos de escalabilidade, o que é desvantajoso para simular redes de maior dimensão. Também podem surgir problemas de modelação quando se consideram novos ambientes e componentes de *hardware*, nomeadamente relacionados com o consumo energético.

De seguida descrevem-se brevemente alguns simuladores.

2.2.3.1. TOSSIM

TOSSIM (TinyOS SIMulator) (Lee, 2003) é o simulador que é instalado de raiz com o TinyOS, sendo bastante utilizado devido à disseminação deste sistema operativo no mercado. Existem modelos de implementação para componentes de *hardware*, como por exemplo CPU, memória e rádio, tirando assim partido da arquitetura baseada em componentes do TinyOS.

A abordagem de baixo nível permite que cada componente seja uma entidade, estas são importadas para uma simulação de eventos discretos, marcando e disparando uma série de eventos, de acordo com o código. Este simulador só é compatível com aplicações TinyOS e com arquiteturas Mica.

2.2.3.2. COOJA

O COOJA – *COntiki Os JAva* (Osterlind et al., 2006) é um simulador baseado em Java concebido para simular RSF em que os dispositivos utilizam o sistema operativo Contiki. Este simulador é capaz de simular redes compostas por dispositivos heterogéneos, não só a nível de *software*, mas também a nível do *hardware* simulado.

Um dispositivo simulado com esta ferramenta possui três propriedades: memória, tipo de dispositivo e periféricos. Podem existir vários dispositivos do mesmo tipo na simulação, partilhando todos as mesmas características, embora durante a sua execução os dados que guardam em memória são diferenciados devido a diferenças de *input* em cada dispositivo. No entanto, quanto maior e diversificada for a RSF simulada, maior poder de processamento será necessário.

2.2.3.3. Avrova

Este simulador é compatível com aplicações para dispositivos baseados em controladores Atmal AVR, em dispositivos MICAz e MICA2. A sua maior vantagem é a independência em termos de linguagens de programação e de sistema operativo de implementação, tornando-se uma mais valia para flexibilidade e portabilidade de simulações (Titzer et al., 2005).

A simulação ao nível da camada de instruções significa que o *firmware* do controlador do dispositivo é executado no simulador, negando a necessidade de adaptar as aplicações para este efeito. Cada componente de *hardware* simulado é representado por uma classe de objetos, deste modo, o modelo de *hardware* simulado é uma combinação da sua hierarquia de componentes.

2.2.3.4. NS – Network Simulator

O NS (*Network Simulator*) é um simulador de eventos discretos direcionado para a investigação de redes de comunicação de sistemas. Esta ferramenta fornece apoio para a simulação de protocolos TCP (*Transmission Control Protocol*), encaminhamento e *multicast* em redes com e sem fios (Sundani et al., 2011). As simulações são executadas numa combinação de linguagem C++ e OTcl, adotando uma abordagem modelar à simulação para melhorar a sua extensibilidade. Nos últimos anos surgiu o NS-3 que permite a implementação de modelos de simulação suficientemente próximos da realidade para permitir que este simulador possa ser usado para emular redes em tempo-real (NS-3 Consortium, 2011).

2.2.3.5. Sumário

Todos os simuladores abordados possuem as suas características chaves e as suas limitações. A Tabela 2.1 sumariza as características dos vários simuladores, servido de comparação entre eles. Tendo em conta estas limitações, uma abordagem orientada a modelos poderá servir para ultrapassá-las de uma forma genérica. A modelação será discutida nas secções seguintes.

Tabela 2.1 – Resumo dos vários tipos de simuladores de DRL e RSF (Sundani et al., 2011;Silva et al., 2016).

Nome	Linguagem de programação	Características	Limitações	Arquiteturas compatíveis
TOSSIM	nesC	Alto nível de precisão a correr o código da aplicação sem alterações. Ferramenta de visualização disponível.	A compilação pode provocar alterações nas propriedades de temporização e interrupção implementadas.	MicaZ
COOJA	Java (simulações em C)	Simula quer *hardware* quer *software*. Podem ser observados algoritmos e comportamentos de maior escala.	Não é extremamente eficiente. Suporta um número limitado de tipos de *nodes*. Dificulta simulações longas e dependentes de tempo.	TI MSP430 e Atmel AVR
Avrora	Java	Pode simular redes com até 10000 dispositivos. Possibilita a validação de propriedades dependentes do tempo em redes de grande dimensão.	50% mais lento que o TOSSIM. Não consegue modelar mobilidade dos nós da rede.	AVRMCU
NS-2	C++/OTcl	Facilidade de adicionar novos protocolos. Grande número de protocolos disponíveis publicamente. Ferramenta de visualização disponível.	Suporta apenas dois protocolos. MAC, 802.11 e um TDMA *single-hop*.	Não aplicável

2.3. Modelos e Metamodelos

Um modelo consiste na definição de uma porção da realidade que seja observável e interpretável. É conseguido utilizando termos abstratos e relações para emular elementos existentes no mundo-real que se pretendem modelar. É possível criar modelos representativos de diferentes aspetos da mesma realidade, utilizando diferentes linguagens, formalismos e paradigmas (Agostinho, 2012).

Um modelo deve estar definido numa linguagem apropriada, pois a simbologia e as relações que representam o sistema que está a ser modelado têm que ser descritas de uma forma coerente semântica e sintaticamente. Atualmente a linguagem de modelação mais utilizada é o UML (*Unified Modelling Language*) (ver Secção 2.3.4). Uma linguagem de modelação é por sua vez descrita por um meta-modelo. Este consiste num modelo que especifica conceitos e relações utilizadas por uma linguagem de programação, o que resulta em afirmações e conclusões sobre o que é possível expressar num modelo válido utilizando uma certa linguagem. Um metamodelo, sendo ele próprio um modelo, tem que ser descrito numa linguagem coerente – metalinguagem (Clark et al. 2002; Agostinho 2012).

Um metamodelo é concebido para ser adequado à criação de modelos com um certo nível de abstração e para conter conceitos descritivos do domínio de aplicação e relevantes para a realização do produto final. A partir destes é possível gerar editores de sintaxe e validadores de semântica necessários para a manipulação e validação de modelos conforme o metamodelo. É ainda possível criar transformações que possibilitam transformar modelos definidos segundo um metamodelo noutros modelos definidos segundo outros metamodelos, ou correr esses modelos diretamente. Com metamodelos é possível criar ferramentas necessárias para desenvolver o produto final, como por exemplo editores de texto e geradores de código (Pomante et al., 2015). Na Figura 2.6 estão ilustradas as relações descritas acima.

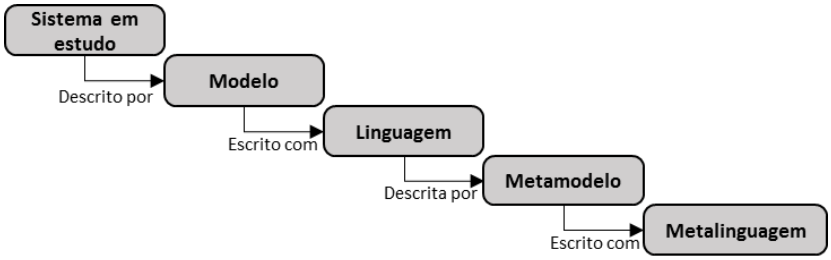

Figura 2.6 – Relações entre os conceitos de Modelação.

2.3.1. Ecore

Linguagens específicas de domínio, DSL – *Domain-Specific Language* em inglês, são a chave para que profissionais de qualquer domínio escrevam as suas próprias aplicações. A meta-modelação foi introduzida para flexibilizar este processo e uma ferramenta bastante popular para modelação é o EMF – *Eclipse Modeling Framework*, que faz parte do projeto de modelação do Eclipse, EMP – *Eclipse Modeling Project*. O Ecore é o metamodelo do EMF (Jorges and Steffen, 2012). Na Figura 2.7 está representada uma vista geral do metamodelo Ecore.

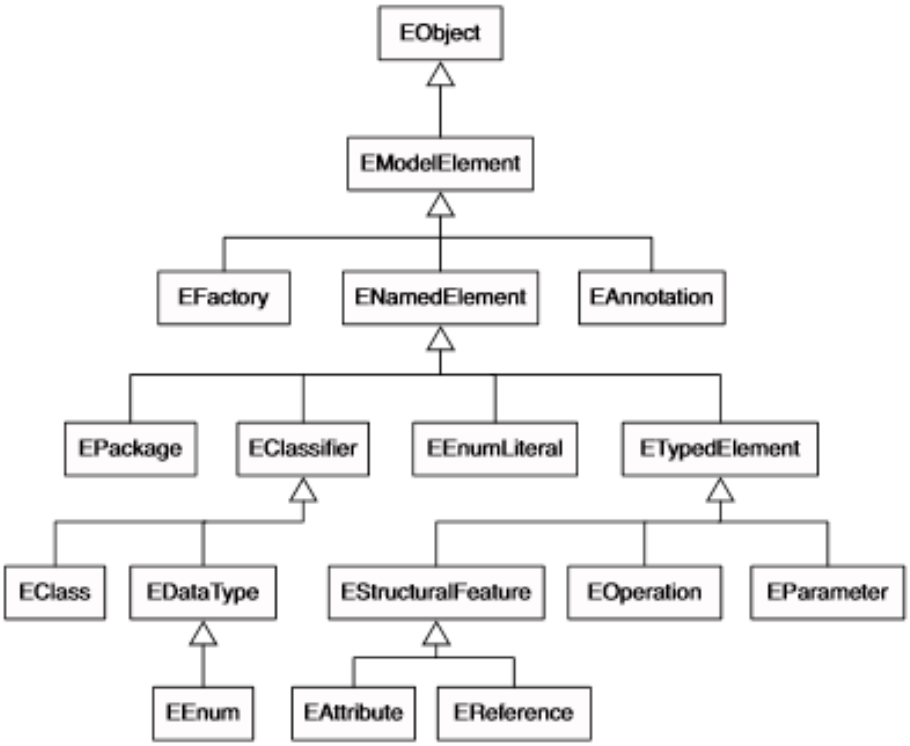

Figura 2.7 – Visão geral do metamodelo Ecore. Fonte: (Jorges and Steffen, 2012)

A reflexividade do Ecore permite modelar hierarquias completas de DSL's como uma "sequência" de modelos, cada um sendo o metamodelo do seguinte. Em relação à geração de código, o EMF fornece ferramentas para gerar código para várias finalidades a partir de modelos Ecore e as suas instâncias.

2.3.2. XML – *eXtensible Markup Language*

Trata-se de uma linguagem de *markup* baseada num subconjunto de outra linguagem, denominada SGML (*Standard Generalised Markup Language*) (Hankins, 1992), e foi concebida por um comité da *World Wide Web* devido à necessidade de generalizar o HTML (*HyperText Markup Language*) (Raggett, 1997), utilizado para formatação de páginas *web*.

O XML é utilizado para guardar informação para leitura quer por humanos quer por máquinas. Por exemplo, guardar e mostrar uma lista de contactos ou integração de vários sistemas independentes através de interfaces definidas em XML. Atualmente existe um vasto número de iniciativas para substituir métodos de formalização proprietários por métodos baseados em XML devido à simplicidade inerente à sua utilização. Esta linguagem é descrita pelo metamodelo representado na Figura 2.8.

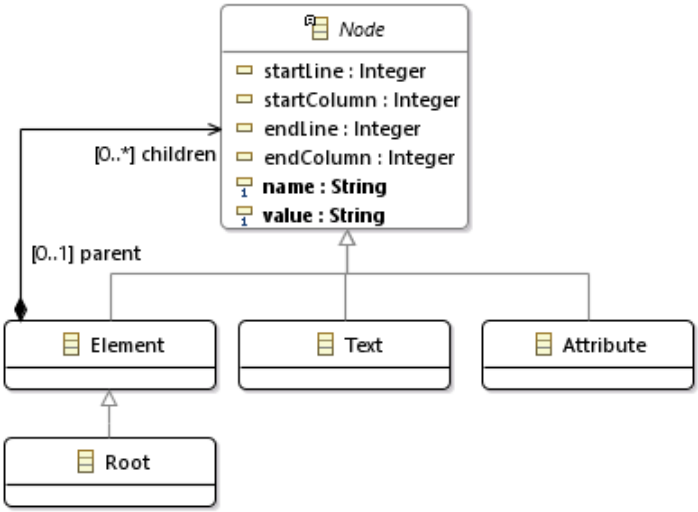

Figura 2.8 – Diagrama de Classes UML representando o metamodelo de XML.

O XML é composto por nós (classe "Node") que podem ou não definir a linha e coluna de inicio e de fim da sua definição no ficheiro XML (atributos "startLine"/"startColumn" e "endLine"/"endColumn" respetivamente) e têm que conter um nome (atributo "name") e um valor associado (atributo "value"). Esta classe pode ser um de três subtipos: elementos (classe "Element"), que podem referenciar outros nós (classe "Node") como subtipos; texto (classe "Text") e atributos (classe "Atribute"). O primeiro elemento de cada ficheiro XML é definido como raiz (classe "Root") pois é o elemento que vai conter toda a informação descrita no ficheiro, ou seja, contem todos os outros nós que fazem parte do ficheiro, seja de que tipo forem.

2.3.3. XMI – *XML Metadata Interchange*

É um método padrão para troca de meta-informação. O objetivo é auxiliar na transformação de modelos de informação para aplicações em UML e outras linguagens e ferramentas de desenvolvimento. O formato XMI padronizou a descrição de conjuntos arbitrários de meta-informação, o que leva a que utilizadores de diferentes áreas e ambientes operacionais possam aceder à informação da mesma maneira (Tieqiang Li et al., 2010). Esta é a base da interoperabilidade e da integração de sistemas.

O XMI fornece uma representação *standard* de objetos XML, permitindo a troca eficaz de objetos entre diferentes entidades, especifica como criar *schemas* XML a partir de modelos, permitindo ao utilizador criar ficheiros XML simples, que podem crescer em complexidade com o desenvolvimento das aplicações que os utilizem (Tieqiang Li et al., 2010).

Um *schema* é um mecanismo para definir a estrutura de uma classe de documentos XML. O objetivo é definir e descrever o significado, a utilização e as relações entre os vários componentes de um documento XML: tipos de dados, elementos, atributos, entidades e notação. Os *schemas* informam sobre o seu próprio significado, utilização e funcionalidade. Logo, um *schema* XML pode ser utilizado para definir vocabulários XML para classes de documentos XML.

2.3.4. UML – *Unified Modelling Language*

Durante os anos 90, a modelação visual de *software* estava contagiada pela incompatibilidade das diferentes nomenclaturas existentes. As poucas ferramentas que existiam, devido à falta de padrões de nomenclatura, não permitiam a verificação da consistência interna dos diagramas ou processamento da informação neles contida. Estes esboços eram úteis como documentação ou para auxiliar no desenho do *software*, mas raramente eram integrados no ciclo de desenvolvimento do *software*. O surgimento do UML alterou este processo e despoletou um crescimento acentuado na modelação de sistemas levando à sua utilização generalizada na engenharia de *software*, sistemas e no setor empresarial (Watson, 2008). Esta linguagem é uma especificação padrão do OMG (*Object Management Group*) e uma linguagem de modelação. Foi concebida para ser utilizada numa vasta gama de áreas profissionais. É definida utilizando MOF (*Meta-Object Facility*) e consiste em treze tipos de diagramas, representados na Figura 2.9 (Feinerer,

2007). Esta figura demonstra as relações entre os diferentes tipos de uma forma hierárquica (Martin Fowler, 2004) (Seidl et al., 2015).

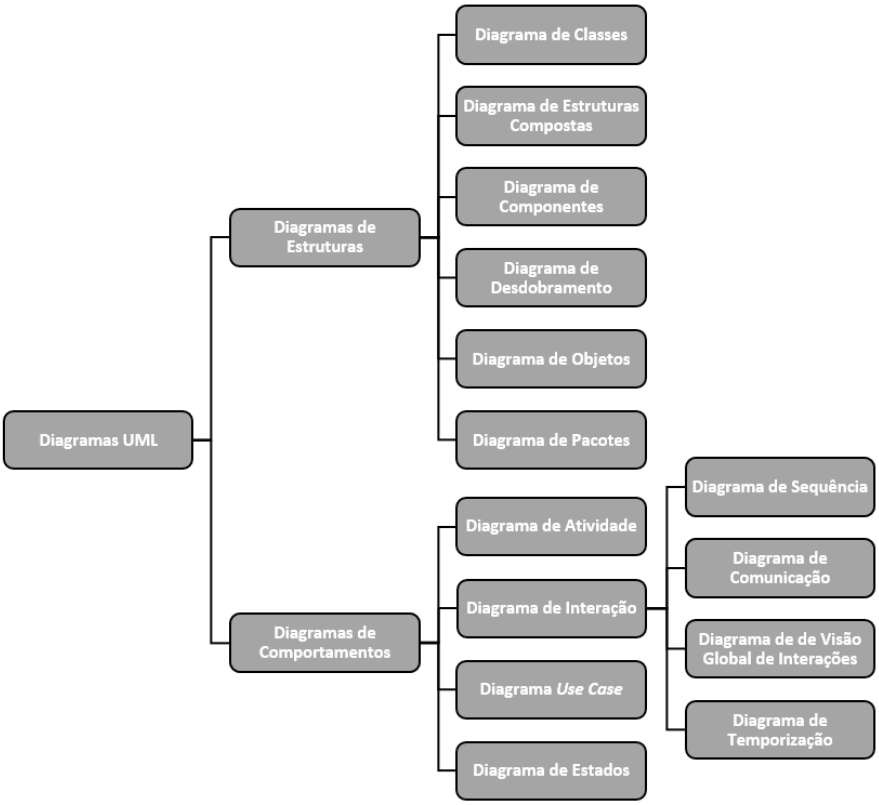

Figura 2.9 – Hierarquia para tipos de diagramas em UML 2.0.

Nesta dissertação só serão abordados os Diagramas de Classes da linguagem UML. Estes descrevem classes e os seus atributos, métodos e associações entre elas em contexto estático. Uma classe é uma representação conceptual de uma entidade a ser modelada. Os objetos de uma certa classe são denominados como instâncias. Cada classe pode possuir vários atributos. Estes atributos podem ser públicos, protegidos e privados para regular o seu acesso por classes externas. Um método é uma operação definida para cada classe, podendo existir vários para a mesma classe. Tipicamente estes acedem e modificam atributos de uma classe. O UML define o conceito de associação, agregação, composições e generalizações. As instâncias de uma associação são denominadas por ligações.

A Figura 2.10 representa a descrição de uma família utilizando a um diagrama de classes UML. Existe uma classe principal “Family” que possui o apelido familiar (atributo “lastName”), comum a todas as pessoas da família. Esta classe contém zero ou mais [0..*] instâncias da classe abstrata “Person” que representa uma pessoa, possuindo nome próprio (“firstName”), apelidos (“middleNames”) e idade (“age”). Cada pessoa pode ser definida como uma criança (classe

"Children") ou adulto (clasee "Parents"), uma criança pode ser rapaz (classe "Son") ou rapariga (classe "Daughter"), o mesmo se aplica aos adultos (classes "Father" e "Mother" respetivamente).

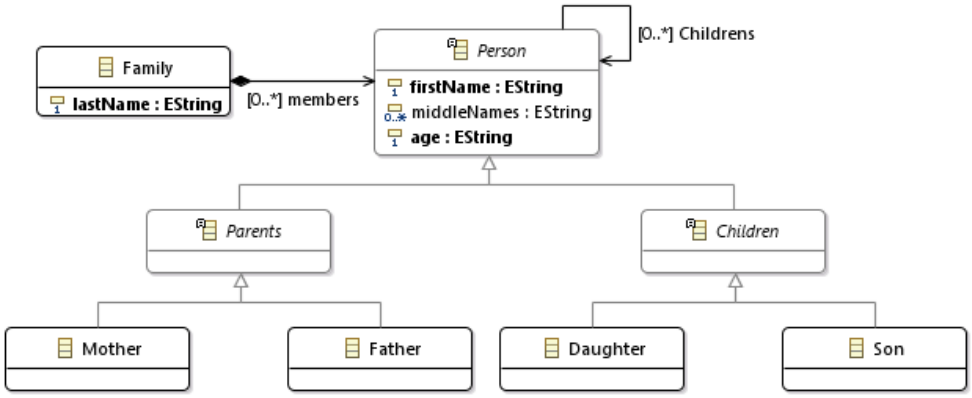

Figura 2.10 – Representação de uma família genérica com um Diagrama de classes UML.

2.3.5. MOF – *Meta-Object Facility*

O *Object Management Group* (OMG) introduziu o MOF (*Meta-Object Facility*) 1.0 em 1997, atualmente na versão 2.0, como ferramenta de gestão de meta-informação para o desenvolvimento e interoperabilidade de modelos e sistemas que utilizam meta-informação, como por exemplo ferramentas de modelação e desenvolvimento de aplicações, *software* de gestão de serviços, entre outros. Um conjunto de tecnologias especificadas pelo OMG utilizam MOF ou derivados para troca e manipulação de meta-informação, como por exemplo UML e XMI.

Um metamodelo utiliza MOF para definir formalmente a sintaxe de alguns conceitos de modelação e especifica alguma da semântica informalmente através de linguagem natural. Esta combinação de definições formais e informais é o chamado metamodelo MOF (Frankel, 2003), também referido como modelo MOF na indústria.

2.4. MDE - *Model-Driven Engineering*

Recentes avanços tecnológicos, quer em termos de linguagens de programação quer em termos de plataformas de desenvolvimento, elevaram o nível de abstração no desenvolvimento de *software*. Hoje em dia são utilizadas linguagens de programação mais orientadas a objetos, como por exemplo C# ou Java, ao contrário de linguagens de mais baixo nível como C e Fortran. Consequentemente as bibliotecas utilizadas nestas linguagens já possuem elas próprias definições

de *middleware* (tolerância a falhas, gestão de recursos, etc), o que facilita o trabalho de desenvolvedores pois "protege" de alguma complexidade inerente ao desenvolvimento de *software*.

Ainda assim, a complexidade das aplicações cresce mais rápido que as linguagens que as definem e isso cria o seu próprio conjunto de problemas. Manter sistemas extremamente complexos atualizados consome tempo e recursos, pois as plataformas de desenvolvimento evoluem rapidamente, e novas aparecem regularmente. É necessário transformar o código para plataformas de desenvolvimento diferentes ou para versões mais recentes das plataformas utilizadas. Outro problema que surge é a diferença entre a intenção do programador e a forma como a linguagem permite que esta seja expressa.

Uma abordagem para exprimir conceitos de domínio de forma mais eficaz é o desenvolvimento de tecnologias MDE, também referido como MDD – *Model-Driven Development* (Pavlos, 2013), que combinam linguagens de modelação específicas de domínio e geradores e motores de transformação de modelos (Schmidt, 2006). Como as linguagens orientadas a objetos que partem do pressuposto de que tudo é um objeto, MDE parte do princípio de que tudo é um modelo, logo tudo pode ser modelado, diminuindo significativamente a complexidade na fase de planeamento de sistemas. Utilizar diagramas para exprimir um certo domínio permite que um individuo, que não conheça a linguagem de baixo nível, possa participar no processo de criação de *software*. Com estas tecnologias é possível desenvolver linguagens específicas de um domínio, DSML – *Domain-Specific Modeling Language*, "à medida" através de metamodelos para corresponder com precisão à semântica de domínio e à sua sintaxe.

2.4.1. MDA – *Model-Driven Architecture*

Lançado em 2001, o MDA é a implementação MDE mais relevante atualmente. É uma iniciativa do *Object Management Group* (OMG) para simplificar o processo de conceção de *software* utilizando uma abordagem orientada a modelos. Os seus três objetivos principais são a portabilidade, interoperabilidade e reusabilidade (Truyen, 2006).

MDA em si não é uma especificação nova, mas sim uma estratégia de desenvolvimento de *software* que é possível devido à existência de outras especificações OMG, como por exemplo UML e MOF. Um sistema MDA divide-se numa hierarquia com três níveis de abstração de informação (Figura 2.11):

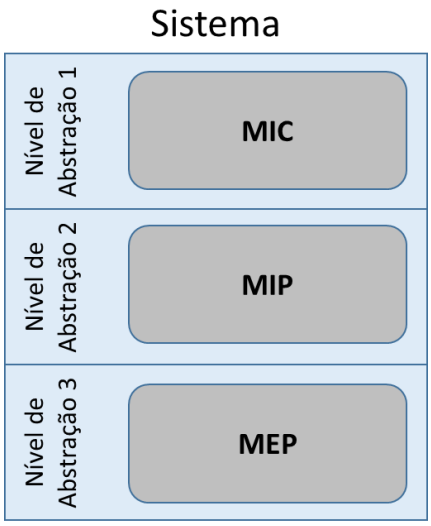

Figura 2.11 – Níveis de Abstração MDA.

- **MIC – Modelo Independente de Computação:** referido em inglês como CIM – *Computation Indenpent Model*, neste nível são especificados os requisitos do sistema a desenvolver e do ambiente onde este terá que trabalhar. É também denominado por modelo de domínio pois é baseado em vocabulário específico ao domínio e é orientado para indivíduos com conhecimento específico a este. Ou seja, serve de "ponte" entre peritos da área de implementação e programadores do *software* a ser implementado.

- **MIP – Modelo Independente de Plataforma:** referido em inglês como PIM – *Platform Independent Model*, este nível representa as especificações de operação do sistema de um modo geral para que seja possível utilizá-lo em várias plataformas diferentes. Estas especificações têm que ser imutáveis entre plataformas. É possível utilizar uma linguagem de modelação genérica ou uma para descrever a estrutura formal e a funcionalidade do sistema em questão, omitindo detalhes técnicos.

- **MEP – Modelo Especifico de Plataforma:** também referido em inglês como PSM – *Platform Especific Model*, adiciona detalhe técnico específico de uma plataforma ao nível superior (MEP), para que se possa implementar o modelo na mesma. O MEP é um modelo do mesmo sistema descrito no MIP, e pode adicionar detalhes técnicos da plataforma de implementação, como por exemplo *middleware*, SO e linguagens de programação (Java, C++, nesC, etc.), ou pode ser usado para melhorar o modelo MIP, resultando num MEP apto a ser implementado diretamente. Este modelo descreve o mesmo sistema que o MIP especificando como esse sistema utiliza a plataforma de implementação.

2.4.2. Transformações de Modelos em MDA

Para traduzir a mesma informação entre os vários níveis de abstração e/ou entre diferentes plataformas alvo, são feitas transformações ao modelo existente para se obter outros modelos equivalentes, sendo que estas podem ser verticais ou horizontais, ver Figura 2.12.

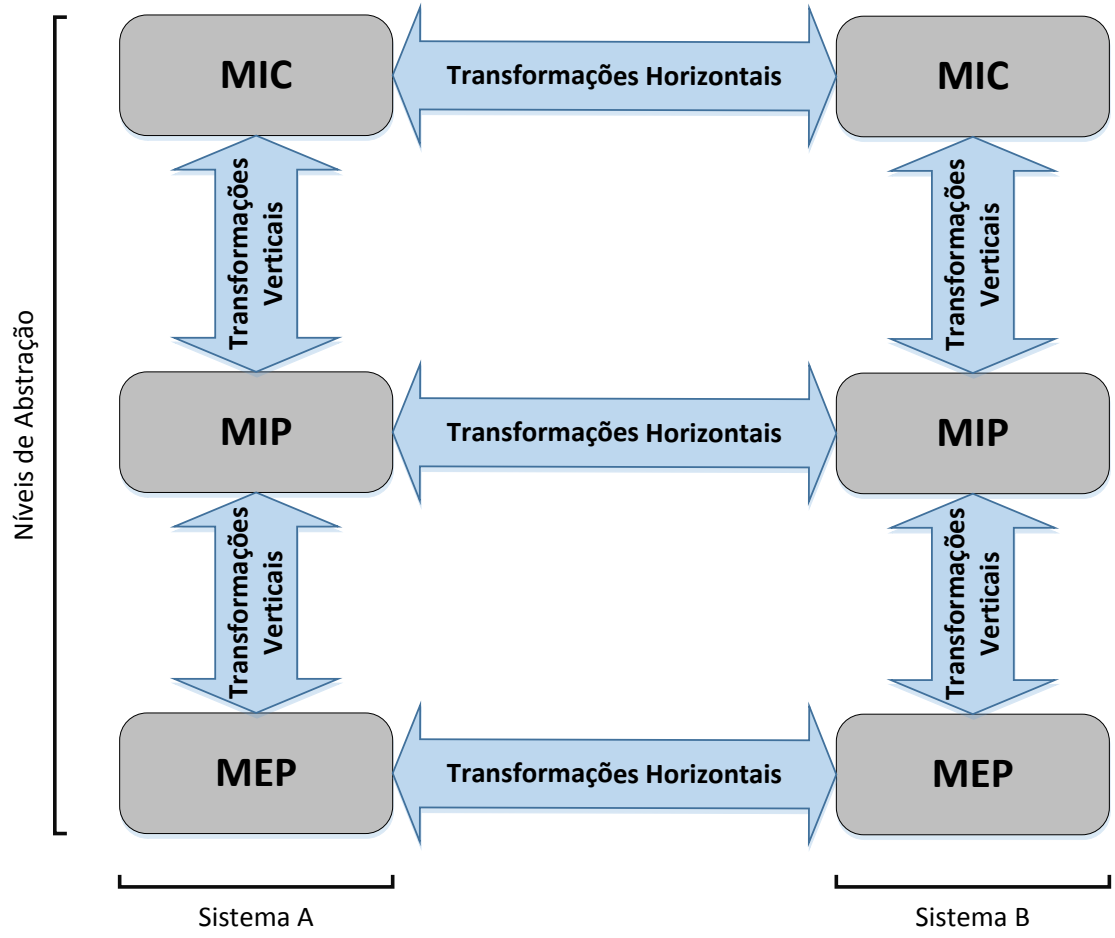

Figura 2.12 – Transformações numa Arquitetura MDA.

Quer o modelo original quer o final têm que ser instâncias de um metamodelo previamente definido. As transformações são realizadas mapeando o modelo original e gerando um novo modelo utilizando informações como especificações do modelo, regras de transformação, etc. Estas transformações são conseguidas através de várias abordagens: marcação, transformação de meta-modelos, transformação de modelos, aplicação de padrões e fusão de modelos.

2.4.2.1. Transformações Horizontais

São transformações de modelos entre diferentes sistemas, mas ao mesmo nível de abstração. Nestes casos é necessário fazer um mapeamento explícito ou implícito do metamodelo. Tipicamente, este processo é feito estaticamente, mas devido à constante mudança de conhecimento, os serviços, modelos e ontologias utilizados nos diferentes sistemas não são estáticos.

Como supracitado, as especificações de cada nível são diferentes, para realizar uma transformação é necessário definir um conjunto de regras para dar significado aos componentes de cada nível de abstração. Estes componentes normalmente são objetos reais, comportamentos, relações ou sistemas que podem ser representados utilizando várias linguagens de modelação que podem definir regras para transformação de modelos.

2.4.2.2. Transformações Verticais

Transformações entre diferentes níveis de abstração, utilizando ferramentas MDA. A maioria destas permitem anotações de modelos em diferentes níveis de abstração. Também fornecem métodos para adaptar as regras de transformação à medida das necessidades do utilizador, assim como as suas notações para as linguagens de programação mais comuns (por exemplo Java, C++, etc.). A quantidade de código gerado depende do nível de detalhe do MEP e do gerador de código.

2.4.3. Linguagens de Transformação de modelos

Para ser possível realizar as transformações descritas é necessário que existam especificações que garantam coerência entre modelo final e inicial. Um modelo inicial pode ser utilizado para obter vários modelos transformados e vice-versa, um modelo final pode ser obtido a partir de um ou vários modelos fonte. Este processo chama-se mapeamento e existem três tipos diferentes: mapeamento de tipos de modelo, mapeamento de instâncias de modelo e um híbrido dos dois anteriores.

Considera-se um mapeamento de tipos quando se especifica o mapeamento entre um tipo de elemento de um modelo inicial para um tipo de elemento no modelo final. Um mapeamento de instancia de modelo é a diferenciação do resultado da transformação de elementos do modelo

inicial dependendo de alguns marcadores (por exemplo, informação adicional sobre a transformação). A maioria destas operações combina estas duas.

Para executar estas operações são utilizadas várias linguagens entre as quais, *Query/View/Transformation* (QVT) e ATLAS Transformation Language (ATL), sendo esta última a mais utilizada.

2.4.3.1. QVT

Especificação OMG baseada em MOF (secção 2.3.1) e *Object Constraint Language* (OCL) (Bajwa et al., 2010), esta linguagem define uma família de linguagens de transformação: Relações (QVT-R) e Mapeamento de Operações (QVT-O) (Guduric et al., 2009).

QVT-R consiste numa especificação declarativa das relações entre modelos MOF. Esta é capaz de fazer correspondências entre padrões complexos e também cria um historial do que acontece durante uma transformação. QVT-O é a linguagem que permite implementações imperativas e pode ser utilizada para implementar uma ou mais relações.

2.4.3.2. ATL

ATL (*ATLAS Transformation Language*) é uma linguagem de transformação de modelos, desenvolvida pelo grupo de estudo ATLAS, equivalente a QVT, com foco na transformação entre modelos. É um tipo de híbrido (declarativo e imperativo) de linguagem de conversão, baseada em OCL, e é bastante aplicada a transformações MDA.

Está dividida em três tipos: *ATL module*, *ATL query* e *ATL library* (Zhang et al., 2010). O primeiro define operações sobre modelos; o segundo permite criar modelos para os tipos de dados primitivos (inteiros, caracteres, etc.), com o objetivo de calcular o valor original no modelo do código fonte; o terceiro oferece a possibilidade de criar bibliotecas de diferentes tipos, fornecendo um método simples para decompor código ATL.

2.5. Conclusões

Da investigação realizada no âmbito desta dissertação resultou o estado de arte aqui apresentado. Primeiro foi abordado o *hardware* dos dispositivos IdC, passando pela sua arquitetura e as suas limitações. Depois foi abordado o seu *software* (sistemas operativos e linguagens de desenvolvimento), dos quais se concluiu que o TinyOS é o mais utilizado, consequentemente a sua linguagem de programação, o nesC, é também a mais utilizada.

De seguida foram abordados os métodos de simulação destes dispositivos e, perante as suas várias limitações, foi abordada a temática da modelação de sistemas. Foram descritos alguns formalismos mais utilizados de modelação (XML, XMI, UML, etc.) e foi feita uma pesquisa sobre abordagens de desenvolvimento de software orientadas a modelos, nomeadamente MDE e MDA.

Estas ultimas abordagens poderão provar-se ferramentas essenciais na descrição e abstração de sistemas e aplicações IdC, facilitando a implementação de aplicações nesta área tecnológica.

2.6. Pergunta de Investigação

Com base na investigação desenvolvida ao longo deste capítulo, foi possível avaliar a tecnologia mais utilizada para desenvolver e implementar aplicações IdC, ferramentas existentes para gerar automaticamente *software* e vantagens e desvantagens das diferentes metodologias estudadas. Como resultado desta pesquisa preliminar, é possível enunciar a seguinte Pergunta de Investigação:

Que técnicas ou métodos se podem usar/criar para ajudar no desenvolvimento de aplicações da Internet-das-Coisas (IdC), de modo a que estes possam ser utilizados por sistemas computacionais de geração de código de uma maneira rápida e fiável?

O próximo capítulo sugere uma hipótese de resposta a esta pergunta de investigação.

3

Solução Proposta

Neste Capítulo será discutida uma hipótese de solução para as pergunta de investigação resultantes do Capítulo anterior. Será também proposta uma metodologia baseada nessa hipótese e todo o seu desenvolvimento.

3.1. Hipótese

A partir da pesquisa efetuada no capítulo anterior, é possível concluir que as abordagens de modelação são uma ferramenta importante atualmente, podendo revelar-se bastante úteis para o desenvolvimento de aplicações IdC.

Existe uma vasta gama de plataformas de implementação de aplicações IdC, quer ao nível de *hardware* (diferentes modelos de dispositivos), quer ao nível de *software* (sistemas operativos dedicados a implementações IdC).

Por outro lado, uma abordagem baseada em modelação pode provar ser uma metodologia bem mais abrangente, devido ao seu elevado nível de abstração. Possibilitando implementações da mesma aplicação em plataformas distintas, aplicando algumas transformações de semântica ao modelo desenvolvido.

Logo é possível formular a seguinte hipótese:

Se for aplicada uma solução baseada em modelos, será possível a sistemas importar e exportar aplicações IdC, então é possível conceber uma maneira rápida e fiável de geração automática de aplicações IdC compatíveis com várias plataformas.

Para testar esta hipótese é necessário primeiro que tudo desenvolver um metamodelo que permita descrever aplicações IdC. Este metamodelo será desenvolvido a partir da linguagem nesC (*network embedded systems* C) que é utilizada para se desenvolver aplicações para o sistema operativo TinyOS. Este, como referido na secção 2.2.1, é o sistema operativo dominante da investigação académica na área da IdC.

Depois será necessário implementar o metamodelo desenvolvido tomando uma abordagem MDA (*Model Driven-Architecture*). Neste caso, será implementada uma transformação do metamodelo nesC para um metamodelo XML e para ficheiros de texto nesC prontos a compilar e instalar nos dispositivos.

3.2. Metamodelo nesC

Este metamodelo foi desenvolvido de acordo com o manual de referência da linguagem nesC 1.3 por David Gay (Gay et al., 2009), com apoio do metamodelo desenvolvido por Rontidis Pavlos (Pavlos, 2013) e por outro metamodelo apresentado em (Rodrigues, 2015).

3.2.1. Componentes

A primeira classe do metamodelo é a classe "Model" que possui dois atributos: nome do modelo ("name") e versão ("version"); possui também uma referência para a classe "Component", que representa um componente do nesC. A Figura 3.1 representa a modelação de cada componente e ligação (classe "Wiring").

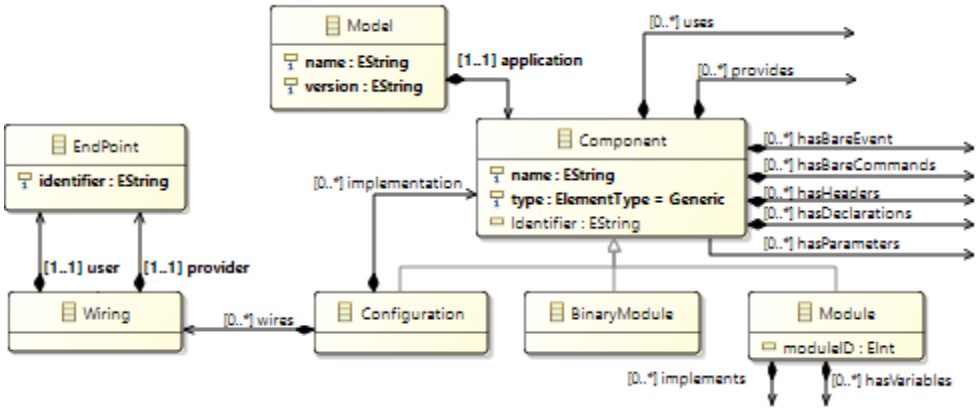

Figura 3.1 – Definição do tipo "Component" e "Wiring".

Um componente tem um nome ("name"), identificador ("identifier") e um tipo ("type"), que pode ser genérico ou normal, neste modelo o tipo de componente está implementado como uma enumeração com dois valores: "Normal" e "Generic". Um componente é genérico quando possui parâmetros de entrada (referência "hasParameters"), estes parâmetros podem ser um dos tipos de argumentos definidos no Capítulo 3.2.6. Um componente normal X é implicitamente instanciado e qualquer referência a esse componente em diferentes configurações aponta para o mesmo componente, por outro lado um componente genérico Y tem que ser instanciado e qualquer configuração que o referencie está a referenciar um componente diferente.

Cada componente é definido pela sua especificação e implementação. A especificação é composta por interfaces, definidas no Capítulo 3.2.2, que o componente usa ou fornece a outros componentes (referências "uses" e provides"), eventos ou comandos definidos no próprio componente, chamados "bare commands/events" (referências "hasBareCommands" e "hasBareEvents") pois não pertencem a nenhuma interface, e declarações de variáveis (referência "hasDeclarations"). Um componente pode também utilizar cabeçalhos (referência "hasHeaders"), como qualquer programa em linguagem C.

A implementação de um componente pode ser uma configuração (classe "Configuration"), módulo binário (classe "Binary Module") ou módulo (classe "Module"). Uma configuração consiste num conjunto de ligações entre componentes, estes podem ser declarados ou instanciados, se forem genéricos. Estas ligações são representadas pela classe "Wiring" que possui duas referências para a classe "Endpoint". Um "Endpoint" representa um elemento de um determinado componente, pois na prática as ligações entre componentes são feitas através de elementos da especificação dos mesmos, ou seja, interfaces, eventos ou comandos. Na Figura 3.2 estão representados alguns exemplos destas ligações entre componentes, neste caso a aplicação "PowerUp" que é instalada com o TinyOS. Como se pode observar, são declarados 3 componentes: "MainC",

"PowerupC" e "LedsC" e de seguida são declaradas as ligações entre eles. O utilizador e fornecedor da interface são dados pela direção das setas nas declarações de ligação entre os vários elementos.

```
Configuration PowerupAppC{}
implementation {
  components MainC, PowerupC, LedsC;
  MainC.Boot <- PowerupC;
  PowerupC -> LedsC.Leds;
}
```

Figura 3.2 – Exemplo de configuração nesC.

Um módulo é um componente implementado com código C que implementa os elementos presentes na especificação do componente. Um módulo pode implementar métodos independentes de interfaces (referência "implements") e pode declarar variáveis próprias (referência "hasVariables") em adição aos atributos herdados da classe "Component": nome, tipo, parâmetros e cabeçalhos. Na figura 3.3 está representado o código do módulo da aplicação "Powerup" que é instalada com o TinyOS por defeito. Primeiro são declaradas as interfaces utilizadas na especificação do módulo, neste caso "Boot" e "Leds", e na implementação é declarado o evento "booted" com o nome da interface à qual pertence. A modelação de interfaces é abordada no capítulo seguinte.

```
module PowerupC @safe()
{
  uses interface Boot;
  uses interface Leds;
}
implementation
{
  event void Boot.booted()  {
    call Leds.led0On();
  }
}
```

Figura 3.3 – Módulo da aplicação "PowerupC".

Um módulo binário é um componente que tem que ser declarado com o prefixo "component" e não possui implementação, esta é fornecida através de uma ligação a um ficheiro binário externo, por exemplo, o resultado de compilação de outro programa nesC.

3.2.2. Interfaces

Na linguagem nesC, uma interface, representada na Figura 3.4, especifica uma interação entre dois componentes, o utilizador e o fornecedor da interface. Estas interações são implementadas por dois tipos de métodos: comandos (referência "hasCommands"), funções invocadas pelo utilizador para o fornecedor, representando pedidos (por exemplo enviar uma mensagem por rádio), e eventos (referência "hasEvents"), funções invocadas do fornecedor para o utilizador, representando respostas (por exemplo, confirmação de mensagem de rádio enviada). A representação de comandos eventos no metamodelo é descrita no Capítulo 3.2.3.

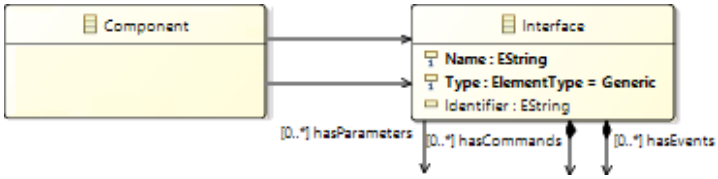

Figura 3.4 – Definição de interfaces.

Uma interface tem obrigatoriamente um nome ("Name") e um tipo ("type"), este é definido da mesma maneira que o tipo de componente pois uma interface pode ser normal ou genérica dependendo se possuir parâmetros de entrada ou não. Opcionalmente, uma interface pode ser definida também por um identificador ("identifier"). Este é utilizado como prefixo no cabeçalho das funções de uma interface implementadas no módulo. Na figura 3.5 estão exemplificadas declarações de interfaces.

```
uses interface Timer<TMilli> as Timer0;
uses interface Timer<TMilli> as Timer1;
uses interface Timer<TMilli> as Timer2;
uses interface Leds;
uses interface Boot;
```

Figura 3.5 – Declaração de interfaces.

A interface "Timer" é genérica pois possui um parâmetro de entrada, "TMilli", e é instanciada três vezes com identificadores diferentes: "Timer0", "Timer1" e "Timer2". As interfaces "Leds" e "Boot" não são genéricas pois não têm parâmetros e não possuem identificador.

3.2.3. Cabeçalhos

Os cabeçalhos em nesC (classe “Header” representada na Figura 3.6) funcionam exatamente como em C: podem definir variáveis (referência “hasVariables”), funções (referência “hasFunctions”) e podem referenciar outros cabeçalhos (referência “includes”).

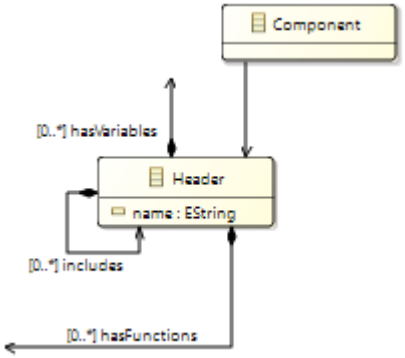

Figura 3.6 – Definição da classe “Header”.

3.2.4. Métodos

No nesC há 4 tipos de métodos: comandos, eventos, funções básicas de linguagem C e tarefas. Neste metamodelo foi decidido implementar uma classe abstrata principal que define um método genérico e 4 classes que representam os quatro tipos de método, como representado na Figura 3.7.

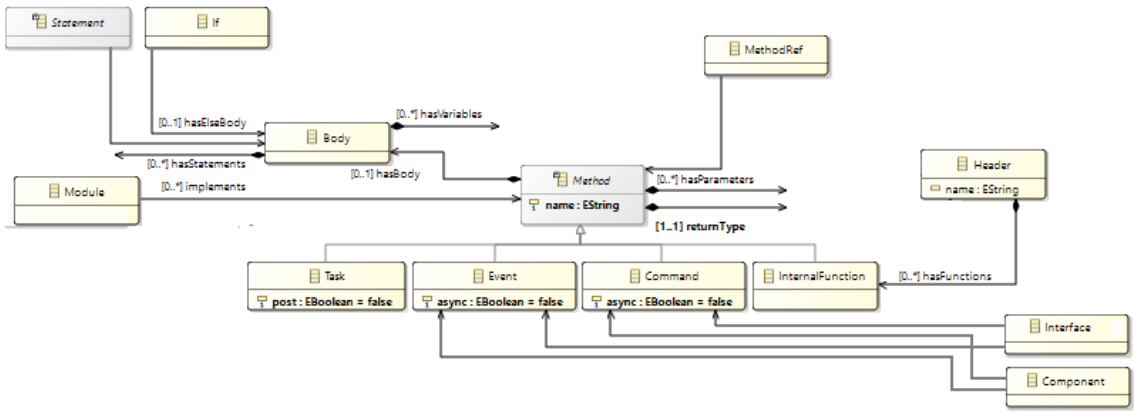

Figura 3.7 – Definição do tipo “Method” e classe “Body”.

Cada método possui um nome (“name”) e tipo (“returnType”) obrigatórios, pode também conter parâmetros de entrada (“hasParameters”) e um corpo (classe “Body”) que, por sua vez, contém declarações ou inicializações de variáveis (“hasVariables”) e operações (“hasStatements”). Visto que duas classes (“Module” e “MethodRef”) podem utilizar qualquer tipo de métodos, a definição da classe “Method” permite reduzir de oito ligações no total para duas, reduzindo a complexidade do metamodelo. Embora não se possa instanciar classes abstratas, qualquer subtipo herda todos os atributos e referências desta classe.

Uma tarefa (classe “Task”), é um tipo método do nesC que corre sincronamente com o *hardware* e não pode ser interrompido até estar finalizado, o tipo de retorno deste método é sempre “void” e não possui parâmetros de entrada. A sua execução pode ser adiada com o prefixo “post” na sua invocação, este está representado como um atributo booleano porque ou existe na invocação do método ou não existe de todo. Na Figura 3.8 está exemplificada uma task chamada toggle() que serve para invocar um comando chamado “led0toggle()” que pertence à interface “Leds”.

```
task void toggle()
{
  call Leds.led0Toggle();
}
```

Figura 3.8 – Exemplo de tarefa.

Já foram abordados os tipos comando (classe “Command”) e evento (classe “Event”), exemplificado na Figura 3.3, estes tipos podem ser invocados com o prefixo “async” com o objetivo de poderem ser iniciados por um manipulador de interrupções, que está representado nas respetivas classes também como um atributo booleano. Por fim uma função C (classe “internalFunction”) possui apenas os atributos pertencentes à classe “Method”.

3.2.5. Operações

Como em linguagem C, no nesC existem as típicas operações (classe “Statement” representada na Figura 3.9) “if”, “else”, “for”, “while” e expressões. Existem uma exclusiva ao nesC denominada “atomic” que pode servir de prefixo a outras operações (referência “statement”), por exemplo “atomic if”, e serve para correr operações como se não existisse operações simultâneas. Isto é útil, por exemplo, para actualizar estruturas de dados. Todas as operações, excluindo as expressões, possuem corpo (referência “hasBody”). A operação “else” está implementada como

uma referência da classe “If” (hasElseBody), que pode ou não ser instanciada. Como no caso dos métodos, a classe “Statement” é abstrata com 5 subtipos para simplificar o modelo.

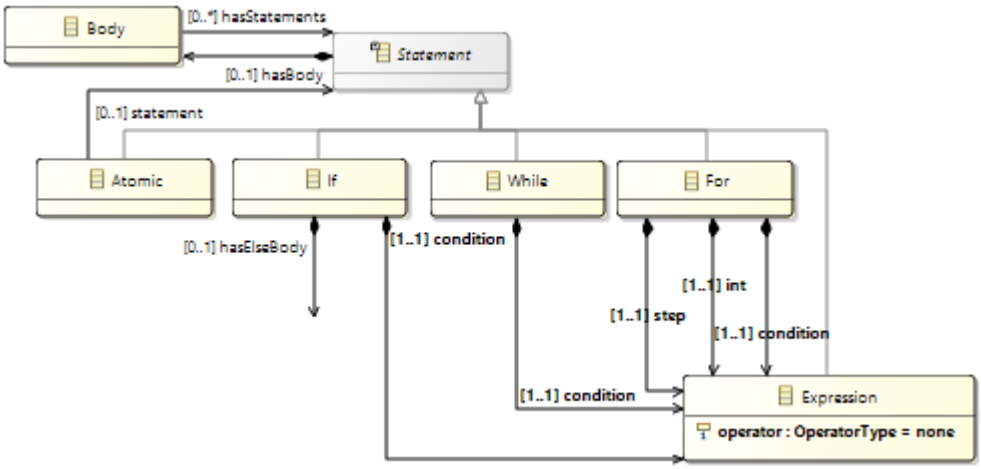

Figura 3.9 – Definição do tipo “Statement”.

Como em C, as classes “If” e “While” possuem uma referência à classe “Expression” pois necessitam de uma condição de execução, i.e. “if(a==False){}” executa se a variável “a” for igual a “False” e “while(a==False){}” é executado repetidamente enquanto “a” for igual a “False”. Em relação à classe “For”, existem três referências diferentes para a classe “Expression”: uma para inicializar a variável de controlo, por exemplo “i=0”, uma para a condição de fim de ciclo, por exemplo “i<10”, e uma para a variação da variável de controlo, por exemplo “i++” para incrementar a variável. Um *for* que utilize estas três expressões seria declarado da seguinte forma: “for(i=0,i<10,i++){}”. As expressões são descritas no capítulo seguinte.

3.2.6. Argumentos

Neste metamodelo são considerados quatro tipos de argumentos: expressões, valores literais, referências a métodos e variáveis. Como no caso dos métodos e das operações, existe uma classe abstrata do tipo argumento (classe “Argument”) e uma para cada subtipo (classes “Expression”, “Literal”, “MethodRef” e “Variable” respetivamente), como está representado na Figura 3.10.

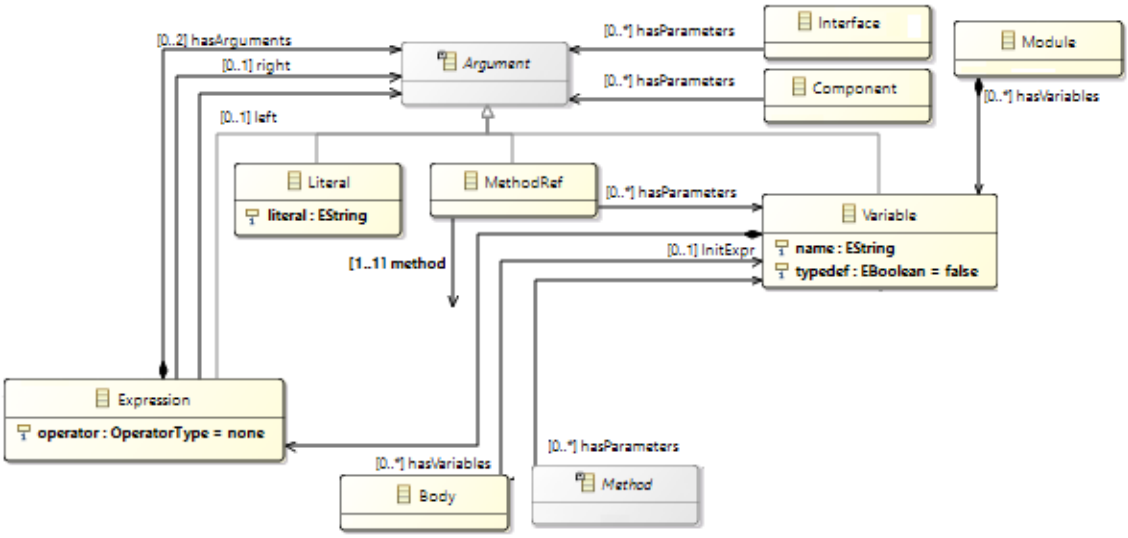

Figura 3.10 – Definição do tipo "Argument".

Uma expressão tem um operador, (atributo "operator") que neste metamodelo é definido apartir de uma enumeração ("OperatorType") com todos os tipos de operador possíveis, e dois possíveis argumentos (referência "hasArguments"), que podem ser cada um atribuído ao lado esquerdo ou direito da expressão (referências "left" e "right" respectivamente). Um valor literal é representado por uma string, por exemplo, um nome de uma variável como parâmetro de entrada de uma função. Uma referência a um método que possui uma ligação à classe "Method" (referência "Method") e referencia também os parâmetros de entrada desse método com uma ligação à classe "Variable" (referência "hasParameters"). Por exemplo, a linha "call Leds.led0Toggle();" é descrita segundo este modelo como uma "Expression" com o operador "Call" e com um argumento do lado direito da expressão que é um "MethodRef" que refere o método "led0Toggle" e não possui parâmetros de entrada. As variáveis (classe "Variable") serão descritas no próximo capítulo.

3.2.7. Variáveis

Para além das variáveis básicas existem 3 subtipos de variáveis em nesC, representados na Figura 3.11: estruturas (classe "Structs"), vetores (classe "Array") e enumerações (classe "Enums"). Todas estas classes têm em comum 2 atributos: nome ("name") e o prefixo "typedef", representado como um booleano, com "falso" como valor por defeito. Este prefixo serve para definir uma variável como se fosse um tipo novo de variável que pode depois ser reutilizado.

Adicionalmente qualquer variável tem que ter um tipo, que está representado pela referência "type" que liga a classe "Variable" à classe "VariableTypes", que será descrita mais à frente. É possível definir a inicialização de uma variável através da referência "InitExpr" que liga esta classe à classe "Expression". Neste caso o termo do lado direito da expressão é definido como o nome da variável sendo necessário definir o tipo de operador e o argumento do lado esquerdo da expressão.

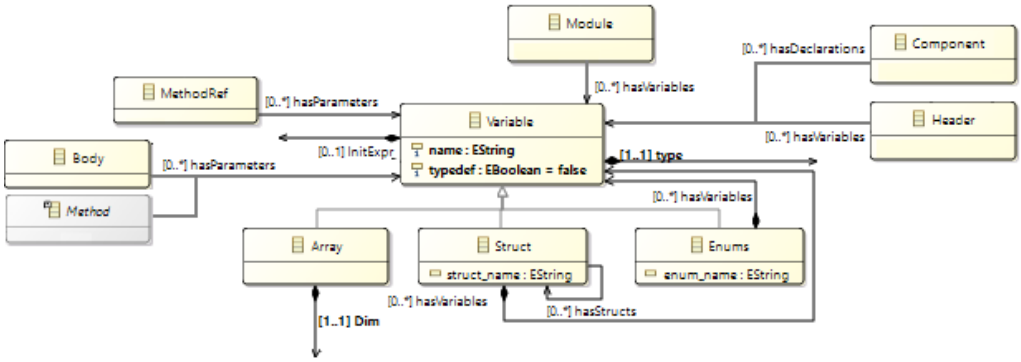

Figura 3.11 – Definição do tipo "Variable".

Relativamente a cada subtipo de variável, para além de herdarem os atributos e referencias da classe "Variable": um vector tem que possuir uma dimensão que está representada por uma referência "Dim" que aponta para a classe "Argument", para que a dimensão de um vector possa ser inicializada por qualquer tipo de argumento; uma estrutura possui um nome extra para além do nome de variável ("struct_name"), pode declarar variáveis dentro dela própria (referência "hasVariables") e pode referênciar outras estruturas (referências "hasStructs"); finalmente uma enumeração também pode possuir um nome extra para além do nome herdado da classe "Variable" e também pode declarar variáveis dentro de si mesma (referência "hasVariables"). Na Figura 3.12 estão exemplificadas uma enumeração e uma estrutura.

```
enum {
  AM_BLINKTORADIO = 6,
  TIMER_PERIOD_MILLI = 250
};

typedef nx_struct BlinkToRadioMsg {
  nx_uint16_t nodeid;
  nx_uint16_t counter;
} BlinkToRadioMsg;
```

Figura 3.12 – Exemplo de subtipos de varíavel.

A enumeração contém duas entradas que são definidas, segundo o metamodelo, como classes “Variable” com “initExpr” contendo um operador “Equals” e um argumento “Literal” do lado direto. A estrutura é definida como “typedef” com o tipo “nx_struct”, que é descrito no capítulo seguinte, e possui nome e nome de estrutura que neste caso são iguais (“BlinkToRadioMsg”). Contém duas variáveis declaradas como as variáveis da enumeração já descrita, com adição ao tipo de variável que desta vez está definido (“nx_uint16_t”).

3.2.8. Tipos de variáveis

Em nesC existem dois subtipos de variáveis: tipos internos, que são os tipos básicos de C como por exemplo inteiros, caracteres ou booleanos, e tipos externos, que são uma extensão da linguagem C que permite a definição de tipos de variáveis com uma representação independente de plataforma. No metamodelo foi decidido implementar uma classe abstrata “VariableTypes” com dois subtipos: “InternalTypes” para definição de tipos internos e “ExternalTypes” para tipos externos, como está representado na Figura 3.13.

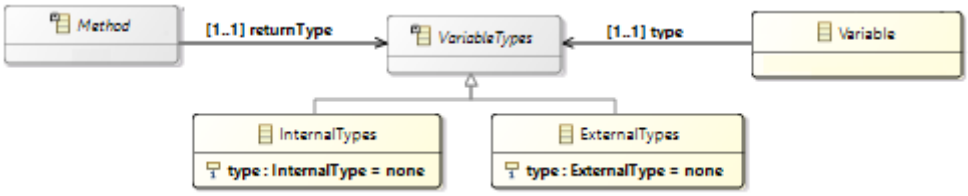

Figura 3.13 – Definição de tipos de variáveis.

Os tipos externos são divididos em três categorias: tipos básicos que complementam tipos internos básicos com tamanho e extremidade fixa, e são declarados como nx_intN, nx_uintN, nxle_intN e nxle_uintN em que “u” representa variáveis sem sinal (“unsigned”) e N representa o número de bits e pode tomar os seguintes valores: 8, 16, 32 e 64. Os prefixos nx e nxle são referentes a extremidade (“endianess”) das variáveis, ou seja, a maneira como são guardadas em memória: nx significa grande extremidade (big endian) que faz com que o byte mais significativo da variável é guardado no endereço de memoria mais pequeno; nxle significa pequena extremidade (“little endian”) em que o byte menos significativo da variável é guardado no menor endereço da memória.

Vetores externos são declarados como vetores internos C com prefixos respectivos, por exemplo, nx_int16_t [10]. Finalmente, também é possível declarar estruturas e uniões externas com os prefixos nx_struct e nx_union respectivamente.

Ambos são definidos a partir de respetivas enumerações com todos os tipos possíveis de variável, que estão representadas na Figura 3.14.

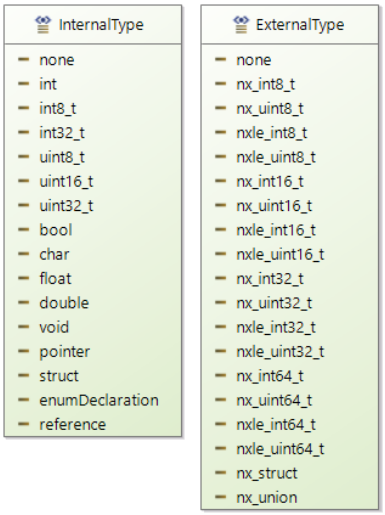

Figura 3.14 – Enumerações de tipos de variáveis.

3.2.9. Metamodelo Final

A Figura 3.15 representa o metamodelo final como um diagrama de classe, a sua representação utilizando linguagem Ecore pode ser consultada na integra no Anexo A.

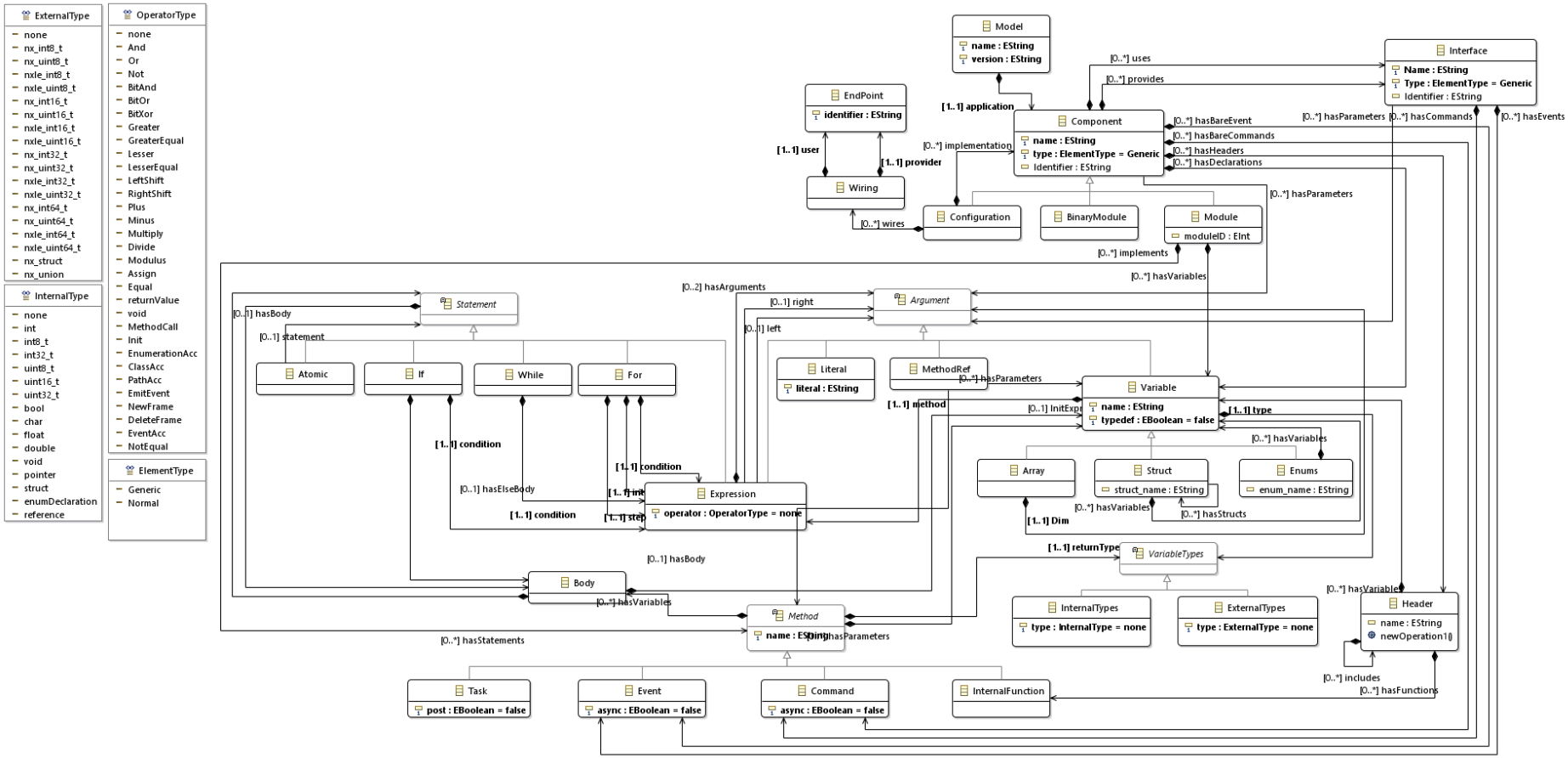

Figura 3.15 – Metamodelo nesC.

4

Implementação

Este capítulo descreve o mapeamento entre o metamodelo nesC (*network embedded system C*) desenvolvido para o metamodelo XML (*eXtensible Markup Language*), e as respetivas transformações entre estes metamodelos. O mapeamento e as transformações foram realizados segundo o paradigma MDA (*Model-Driven Architecture*). Foi também desenvolvido a geração de ficheiros de texto nesC (.nc) para instalação em dispositivos, a partir do metamodelo nesC.

4.1. Ferramentas de desenvolvimento

O desenvolvimento e validação do metamodelo nesC foram concebidos utilizando o *software Eclipse Modelling Tools* versão *Mars.2*, utilizando elementos do *Eclipse Modelling Framework*, nomeadamente, ferramentas de desenvolvimento Ecore. Adicionalmente também foi utilizado o plug-in ATL que contém todas as ferramentas disponíveis para o desenvolvimento de transformações de modelos com a linguagem ATL (Wagelaar, 2016).

4.2. Mapeamento para XML

Esta secção aborda o mapeamento XML. Este é conseguido desenvolvendo uma transformação modelo para modelo a partir do metamodelo nesC para o metamodelo XML descrito no Capítulo 2.3.2, utilizando a linguagem ATL (*Atlas Transformation Language*).

Utilizando a abordagem MDA (*Model-Driven Architecure*), a obtenção de um conjunto de dados descritos de acordo com uma certa especificação (neste caso nesC), noutro tipo de linguagem/especificação requer a elaboração do mapeamento entre as duas linguagens, ou seja, um conjunto de regras entre os metamodelos a utilizar. Estas definem correspondências entre elementos dos dois metamodelos, mantendo a coerência e validade da informação transmitida, e são implementadas através de um ficheiro ATL (*ATLAS Transformation Language*). Por fim é realizada uma extração dos dados para a representação em XML.

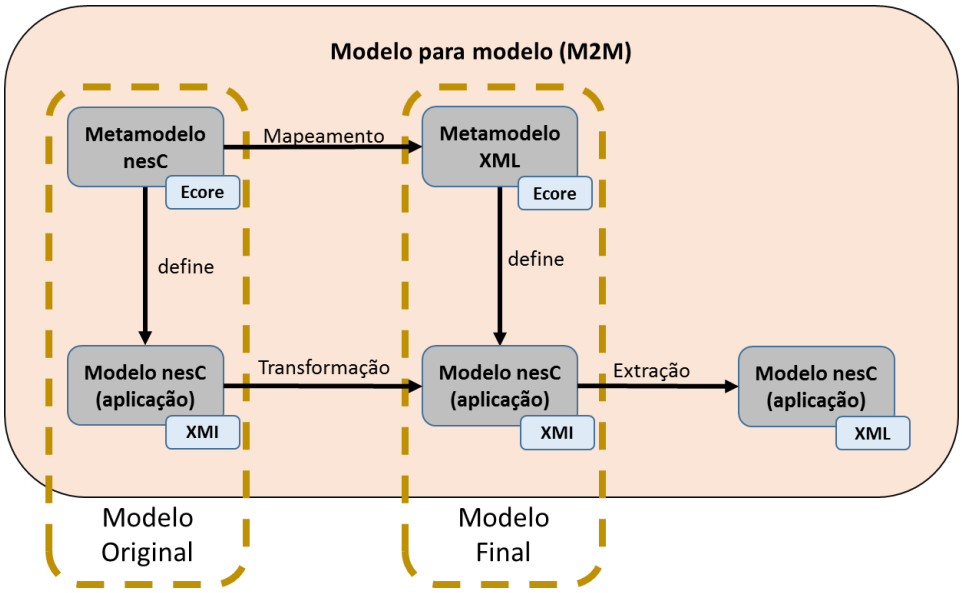

Figura 4.1 – Transformação modelo para modelo.

A Figura 4.1 descreve o processo de transformação. Para realizar o mapeamento, primeiro é necessária uma instanciação do modelo de origem (nesC), o que na prática se traduz em instanciar uma dada aplicação nesC segundo o seu metamodelo.

Um ficheiro ATL é constituído por diversas regras que convertem elementos do modelo de origem (nesC) em elementos equivalentes no modelo final (XML). Estas regras podem utilizar funções secundárias denominadas *helpers*. Ajudam, por exemplo, à verificação de valores de alguns atributos. O código da primeira regra do ficheiro encontra-se na Figura 4.2.

```
rule Model {
    from
        M: nesC!Model
    using {
        name: XML!Attribute = thisModule.createAttribute('name', M.name.toString());
        version: XML!Attribute = thisModule.createAttribute('version', M.version.
                toString());
        application: nesC!Configuration = if ( M.application.
                oclIsTypeOf(nesC!Configuration) ) then
                                                M.application
                                            else
                                                OclUndefined
                                            endif;
    }
    to
        root: XML!Root (
            name <- 'nesC:Model',
            children <- Sequence{name,
                        version,
                        application}
        )
}
```

Figura 4.2 – Regra “Model”.

Esta regra utiliza a classe “Model” do metamodelo nesC, que representa a secção principal no ficheiro XMI, selecionando os seus atributos (“name”, “version” e “application”) e associa-os à classe “XML!Root” do ficheiro de saída da transformação. A classe “Model” e a classe “Root” são as classes principais dos respetivos metamodelos. Ou seja, são a raíz, o ínicio de toda a informação contida num modelo. A partir destas classes é possível traçar um caminho até qualquer campo/dado pertencente ao modelo em causa.

Em todas as regras definidas para este mapeamento, qualquer atributo do metamodelo nesC, exceto referências a classes, é convertido para um atributo XML, “XML!Attribute”, através da regra “createAttribute”, que está presente na Figura 4.3. Esta recebe o nome do atributo em questão e o seu valor, define-os como nome e valor de um atributo XML, respetivamente, e devolve o atributo XML como objeto.

```
rule createAttribute(name: String, value: String){
    to
        attr: XML!Attribute (
            name <- name,
            value <- value
        )

    do {
        attr;
    }
}
```

Figura 4.3 – Regra "createAttribute".

O atributo “application” é uma referência a uma outra classe do metamodelo nesC, neste caso “Configuration”. Referências são representadas como elementos XML, “XML!Element”. Esta regra confirma se o atributo está a apontar para a classe “Configuration”, alocando espaço

para a sua definição. Se o este atributo não referenciar uma "Configuration", a regra define o atributo como "OclUndefined", significando que não está definido na instanciação. Esta regra estabelece o mapeamento descrito na Tabela 4.1.

Tabela 4.1 – Mapeamento da regra "Model".

Modelo Inicial - nesC	Modelo Final - XML
Model: M	Raiz **Nome**: "nesC:Model" **Atributos**: M.name; M.version; M.application
Model.name(string) Model.version(string) Model.application(Configuration)	Atributo Atributo Elemento

A regra "Configuration" está representada na Figura 4.4. Esta regra, como todas as outras, segue o mesmo padrão da regra "Model". Em primeiro lugar é definida a variável para a classe do metamodelo de entrada que se quer mapear, depois são definidos os atributos a criar e finalmente são associados à classe correspondente no metamodelo de destino.

```
rule Configuration {
    from
        C: nesC!Configuration
    using {
        name: XML!Attribute = thisModule.createAttribute('name', C.name.toString());
        type: XML!Attribute = thisModule.getType(C);
    }
    to
        e: XML!Element (
            name <- 'nesC:Configuration',
            children <- Sequence{name, type, C.implementation, C.wires, C.hasHeaders}
        )
}
```

Figura 4.4 – Regra "Configuration".

Nesta classe o atributo "type" é retornado por um helper "getType", representado na Figura 4.5. Este consiste numa operação "se" que verifica se o atributo é do tipo "Normal" ou "Generic". Se for o primeiro é dado como indefinido de modo a ficar ocultado no ficheiro de saída, caso contrário é escrito no ficheiro. Esta operação é feita de modo a facilitar a leitura do ficheiro XML final pelo utilizador.

```
helper def: getType(C: nesC!Component): String =
    if (C.type.toString() = 'Normal') then
        OclUndefined
    else
        thisModule.createAttribute('type', C.type.toString())
    endif;
```

Figura 4.5 – Helper "getType".

Considerando que o valor por defeito deste atributo é "Normal" e que por programa nesC existem vários componentes diferentes, todos com um tipo definido, não faz sentido descriminar no ficheiro final todos os tipos de componentes admitindo que à partida todos são do tipo "Normal" e que é possível fazer uma verificação semelhante deste atributo aquando do tratamento do ficheiro XML final. É informação que se torna redundante, servindo de "poluição visual" na leitura do ficheiro e aumentando a quantidade de dados a transmitir ou guardar.

Uma função semelhante a esta é utilizada para a classe "Interface", por também possuir um atributo "type". No entanto, o parâmetro de entrada é do tipo "nesC!Interface" em vez de "nesC!Component". A Tabela 4.2 representa o mapeamento para XML da Classe "Configuration". As outras classes do tipo "Component" ("BinaryModule" e "Module") seguem o mesmo raciocínio que a classe "Configuration".

Tabela 4.2 – Mapeamento da classe "Configuration".

Modelo Inicial - nesC	Modelo Final - XML
Configuration: Conf	Elemento **Nome**: "nesc:Configuration" **Atributos**: Conf.Name; Conf.Type; Conf.Wires; Conf.Implementation.
Configuration.name(string) Configuration.type(ElementType) Configuration.wires(Wiring) Configuration.implementation(Component)	Atributo Atributo Elemento Elemento

A regra da classe "Wiring" que representa as ligações entre os vários componentes efetuadas na configuração de um programa nesC é apresentada na Figura 4.6.

```
rule Wiring {
    from
        W: nesC!Wiring
    using {
        user: XML!Attribute = thisModule.createAttribute('user', W.user.identifier.toString());
        provider: XML!Attribute = thisModule.createAttribute('provider', W.provider.identifier.toString());
    }
    to
        e: XML!Element (
            name <- 'nesC:Wiring',
            children <- Sequence{user, provider}
        )
}
```

Figura 4.6 – Regra "Wiring".

Em vez de se mapear a classe "Wiring" e a classe "Endpoint" do metamodelo, optou-se por fazê-lo só para a primeira, pois os atributos que possui são duas referências para a classe "Endpoit": "user" e "provider". Estes atributos na prática são duas *strings* que podem ser facilmente atribuídas diretamente à classe "Wiring" simplificando o código. A Tabela 4.3 representa o mapeamento desta classe.

Tabela 4.3 – Mapeamento da classe "Wiring".

Modelo Inicial - nesC	Modelo Final - XML
Wiring: W	Elemento **Nome**: "nesC:Wiring" **Atributos**: W.user; W.provider
Wiring.user(EndPoint) Wiring.provider(EndPoint)	Atributo Atributo

Qualquer classe abstrata ("Statement", "Argument", "Method" e "VariableType") presente no metamodelo nesC não pode ser instanciada, logo nunca podem ser diretamente mapeadas. Por outro lado, os seus atributos, se existirem, são mapeados quando utilizados por qualquer subclasse. A Tabela 4.4 representa o "mapeamento" da classe "Method", como se pode verificar apenas os atributos e ligações a outras classes são definidos, não há correspondência direta entre a classe e um elemento XML.

Tabela 4.4 – Mapeamento da classe "Method".

Method: Me	
Me.Name(string)	Atributo
Me.hasBody(Body)	Elemento
Me.hasParameters(Variable)	Elemento
Me.returnType(VariableType)	Elemento

As subclasses da classe “Method” herdam toda a informação presente nesta tabela com adição de atributos presentes na sua própria definição. No caso da classe “Command”, cuja regra está representada na Figura 4.7, possui o atributo “async” que é um booleano. É verificado o seu valor, se for “falso” não é mostrado pela mesma razão da ocultação do atributo “type” na classe “Component” e nas respetivas subclasses.

```
rule Command {
    from
        C: nesC!Command
    using {
        name: XML!Attribute = thisModule.createAttribute('name', C.name.toString());
        async: XML!Attribute =  if (C.async.toString() = 'false') then
                                    OclUndefined
                                else
                                    thisModule.createAttribute('async', C.async.toString())
                                endif;
        }
    to
        e: XML!Element (
            name <- 'nesC:Command',
            children <- Sequence{name, C.returnType, async, C.hasParameters, C.hasBody}
        )
}
```

Figura 4.7 – Regra “Command”.

Uma regra idêntica é aplicada à classe “Event” visto que, em termos de semântica, a única diferença será o nome do tipo de método, ou seja, *command* ou *event*, que acompanha a sua declaração. No caso da subclasse “Task” existe o atributo “post”, um booleano com o valor “false” por defeito, em adição a todos os atributos da classe “Method”. Este atributo sofre a mesma verificação que o atributo “async” pela mesma razão de ocultar informação redundante no ficheiro XML. Para a classe “InternalFunction” não existe qualquer atributo para além dos já existentes na classe “Method”.

O mapeamento da classe “Command” está representado na Tabela 4.5, como anteriormente, o mesmo mapeamento é aplicado à classe “Event” e com as diferenças já apontadas aos restantes tipos de métodos.

Tabela 4.5 – Mapeamento da classe “Command”.

Modelo Inicial - nesC	Modelo Final - XML
Command: Comm	Elemento **Nome**: “nesC:Command” **Atributos**: Comm.Async; (mais todos os atributos e elementos herdados da classe “Method”)
Comm.async(Boolean)	Atributo

Todas as subclasses da classe “Statement” (“Atomic”, “If”, “While” e “Expression”) possuem apenas referências, logo os seus mapeamentos são bastante semelhantes aos já aqui representados. O mesmo se aplica às subclasses da classe “Argument”. A classe “Variable” tem a particularidade de referenciar a classe “VariableTypes”. Mais uma vez, essa classe é abstrata e não pode ser instanciada. As suas subclasses: “InternalTypes” e “ExternalTypes” são sim mapeados e as suas regras estão representadas na Figura 4.8.

```
rule VariableTypes_Internal {
    from
        IT : nesC!InternalTypes
    to
        attr: XML!Attribute(
            name <- 'nesC:InternalTypes',
            value <- IT.type.toString()
        )
}

rule VariableTypes_External {
    from
        ET : nesC!ExternalTypes
    to
        attr: XML!Attribute(
            name <- 'nesC:ExternalTypes',
            value <- ET.type.toString()
        )
}
```

Figura 4.8 – Regras “VaribableTypes”.

Estas regras são executadas de cada vez que é detetado algum dos tipos de variável (“nesC!InternalTypes” ou “nesC!ExternalTypes”). O tipo de variável é mapeado como um atributo XML da classe “Variable”, e não uma nova classe, embora no metamodelo original sejam classes diferentes. Esta operação simplifica a leitura do ficheiro XML final. O mapeamento da classe Variable está representado na Tabela 4.6. Outra particularidade desta classe é o facto que o atributo “typedef” também pode ser obtido por uma função semelhante à “getType” representado na Figura 4.5. O atributo é ocultado se possuir o valor por defeito, neste caso ser “false”, de modo a simplificar a leitura do ficheiro final.

Tabela 4.6 – Mapeamento da classe "Variable".

Modelo Inicial - nesC	Modelo Final - XML
Variable: V	Elemento **Name**: “nesC:Variable” **Children**: V.Name; V.Typedef; V.initExpr; V.Type;
V.Name(string) V.Typedef(boolean) V.initExpr(Expression) V.Type(VariableType)	Atributo Atributo Atributo Elemento

Todos os mapeamentos de classes que não foram aqui descritos podem ser consultados no Anexo B, que inclui a tabela integral de mapeamento do metamodelo nesC para XML. Uma vez aplicado o mapeamento aqui definido, que estabelece as regras para transformar modelos nesC em XML, é necessário extrair os dados para a representação habitual de ficheiros XML. Esta operação é feita através da utilização de uma biblioteca disponível (ficheiro JAR).

4.3. Geração de ficheiros de texto nesC

A geração de ficheiros de texto a partir do metamodelo nesC, representada na Figura 4.9, é mais simples que a transformação de modelos. Como no mapeamento XML, é necessário instanciar uma aplicação nesC segundo o metamodelo original. Depois é necessário elaborar um ficheiro ATL para transformar essa instanciação em ficheiros de texto correspondestes.

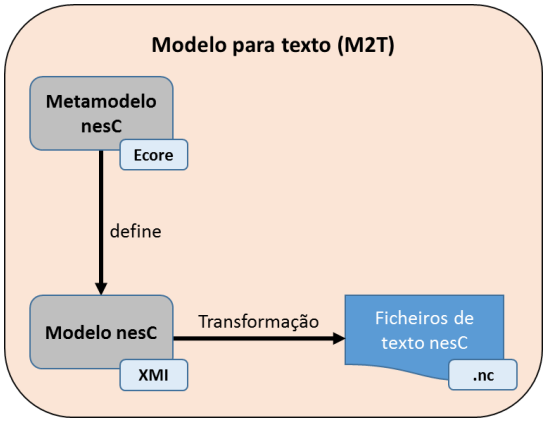

Figura 4.9 – Transformação modelo para texto.

No caso do nesC, existirão, no mínimo, dois ficheiros ".nc": configuração e módulo. Poderão existir bibliotecas associadas, ficheiros ".h", a estes ficheiros que também serão gerados pela transformação ATL. A regra que cria os ficheiros está representada na Figura 4.10. Os ficheiros são criados com os nomes de configuração (1), módulo (2) e cabeçalhos (3) definidos na instanciação, numa diretoria previamente definida ("FILES_PATH").

```
helper context nesC!Model def: CreateFiles() : String =
    let configuration : nesC!Configuration = self.application in
    if (configuration <> OclUndefined)
    then
        let result : String =
            configuration.toString(Sequence{''}).writeTo(thisModule.FILES_PATH + configuration.name + '.nc') in
        let _modules : Sequence(nesC!Module) = configuration.implementation->select(i | i.oclIsTypeOf(nesC!Module)) in
        if (_modules <> OclUndefined)
        then
            let res : String =
                _modules->collect(i | i.toString(Sequence{''}).writeTo(thisModule.FILES_PATH + i.name + '.nc')) in
            let libs : Sequence(nesC!Header) = _modules->iterate(s; acc: Sequence(nesC!Header) = Sequence{} |
                                                    acc->union(s.hasHeaders)
                                                ) in
            if (libs <> OclUndefined)
            then
                libs->collect(i | i.toString(Sequence{''}).writeTo(thisModule.FILES_PATH + i.name + '.h'))
            else
                OclUndefined
            endif
        else
            OclUndefined
        endif
    else
        OclUndefined
    endif;
```

Figura 4.10 – Função "CreateFiles".

Por não existir modelo-alvo, esta transformação consiste num conjunto de *helpers* que selecionam atributos presentes na instanciação do metamodelo e escrevem os seus valores no respetivo ficheiro de saída. Na Figura 4.11 está representado o código da função "toString" que converte a instanciação de uma configuração ("nesC!Configuration") em texto.

```
helper context nesC!Configuration def : toString(str : Sequence(String)) : String =
    let components : Sequence(nesC!Component) = self.implementation in
    let wires : Sequence(nesC!Wiring) = self.wires in
    '\n\n' +
    'configuration ' + self.name + '{\n' +
    '}\n' +
    'implementation{\n' +
    if (components <> OclUndefined)
    then
        components->iterate(s; acc: String = '' |
            acc + s.declareComponents(str)
        )
    else
        ''
    endif
    + '\n' +
    if (wires <> OclUndefined)
    then
        wires->iterate(s; acc: String = '' |
            acc + thisModule.getTABS(str) + s.user.identifier + ' -> ' + s.provider.identifier + ';\n'
        )
    else
        ''
    endif
    + '}\n\n';
```

Figura 4.11 – Função "toString" para configurações.

Primeiro esta função cria a secção “configuration” com o nome da configuração definido na instanciação. Depois cria a secção “implementation” do ficheiro onde declara os componentes utilizados escrevendo o nome dos componentes implementados pela configuração. São então ligados a esta através da referência “implementation”, e depois escreve todas as ligações entre os componentes recorrendo às referências “wires”.

No caso do módulo e dos cabeçalhos, existem funções muito semelhantes a esta. Para módulos, em vez de escrever “configuration” seguido do nome é escrito “module” com o nome do módulo em questão. No caso dos cabeçalhos, são escritas as palavras reservadas obrigatórias (#ifdef, #define, etc), seguidas da definição das variáveis e funções contidas no cabeçalho através das referências “includes” e “hasVariables”.

A escrita de métodos funciona de uma forma semelhante, para cada evento é escrito o tipo de retorno do método, o seu nome, os parâmetros de entrada, se existirem, e é feita a ligação ao corpo do método onde são declaradas variáveis e operações utilizadas pelo método.

No caso de eventos ou comandos é necessário escrever o identificador da interface que os contém, ou nome no caso de o primeiro não existir, como prefixo no cabeçalho da função. Para isto existe uma função, Figura 4.12, que verifica se o método é um evento ou comando, utilizando um iterador para escolher o elemento certo da sequência de métodos que contém. É identificado de seguida se a interface correspondente possui identificador, retornando-o. Se não existir identificador retorna o nome da mesma.

```
helper context nesC!Interface def : getInterface_Identifier(method : nesC!Method) : String =
    let local_method : nesC!Method =
        if (method.oclIsTypeOf(nesC!Event))
        then
            self.hasEvents->select(i | i.name = method.name)->first()
        else
            self.hasCommands->select(i | i.name = method.name)->first()
        endif in
    if (local_method->oclIsUndefined())
    then
        ''
    else
        if (self.Identifier.oclIsUndefined())
        then
            self.Name
        else
            self.Identifier
        endif
    endif;
```

Figura 4.12 – Função “getInterface_Identifier”.

Para retornar os tipos utilizados das enumerações, definidas no metamodelo, existe uma função especifica (por exemplo “getOper()”) para cada uma das enumerações (por exemplo OperatorType). Cada função possui estaticamente todos os tipos contidos na respetiva enumeração, selecionando o atributo em questão e retornando uma *string* com o valor correspondente.

No caso dos tipos de variáveis, visto que as referências que existem são para a classe abstrata “VariableTypes” que depois se divide em tipos internos e externos, foi necessário implementar uma função para selecionar o subtipo certo de variável como está representado na Figura 4.13. Esta verifica se o tipo de variável é interno, se for executa a função “getInternal” que retorna tipos internos, caso contrário executa a função “getExternal” que retorna tipos externos.

```
helper context nesC!VariableTypes def : getVarType() : String =
    if (self.oclIsTypeOf(nesC!InternalTypes))
    then
        self.getInternal()
    else
        self.getExternal()
    endif;
```

Figura 4.13 – Função “getVarType”.

5

Validação

Neste capítulo será descrita a validação das transformações implementadas no capítulo anterior. Esta validação é feita instanciando quatro aplicações seguindo o metamodelo nesC (*network embedded systems C*) desenvolvido no capítulo três e aplicando as transformações do capítulo quatro.

5.1. Compilação do Código Gerado

As transformações serão testadas através de quatro aplicações: "EmptyInstallation", "ToggleLeds", "Timers" e "SendMsgXbytes" que serão descritas posteriormente. Os ficheiros de texto gerados foram compilados utilizando ficheiros "Makefile" e o comando "make telosb" num terminal de uma instalação do sistema operativo Ubuntu 12.04 com a versão 2.1.2 do TinyOS instalada. "Telosb" é uma arquitetura de hardware para Dispositivos de Recursos Limitados (DRL) que suporta TinyOS. A escolha é arbitrária, na realidade existem 14 tipos de arquiteturas de hardware que suportam este sistema operativo (TinyOS Wiki, 2012), e por extensão a linguagem nesC. Qualquer aplicação pode ser compilada para qualquer um dos tipos de hardware mudando o nome da plataforma no comando de compilação.

O ficheiro "Makefile" não é gerado pela transformação nesC para texto pois possui um conjunto próprio de propriedades e parâmetros que podem ser definidos consoante as necessidades do programa. Por exemplo, um módulo de rádio de um dispositivo AdvanticSys MTM-CM3000 (Advanticsys, 2012) pode comunicar entre 2.4 GHz e 2.485GHz. Esta gama de frequências é divida em 16 canais de transmissão. O canal a ser utilizado pode ser definido declarando o

sinalizador “CFLAGS += -DCC2420_DEF_CHANNEL=X” no “Makefile”, onde X pode ser um número entre onze e vinte e seis. Para gerar este ficheiro seria necessário modelar todas as propriedades que podem ser incluídas na compilação e implementar a sua transformação ATL (*Atlas Transformation Language*).

5.2. Aplicação “Vazia”

Esta aplicação é um programa que serve somente para ligar o sistema operativo do dispositivo, não faz qualquer operação. O texto da descrição XMI (XML *Metadata Interchange*) no módulo desta aplicação está representado na Figura 5.1. A descrição pode ser consultada na integra no anexo C.

```
<implementation xsi:type="nesC:Module" name="EmptyInstallationC" type="Normal" Iden-
tifier="App" implements="//@application/@implementation.0/@uses.0/@hasEvents.0">
  <uses Name="Boot" Type="Normal">
    <hasEvents name="booted">
      <returnType xsi:type="nesC:InternalTypes" type="void"/>
    </hasEvents>
  </uses>
</implementation>
```

Figura 5.1 – Instanciação da aplicação “EmptyInstallation”.

O módulo é do tipo “nesC:Module” e é descrito pelo nome (“name”), tipo (“type”) e identificador (“Identifier”). Utiliza uma interface (secção “uses” na imagem), também descrita por nome, neste caso “Boot”, e tipo, que é “Normal” como o módulo. Esta interface contém um evento (secção “hasEvents”) chamado “booted”, que é do tipo “void”.

Na Figura 5.2 está representada a descrição XMI do módulo da Figura 5.1 depois de ser executada a transformação nesC para XML (eXtensible Markup Language) desenvolvida no capítulo anterior (transformação vertical na figura 4.1).

```
<children xsi:type="xmi:Element" name="nesC:Module">
  <children xsi:type="xmi:Attribute" name="name" value="EmptyInstallationC"/>
  <children xsi:type="xmi:Attribute" name="identifier" value="App"/>
  <children xsi:type="xmi:Element" name="nesC:Interface">
    <children xsi:type="xmi:Attribute" name="name" value="Boot"/>
    <children xsi:type="xmi:Element" name="nesC:Event">
      <children xsi:type="xmi:Attribute" name="name" value="booted"/>
      <children xsi:type="xmi:Attribute" name="nesC:InternalTypes" value="void"/>
    </children>
  </children>
</children>
```

Figura 5.2 – Transformação nesC para XML da aplicação “EmptyInstallation”.

Como previsto, o módulo é convertido de “nesC:Module” para um elemento XML, “XML:Element” com o nome “nesC:Module”. Este elemento passa a ter todos os atributos e referências do módulo nesC. Todos os elementos e atributos têm a sua própria secção porque são classes separadas do metamodelo XML utilizado, descrito no Capítulo 2.3.2. Como descrito no capítulo anterior, o tipo de elemento é escondido na transformação. Na Figura 5.2 não existe qualquer referência ao tipo de módulo ou interface pois são ambos do tipo “Normal”.

Na Figura 5.3 está representado o resultado do ultimo passo do mapeamento XML, a extração do ficheiro XML a partir do ficheiro XMI gerado pela transformação ATL. Cada secção têm o nome da classe correspondente no metamodelo nesC e os atributos de cada classe estão contidos dentro da respetiva secção.

```
<nesC:Module name="EmptyInstallationC" identifier="App">
  <nesC:Interface name="Boot">
    <nesC:Event name="booted" nesC:InternalTypes="void"/>
  </nesC:Interface>
</nesC:Module>
```

Figura 5.3 – Mapeamento XML da aplicação “EmptyInstallation”.

Em relação à geração de ficheiros de texto a partir da instanciação desta aplicação, o código nesC gerado pela transformação ATL para texto correspondente à descrição presente na Figura 5.1, e o código gerado está representado na Figura 5.4. O módulo tem uma secção onde as interfaces utilizadas são declaradas (“uses{}”) e de seguida existe a secção de implementação onde os métodos do módulo e interfaces utilizadas são implementados. Esta aplicação consiste em dois ficheiros, configuração e módulo, representado na Figura 5.4. Esta aplicação é constituída por dois ficheiros: o ficheiro de configuração “EmptyInstallationAppC” e o módulo “EmptyInstallationC”. Todo o código gerado/compilado pode ser consultado no Anexo C.

```
module EmptyInstallationC @safe() {
    uses {
        interface Boot;
    }
}
implementation{
    event void Boot.booted(){
    }
}
```

Figura 5.4 – Código nesC gerado a partir da aplicação “EmptyInstallation”.

Na Figura 5.5 está representado o resultado da compilação do código gerado a partir da instanciação da aplicação através da transformação descrita no Capítulo 4.3. Como é possível observar, o código compilou sem erros.

```
tese@tese-VirtualBox:~/Desktop/tese/EmptyInstallation$ make telosb
mkdir -p build/telosb
    compiling EmptyInstallationAppC to a telosb binary

    WARNING: Minimum recommended msp430-gcc version for this TinyOS release is 4
.6.3!!!

ncc -o build/telosb/main.exe  -Os -fnesc-separator=__ -Wall -Wshadow -Wnesc-all
-target=telosb -fnesc-cfile=build/telosb/app.c -board= -DDEFINED_TOS_AM_GROUP=0x
22 -DIDENT_APPNAME=\"EmptyInstallati\" -DIDENT_USERNAME=\"tese\" -DIDENT_HOSTNAM
E=\"tese-VirtualBox\" -DIDENT_USERHASH=0x09e493eaL -DIDENT_TIMESTAMP=0x57dd7666L
 -DIDENT_UIDHASH=0x138fde82L  EmptyInstallationAppC.nc -lm
    compiled EmptyInstallationAppC to build/telosb/main.exe
            1320 bytes in ROM
               6 bytes in RAM
msp430-objcopy --output-target=ihex build/telosb/main.exe build/telosb/main.ihex
    writing TOS image
```

Figura 5.5 – Compilação da aplicação “EmptyInstallation”.

5.3. Alternar Leds

Esta aplicação liga e desliga os leds do dispositivo onde é instalada e é constituída por três ficheiros. Os LEDs que são ligados podem ser escolhidos através das variáveis “LED0_Active”, “LED1_Active” e “LED2_Active”. Todos os LEDs são utilizados. É possível definir um período de espera entre ativação de LEDs com a variável “TimePeriod”, o seu valor por defeito é 1000 milissegundos. Na Figura 5.6 está presente um excerto da descrição XMI da instanciação desta aplicação.

```
<hasStatements xsi:type="nesC:If">
  <hasBody>
    <hasStatements xsi:type="nesC:Expression" right="//@application/@implementa-
tion.0/@uses.2/@hasEvents.0/@hasBody/@hasStatements.0/@hasBody/@hasStatements.0/@ha-
sArguments.0" operator="MethodCall">
      <hasArguments xsi:type="nesC:MethodRef" method="//@application/@implementa-
tion.0/@uses.1/@hasCommands.0"/>
    </hasStatements>
  </hasBody>
  <condition left="//@application/@implementa-
tion.0/@uses.2/@hasEvents.0/@hasBody/@hasStatements.0/@condition/@hasArguments.0">
    <hasArguments xsi:type="nesC:Literal" literal="LED0_Active"/>
  </condition>
</hasStatements>
```

Figura 5.6 – Instanciação da aplicação “ToggleLeds”.

É representada uma operação "se", ou seja, "nesC:If", que verifica a condição para ligar o LED0. Primeiro está descrito o corpo da operação, representado pela secção "hasBody". Este possui uma expressão, representada pela secção "hasStatements". Neste caso o operador da expressão é do tipo "MethodCall" que significa que é uma referência a um método declarado noutra parte do programa. Consequentemente, esta expressão só tem um argumento que é a referência ao método, representada pela secção "hasArguments".

Depois do corpo está descrita a condição da operação "se" na secção "condition", que contém dois argumentos, lado esquerdo ("left") e direito ("right") do operador ("operator") que neste caso é uma igualdade ("Equal"). Ambos os argumentos estão descritos nas secções "hasArguments" respetivas, contidas na secção "condition".

Na Figura 5.7 a instanciação segundo o metamodelo XML desta aplicação. Como anteriormente, todos os atributos e classe mapeados estão nas suas próprias secções, embora haja algumas mudanças na ordem das secções. Ao contrário da instanciação, depois da transformação a condição da operação "se" (primeira secção "nesC:Expression") aparece antes da definição do corpo da operação (secção "nesC:Body").

```
<children xsi:type="xmi:Element" name="nesC:If">
  <children xsi:type="xmi:Element" name="nesC:Expression">
    <children xsi:type="xmi:Attribute" name="operator" value="none"/>
    <children xsi:type="xmi:Element" name="nesC:Literal">
      <children xsi:type="xmi:Attribute" name="literal" value="LED1_Active"/>
    </children>
  </children>
  <children xsi:type="xmi:Element" name="nesC:Body">
    <children xsi:type="xmi:Element" name="nesC:Expression">
      <children xsi:type="xmi:Attribute" name="operator" value="MethodCall"/>
      <children xsi:type="xmi:Element" name="nesC:MethodRef">
        <children xsi:type="xmi:Element" name="nesC:Command">
          <children xsi:type="xmi:Attribute" name="name" value="led1Toggle"/>
          <children xsi:type="xmi:Attribute" name="nesC:InternalTypes" value="void"/>
        </children>
      </children>
    </children>
  </children>
</children>
```

Figura 5.7 – Instanciação XML da aplicação "ToggleLeds".

O resultado da extração do ficheiro XML a partir do ficheiro XMI está representado na Figura 5.8. Todo o código destes ficheiros está disponível para consulta no Anexo D.

```
<nesC:If>
  <nesC:Expression operator="none">
    <nesC:Literal literal="LED0_Active"/>
  </nesC:Expression>
  <nesC:Body>
    <nesC:Expression operator="MethodCall">
      <nesC:MethodRef>
        <nesC:Command name="led0Toggle" nesC:InternalTypes="void"/>
      </nesC:MethodRef>
    </nesC:Expression>
  </nesC:Body>
</nesC:If>
```

Figura 5.8 – Mapeamento XML da aplicação "ToggleLeds".

Na Figura 5.9 está representado o código, gerado através da transformação do metamodelo nesC para texto, correspondente ao texto presente na Figura 5.6. Esta aplicação é constituída por 3 ficheiros: "ToggleLedsAppC", ToggleLedsC" e "lib.h". Todos os ficheiros podem ser consultados no Anexo D.

```
if (LED0_Active){
        call Leds.led0Toggle();
}
```

Figura 5.9 – Código nesC gerado a partir da aplicação "ToggleLeds".

O resultado da compilação desta aplicação está representado na Figura 5.10.

```
tese@tese-VirtualBox:~/Desktop/tese/Toggle LEds$ make telosb
mkdir -p build/telosb
    compiling ToggleLedsAppC to a telosb binary

    WARNING: Minimum recommended msp430-gcc version for this TinyOS release is 4
.6.3!!!

ncc -o build/telosb/main.exe  -Os -fnesc-separator=__ -Wall -Wshadow -Wnesc-all
-target=telosb -fnesc-cfile=build/telosb/app.c -board= -DDEFINED_TOS_AM_GROUP=0x
22 -DIDENT_APPNAME=\"ToggleLedsAppC\" -DIDENT_USERNAME=\"tese\" -DIDENT_HOSTNAME
=\"tese-VirtualBox\" -DIDENT_USERHASH=0x09e493eaL -DIDENT_TIMESTAMP=0x57dd41faL
-DIDENT_UIDHASH=0xd495205bL  ToggleLedsAppC.nc -lm
    compiled ToggleLedsAppC to build/telosb/main.exe
            2328 bytes in ROM
              36 bytes in RAM
msp430-objcopy --output-target=ihex build/telosb/main.exe build/telosb/main.ihex
    writing TOS image
```

Figura 5.10 – Compilação da aplicação "ToggleLeds".

5.4. Temporizador

Esta aplicação define um temporizador com um certo período, definido pela variável "TimePeriod". Esta variável está definida no cabeçalho "lib.h", cujo texto XMI da instanciação desta aplicação está representado na Figura 5.11. O seu valor por defeito é mil milissegundos.

```
<hasHeaders name="lib">
  <hasVariables xsi:type="nesC:Enums" name="" enum_name="">
    <hasVariables name="TimePeriod">
      <InitExpr right="//@application/@implementation.0/@hasHeaders.0/@hasVar-
iables.0/@hasVariables.0/@InitExpr/@hasArguments.0" operator="Assign">
        <hasArguments xsi:type="nesC:Literal" literal="1000"/>
      </InitExpr>
    </hasVariables>
  </hasVariables>
  <includes name="Timer">
    <hasVariables name="TMilli">
      <type xsi:type="nesC:InternalTypes" type="void"/>
    </hasVariables>
  </includes>
</hasHeaders>
```

Figura 5.11 – Instanciação do cabeçalho da aplicação "Timer".

Este cabeçalho contém uma enumeração (primeira secção "hasVariables"). Esta contém por sua vez a variável "TimePeriod" que possui uma expressão de inicialização (secção "InitExpr"). O cabeçalho contém ainda outro cabeçalho que contém a variável "TMili". Esta variável é utilizada pela interface Timer usada pelo módulo desta aplicação. O resultado da transformação nesC para XML é apresentado na Figura 5.12.

```
<children xsi:type="xmi:Element" name="nesC:Header">
  <children xsi:type="xmi:Attribute" name="name" value="lib"/>
  <children xsi:type="xmi:Element" name="nesC:Enum">
    <children xsi:type="xmi:Attribute" name="name" value=""/>
    <children xsi:type="xmi:Attribute" name="enum_name" value=""/>
    <children xsi:type="xmi:Element" name="nesC:Variable">
      <children xsi:type="xmi:Attribute" name="name" value="TimePeriod"/>
      <children xsi:type="xmi:Element" name="nesC:Expression">
        <children xsi:type="xmi:Attribute" name="operator" value="Assign"/>
        <children xsi:type="xmi:Element" name="nesC:Literal">
          <children xsi:type="xmi:Attribute" name="literal" value="1000"/>
        </children>
      </children>
    </children>
  </children>
  <children xsi:type="xmi:Element" name="nesC:Header">
    <children xsi:type="xmi:Attribute" name="name" value="Timer"/>
    <children xsi:type="xmi:Element" name="nesC:Variable">
      <children xsi:type="xmi:Attribute" name="name" value="TMilli"/>
      <children xsi:type="xmi:Attribute" name="nesC:InternalTypes"
value="void"/>
    </children>
  </children>
</children>
```

Figura 5.12 – Instanciação XML do cabeçalho da aplicação "Timer" (parcial).

O resultado da extração XML do ficheiro transformado está representada na Figura 5.13. O código XML pode ser consultado na integra no anexo E.

```
<nesC:Header name="lib">
  <nesC:Enum name="" enum_name="">
    <nesC:Variable name="TimePeriod">
      <nesC:Expression operator="Assign">
        <nesC:Literal literal="1000"/>
      </nesC:Expression>
    </nesC:Variable>
  </nesC:Enum>
  <nesC:Header name="Timer">
    <nesC:Variable name="TMilli" nesC:InternalTypes="void"/>
  </nesC:Header>
</nesC:Header>
```

Figura 5.13 – Representação XML da aplicação "Timers" (parcial).

O código referente a este cabeçalho, que é gerado pela transformação ATL descrita no Capítulo 4.3 é apresentado na Figura 5.14.

```
#ifndef LIB_H
#define LIB_H
#include "Timer.h"
enum {
        TimePeriod = 1000
};
#endif /* LIB_H */
```

Figura 5.14 – Código do cabeçalho “lib.h” gerado pela transformação ATL.

A aplicação é constituída por 3 ficheiros de texto: uma configuração (“TimersAppC.nc”), um módulo (“TimersC”) e um cabeçalho (“lib.h”). Todos estes ficheiros são gerados pela transformação nesC para texto, e estão disponíveis na integra no anexo E, sendo necessário criar o ficheiro “Makefile” para compilar o programa. O resultado desta compilação está apresentado na Figura 5.15.

```
tese@tese-VirtualBox:~/Desktop/tese/SendTimers$ make telosb
mkdir -p build/telosb
    compiling TimersAppC to a telosb binary

    WARNING: Minimum recommended msp430-gcc version for this TinyOS release is 4
.6.3!!!

ncc -o build/telosb/main.exe  -Os -fnesc-separator=__ -Wall -Wshadow -Wnesc-all
-target=telosb -fnesc-cfile=build/telosb/app.c -board= -DDEFINED_TOS_AM_GROUP=0x
22 -DIDENT_APPNAME=\"TimersAppC\" -DIDENT_USERNAME=\"tese\" -DIDENT_HOSTNAME=\"t
ese-VirtualBox\" -DIDENT_USERHASH=0x09e493eaL -DIDENT_TIMESTAMP=0x57dd70b6L -DID
ENT_UIDHASH=0x0e4b009eL  TimersAppC.nc -lm
    compiled TimersAppC to build/telosb/main.exe
            2250 bytes in ROM
              36 bytes in RAM
msp430-objcopy --output-target=ihex build/telosb/main.exe build/telosb/main.ihex
    writing TOS image
```

Figura 5.15 – Compilação da aplicação “Timers”.

5.5. Enviar Mensagens

Esta aplicação transmite uma mensagem com um certo número de bytes. O total de bytes a ser enviado pode ser definido pela variável “NumberOfBytes” presente no cabeçalho utilizado por esta aplicação. O intervalo de envio pode ser fixado pela variável “TimePeriod”, também declarada na biblioteca “lib.h”. O seu valor por defeito é mil milissegundos.

Na Figura 5.16 está representado um excerto da descrição XMI gerada pela instanciação deste programa segundo o metamodelo nesC.

```
<wires>
  <provider identifier="MainC"/>
  <user identifier="App.Boot"/>
</wires>
<wires>
  <provider identifier="LedsC"/>
  <user identifier="App.Leds"/>
</wires>
<wires>
  <provider identifier="Wireless"/>
  <user identifier="App.AMControl"/>
<wires>
  <provider identifier="AMSend"/>
  <user identifier="App.AMSend"/>
</wires>
<wires>
  <provider identifier="Timer"/>
  <user identifier="App.Timer"/>
</wires>
```

Figura 5.16 – Instanciação das ligações entre componentes da aplicação "SendMsgXbytes".

Este excerto representa o conjunto de ligações feitas entre os vários componentes utilizados pela aplicação. Por exemplo, na primeira seção "wires", o fornecedor desta ligação é o componente "MainC" e o utilizador é o elemento "App.Boot", sendo "App" o identificador do módulo da aplicação e "Boot" uma interface utilizada por esse módulo.

Segundo o metamodelo nesC, "user" e "provider" são duas referências para a classe "EndPoint". Como descrito no Capítulo 4.2, estas são mapeadas para classes "Atributo" do metamodelo XML, como se pode confirmar na Figura 5.6, que representa a transformação da descrição representada na Figura 5.16. Cada secção "wires" é mapeada para um elemento XML contendo dois atributos: nome do atributo, "user" ou "provider", e valor do atributo.

```
<children xsi:type="xmi:Element" name="nesC:Wiring">
  <children xsi:type="xmi:Attribute" name="user" value="App.Boot"/>
  <children xsi:type="xmi:Attribute" name="provider" value="MainC"/>
</children>
<children xsi:type="xmi:Element" name="nesC:Wiring">
  <children xsi:type="xmi:Attribute" name="user" value="App.Leds"/>
  <children xsi:type="xmi:Attribute" name="provider" value="LedsC"/>
</children>
<children xsi:type="xmi:Element" name="nesC:Wiring">
  <children xsi:type="xmi:Attribute" name="user" value="App.AMControl"/>
  <children xsi:type="xmi:Attribute" name="provider" value="Wireless"/>
</children>
<children xsi:type="xmi:Element" name="nesC:Wiring">
  <children xsi:type="xmi:Attribute" name="user" value="App.AMSend"/>
  <children xsi:type="xmi:Attribute" name="provider" value="AMSend"/>
</children>
<children xsi:type="xmi:Element" name="nesC:Wiring">
  <children xsi:type="xmi:Attribute" name="user" value="App.Timer"/>
  <children xsi:type="xmi:Attribute" name="provider" value="Timer"/>
</children>
```

Figura 5.17 – Instanciação XML da aplicação "SendMsgXbytes" (parcial).

Na Figura 5.18 está representado o resultado final (parcial) do mapeamento XML da aplicação "SendMsgXbytes". Como previsto, cada ligação é representada por uma secção "nesC:Wiring" com os devidos atributos "user" e "provider".

```
<nesC:Wiring user="App.Boot" provider="MainC"/>
<nesC:Wiring user="App.Leds" provider="LedsC"/>
<nesC:Wiring user="App.AMControl" provider="Wireless"/>
<nesC:Wiring user="App.AMSend" provider="AMSend"/>
<nesC:Wiring user="App.Timer" provider="Timer"/>
```

Figura 5.18 – Representação XML da aplicação "SendMsgXbytes" (parcial).

O código referente à representação XML apresentada na figura anterior, que foi gerado a partir da instanciação desta aplicação está descriminado na Figura 5.19. Esta aplicação é constituída por três ficheiros: o ficheiro de configuração "SendMsgXbytesAppC", o módulo "SendMsgXbytesC" e a biblioteca "lib.h.". Todos estes ficheiros estão incluídos no Anexo F.

```
App.Boot -> MainC;
App.Leds -> LedsC;
App.AMControl -> Wireless;
App.AMSend -> AMSend;
App.Timer -> Timer;
```

Figura 5.19 – Código gerado a partir da transformação ATL (ligação entre componentes do nesC).

A seta "->" representa a ligação entre os componentes. À sua esquerda encontra-se o utilizador e à esquerda o fornecedor. Na Figura 5.20 está representado o resultado final do mapeamento XML da aplicação "SendMsgXbytes".

```
tese@tese-VirtualBox:~/Desktop/tese/SendMsgXbytes$ make telosb
mkdir -p build/telosb
    compiling SendMsgXbytesAppC to a telosb binary

    WARNING: Minimum recommended msp430-gcc version for this TinyOS release is 4.6.
3!!!

ncc -o build/telosb/main.exe  -Os -fnesc-separator=__ -Wall -Wshadow -Wnesc-all -ta
rget=telosb -fnesc-cfile=build/telosb/app.c -board= -DDEFINED_TOS_AM_GROUP=0x22 -DI
DENT_APPNAME=\"SendMsgXbytesAp\" -DIDENT_USERNAME=\"tese\" -DIDENT_HOSTNAME=\"tese-
VirtualBox\" -DIDENT_USERHASH=0x09e493eaL -DIDENT_TIMESTAMP=0x57dec396L -DIDENT_UID
HASH=0x7b8e3185L  SendMsgXbytesAppC.nc -lm
/opt/tinyos-2.1.2/tos/chips/cc2420/lpl/DummyLplC.nc:39:2: warning: #warning "*** LO
W POWER COMMUNICATIONS DISABLED ***"
    compiled SendMsgXbytesAppC to build/telosb/main.exe
          11082 bytes in ROM
            353 bytes in RAM
msp430-objcopy --output-target=ihex build/telosb/main.exe build/telosb/main.ihex
    writing TOS image
```

Figura 5.20 – Compilação da aplicação "SendMsgXbytes".

5.6. Sumário

Comparando os ficheiros XMI, resultantes da instanciação de cada uma das aplicações de acordo com o metamodelo nesC, com os obtidos segundo o metamodelo XML, concluiu-se que a informação se manteve coerente, pois a estrutura geral do ficheiro inicial corresponde à do ficheiro gerado pela transformação, isto é, cada elemento do ficheiro inicial possui uma correspondência direta no ficheiro final. Estes ficheiros podem ser consultados na integra nos Anexos C, D, E e F.

Em relação à geração automática de texto, analisando as figuras referentes à compilação das aplicações (Figuras 5.5, 5.10, 5.15 e 5.20) conclui-se que o código compilou sem erros, logo, a geração de código foi bem-sucedida. De salientar que, em nesC, as bibliotecas funcionam da mesma maneira que na linguagem C: cabeçalhos contidos na mesma diretoria do módulo que os utiliza são declarados com aspas (*i.e. #include "lib.h"*) e cabeçalhos do sistema são declarados com <> (i.e. *#include <lib.h>*), podendo também ser declarados como os primeiros. Dado que a linguagem ATL não permite verificar o caminho dos ficheiros criados dentro de funções, durante a geração dos ficheiros os cabeçalhos incluídos nos ficheiros gerados são todos declarados com aspas, esta abordagem permite que ambos os tipos de cabeçalhos sejam "encontrados" pelo compilador. Caso contrário, o compilador não deteta cabeçalhos presentes na mesma diretoria que o módulo que os utiliza.

Após analisados os resultados das transformações XML, assim como o código nesC gerado, é possível concluir que o metamodelo implementado provou ser viável para transformação noutros tipos de metamodelos e geração de texto.

De realçar que a linguagem utilizada para validação do metamodelo nesC foi XML. No entanto, é possível transformar instanciações do metamodelo desenvolvido noutras linguagens, nomeadamente, Simulink (para simulação de aplicações nesC em dispositivos com recursos limitados). Com esse objetivo em mente, é possível integrar este modelo na arquitetura para simulação de dispositivos desenvolvida na dissertação de mestrado intitulada "*A Dynamic Model-based Simulation Framework for Resource Constrained Devices* (Rodrigues, 2015). O metamodelo nesC desenvolvido seria inserido como modelo de *software* na secção de modelos da arquitetura, de forma a gerar blocos Simulink para simular a execução do código do programa a testar, a fim de verificar o seu comportamento num dispositivo antes de se passar à fase de implementação de dispositivos numa situação real.

6

Conclusões e Trabalho Futuro

Dado o estado atual do desenvolvimento de aplicações para a Internet-das-Coisas (IdC), esta dissertação tem como objetivo a criação de um formalismo genérico da linguagem de programação nesC (*network embedded system C*), possibilitando a geração automática de código pronto a compilar. A transformação deste formalismo para outras linguagens, promove a interoperabilidade entre esta linguagem e outros sistemas. Para tal, é assumida uma abordagem orientada a modelos, nomeadamente MDA (*Model-Driven Architecture*). O conceito MDA em si é uma abordagem de alto-nível ao desenvolvimento de *software* pois permite aplicar vários graus de abstração à conceção e desenvolvimento de sistemas. Devido ao facto das limitações existentes hoje em dia para aplicações IdC, nomeadamente em termos de plataformas suportadas e linguagem de programação suportada, a abordagem MDA, desenvolvida no âmbito desta dissertação, pode ser utilizada para simplificar o desenvolvimento de aplicações IdC visto ser independente de plataforma.

O desenvolvimento desta dissertação passou por diversas etapas. Em primeiro lugar, foi realizado o metamodelo da linguagem nesC, descrito através da linguagem Ecore e representado utilizando diagramas de classes UML (*Unified Modelling Language*) e descrição Ecore. Este metamodelo define os elementos da linguagem nesC, tais como componentes, interfaces e metodos, como classes e as relações entre eles. Depois foi implementada uma transformação entre o metamodelo nesC desenvolvido e o metamodelo da linguagem XML utilizando a linguagem de transformação de modelos ATL (*ATLAS Transformation Language*). Foi também desenvolvida outra transformação ATL mas para a geração de ficheiros de texto a partir de instanciações do metamodelo nesC.

As transformações foram validadas instanciando quatro aplicações diferentes, provando que, aplicando uma abordagem MDA, é possível obter descrições da mesma aplicação em linguagens diferentes e é possível criar um mecanismo de geração automática de código nesC. Utilizando esta abordagem, com o metamodelo nesC desenvolvido é possível conceber aplicações que corram em catorze plataformas de hardware diferentes (TinyOS Wiki, 2012) compatíveis com TinyOS, ao passo que, por exemplo, o Simulador TOSSIM que é o simulador por defeito do TinyOS só consegue simular arquiteturas mica e micaz (Lee, 2003).

O metamodelo desenvolvido possui todas as características da linguagem à exceção de dois aspetos: Módulos Binários e Atributos nesC. Módulos Binários são ligações a ficheiros objeto ".o", que são, por exemplo, compilações de outras aplicações. Estes módulos são incluídos no metamodelo como uma classe em branco pois envolvem a implementação e ligação a código extern. Quanto aos atributos nesC, estes não foram implementados pois necessitam de implementação de código C e para isso seria necessário elaborar/utilizar um metamodelo C ou expandir o metamodelo nesC existente. De qualquer forma, estes atributos poderiam ser implementados como mais uma enumeração no metamodelo e atributos nas devidas classes que os pudessem utilizar.

O trabalho desenvolvido nesta dissertação permite facilitar a simulação de dispositivos IdC na medida em que o metamodelo nesC está definido e validado, sendo necessário implementar mais uma transformação ATL entre modelos, por exemplo entre o metamodelo nesC e um metamodelo Simulink.

Outra abordagem a considerar depois da elaboração desta dissertação é continuar a melhorar a geração de ficheiros de texto a partir do metamodelo, desenvolvendo, por exemplo, uma interface gráfica para o utilizador poder escolher os elementos do metamodelo que quer utilizar na sua aplicação e gerar o respetivo código pronto a compilar e a instalar nos dispositivos compatíveis com nesC.

Bibliografia

Advanticsys, 2012. CM3000 [WWW Document]. URL http://advanticsys.com/wiki/index.php?title=CM3000 (accessed 9.15.16).

Agostinho, C., 2012. Sustainability of Systems Interoperability in Dynamic Business Networks. Faculdade de Ciências e Tecnologia - Universidade Nova de Lisboa.

Bajwa, I.S., Bordbar, B., Lee, M.G., 2010. OCL Constraints Generation from Natural Language Specification, in: 2010 14th IEEE International Enterprise Distributed Object Computing Conference. IEEE, pp. 204–213. doi:10.1109/EDOC.2010.33

Clark, T., Evans, A., Kent, S., 2002. Engineering Modelling Languages: A Precise Meta-Modelling Approach. pp. 159–173. doi:10.1007/3-540-45923-5_11

Dunkels, A., Schmidt, O., Voigt, T., Ali, M., 2006. Protothreads, in: Proceedings of the 4th International Conference on Embedded Networked Sensor Systems - SenSys '06. ACM Press, New York, New York, USA, p. 29. doi:10.1145/1182807.1182811

Feinerer, I., 2007. A Formal Treatment of UML Class Diagrams as an Efficient Method for Configuration Management. Institute of Computer Languages Vienna University of Technology.

Frankel, D.S., 2003. Model Driven Architecture: Applying MDA to Enterprise Computing.

Gajjar, S., Choksi, N., Sarkar, M., Dasgupta, K., 2014. Comparative analysis of wireless sensor network motes, in: 2014 International Conference on Signal Processing and Integrated Networks (SPIN). IEEE, pp. 426–431. doi:10.1109/SPIN.2014.6776991

Gay, D., Levis, P., Culler, D., Brewer, E., 2009. nesC 1.3 Language Reference Manual 1–46.

Gay, D., Levis, P., von Behren, R., Welsh, M., Brewer, E., Culler, D., 2003. The nesC language: A holistic approach to networked embedded systems. Proc. ACM SIGPLAN 2003 Conf. Program. Lang. Des. Implement. - PLDI '03 1. doi:10.1145/781131.781133

Gubbi, J., Buyya, R., Marusic, S., Palaniswami, M., 2013. Internet of Things (IoT): A vision, architectural elements, and future directions. Futur. Gener. Comput. Syst. 29, 1645–1660. doi:10.1016/j.future.2013.01.010

Guduric, P., Puder, A., Todtenhoefer, R., 2009. A Comparison between Relational and Operational QVT Mappings, in: 2009 Sixth International Conference on Information

Technology: New Generations. IEEE, pp. 266–271. doi:10.1109/ITNG.2009.156

Hankins, C., 1992. The SGML Handbook. Comput. J. 35, 40–40. doi:10.1093/comjnl/35.1.40

Ishaq, I., Hoebeke, J., Moerman, I., Demeester, P., 2012. Internet of Things Virtual Networks: Bringing Network Virtualization to Resource-Constrained Devices, in: 2012 IEEE International Conference on Green Computing and Communications. IEEE, pp. 293–300. doi:10.1109/GreenCom.2012.152

Jorges, S., Steffen, B., 2012. Exploiting Ecore's Reflexivity for Bootstrapping Domain-Specific Code-Generators, in: 2012 35th Annual IEEE Software Engineering Workshop. IEEE, pp. 72–81. doi:10.1109/SEW.2012.14

Kortuem, G., Kawsar, F., Fitton, D., Sundramoorthy, V., 2010. Smart objects as building blocks for the Internet of things. IEEE Internet Comput. 14, 44–51. doi:10.1109/MIC.2009.143

Lajara, R., Pelegrí-Sebastiá, J., Solano, J.J.P., 2010. Power Consumption Analysis of Operating Systems for Wireless Sensor Networks. Sensors 10, 5809–5826. doi:10.3390/s100605809

Lee, P.L. and N., 2003. TOSSIM: A Simulator for TinyOS Networks.

Levis, P., Madden, S., Polastre, J., Szewczyk, R., Whitehouse, K., Woo, A., Gay, D., Hill, J., Welsh, M., Brewer, E., Culler, D., 2005. TinyOS: An Operating System for Sensor Networks, in: Ambient Intelligence. Springer-Verlag, Berlin/Heidelberg, pp. 115–148. doi:10.1007/3-540-27139-2_7

Lin, Z., Hui, S., Qiang, S., Changjian, G., Xueyou, H., Xiaowei, X., 2009. Design and implementation of wireless sensor network node in environment monitoring, in: 2009 2nd IEEE International Conference on Broadband Network & Multimedia Technology. IEEE, pp. 715–718. doi:10.1109/ICBNMT.2009.5348542

Martin Fowler, 2004. UML Distilled: A Brief Guide to the Standard Object Modeling Language.

NS-3 Consortium, 2011. What is NS-3 [WWW Document]. URL https://www.nsnam.org/overview/what-is-ns-3/ (accessed 9.10.16).

Osterlind, F., Dunkels, A., Eriksson, J., Finne, N., Voigt, T., 2006. Cross-Level Sensor Network Simulation with COOJA, in: Proceedings. 2006 31st IEEE Conference on Local Computer Networks. IEEE, pp. 641–648. doi:10.1109/LCN.2006.322172

Paul, T., Kumar, G.S., 2009. Safe Contiki OS: Type and Memory Safety for Contiki OS, in: 2009 International Conference on Advances in Recent Technologies in Communication and Computing. IEEE, pp. 169–171. doi:10.1109/ARTCom.2009.126

Pavlos, R., 2013. Model Driven Development in Sensor Networks.

Pomante, L., Candia, S., Incerto, E., 2015. A Model-Driven approach for the development of an IDE for Spacecraft on-board software, in: 2015 IEEE Aerospace Conference. IEEE, pp. 1–17. doi:10.1109/AERO.2015.7119032

Raggett, D., 1997. HTML 3.2 Reference Specification [WWW Document]. URL https://www.w3.org/TR/REC-html32-19970114 (accessed 2.2.16).

Rodrigues, J., 2015. A Dynamic Model-based Simulation Framework for Resource Constrained Devices.

Savolainen, T., Soininen, J., Silverajan, B., 2013. IPv6 Addressing Strategies for IoT. IEEE Sens. J. 13, 3511–3519. doi:10.1109/JSEN.2013.2259691

Schmidt, D.C., 2006. Guest Editor's Introduction: Model-Driven Engineering. Computer (Long. Beach. Calif). 39, 25–31. doi:10.1109/MC.2006.58

Seidl, M., Scholz, M., Huemer, C., Kappel, G., 2015. UML @ Classroom, Undergraduate Topics in Computer Science. Springer International Publishing, Cham. doi:10.1007/978-3-319-12742-2

Silva, E.M., Malo, P., Albano, M., 2016. Energy consumption awareness for resource-constrained devices, in: 2016 European Conference on Networks and Communications (EuCNC). IEEE, pp. 74–78. doi:10.1109/EuCNC.2016.7561008

Sundani, H., Li, H., Devabhaktuni, V.K., Alam, M., Bhattacharya, P., 2011. Wireless Sensor Network Simulators A Survey and Comparisons. Int. J. Comput. Networks (IJCN), Vol. Issue 2, 249–265.

Tieqiang Li, Jianbin Liu, Manze Li, Shuai Zhang, 2010. Research on information transformation based on XMI, in: 2010 3rd International Conference on Computer Science and Information Technology. IEEE, pp. 356–359. doi:10.1109/ICCSIT.2010.5563946

TinyOS Wiki, 2012. Platform Hardware [WWW Document]. URL http://tinyos.stanford.edu/tinyos-wiki/index.php/Platform_Hardware (accessed 9.15.16).

Titzer, B.L., Lee, D.K., Palsberg, J., 2005. Avrora: scalable sensor network simulation with precise timing, in: IPSN 2005. Fourth International Symposium on Information Processing in Sensor Networks, 2005. IEEE, pp. 477–482. doi:10.1109/IPSN.2005.1440978

Truyen, F., 2006. The Fast Guide to Model Driven Architecture: The Basics of Model Driven Architecture.

Vermesan, O., Friess, P., 2014. Internet of Things: From Research and Innovation to Market Deployment. River Publishers.

Vortler, T., Hockner, B., Hofstedt, P., Klotz, T., 2015. Formal Verification of Software for the Contiki Operating System Considering Interrupts, in: 2015 IEEE 18th International Symposium on Design and Diagnostics of Electronic Circuits & Systems. IEEE, pp. 295–298. doi:10.1109/DDECS.2015.59

Wagelaar, D., 2016. MMT/ATL Transformation Language (ATL) [WWW Document]. URL http://www.eclipse.org/atl/ (accessed 8.25.16).

Watson, A., 2008. Visual Modelling: past, present and future.

Welsh, M., Lorincz, K., 2006. MoteTrack: A Robust, Decentralized Approach to RF-Based Location Tracking [WWW Document]. URL http://www.eecs.harvard.edu/~konrad/projects/motetrack/ (accessed 1.15.16).

Zhang, J., Wei, F., Li, H., Chen, Y., 2010. Using ATL to receive model transformation in MDA. Proc. - 2010 Int. Conf. Inf. Sci. Manag. Eng. ISME 2010 2, 135–138. doi:10.1109/ISME.2010.57

Anexo A – Representação Ecore do metamodelo nesC

No texto seguinte está representada, na integra, a descrição em código Ecore do metamodelo nesC desenvolvido no capítulo 3, cuja representação em diagrama de classes UML está na Figura 3.15.

```
<?xml version="1.0" encoding="UTF-8"?>
<ecore:EPackage                    xmi:version="2.0"                xmlns:xmi="http://www.omg.org/XMI"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:ecore="http://www.eclipse.org/emf/2002/Ecore"                name="NetworkEmbeddedSystemC"
nsURI="http://cts.uninova.pt/nesC" nsPrefix="nesC">
  <eClassifiers xsi:type="ecore:EClass" name="Statement" abstract="true">
    <eStructuralFeatures xsi:type="ecore:EReference" name="hasBody" eType="#//Body"
        containment="true"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EEnum" name="ElementType">
    <eLiterals name="Generic" value="1"/>
    <eLiterals name="Normal"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EEnum" name="OperatorType">
    <eLiterals name="none"/>
    <eLiterals name="And" value="1"/>
    <eLiterals name="Or" value="2"/>
    <eLiterals name="Not" value="3"/>
    <eLiterals name="BitAnd" value="4"/>
    <eLiterals name="BitOr" value="5"/>
    <eLiterals name="BitXor" value="6"/>
    <eLiterals name="Greater" value="7"/>
    <eLiterals name="GreaterEqual" value="8"/>
    <eLiterals name="Lesser" value="9"/>
    <eLiterals name="LesserEqual" value="10"/>
    <eLiterals name="LeftShift" value="11"/>
    <eLiterals name="RightShift" value="12"/>
    <eLiterals name="Plus" value="13"/>
    <eLiterals name="Minus" value="14"/>
    <eLiterals name="Multiply" value="15"/>
    <eLiterals name="Divide" value="16"/>
    <eLiterals name="Modulus" value="17"/>
    <eLiterals name="Assign" value="18"/>
    <eLiterals name="Equal" value="19"/>
    <eLiterals name="returnValue" value="20"/>
    <eLiterals name="void" value="21"/>
    <eLiterals name="MethodCall" value="22"/>
    <eLiterals name="Init" value="23"/>
    <eLiterals name="EnumerationAcc" value="24"/>
    <eLiterals name="ClassAcc" value="25"/>
    <eLiterals name="PathAcc" value="26"/>
    <eLiterals name="EmitEvent" value="27"/>
```

```
    <eLiterals name="NewFrame" value="28"/>
    <eLiterals name="DeleteFrame" value="29"/>
    <eLiterals name="EventAcc" value="30"/>
    <eLiterals name="NotEqual" value="31"/>
    <eLiterals name="Cast" value="32"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EEnum" name="InternalType">
    <eLiterals name="none"/>
    <eLiterals name="int" value="1"/>
    <eLiterals name="int8_t" value="2"/>
    <eLiterals name="int32_t" value="3"/>
    <eLiterals name="uint8_t" value="4"/>
    <eLiterals name="uint16_t" value="5"/>
    <eLiterals name="uint32_t" value="6"/>
    <eLiterals name="bool" value="7"/>
    <eLiterals name="char" value="8"/>
    <eLiterals name="float" value="9"/>
    <eLiterals name="double" value="10"/>
    <eLiterals name="void" value="11"/>
    <eLiterals name="pointer" value="12"/>
    <eLiterals name="struct" value="13"/>
    <eLiterals name="enumDeclaration" value="14"/>
    <eLiterals name="reference" value="15"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="If" eSuperTypes="#//Statement">
    <eStructuralFeatures xsi:type="ecore:EReference" name="condition" lowerBound="1"
        eType="#//Expression" containment="true"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="hasElseBody" eType="#//Body"
        containment="true"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="Atomic" eSuperTypes="#//Statement">
    <eStructuralFeatures xsi:type="ecore:EReference" name="statement" eType="#//Statement"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="For" eSuperTypes="#//Statement">
    <eStructuralFeatures xsi:type="ecore:EReference" name="condition" lowerBound="1"
        eType="#//Expression" containment="true"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="int" lowerBound="1" eType="#//Expression"
        containment="true"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="step" lowerBound="1" eType="#//Expression"
        containment="true"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="While" eSuperTypes="#//Statement">
    <eStructuralFeatures xsi:type="ecore:EReference" name="condition" lowerBound="1"
        eType="#//Expression" containment="true"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="Model">
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="name" lowerBound="1" eType="ecore:EDataType
http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="version" lowerBound="1"
        eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="application" lowerBound="1"
        eType="#//Component" containment="true"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="Component">
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="name" lowerBound="1" eType="ecore:EDataType
http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures  xsi:type="ecore:EAttribute"  name="type"  lowerBound="1"  eType="#//Ele-
mentType"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="provides" upperBound="-1"
        eType="#//Interface" containment="true"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="uses" upperBound="-1" eType="#//Interface"
        containment="true"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="hasDeclarations" upperBound="-1"
        eType="#//Variable" containment="true"/>
    <eStructuralFeatures   xsi:type="ecore:EAttribute"   name="Identifier"   eType="ecore:EDataType
http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="hasHeaders" upperBound="-1"
        eType="#//Header" containment="true"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="hasParameters" upperBound="-1"
        eType="#//Argument"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="hasBareCommands" upperBound="-1"
        eType="#//Command" containment="true"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="hasBareEvent" upperBound="-1"
        eType="#//Event" containment="true"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="EndPoint">
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="identifier" lowerBound="1"
        eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="Wiring">
    <eStructuralFeatures xsi:type="ecore:EReference" name="provider" lowerBound="1"
        eType="#//EndPoint" containment="true"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="user" lowerBound="1" eType="#//EndPoint"
        containment="true"/>
```

```
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="Configuration" eSuperTypes="#//Component">
    <eStructuralFeatures xsi:type="ecore:EReference" name="wires" upperBound="-1"
        eType="#//Wiring" containment="true"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="implementation" upperBound="-1"
        eType="#//Component" containment="true"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="Module" eSuperTypes="#//Component">
    <eStructuralFeatures xsi:type="ecore:EReference" name="hasVariables" upperBound="-1"
        eType="#//Variable" containment="true"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="implements" upperBound="-1"
        eType="#//Method"/>
    <eStructuralFeatures    xsi:type="ecore:EAttribute"    name="moduleID"    eType="ecore:EDataType
http://www.eclipse.org/emf/2002/Ecore#//EInt"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="Method" abstract="true">
    <eStructuralFeatures xsi:type="ecore:EReference" name="hasBody" eType="#//Body"
        containment="true"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="hasParameters" upperBound="-1"
        eType="#//Variable" containment="true"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="name" lowerBound="1" eType="ecore:EDataType
http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="returnType" lowerBound="1"
        eType="#//VariableTypes" containment="true"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="Interface">
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="Name" lowerBound="1" eType="ecore:EDataType
http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures   xsi:type="ecore:EAttribute"   name="Type"   lowerBound="1"   eType="#//Ele-
mentType"/>
    <eStructuralFeatures    xsi:type="ecore:EAttribute"    name="Identifier"    eType="ecore:EDataType
http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="hasEvents" upperBound="-1"
        eType="#//Event" containment="true"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="hasCommands" upperBound="-1"
        eType="#//Command" containment="true"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="hasParameters" upperBound="-1"
        eType="#//Argument"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="Expression" eSuperTypes="#//Statement #//Argument">
    <eStructuralFeatures xsi:type="ecore:EReference" name="left" eType="#//Argument"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="right" eType="#//Argument"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="hasArguments" upperBound="2"
        eType="#//Argument" containment="true"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="operator" lowerBound="1"
        eType="#//OperatorType"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="Argument" abstract="true"/>
  <eClassifiers xsi:type="ecore:EClass" name="Literal" eSuperTypes="#//Argument">
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="literal" lowerBound="1"
        eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EString"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="MethodRef" eSuperTypes="#//Argument">
    <eStructuralFeatures xsi:type="ecore:EReference" name="method" lowerBound="1"
        eType="#//Method"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="hasParameters" upperBound="-1"
        eType="#//Variable"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="Variable" eSuperTypes="#//Argument">
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="name" lowerBound="1" eType="ecore:EDataType
http://www.eclipse.org/emf/2002/Ecore#//EString"/>
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="typedef" lowerBound="1"
        eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EBoolean"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="InitExpr" eType="#//Expression"
        containment="true"/>
    <eStructuralFeatures  xsi:type="ecore:EReference"  name="type"  lowerBound="1"  eType="#//Variable-
Types"
        containment="true"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="Array" eSuperTypes="#//Variable">
    <eStructuralFeatures xsi:type="ecore:EReference" name="Dim" lowerBound="1" eType="#//Argument"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="Struct" eSuperTypes="#//Variable">
    <eStructuralFeatures xsi:type="ecore:EReference" name="hasStructs" upperBound="-1"
        eType="#//Struct"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="hasVariables" upperBound="-1"
        eType="#//Variable" containment="true"/>
    <eStructuralFeatures   xsi:type="ecore:EAttribute"   name="struct_name"   eType="ecore:EDataType
http://www.eclipse.org/emf/2002/Ecore#//EString"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="Enums" eSuperTypes="#//Variable">
    <eStructuralFeatures xsi:type="ecore:EReference" name="hasVariables" upperBound="-1"
        eType="#//Variable" containment="true"/>
```

```
    <eStructuralFeatures    xsi:type="ecore:EAttribute"    name="enum_name"    eType="ecore:EDataType
http://www.eclipse.org/emf/2002/Ecore#//EString"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="Header">
    <eStructuralFeatures xsi:type="ecore:EReference" name="hasVariables" upperBound="-1"
        eType="#//Variable" containment="true"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="includes" upperBound="-1"
        eType="#//Header" containment="true"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="hasFunctions" upperBound="-1"
        eType="#//InternalFunction" containment="true"/>
    <eStructuralFeatures      xsi:type="ecore:EAttribute"      name="name"      eType="ecore:EDataType
http://www.eclipse.org/emf/2002/Ecore#//EString"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="Body">
    <eStructuralFeatures xsi:type="ecore:EReference" name="hasVariables" upperBound="-1"
        eType="#//Variable" containment="true"/>
    <eStructuralFeatures xsi:type="ecore:EReference" name="hasStatements" upperBound="-1"
        eType="#//Statement" containment="true"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="Task" eSuperTypes="#//Method">
    <eStructuralFeatures xsi:type="ecore:EAttribute" name="post" lowerBound="1" eType="ecore:EDataType
http://www.eclipse.org/emf/2002/Ecore#//EBoolean"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="InternalFunction" eSuperTypes="#//Method"/>
  <eClassifiers xsi:type="ecore:EClass" name="Command" eSuperTypes="#//Method">
    <eStructuralFeatures        xsi:type="ecore:EAttribute"        name="async"        lowerBound="1"
eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EBoolean"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="Event" eSuperTypes="#//Method">
    <eStructuralFeatures        xsi:type="ecore:EAttribute"        name="async"        lowerBound="1"
eType="ecore:EDataType http://www.eclipse.org/emf/2002/Ecore#//EBoolean"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="BinaryModule" eSuperTypes="#//Component"/>
  <eClassifiers xsi:type="ecore:EClass" name="VariableTypes" abstract="true"/>
  <eClassifiers xsi:type="ecore:EClass" name="ExternalTypes" eSuperTypes="#//VariableTypes">
    <eStructuralFeatures  xsi:type="ecore:EAttribute"  name="type"  lowerBound="1"  eType="#//External-
Type"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EClass" name="InternalTypes" eSuperTypes="#//VariableTypes">
    <eStructuralFeatures  xsi:type="ecore:EAttribute"  name="type"  lowerBound="1"  eType="#//Internal-
Type"/>
  </eClassifiers>
  <eClassifiers xsi:type="ecore:EEnum" name="ExternalType">
    <eLiterals name="none"/>
    <eLiterals name="nx_int8_t" value="1"/>
    <eLiterals name="nx_uint8_t" value="2"/>
    <eLiterals name="nxle_int8_t" value="3"/>
    <eLiterals name="nxle_uint8_t" value="4"/>
    <eLiterals name="nx_int16_t" value="5"/>
    <eLiterals name="nx_uint16_t" value="6"/>
    <eLiterals name="nxle_int16_t" value="7"/>
    <eLiterals name="nxle_uint16_t" value="8" literal="nxle_uint16_t"/>
    <eLiterals name="nx_int32_t" value="9" literal="nx_int32_t"/>
    <eLiterals name="nx_uint32_t" value="10"/>
    <eLiterals name="nxle_int32_t" value="11"/>
    <eLiterals name="nxle_uint32_t" value="12"/>
    <eLiterals name="nx_int64_t" value="13"/>
    <eLiterals name="nx_uint64_t" value="14"/>
    <eLiterals name="nxle_int64_t" value="15"/>
    <eLiterals name="nxle_uint64_t" value="16"/>
    <eLiterals name="nx_struct" value="17"/>
    <eLiterals name="nx_union" value="18"/>
  </eClassifiers>
</ecore:EPackage>
```

Anexo B – Tabela de mapeamento nesC para XML

Tabela completa do mapeamento entre o metamodelo nesC desenvolvido e o metamodelo XML, desenvolvido no capitulo quatro.

Modelo Original - nesC	Modelo Final - XML
Model: M	Raiz **Nome**: "nesC:Model" **Atributos**: M.name; M.version; M.application
Model.name(string) Model.version(string) Model.application(Configuration)	Atributo Atributo Elemento
Configuration: Conf	Elemento **Nome**: "nesc:Configuration" **Atributos**: Conf.Implementation; Conf.Wires; (mais todos os atributos e elementos herdados da classe "Component")
Configuration.name(string) Configuration.type(ElementType) Configuration.wires(Wiring) Configuration.implementation(Component)	Atributo Atributo Elemento Elemento
Wiring: W	Elemento **Nome**: "nesC:Wiring" **Atributos**: W.user; W.provider
Wiring.user(EndPoint) Wiring.provider(EndPoint)	Atributo Atributo
Component: C	Elemento **Nome**: "nesC:Component" **Atributos**: C.Name; C.Type; C.Identifier; C.hasParameters; C.uses; C.provides; C.hasBareEvents; C.hasBareCommands; C.hasDeclarations; C.hasHeaders
Component.Name(string) Component.Type(ElementType) Component.Identifier(string) Component.hasParameters(Argument) Component.uses(Interface) Component.provides(Interface) Component.hasBareEvents(Event) Component.hasBareCommands(Command) Component.hasDeclarations(Variable) Component.hasHeaders(Header)	Atributo Atributo Atributo Elemento Elemento Elemento Elemento Elemento Elemento Elemento

BinaryModule: BM	Elemento **Nome:** "nesC:BinaryModule"
Module: Mo	Elemento **Nome:** "nesC:Module" **Atributos:** Mo.ModuleID; (mais todos os atributos e elementos herdados da classe "Component")
Mo.ModuleID(int)	A Atributo
Interface: Itf	Elemento **Nome:** "nesC:Interface" **Atributos:** Itf.Name; Itf.Type; Itf.Identifier; Itf.hasParameters; Itf.hasCommands; Itf.hasEvents
Interface.Name(string) Interface.Type(ElementType) Interface.Identifier(string) Interface.hasParameters(Variable) Interface.hasCommands(Command) Interface.hasEvents(Event)	Atributo Atributo Atributo Elemento Elemento Elemento
Method: Me Me.Name(string) Me.returnType(VariableType) Me.hasBody(Body) Me.hasParameters(Variable)	 Atributo Atributo Elemento Elemento
Task: T	Elemento **Nome**: "nesC:Task" **Atributos**: T.Post; (mais todos os atributos e elementos herdados da classe "Method")
T.Post(Boolean)	Atributo
Command: Comm	Elemento **Nome**: "nesC:Command" **Atributos**: Comm.Async; (mais todos os atributos e elementos herdados da classe "Method")
Comm.async(Boolean)	Atributo
Event: Ev	Elemento **Nome**: "nesC:Event" **Atributos**: Ev.Async; (mais todos os atributos e elementos herdados da classe "Method")
Ev.async(Boolean)	Atributo
InternalFunction: Inf	Elemento **Nome**: "nesC:InternalFunction" **Atributos**: (todos os atributos e elementos herdados da classe "Method")
Body: B	Elemento **Nome:** "nesC:Body" **Atributos**: B.hasStatements; B.hasVariables
B.hasStatements(Statement) B.hasVariables(Variable)	Elemento Elemento
Statements: S S.hasBody	 Elemento
Atomic: A	Elemento **Nome**: "nesC:Atomic" **Atributos**: A.Statement; (mais todos os atributos e elementos herdados da classe "Statement")
A.statement(Statement)	Elemento
IF: If	Elemento **Nome**: "nesC:If" **Atributos**: If.hasElseBody; If.Condition; (mais todos os atributos e elementos herdados da classe "Statement")
If.hasElseBody(Body) If.condition(Expression)	Elemento Elemento
While: Wh	Elemento **Nome**: "nesC:While" **Atributos**: Wh.Condition; (mais todos os atributos e elementos herdados da classe "Statement")
Wh.condition(Expression)	Elemento

For: F	Elemento **Nome**: “nesC:For” **Atributos**: F.Int; F.Condition; F.Step; (mais todos os atributos e elementos herdados da classe “Statement”)
F.int(Expression) F.int(Expression) F.int(Expression)	Elemento Elemento Elemento
Expression: Ex	Elemento **Nome**: “nesC:Expression” **Atributos**: Ex.OperatorType; Ex.Left; Ex.Right; Ex.hasArguments
Ex.OpertatorType(OperatorType) Ex.Left(Argument) Ex.Right(Argument) Ex.hasArguments(Argument)	Atributo Elemento Elemento Elemento
Literal: L	Elemento **Nome**: “nesC:Literal” **Atributos**: L.literal;
L.literal(string)	Atributo
MethodRef: MR	Elemento **Nome**: “nesC:MethodRef” **Atributos**: MR.Method; MR.hasParameters
MR.Method(Method) MR.hasParameters(Variable)	Elemento Elemento
Variable: V	Elemento **Nome**: “nesC:Variable” **Atributos**: V.Name; V.Typedef; V.initExpr; V.Type;
V.Name(string) V.Typedef(boolean) V.Type(VariableType) V.initExpr(Expression)	Atributo Atributo Atributo Elemento
Struct: S	Elemento **Nome**: “nesC:Struct” **Atributos**: S.Struct_name; S.hasVariables; S.hasStructs; (mais todos os atributos e elementos herdados da classe “Variable”)
S.Struct_name(string) S.hasVariables(Variable) S.hasStructs(Struct)	Atributo Elemento Elemento
Array: A	Elemento **Nome**: “nesC:Array” **Atributos**: A.Dim; (mais todos os atributos e elementos herdados da classe “Variable”)
A.Dim(Argument)	Elemento
Enums: En	Elemento **Nome**: “nesC:Enums” **Atributos**: En.Enum_name; En.hasVariables; En.Name;
En.Name(string) En.Enum_name(string) En.hasVariables(Variable)	Atributo Atributo Elemento

Anexo C – “EmptyInstallation”

Neste anexo está presente todo o código referente à aplicação “EmptyInstallation” ao longo das várias etapas desta dissertação, validação do metamodelo nesC proposto e transformações desenvolvidas.

“EmptyInstalation_nesC.xmi” – Instanciação nesC

```
<?xml version="1.0" encoding="ASCII"?>
<nesC:Model xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:nesC="http://cts.un-
inova.pt/nesC" name="Network Embedded System C" version="1.0">
  <application xsi:type="nesC:Configuration" name="EmptyInstallationAppC" type="Normal">
    <wires>
      <provider identifier="mainC"/>
      <user identifier="App.Boot"/>
    </wires>
    <implementation xsi:type="nesC:Module" name="EmptyInstallationC" type="Normal" Identifier="App"
implements="//@application/@implementation.0/@uses.0/@hasEvents.0">
      <uses Name="Boot" Type="Normal">
        <hasEvents name="booted">
          <returnType xsi:type="nesC:InternalTypes" type="void"/>
        </hasEvents>
      </uses>
    </implementation>
    <implementation name="mainC" type="Normal"/>
  </application>
</nesC:Model>
```

“EmptyInstalation_XMI.xmi” – Instanciação XML

```
<?xml version="1.0" encoding="ASCII"?>
<xmi:Root xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" name="nesC:Model">
  <children xsi:type="xmi:Attribute" name="name" value="Network Embedded System C"/>
  <children xsi:type="xmi:Attribute" name="version" value="1.0"/>
  <children xsi:type="xmi:Element" name="nesC:Configuration">
    <children xsi:type="xmi:Attribute" name="name" value="EmptyInstallationAppC"/>
    <children xsi:type="xmi:Element" name="nesC:Module">
      <children xsi:type="xmi:Attribute" name="name" value="EmptyInstallationC"/>
      <children xsi:type="xmi:Attribute" name="identifier" value="App"/>
      <children xsi:type="xmi:Element" name="nesC:Interface">
        <children xsi:type="xmi:Attribute" name="name" value="Boot"/>
        <children xsi:type="xmi:Element" name="nesC:Event">
          <children xsi:type="xmi:Attribute" name="name" value="booted"/>
          <children xsi:type="xmi:Attribute" name="nesC:InternalTypes" value="void"/>
        </children>
      </children>
    </children>
```

```
    <children xsi:type="xmi:Element" name="nesC:Component">
      <children xsi:type="xmi:Attribute" name="name" value="mainC"/>
    </children>
    <children xsi:type="xmi:Element" name="nesC:Wiring">
      <children xsi:type="xmi:Attribute" name="user" value="App.Boot"/>
      <children xsi:type="xmi:Attribute" name="provider" value="mainC"/>
    </children>
  </children>
</xmi:Root>
```

“EmptyInstalation_XML.xml” – Representação XML

```
<?xml version = '1.0' encoding = 'ISO-8859-1' ?>
<nesC:Model name="Network Embedded System C" version="1.0">
  <nesC:Configuration name="EmptyInstallationAppC">
    <nesC:Module name="EmptyInstallationC" identifier="App">
      <nesC:Interface name="Boot">
        <nesC:Event name="booted" nesC:InternalTypes="void"/>
      </nesC:Interface>
    </nesC:Module>
    <nesC:Component name="mainC"/>
    <nesC:Wiring user="App.Boot" provider="mainC"/>
  </nesC:Configuration>
</nesC:Model>
```

“EmptyInstalation” – Código nesC Gerado

Configuração: “EmptyInstallationAppC.nc”

```
configuration EmptyInstallationAppC{
}
implementation{
        components EmptyInstallationC as App;
        components mainC;

        App.Boot -> mainC;
}
```

Módulo: “EmptyInstallationC.nc”

```
module EmptyInstallationC @safe() {
        uses {
                interface Boot;
        }
}
implementation{
        event void Boot.booted(){
        }
}
```

Anexo D – “ToggleLeds”

Neste anexo está presente todo o código referente à aplicação “ToggleLeds” ao longo das várias etapas desta dissertação, validação do metamodelo nesC proposto e transformações desenvolvidas.

“ToggleLeds_nesC.xmi” – Instanciação nesC

```
<?xml version="1.0" encoding="ASCII"?>
<nesC:Model xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:nesC="http://cts.unino-
va.pt/nesC" name="Network Embedded System C" version="1.0">
  <application xsi:type="nesC:Configuration" name="ToggleLedsAppC" type="Normal">
    <wires>
      <provider identifier="MainC"/>
      <user identifier="App.Boot"/>
    </wires>
    <wires>
      <provider identifier="LedsC"/>
      <user identifier="App.Leds"/>
    </wires>
    <wires>
      <provider identifier="Timer"/>
      <user identifier="App.Timer"/>
    </wires>
    <implementation xsi:type="nesC:Module" name="ToggleLedsC" type="Normal" Identifier="App" imple-
ments="//@application/@implementation.0/@uses.2/@hasEvents.0 //@application/@implementa-
tion.0/@uses.0/@hasEvents.0">
      <uses Name="Boot" Type="Normal">
        <hasEvents name="booted">
          <hasBody>
            <hasStatements xsi:type="nesC:Expression" right="//@application/@implementa-
tion.0/@uses.0/@hasEvents.0/@hasBody/@hasStatements.0/@hasArguments.0" operator="MethodCall">
              <hasArguments xsi:type="nesC:MethodRef" method="//@application/@implementa-
tion.0/@uses.2/@hasCommands.0" hasParameters="//@application/@implementation.0/@hasHeaders.0/@hasVar-
iables.0/@hasVariables.0"/>
            </hasStatements>
          </hasBody>
          <returnType xsi:type="nesC:InternalTypes" type="void"/>
        </hasEvents>
      </uses>
      <uses Name="Leds">
        <hasCommands name="led0Toggle">
          <returnType xsi:type="nesC:InternalTypes" type="void"/>
        </hasCommands>
        <hasCommands name="led1Toggle">
          <returnType xsi:type="nesC:InternalTypes" type="void"/>
        </hasCommands>
        <hasCommands name="led2Toggle">
          <returnType xsi:type="nesC:InternalTypes" type="void"/>
        </hasCommands>
```

```
      </uses>
      <uses Name="Timer" Type="Normal" Identifier="Timer" hasParameters="//@application/@implementation.0/@hasHeaders.0/@includes.0/@hasVariables.0">
        <hasEvents name="fired">
          <hasBody>
            <hasStatements xsi:type="nesC:If">
              <hasBody>
                <hasStatements xsi:type="nesC:Expression" right="//@application/@implementation.0/@uses.2/@hasEvents.0/@hasBody/@hasStatements.0/@hasBody/@hasStatements.0/@hasArguments.0" operator="MethodCall">
                  <hasArguments xsi:type="nesC:MethodRef" method="//@application/@implementation.0/@uses.1/@hasCommands.0"/>
                </hasStatements>
              </hasBody>
              <condition left="//@application/@implementation.0/@uses.2/@hasEvents.0/@hasBody/@hasStatements.0/@condition/@hasArguments.0">
                <hasArguments xsi:type="nesC:Literal" literal="LED0_Active"/>
              </condition>
            </hasStatements>
            <hasStatements xsi:type="nesC:If">
              <hasBody>
                <hasStatements xsi:type="nesC:Expression" right="//@application/@implementation.0/@uses.2/@hasEvents.0/@hasBody/@hasStatements.1/@hasBody/@hasStatements.0/@hasArguments.0" operator="MethodCall">
                  <hasArguments xsi:type="nesC:MethodRef" method="//@application/@implementation.0/@uses.1/@hasCommands.1"/>
                </hasStatements>
              </hasBody>
              <condition left="//@application/@implementation.0/@uses.2/@hasEvents.0/@hasBody/@hasStatements.1/@condition/@hasArguments.0">
                <hasArguments xsi:type="nesC:Literal" literal="LED1_Active"/>
              </condition>
            </hasStatements>
            <hasStatements xsi:type="nesC:If">
              <hasBody>
                <hasStatements xsi:type="nesC:Expression" right="//@application/@implementation.0/@uses.2/@hasEvents.0/@hasBody/@hasStatements.2/@hasBody/@hasStatements.0/@hasArguments.0" operator="MethodCall">
                  <hasArguments xsi:type="nesC:MethodRef" method="//@application/@implementation.0/@uses.1/@hasCommands.2"/>
                </hasStatements>
              </hasBody>
              <condition left="//@application/@implementation.0/@uses.2/@hasEvents.0/@hasBody/@hasStatements.2/@condition/@hasArguments.0">
                <hasArguments xsi:type="nesC:Literal" literal="LED2_Active"/>
              </condition>
            </hasStatements>
          </hasBody>
          <returnType xsi:type="nesC:InternalTypes" type="void"/>
        </hasEvents>
        <hasCommands name="startPeriodic">
          <returnType xsi:type="nesC:InternalTypes" type="void"/>
        </hasCommands>
      </uses>
      <hasHeaders name="lib">
        <hasVariables xsi:type="nesC:Enums" name="" enum_name="">
          <type xsi:type="nesC:InternalTypes" type="enumDeclaration"/>
          <hasVariables name="TimePeriod">
            <InitExpr right="//@application/@implementation.0/@hasHeaders.0/@hasVariables.0/@hasVariables.0/@InitExpr/@hasArguments.0" operator="Assign">
              <hasArguments xsi:type="nesC:Literal" literal="1000"/>
            </InitExpr>
          </hasVariables>
          <hasVariables name="LED0_Active">
            <InitExpr right="//@application/@implementation.0/@hasHeaders.0/@hasVariables.0/@hasVariables.1/@InitExpr/@hasArguments.0" operator="Assign">
              <hasArguments xsi:type="nesC:Literal" literal="TRUE"/>
            </InitExpr>
          </hasVariables>
          <hasVariables name="LED1_Active">
            <InitExpr right="//@application/@implementation.0/@hasHeaders.0/@hasVariables.0/@hasVariables.2/@InitExpr/@hasArguments.0" operator="Assign">
              <hasArguments xsi:type="nesC:Literal" literal="TRUE"/>
            </InitExpr>
          </hasVariables>
          <hasVariables name="LED2_Active">
            <InitExpr right="//@application/@implementation.0/@hasHeaders.0/@hasVariables.0/@hasVariables.3/@InitExpr/@hasArguments.0" operator="Assign">
              <hasArguments xsi:type="nesC:Literal" literal="TRUE"/>
            </InitExpr>
          </hasVariables>
        </hasVariables>
        <includes name="Timer">
          <hasVariables name="TMilli"/>
```

```
        </includes>
      </hasHeaders>
    </implementation>
    <implementation name="MainC" type="Normal"/>
    <implementation name="LedsC" type="Normal"/>
    <implementation name="TimerMilliC" Identifier="Timer"/>
  </application>
</nesC:Model>
```

“ToggleLeds_XMI.xmi” – Instanciação XML

```
<?xml version="1.0" encoding="ASCII"?>
<xmi:Root                    xmi:version="2.0"                    xmlns:xmi="http://www.omg.org/XMI"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" name="nesC:Model">
  <children xsi:type="xmi:Attribute" name="name" value="Network Embedded System C"/>
  <children xsi:type="xmi:Attribute" name="version" value="1.0"/>
  <children xsi:type="xmi:Element" name="nesC:Configuration">
    <children xsi:type="xmi:Attribute" name="name" value="ToggleLedsAppC"/>
    <children xsi:type="xmi:Element" name="nesC:Module">
      <children xsi:type="xmi:Attribute" name="name" value="ToggleLedsC"/>
      <children xsi:type="xmi:Attribute" name="identifier" value="App"/>
      <children xsi:type="xmi:Element" name="nesC:Interface">
        <children xsi:type="xmi:Attribute" name="name" value="Boot"/>
        <children xsi:type="xmi:Element" name="nesC:Event">
          <children xsi:type="xmi:Attribute" name="name" value="booted"/>
          <children xsi:type="xmi:Attribute" name="nesC:InternalTypes" value="void"/>
          <children xsi:type="xmi:Element" name="nesC:Body">
            <children xsi:type="xmi:Element" name="nesC:Expression">
              <children xsi:type="xmi:Attribute" name="operator" value="MethodCall"/>
              <children xsi:type="xmi:Element" name="nesC:MethodRef">
                <children xsi:type="xmi:Element" name="nesC:Command">
                  <children xsi:type="xmi:Attribute" name="name" value="startPeriodic"/>
                  <children xsi:type="xmi:Attribute" name="nesC:InternalTypes" value="void"/>
                </children>
              </children>
            </children>
          </children>
        </children>
      </children>
      <children xsi:type="xmi:Element" name="nesC:Interface">
        <children xsi:type="xmi:Attribute" name="name" value="Leds"/>
        <children xsi:type="xmi:Attribute" name="type" value="Generic"/>
      </children>
      <children xsi:type="xmi:Element" name="nesC:Interface">
        <children xsi:type="xmi:Attribute" name="name" value="Timer"/>
        <children xsi:type="xmi:Attribute" name="identifier" value="Timer"/>
        <children xsi:type="xmi:Element" name="nesC:Event">
          <children xsi:type="xmi:Attribute" name="name" value="fired"/>
          <children xsi:type="xmi:Attribute" name="nesC:InternalTypes" value="void"/>
          <children xsi:type="xmi:Element" name="nesC:Body">
            <children xsi:type="xmi:Element" name="nesC:If">
              <children xsi:type="xmi:Element" name="nesC:Expression">
                <children xsi:type="xmi:Attribute" name="operator" value="none"/>
                <children xsi:type="xmi:Element" name="nesC:Literal">
                  <children xsi:type="xmi:Attribute" name="literal" value="LED0_Active"/>
                </children>
              </children>
              <children xsi:type="xmi:Element" name="nesC:Body">
                <children xsi:type="xmi:Element" name="nesC:Expression">
                  <children xsi:type="xmi:Attribute" name="operator" value="MethodCall"/>
                  <children xsi:type="xmi:Element" name="nesC:MethodRef">
                    <children xsi:type="xmi:Element" name="nesC:Command">
                      <children xsi:type="xmi:Attribute" name="name" value="led0Toggle"/>
                      <children xsi:type="xmi:Attribute" name="nesC:InternalTypes" value="void"/>
                    </children>
                  </children>
                </children>
              </children>
            </children>
            <children xsi:type="xmi:Element" name="nesC:If">
              <children xsi:type="xmi:Element" name="nesC:Expression">
                <children xsi:type="xmi:Attribute" name="operator" value="none"/>
                <children xsi:type="xmi:Element" name="nesC:Literal">
                  <children xsi:type="xmi:Attribute" name="literal" value="LED1_Active"/>
                </children>
              </children>
              <children xsi:type="xmi:Element" name="nesC:Body">
                <children xsi:type="xmi:Element" name="nesC:Expression">
                  <children xsi:type="xmi:Attribute" name="operator" value="MethodCall"/>
                  <children xsi:type="xmi:Element" name="nesC:MethodRef">
                    <children xsi:type="xmi:Element" name="nesC:Command">
```

```
                <children xsi:type="xmi:Attribute" name="name" value="led1Toggle"/>
                <children xsi:type="xmi:Attribute" name="nesC:InternalTypes" value="void"/>
              </children>
            </children>
          </children>
        </children>
        <children xsi:type="xmi:Element" name="nesC:If">
          <children xsi:type="xmi:Element" name="nesC:Expression">
            <children xsi:type="xmi:Attribute" name="operator" value="none"/>
            <children xsi:type="xmi:Element" name="nesC:Literal">
              <children xsi:type="xmi:Attribute" name="literal" value="LED2_Active"/>
            </children>
          </children>
          <children xsi:type="xmi:Element" name="nesC:Body">
            <children xsi:type="xmi:Element" name="nesC:Expression">
              <children xsi:type="xmi:Attribute" name="operator" value="MethodCall"/>
              <children xsi:type="xmi:Element" name="nesC:MethodRef">
                <children xsi:type="xmi:Element" name="nesC:Command">
                  <children xsi:type="xmi:Attribute" name="name" value="led2Toggle"/>
                  <children xsi:type="xmi:Attribute" name="nesC:InternalTypes" value="void"/>
                </children>
              </children>
            </children>
          </children>
        </children>
      </children>
    </children>
  </children>
  <children xsi:type="xmi:Element" name="nesC:Header">
    <children xsi:type="xmi:Attribute" name="name" value="lib"/>
    <children xsi:type="xmi:Element" name="nesC:Enum">
      <children xsi:type="xmi:Attribute" name="name" value=""/>
      <children xsi:type="xmi:Attribute" name="enum_name" value=""/>
      <children xsi:type="xmi:Attribute" name="nesC:InternalTypes" value="enumDeclaration"/>
      <children xsi:type="xmi:Element" name="nesC:Variable">
        <children xsi:type="xmi:Attribute" name="name" value="TimePeriod"/>
        <children xsi:type="xmi:Element" name="nesC:Expression">
          <children xsi:type="xmi:Attribute" name="operator" value="Assign"/>
          <children xsi:type="xmi:Element" name="nesC:Literal">
            <children xsi:type="xmi:Attribute" name="literal" value="1000"/>
          </children>
        </children>
      </children>
      <children xsi:type="xmi:Element" name="nesC:Variable">
        <children xsi:type="xmi:Attribute" name="name" value="LED0_Active"/>
        <children xsi:type="xmi:Element" name="nesC:Expression">
          <children xsi:type="xmi:Attribute" name="operator" value="Assign"/>
          <children xsi:type="xmi:Element" name="nesC:Literal">
            <children xsi:type="xmi:Attribute" name="literal" value="TRUE"/>
          </children>
        </children>
      </children>
      <children xsi:type="xmi:Element" name="nesC:Variable">
        <children xsi:type="xmi:Attribute" name="name" value="LED1_Active"/>
        <children xsi:type="xmi:Element" name="nesC:Expression">
          <children xsi:type="xmi:Attribute" name="operator" value="Assign"/>
          <children xsi:type="xmi:Element" name="nesC:Literal">
            <children xsi:type="xmi:Attribute" name="literal" value="TRUE"/>
          </children>
        </children>
      </children>
      <children xsi:type="xmi:Element" name="nesC:Variable">
        <children xsi:type="xmi:Attribute" name="name" value="LED2_Active"/>
        <children xsi:type="xmi:Element" name="nesC:Expression">
          <children xsi:type="xmi:Attribute" name="operator" value="Assign"/>
          <children xsi:type="xmi:Element" name="nesC:Literal">
            <children xsi:type="xmi:Attribute" name="literal" value="TRUE"/>
          </children>
        </children>
      </children>
    </children>
    <children xsi:type="xmi:Element" name="nesC:Header">
      <children xsi:type="xmi:Attribute" name="name" value="Timer"/>
      <children xsi:type="xmi:Element" name="nesC:Variable">
        <children xsi:type="xmi:Attribute" name="name" value="TMilli"/>
      </children>
    </children>
  </children>
</children>
<children xsi:type="xmi:Element" name="nesC:Component">
  <children xsi:type="xmi:Attribute" name="name" value="MainC"/>
</children>
<children xsi:type="xmi:Element" name="nesC:Component">
```

```
      <children xsi:type="xmi:Attribute" name="name" value="LedsC"/>
    </children>
    <children xsi:type="xmi:Element" name="nesC:Component">
      <children xsi:type="xmi:Attribute" name="name" value="TimerMilliC"/>
      <children xsi:type="xmi:Attribute" name="type" value="Generic"/>
    </children>
    <children xsi:type="xmi:Element" name="nesC:Wiring">
      <children xsi:type="xmi:Attribute" name="user" value="App.Boot"/>
      <children xsi:type="xmi:Attribute" name="provider" value="MainC"/>
    </children>
    <children xsi:type="xmi:Element" name="nesC:Wiring">
      <children xsi:type="xmi:Attribute" name="user" value="App.Leds"/>
      <children xsi:type="xmi:Attribute" name="provider" value="LedsC"/>
    </children>
    <children xsi:type="xmi:Element" name="nesC:Wiring">
      <children xsi:type="xmi:Attribute" name="user" value="App.Timer"/>
      <children xsi:type="xmi:Attribute" name="provider" value="Timer"/>
    </children>
  </children>
</xmi:Root>
```

“ToggleLeds_XML.xml” – Representação XML

```
<?xml version = '1.0' encoding = 'ISO-8859-1' ?>
<nesC:Model name="Network Embedded System C" version="1.0">
  <nesC:Configuration name="ToggleLedsAppC">
    <nesC:Module name="ToggleLedsC" identifier="App">
      <nesC:Interface name="Boot">
        <nesC:Event name="booted" nesC:InternalTypes="void">
          <nesC:Body>
            <nesC:Expression operator="MethodCall">
              <nesC:MethodRef>
                <nesC:Command name="startPeriodic" nesC:InternalTypes="void"/>
              </nesC:MethodRef>
            </nesC:Expression>
          </nesC:Body>
        </nesC:Event>
      </nesC:Interface>
      <nesC:Interface name="Leds" type="Generic"/>
      <nesC:Interface name="Timer" identifier="Timer">
        <nesC:Event name="fired" nesC:InternalTypes="void">
          <nesC:Body>
            <nesC:If>
              <nesC:Expression operator="none">
                <nesC:Literal literal="LED0_Active"/>
              </nesC:Expression>
              <nesC:Body>
                <nesC:Expression operator="MethodCall">
                  <nesC:MethodRef>
                    <nesC:Command name="led0Toggle" nesC:InternalTypes="void"/>
                  </nesC:MethodRef>
                </nesC:Expression>
              </nesC:Body>
            </nesC:If>
            <nesC:If>
              <nesC:Expression operator="none">
                <nesC:Literal literal="LED1_Active"/>
              </nesC:Expression>
              <nesC:Body>
                <nesC:Expression operator="MethodCall">
                  <nesC:MethodRef>
                    <nesC:Command name="led1Toggle" nesC:InternalTypes="void"/>
                  </nesC:MethodRef>
                </nesC:Expression>
              </nesC:Body>
            </nesC:If>
            <nesC:If>
              <nesC:Expression operator="none">
                <nesC:Literal literal="LED2_Active"/>
              </nesC:Expression>
              <nesC:Body>
                <nesC:Expression operator="MethodCall">
                  <nesC:MethodRef>
                    <nesC:Command name="led2Toggle" nesC:InternalTypes="void"/>
                  </nesC:MethodRef>
                </nesC:Expression>
              </nesC:Body>
            </nesC:If>
          </nesC:Body>
        </nesC:Event>
      </nesC:Interface>
```

```
      <nesC:Header name="lib">
        <nesC:Enum name="" enum_name="" nesC:InternalTypes="enumDeclaration">
          <nesC:Variable name="TimePeriod">
            <nesC:Expression operator="Assign">
              <nesC:Literal literal="1000"/>
            </nesC:Expression>
          </nesC:Variable>
          <nesC:Variable name="LED0_Active">
            <nesC:Expression operator="Assign">
              <nesC:Literal literal="TRUE"/>
            </nesC:Expression>
          </nesC:Variable>
          <nesC:Variable name="LED1_Active">
            <nesC:Expression operator="Assign">
              <nesC:Literal literal="TRUE"/>
            </nesC:Expression>
          </nesC:Variable>
          <nesC:Variable name="LED2_Active">
            <nesC:Expression operator="Assign">
              <nesC:Literal literal="TRUE"/>
            </nesC:Expression>
          </nesC:Variable>
        </nesC:Enum>
        <nesC:Header name="Timer">
          <nesC:Variable name="TMilli"/>
        </nesC:Header>
      </nesC:Header>
    </nesC:Module>
    <nesC:Component name="MainC"/>
    <nesC:Component name="LedsC"/>
    <nesC:Component name="TimerMilliC" type="Generic"/>
    <nesC:Wiring user="App.Boot" provider="MainC"/>
    <nesC:Wiring user="App.Leds" provider="LedsC"/>
    <nesC:Wiring user="App.Timer" provider="Timer"/>
  </nesC:Configuration>
</nesC:Model>
```

“ToggleLeds” – Código nesC Gerado

Configuração: “ToggleLedsAppC.nc”

```
configuration ToggleLedsAppC{
}
implementation{
        components ToggleLedsC as App;
        components MainC;
        components LedsC;
        components new TimerMilliC() as Timer;
        App.Boot -> MainC;
        App.Leds -> LedsC;
        App.Timer -> Timer;
}
```

Módulo: “ToggleLedsC.nc”

```
#include "lib.h"

module ToggleLedsC @safe() {
    uses {
        interface Boot;
        interface Leds;
        interface Timer<TMilli> as Timer;
    }
}
implementation{
    event void Timer.fired(){
        if (LED0_Active ){
            call Leds.led0Toggle();
        }
        if (LED1_Active ){
            call Leds.led1Toggle();
        }
        if (LED2_Active ){
            call Leds.led2Toggle();
        }
    }
    event void Boot.booted(){
        call Timer.startPeriodic(TimePeriod);
```

```
        }
}
```

Cabeçalho: “lib.h”

```
#ifndef LIB_H
#define LIB_H

#include "Timer.h"
enum {
                TimePeriod = 1000,
                LED0_Active = TRUE,
                LED1_Active = TRUE,
                LED2_Active = TRUE
};

#endif /* LIB_H */
```

Anexo E – “Timers”

Neste anexo está presente todo o código referente à aplicação “Timers” ao longo das várias etapas desta dissertação, validação do metamodelo nesC proposto e transformações desenvolvidas.

“Timers_nesC.xmi” – Instanciação nesC

```
<?xml version="1.0" encoding="ASCII"?>
<nesC:Model      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"      xmlns:nesC="http://cts.unin-
ova.pt/nesC" name="Network Embedded System C" version="1.0">
  <application xsi:type="nesC:Configuration" name="TimersAppC" type="Normal">
    <wires>
      <provider identifier="MainC"/>
      <user identifier="App.Boot"/>
    </wires>
    <wires>
      <provider identifier="Timer"/>
      <user identifier="App.Timer"/>
    </wires>
    <implementation  xsi:type="nesC:Module"  name="TimersC"  type="Normal"  Identifier="App"  imple-
ments="//@application/@implementation.0/@uses.0/@hasEvents.0              //@application/@implementa-
tion.0/@uses.1/@hasEvents.0">
      <uses Name="Boot" Type="Normal">
        <hasEvents name="booted">
          <hasBody>
            <hasStatements        xsi:type="nesC:Expression"        right="//@application/@implementa-
tion.0/@uses.0/@hasEvents.0/@hasBody/@hasStatements.0/@hasArguments.0" operator="MethodCall">
              <hasArguments       xsi:type="nesC:MethodRef"       method="//@application/@implementa-
tion.0/@uses.1/@hasCommands.0" hasParameters="//@application/@implementation.0/@hasHeaders.0/@hasVar-
iables.0/@hasVariables.0"/>
            </hasStatements>
          </hasBody>
          <returnType xsi:type="nesC:InternalTypes" type="void"/>
        </hasEvents>
      </uses>
      <uses Name="Timer" Type="Normal" Identifier="Timer" hasParameters="//@application/@implementa-
tion.0/@hasHeaders.0/@includes.0/@hasVariables.0">
        <hasEvents name="fired">
          <returnType xsi:type="nesC:InternalTypes" type="void"/>
        </hasEvents>
        <hasCommands         hasParameters="//@application/@implementation.0/@hasHeaders.0/@hasVaria-
bles.0/@hasVariables.0" name="startPeriodic">
          <returnType xsi:type="nesC:InternalTypes" type="void"/>
        </hasCommands>
      </uses>
      <hasHeaders name="lib">
        <hasVariables xsi:type="nesC:Enums" name="" enum_name="">
          <hasVariables name="TimePeriod">
```

```
            <InitExpr right="//@application/@implementation.0/@hasHeaders.0/@hasVariables.0/@hasVari-
ables.0/@InitExpr/@hasArguments.0" operator="Assign">
              <hasArguments xsi:type="nesC:Literal" literal="1000"/>
            </InitExpr>
          </hasVariables>
        </hasVariables>
        <includes name="Timer">
          <hasVariables name="TMilli">
            <type xsi:type="nesC:InternalTypes" type="void"/>
          </hasVariables>
        </includes>
      </hasHeaders>
    </implementation>
    <implementation name="MainC" type="Normal"/>
    <implementation name="TimerMilliC" Identifier="Timer"/>
  </application>
</nesC:Model>
```

“Timers_XMI.xmi” – Instanciação XML

```
<?xml version="1.0" encoding="ASCII"?>
<xmi:Root                    xmi:version="2.0"                    xmlns:xmi="http://www.omg.org/XMI"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" name="nesC:Model">
  <children xsi:type="xmi:Attribute" name="name" value="Network Embedded System C"/>
  <children xsi:type="xmi:Attribute" name="version" value="1.0"/>
  <children xsi:type="xmi:Element" name="nesC:Configuration">
    <children xsi:type="xmi:Attribute" name="name" value="TimersAppC"/>
    <children xsi:type="xmi:Element" name="nesC:Module">
      <children xsi:type="xmi:Attribute" name="name" value="TimersC"/>
      <children xsi:type="xmi:Attribute" name="identifier" value="App"/>
      <children xsi:type="xmi:Element" name="nesC:Interface">
        <children xsi:type="xmi:Attribute" name="name" value="Boot"/>
        <children xsi:type="xmi:Element" name="nesC:Event">
          <children xsi:type="xmi:Attribute" name="name" value="booted"/>
          <children xsi:type="xmi:Attribute" name="nesC:InternalTypes" value="void"/>
          <children xsi:type="xmi:Element" name="nesC:Body">
            <children xsi:type="xmi:Element" name="nesC:Expression">
              <children xsi:type="xmi:Attribute" name="operator" value="MethodCall"/>
              <children xsi:type="xmi:Element" name="nesC:MethodRef">
                <children xsi:type="xmi:Element" name="nesC:Command">
                  <children xsi:type="xmi:Attribute" name="name" value="startPeriodic"/>
                  <children xsi:type="xmi:Attribute" name="nesC:InternalTypes" value="void"/>
                </children>
              </children>
            </children>
          </children>
        </children>
      </children>
      <children xsi:type="xmi:Element" name="nesC:Interface">
        <children xsi:type="xmi:Attribute" name="name" value="Timer"/>
        <children xsi:type="xmi:Attribute" name="identifier" value="Timer"/>
        <children xsi:type="xmi:Element" name="nesC:Event">
          <children xsi:type="xmi:Attribute" name="name" value="fired"/>
          <children xsi:type="xmi:Attribute" name="nesC:InternalTypes" value="void"/>
        </children>
      </children>
      <children xsi:type="xmi:Element" name="nesC:Header">
        <children xsi:type="xmi:Attribute" name="name" value="lib"/>
        <children xsi:type="xmi:Element" name="nesC:Enum">
          <children xsi:type="xmi:Attribute" name="name" value=""/>
          <children xsi:type="xmi:Attribute" name="enum_name" value=""/>
          <children xsi:type="xmi:Element" name="nesC:Variable">
            <children xsi:type="xmi:Attribute" name="name" value="TimePeriod"/>
            <children xsi:type="xmi:Element" name="nesC:Expression">
              <children xsi:type="xmi:Attribute" name="operator" value="Assign"/>
              <children xsi:type="xmi:Element" name="nesC:Literal">
                <children xsi:type="xmi:Attribute" name="literal" value="1000"/>
              </children>
            </children>
          </children>
        </children>
        <children xsi:type="xmi:Element" name="nesC:Header">
          <children xsi:type="xmi:Attribute" name="name" value="Timer"/>
          <children xsi:type="xmi:Element" name="nesC:Variable">
            <children xsi:type="xmi:Attribute" name="name" value="TMilli"/>
            <children xsi:type="xmi:Attribute" name="nesC:InternalTypes" value="void"/>
          </children>
        </children>
      </children>
    </children>
    <children xsi:type="xmi:Element" name="nesC:Component">
```

```
      <children xsi:type="xmi:Attribute" name="name" value="MainC"/>
    </children>
    <children xsi:type="xmi:Element" name="nesC:Component">
      <children xsi:type="xmi:Attribute" name="name" value="TimerMilliC"/>
      <children xsi:type="xmi:Attribute" name="type" value="Generic"/>
    </children>
    <children xsi:type="xmi:Element" name="nesC:Wiring">
      <children xsi:type="xmi:Attribute" name="user" value="App.Boot"/>
      <children xsi:type="xmi:Attribute" name="provider" value="MainC"/>
    </children>
    <children xsi:type="xmi:Element" name="nesC:Wiring">
      <children xsi:type="xmi:Attribute" name="user" value="App.Timer"/>
      <children xsi:type="xmi:Attribute" name="provider" value="Timer"/>
    </children>
  </children>
</xmi:Root>
```

“Timers_XML.xml” – Representação XML

```
<?xml version = '1.0' encoding = 'ISO-8859-1' ?>
<nesC:Model name="Network Embedded System C" version="1.0">
  <nesC:Configuration name="TimersAppC">
    <nesC:Module name="TimersC" identifier="App">
      <nesC:Interface name="Boot">
        <nesC:Event name="booted" nesC:InternalTypes="void">
          <nesC:Body>
            <nesC:Expression operator="MethodCall">
              <nesC:MethodRef>
                <nesC:Command name="startPeriodic" nesC:InternalTypes="void"/>
              </nesC:MethodRef>
            </nesC:Expression>
          </nesC:Body>
        </nesC:Event>
      </nesC:Interface>
      <nesC:Interface name="Timer" identifier="Timer">
        <nesC:Event name="fired" nesC:InternalTypes="void"/>
      </nesC:Interface>
      <nesC:Header name="lib">
        <nesC:Enum name="" enum_name="">
          <nesC:Variable name="TimePeriod">
            <nesC:Expression operator="Assign">
              <nesC:Literal literal="1000"/>
            </nesC:Expression>
          </nesC:Variable>
        </nesC:Enum>
        <nesC:Header name="Timer">
          <nesC:Variable name="TMilli" nesC:InternalTypes="void"/>
        </nesC:Header>
      </nesC:Header>
    </nesC:Module>
    <nesC:Component name="MainC"/>
    <nesC:Component name="TimerMilliC" type="Generic"/>
    <nesC:Wiring user="App.Boot" provider="MainC"/>
    <nesC:Wiring user="App.Timer" provider="Timer"/>
  </nesC:Configuration>
</nesC:Model>
```

“Timers” – Código nesC Gerado

Configuração: “TimersAppC”

```
configuration TimersAppC{
}
implementation{
        components TimersC as App;
        components MainC;
        components new TimerMilliC() as Timer;
        App.Boot -> MainC;
        App.Timer -> Timer;
}
```

Módulo: “TimersC”

```
#include "lib.h"
```

```
module TimersC @safe() {
	uses {
		interface Boot;
		interface Timer<TMilli> as Timer;
	}
}
implementation{
	event void Boot.booted(){
		call Timer.startPeriodic(TimePeriod);
	}
	event void Timer.fired(){
	}
}
```

Cabeçalho: “lib.h”

```
#ifndef LIB_H
#define LIB_H

#include "Timer.h"
enum {
	TimePeriod = 1000
};

#endif /* LIB_H */
```

Anexo F – “SendMsgXbytes”

Neste anexo está presente todo o código referente à aplicação “SendMsgXbytes” ao longo das várias etapas desta dissertação, validação do metamodelo nesC proposto e transformações desenvolvidas.

“SendMsgXbytes_nesC.xmi” – Instanciação nesC

```
<?xml version="1.0" encoding="ASCII"?>
<nesC:Model xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:nesC="http://cts.un-
inova.pt/nesC" name="Network Embedded System C" version="2.0">
  <application xsi:type="nesC:Configuration" name="SendMsgXbytesAppC" type="Normal">
    <wires>
      <provider identifier="MainC"/>
      <user identifier="App.Boot"/>
    </wires>
    <wires>
      <provider identifier="LedsC"/>
      <user identifier="App.Leds"/>
    </wires>
    <wires>
      <provider identifier="Wireless"/>
      <user identifier="App.AMControl"/>
    </wires>
    <wires>
      <provider identifier="AMSend"/>
      <user identifier="App.AMSend"/>
    </wires>
    <wires>
      <provider identifier="Timer"/>
      <user identifier="App.Timer"/>
    </wires>
    <implementation xsi:type="nesC:Module" name="SendMsgXbytesC" type="Normal" Identifier="App" im-
plements="//@application/@implementation.0/@uses.0/@hasEvents.0 //@application/@implementa-
tion.0/@uses.2/@hasEvents.0 //@application/@implementation.0/@uses.2/@hasEvents.1 //@application/@im-
plementation.0/@uses.3/@hasEvents.0 //@application/@implementation.0/@uses.4/@hasEvents.0">
      <uses Name="Boot" Type="Normal">
        <hasEvents name="booted">
          <hasBody>
            <hasStatements xsi:type="nesC:Expression" left="//@application/@implementation.0/@hasVar-
iables.1" right="//@application/@implementation.0/@uses.0/@hasEvents.0/@hasBody/@hasStatements.0/@ha-
sArguments.0" operator="Assign">
              <hasArguments xsi:type="nesC:Literal" literal="FALSE"/>
            </hasStatements>
            <hasStatements xsi:type="nesC:Expression" left="//@application/@implementa-
tion.0/@uses.0/@hasEvents.0/@hasBody/@hasStatements.1/@hasArguments.0" right="//@application/@imple-
mentation.0/@uses.0/@hasEvents.0/@hasBody/@hasStatements.1/@hasArguments.1" operator="MethodCall">
              <hasArguments xsi:type="nesC:Literal" literal=""/>
              <hasArguments xsi:type="nesC:MethodRef" method="//@application/@implementa-
tion.0/@uses.2/@hasCommands.0"/>
```

```
            </hasStatements>
          </hasBody>
          <returnType xsi:type="nesC:InternalTypes" type="void"/>
        </hasEvents>
      </uses>
      <uses Name="Leds" Type="Normal">
        <hasCommands name="led0Toggle">
          <returnType xsi:type="nesC:InternalTypes" type="void"/>
        </hasCommands>
      </uses>
      <uses Name="SplitControl" Type="Normal" Identifier="AMControl">
        <hasEvents name="startDone">
          <hasBody>
            <hasStatements xsi:type="nesC:If">
              <hasBody>
                <hasStatements xsi:type="nesC:Expression" left="//@application/@implementa-
tion.0/@uses.2/@hasEvents.0/@hasBody/@hasStatements.0/@hasBody/@hasStatements.0/@hasArguments.0"
right="//@application/@implementation.0/@uses.2/@hasEvents.0/@hasBody/@hasStatements.0/@hasBody/@has-
Statements.0/@hasArguments.1" operator="MethodCall">
                  <hasArguments xsi:type="nesC:Literal" literal=""/>
                  <hasArguments xsi:type="nesC:MethodRef" method="//@application/@implementa-
tion.0/@uses.2/@hasCommands.0"/>
                </hasStatements>
              </hasBody>
              <condition left="//@application/@implementation.0/@uses.2/@hasEvents.0/@hasBody/@has-
Statements.0/@condition/@hasArguments.0" right="//@application/@implementa-
tion.0/@uses.2/@hasEvents.0/@hasBody/@hasStatements.0/@condition/@hasArguments.1" operator="NotE-
qual">
                <hasArguments xsi:type="nesC:Literal" literal="error"/>
                <hasArguments xsi:type="nesC:Literal" literal="SUCCESS"/>
              </condition>
              <hasElseBody>
                <hasStatements xsi:type="nesC:Expression" left="//@application/@implementa-
tion.0/@uses.2/@hasEvents.0/@hasBody/@hasStatements.0/@hasElseBody/@hasStatements.0/@hasArguments.0"
right="//@application/@implementation.0/@uses.2/@hasEvents.0/@hasBody/@hasStatements.0/@hasElse-
Body/@hasStatements.0/@hasArguments.1" operator="MethodCall">
                  <hasArguments xsi:type="nesC:Literal" literal=""/>
                  <hasArguments xsi:type="nesC:MethodRef" method="//@application/@implementa-
tion.0/@uses.4/@hasCommands.0" hasParameters="//@application/@implementation.0/@hasHeaders.0/@hasVar-
iables.0/@hasVariables.1"/>
                </hasStatements>
              </hasElseBody>
            </hasStatements>
          </hasBody>
          <hasParameters xsi:type="nesC:Struct" name="error" struct_name="error_t"/>
          <returnType xsi:type="nesC:InternalTypes" type="void"/>
        </hasEvents>
        <hasEvents name="stopDone">
          <hasParameters xsi:type="nesC:Struct" name="error" struct_name="error_t"/>
          <returnType xsi:type="nesC:InternalTypes" type="void"/>
        </hasEvents>
        <hasCommands name="start">
          <returnType xsi:type="nesC:InternalTypes" type="void"/>
        </hasCommands>
      </uses>
      <uses Name="AMSend" Type="Normal">
        <hasEvents name="sendDone">
          <hasBody>
            <hasStatements xsi:type="nesC:If">
              <hasBody>
                <hasStatements xsi:type="nesC:Expression" left="//@application/@implementa-
tion.0/@hasVariables.1" right="//@application/@implementation.0/@uses.3/@hasEvents.0/@hasBody/@has-
Statements.0/@hasBody/@hasStatements.0/@hasArguments.0" operator="Assign">
                  <hasArguments xsi:type="nesC:Literal" literal="FALSE"/>
                </hasStatements>
                <hasStatements xsi:type="nesC:Expression" left="//@application/@implementa-
tion.0/@uses.3/@hasEvents.0/@hasBody/@hasStatements.0/@hasBody/@hasStatements.1/@hasArguments.0"
right="//@application/@implementation.0/@uses.3/@hasEvents.0/@hasBody/@hasStatements.0/@hasBody/@has-
Statements.1/@hasArguments.1" operator="MethodCall">
                  <hasArguments xsi:type="nesC:Literal" literal=""/>
                  <hasArguments xsi:type="nesC:MethodRef" method="//@application/@implementa-
tion.0/@uses.1/@hasCommands.0"/>
                </hasStatements>
              </hasBody>
              <condition left="//@application/@implementation.0/@uses.3/@hasEvents.0/@hasBody/@has-
Statements.0/@condition/@hasArguments.0" right="//@application/@implementa-
tion.0/@uses.3/@hasEvents.0/@hasBody/@hasStatements.0/@condition/@hasArguments.1" operator="Equal">
                <hasArguments xsi:type="nesC:Variable" name="AM_pkt">
                  <type xsi:type="nesC:InternalTypes" type="reference"/>
                </hasArguments>
                <hasArguments xsi:type="nesC:Literal" literal="msg"/>
              </condition>
            </hasStatements>
          </hasBody>
```

```
          <hasParameters xsi:type="nesC:Struct" name="msg" struct_name="message_t">
            <type xsi:type="nesC:InternalTypes" type="pointer"/>
          </hasParameters>
          <hasParameters xsi:type="nesC:Struct" name="error" struct_name="error_t"/>
          <returnType xsi:type="nesC:InternalTypes" type="void"/>
        </hasEvents>
        <hasCommands name="send">
          <returnType xsi:type="nesC:InternalTypes" type="bool"/>
        </hasCommands>
        <hasCommands name="getPayload"/>
      </uses>
      <uses Name="Timer" Type="Normal" Identifier="Timer" hasParameters="//@application/@implementa-
tion.0/@hasHeaders.0/@includes.0/@hasVariables.0">
        <hasEvents name="fired">
          <hasBody>
            <hasVariables name="i">
              <InitExpr left="//@application/@implementation.0/@uses.4/@hasEvents.0/@hasBody/@hasVar-
iables.0/@InitExpr/@hasArguments.0" right="//@application/@implementa-
tion.0/@uses.4/@hasEvents.0/@hasBody/@hasVariables.0/@InitExpr/@hasArguments.1" operator="Init">
                <hasArguments xsi:type="nesC:Literal" literal=""/>
                <hasArguments xsi:type="nesC:Literal" literal="0"/>
              </InitExpr>
              <type xsi:type="nesC:InternalTypes" type="int"/>
            </hasVariables>
            <hasVariables xsi:type="nesC:Struct" name="pkt" struct_name="WirellessMessage">
              <InitExpr right="//@application/@implementa-
tion.0/@uses.4/@hasEvents.0/@hasBody/@hasVariables.1/@InitExpr/@hasArguments.0" operator="Init">
                <hasArguments xsi:type="nesC:Expression" left="//@application/@implementa-
tion.0/@uses.4/@hasEvents.0/@hasBody/@hasVariables.1/@InitExpr/@hasArguments.0/@hasArguments.0"
right="//@application/@implementation.0/@uses.4/@hasEvents.0/@hasBody/@hasVariables.1/@InitExpr/@ha-
sArguments.0/@hasArguments.1" operator="Cast">
                  <hasArguments xsi:type="nesC:Struct" name="" struct_name="WirellessMessage">
                    <type xsi:type="nesC:InternalTypes" type="pointer"/>
                  </hasArguments>
                  <hasArguments xsi:type="nesC:Expression" right="//@application/@implementa-
tion.0/@uses.4/@hasEvents.0/@hasBody/@hasVariables.1/@InitExpr/@hasArguments.0/@hasArguments.1/@ha-
sArguments.0" operator="MethodCall">
                    <hasArguments xsi:type="nesC:MethodRef" method="//@application/@implementa-
tion.0/@uses.3/@hasCommands.1" hasParameters="//@application/@implementa-
tion.0/@uses.4/@hasEvents.0/@hasBody/@hasVariables.1/@InitExpr/@hasArguments.0/@hasArguments.1/@ha-
sArguments.1 //@application/@implementation.0/@uses.4/@hasEvents.0/@hasBody/@hasVariables.1/@In-
itExpr/@hasArguments.0/@hasArguments.1/@hasArguments.2"/>
                    <hasArguments xsi:type="nesC:Variable" name="AM_pkt">
                      <type xsi:type="nesC:InternalTypes" type="reference"/>
                    </hasArguments>
                    <hasArguments xsi:type="nesC:Variable" name="sizeof(WirellessMessage)"/>
                  </hasArguments>
                </hasArguments>
              </InitExpr>
              <type xsi:type="nesC:InternalTypes" type="pointer"/>
            </hasVariables>
            <hasStatements xsi:type="nesC:If">
              <hasBody>
                <hasStatements xsi:type="nesC:Expression" left="//@application/@implementa-
tion.0/@uses.4/@hasEvents.0/@hasBody/@hasStatements.0/@hasBody/@hasStatements.0/@hasArguments.0" op-
erator="returnValue">
                  <hasArguments xsi:type="nesC:Literal" literal=""/>
                </hasStatements>
              </hasBody>
              <condition left="//@application/@implementation.0/@uses.4/@hasEvents.0/@hasBody/@has-
Statements.0/@condition/@hasArguments.0" right="//@application/@implementa-
tion.0/@uses.4/@hasEvents.0/@hasBody/@hasStatements.0/@condition/@hasArguments.1" operator="Equal">
                <hasArguments xsi:type="nesC:Literal" literal="pkt"/>
                <hasArguments xsi:type="nesC:Literal" literal="NULL"/>
              </condition>
            </hasStatements>
            <hasStatements xsi:type="nesC:For">
              <hasBody>
                <hasStatements xsi:type="nesC:Expression" left="//@application/@implementa-
tion.0/@uses.4/@hasEvents.0/@hasBody/@hasStatements.1/@hasBody/@hasStatements.0/@hasArguments.0"
right="//@application/@implementation.0/@uses.4/@hasEvents.0/@hasBody/@hasStatements.1/@hasBody/@has-
Statements.0/@hasArguments.1" operator="Assign">
                  <hasArguments xsi:type="nesC:Literal" literal="pkt->value[i]"/>
                  <hasArguments xsi:type="nesC:Literal" literal="170"/>
                </hasStatements>
              </hasBody>
              <condition left="//@application/@implementa-
tion.0/@uses.4/@hasEvents.0/@hasBody/@hasVariables.0" right="//@application/@implementa-
tion.0/@hasHeaders.0/@hasVariables.0/@hasVariables.2" operator="Lesser"/>
              <int left="//@application/@implementation.0/@uses.4/@hasEvents.0/@hasBody/@hasVaria-
bles.0" right="//@application/@implementation.0/@uses.4/@hasEvents.0/@hasBody/@hasVariables.0/@In-
itExpr/@hasArguments.1" operator="Assign"/>
```

```
            <step left="//@application/@implementation.0/@uses.4/@hasEvents.0/@hasBody/@hasVaria-
bles.0" right="//@application/@implementation.0/@uses.4/@hasEvents.0/@hasBody/@hasState-
ments.1/@step/@hasArguments.0" operator="Plus">
              <hasArguments xsi:type="nesC:Expression" operator="Plus"/>
            </step>
          </hasStatements>
          <hasStatements xsi:type="nesC:If">
            <hasBody>
              <hasStatements xsi:type="nesC:Expression" left="//@application/@implementa-
tion.0/@hasVariables.1" right="//@application/@implementation.0/@uses.4/@hasEvents.0/@hasBody/@has-
Statements.2/@hasBody/@hasStatements.0/@hasArguments.0" operator="Assign">
                <hasArguments xsi:type="nesC:Literal" literal="TRUE"/>
              </hasStatements>
            </hasBody>
            <condition left="//@application/@implementation.0/@uses.4/@hasEvents.0/@hasBody/@has-
Statements.2/@condition/@hasArguments.0" right="//@application/@implementa-
tion.0/@uses.2/@hasEvents.0/@hasBody/@hasStatements.0/@condition/@hasArguments.1" operator="Equal">
              <hasArguments xsi:type="nesC:Expression" right="//@application/@implementa-
tion.0/@uses.4/@hasEvents.0/@hasBody/@hasStatements.2/@condition/@hasArguments.0/@hasArguments.3" op-
erator="MethodCall">
                <hasArguments xsi:type="nesC:Variable" name="AM_BROADCAST_ADDR"/>
                <hasArguments xsi:type="nesC:Variable" name="AM_pkt">
                  <type xsi:type="nesC:InternalTypes" type="reference"/>
                </hasArguments>
                <hasArguments xsi:type="nesC:Variable" name="sizeof(WirellessMessage)"/>
                <hasArguments xsi:type="nesC:MethodRef" method="//@application/@implementa-
tion.0/@uses.3/@hasCommands.0" hasParameters="//@application/@implementa-
tion.0/@uses.4/@hasEvents.0/@hasBody/@hasStatements.2/@condition/@hasArguments.0/@hasArguments.0
//@application/@implementation.0/@uses.4/@hasEvents.0/@hasBody/@hasStatements.2/@condition/@hasArgu-
ments.0/@hasArguments.1 //@application/@implementation.0/@uses.4/@hasEvents.0/@hasBody/@hasState-
ments.2/@condition/@hasArguments.0/@hasArguments.2"/>
              </hasArguments>
            </condition>
          </hasStatements>
        </hasBody>
        <returnType xsi:type="nesC:InternalTypes" type="void"/>
      </hasEvents>
      <hasCommands name="startPeriodic">
        <returnType xsi:type="nesC:InternalTypes" type="void"/>
      </hasCommands>
    </uses>
    <hasHeaders name="lib">
      <hasVariables xsi:type="nesC:Enums" name="" enum_name="">
        <hasVariables name="WIRELESS_MSG_ID">
          <InitExpr right="//@application/@implementation.0/@hasHeaders.0/@hasVariables.0/@hasVari-
ables.0/@InitExpr/@hasArguments.0" operator="Assign">
            <hasArguments xsi:type="nesC:Literal" literal="99"/>
          </InitExpr>
        </hasVariables>
        <hasVariables name="TimePeriod">
          <InitExpr right="//@application/@implementation.0/@hasHeaders.0/@hasVariables.0/@hasVari-
ables.1/@InitExpr/@hasArguments.0" operator="Assign">
            <hasArguments xsi:type="nesC:Literal" literal="100"/>
          </InitExpr>
        </hasVariables>
        <hasVariables name="NumberOfBytes">
          <InitExpr right="//@application/@implementation.0/@hasHeaders.0/@hasVariables.0/@hasVari-
ables.2/@InitExpr/@hasArguments.0" operator="Assign">
            <hasArguments xsi:type="nesC:Literal" literal="28"/>
          </InitExpr>
        </hasVariables>
        <hasVariables name="MAX_PAYLOAD">
          <InitExpr right="//@application/@implementation.0/@hasHeaders.0/@hasVariables.0/@hasVari-
ables.3/@InitExpr/@hasArguments.0" operator="Assign">
            <hasArguments xsi:type="nesC:Literal" literal="28"/>
          </InitExpr>
        </hasVariables>
      </hasVariables>
      <hasVariables xsi:type="nesC:Struct" name="WirellessMessage" typedef="true"
struct_name="WirellessMessage">
        <type xsi:type="nesC:ExternalTypes" type="nx_struct"/>
        <hasVariables xsi:type="nesC:Array" name="value" Dim="//@application/@implementa-
tion.0/@hasHeaders.0/@hasVariables.0/@hasVariables.2">
          <type xsi:type="nesC:ExternalTypes" type="nx_uint8_t"/>
        </hasVariables>
      </hasVariables>
      <includes name="Timer">
        <hasVariables name="TMilli"/>
      </includes>
    </hasHeaders>
    <hasVariables xsi:type="nesC:Struct" name="AM_pkt" struct_name="message_t"/>
    <hasVariables name="AMlocked">
      <type xsi:type="nesC:InternalTypes" type="bool"/>
    </hasVariables>
```

```
    </implementation>
    <implementation name="MainC" type="Normal"/>
    <implementation name="LedsC" type="Normal"/>
    <implementation name="ActiveMessageC" type="Normal" Identifier="Wireless"/>
    <implementation name="AMSenderC" Identifier="AMSend" hasParameters="//@application/@implementa-
tion.0/@hasHeaders.0/@hasVariables.0/@hasVariables.0"/>
    <implementation name="TimerMilliC" Identifier="Timer"/>
  </application>
</nesC:Model>
```

“SendMsgXbytes_XMI.xmi” – Instanciação XML

```
<?xml version="1.0" encoding="ASCII"?>
<xmi:Root                    xmi:version="2.0"                    xmlns:xmi="http://www.omg.org/XMI"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" name="nesC:Model">
  <children xsi:type="xmi:Attribute" name="name" value="Network Embedded System C"/>
  <children xsi:type="xmi:Attribute" name="version" value="2.0"/>
  <children xsi:type="xmi:Element" name="nesC:Configuration">
    <children xsi:type="xmi:Attribute" name="name" value="SendMsgXbytesAppC"/>
    <children xsi:type="xmi:Element" name="nesC:Module">
      <children xsi:type="xmi:Attribute" name="name" value="SendMsgXbytesC"/>
      <children xsi:type="xmi:Attribute" name="identifier" value="App"/>
      <children xsi:type="xmi:Element" name="nesC:Interface">
        <children xsi:type="xmi:Attribute" name="name" value="Boot"/>
        <children xsi:type="xmi:Element" name="nesC:Event">
          <children xsi:type="xmi:Attribute" name="name" value="booted"/>
          <children xsi:type="xmi:Attribute" name="nesC:InternalTypes" value="void"/>
          <children xsi:type="xmi:Element" name="nesC:Body">
            <children xsi:type="xmi:Element" name="nesC:Expression">
              <children xsi:type="xmi:Attribute" name="operator" value="Assign"/>
              <children xsi:type="xmi:Element" name="nesC:Literal">
                <children xsi:type="xmi:Attribute" name="literal" value="FALSE"/>
              </children>
            </children>
            <children xsi:type="xmi:Element" name="nesC:Expression">
              <children xsi:type="xmi:Attribute" name="operator" value="MethodCall"/>
              <children xsi:type="xmi:Element" name="nesC:Literal">
                <children xsi:type="xmi:Attribute" name="literal" value=""/>
              </children>
              <children xsi:type="xmi:Element" name="nesC:MethodRef"/>
            </children>
          </children>
        </children>
      </children>
      <children xsi:type="xmi:Element" name="nesC:Interface">
        <children xsi:type="xmi:Attribute" name="name" value="Leds"/>
      </children>
      <children xsi:type="xmi:Element" name="nesC:Interface">
        <children xsi:type="xmi:Attribute" name="name" value="SplitControl"/>
        <children xsi:type="xmi:Attribute" name="identifier" value="AMControl"/>
        <children xsi:type="xmi:Element" name="nesC:Event">
          <children xsi:type="xmi:Attribute" name="name" value="startDone"/>
          <children xsi:type="xmi:Attribute" name="nesC:InternalTypes" value="void"/>
          <children xsi:type="xmi:Element" name="nesC:Struct">
            <children xsi:type="xmi:Attribute" name="name" value="error"/>
            <children xsi:type="xmi:Attribute" name="struct_name" value="error_t"/>
          </children>
          <children xsi:type="xmi:Element" name="nesC:Body">
            <children xsi:type="xmi:Element" name="nesC:If">
              <children xsi:type="xmi:Element" name="nesC:Expression">
                <children xsi:type="xmi:Attribute" name="operator" value="NotEqual"/>
                <children xsi:type="xmi:Element" name="nesC:Literal">
                  <children xsi:type="xmi:Attribute" name="literal" value="error"/>
                </children>
              </children>
              <children xsi:type="xmi:Element" name="nesC:Body">
                <children xsi:type="xmi:Element" name="nesC:Expression">
                  <children xsi:type="xmi:Attribute" name="operator" value="MethodCall"/>
                  <children xsi:type="xmi:Element" name="nesC:Literal">
                    <children xsi:type="xmi:Attribute" name="literal" value=""/>
                  </children>
                  <children xsi:type="xmi:Element" name="nesC:MethodRef">
                    <children xsi:type="xmi:Element" name="nesC:Command">
                      <children xsi:type="xmi:Attribute" name="name" value="start"/>
                      <children xsi:type="xmi:Attribute" name="nesC:InternalTypes" value="void"/>
                    </children>
                  </children>
                </children>
              </children>
              <children xsi:type="xmi:Element" name="nesC:Body">
                <children xsi:type="xmi:Element" name="nesC:Expression">
                  <children xsi:type="xmi:Attribute" name="operator" value="MethodCall"/>
```

```
            <children xsi:type="xmi:Element" name="nesC:Literal">
              <children xsi:type="xmi:Attribute" name="literal" value=""/>
            </children>
            <children xsi:type="xmi:Element" name="nesC:MethodRef">
              <children xsi:type="xmi:Element" name="nesC:Command">
                <children xsi:type="xmi:Attribute" name="name" value="startPeriodic"/>
                <children xsi:type="xmi:Attribute" name="nesC:InternalTypes" value="void"/>
              </children>
            </children>
          </children>
        </children>
      </children>
    </children>
  </children>
  <children xsi:type="xmi:Element" name="nesC:Event">
    <children xsi:type="xmi:Attribute" name="name" value="stopDone"/>
    <children xsi:type="xmi:Attribute" name="nesC:InternalTypes" value="void"/>
    <children xsi:type="xmi:Element" name="nesC:Struct">
      <children xsi:type="xmi:Attribute" name="name" value="error"/>
      <children xsi:type="xmi:Attribute" name="struct_name" value="error_t"/>
    </children>
  </children>
</children>
<children xsi:type="xmi:Element" name="nesC:Interface">
  <children xsi:type="xmi:Attribute" name="name" value="AMSend"/>
  <children xsi:type="xmi:Element" name="nesC:Event">
    <children xsi:type="xmi:Attribute" name="name" value="sendDone"/>
    <children xsi:type="xmi:Attribute" name="nesC:InternalTypes" value="void"/>
    <children xsi:type="xmi:Element" name="nesC:Struct">
      <children xsi:type="xmi:Attribute" name="nesC:InternalTypes" value="pointer"/>
      <children xsi:type="xmi:Attribute" name="name" value="msg"/>
      <children xsi:type="xmi:Attribute" name="struct_name" value="message_t"/>
    </children>
    <children xsi:type="xmi:Element" name="nesC:Struct">
      <children xsi:type="xmi:Attribute" name="name" value="error"/>
      <children xsi:type="xmi:Attribute" name="struct_name" value="error_t"/>
    </children>
    <children xsi:type="xmi:Element" name="nesC:Body">
      <children xsi:type="xmi:Element" name="nesC:If">
        <children xsi:type="xmi:Element" name="nesC:Expression">
          <children xsi:type="xmi:Attribute" name="operator" value="Equal"/>
          <children xsi:type="xmi:Element" name="nesC:Variable">
            <children xsi:type="xmi:Attribute" name="name" value="AM_pkt"/>
            <children xsi:type="xmi:Attribute" name="nesC:InternalTypes" value="reference"/>
          </children>
          <children xsi:type="xmi:Element" name="nesC:Literal">
            <children xsi:type="xmi:Attribute" name="literal" value="msg"/>
          </children>
        </children>
        <children xsi:type="xmi:Element" name="nesC:Body">
          <children xsi:type="xmi:Element" name="nesC:Expression">
            <children xsi:type="xmi:Attribute" name="operator" value="Assign"/>
            <children xsi:type="xmi:Element" name="nesC:Literal">
              <children xsi:type="xmi:Attribute" name="literal" value="FALSE"/>
            </children>
          </children>
          <children xsi:type="xmi:Element" name="nesC:Expression">
            <children xsi:type="xmi:Attribute" name="operator" value="MethodCall"/>
            <children xsi:type="xmi:Element" name="nesC:Literal">
              <children xsi:type="xmi:Attribute" name="literal" value=""/>
            </children>
            <children xsi:type="xmi:Element" name="nesC:MethodRef">
              <children xsi:type="xmi:Element" name="nesC:Command">
                <children xsi:type="xmi:Attribute" name="name" value="led0Toggle"/>
                <children xsi:type="xmi:Attribute" name="nesC:InternalTypes" value="void"/>
              </children>
            </children>
          </children>
        </children>
      </children>
    </children>
  </children>
</children>
<children xsi:type="xmi:Element" name="nesC:Interface">
  <children xsi:type="xmi:Attribute" name="name" value="Timer"/>
  <children xsi:type="xmi:Attribute" name="identifier" value="Timer"/>
  <children xsi:type="xmi:Element" name="nesC:Event">
    <children xsi:type="xmi:Attribute" name="name" value="fired"/>
    <children xsi:type="xmi:Attribute" name="nesC:InternalTypes" value="void"/>
    <children xsi:type="xmi:Element" name="nesC:Body">
      <children xsi:type="xmi:Element" name="nesC:If">
        <children xsi:type="xmi:Element" name="nesC:Expression">
          <children xsi:type="xmi:Attribute" name="operator" value="Equal"/>
          <children xsi:type="xmi:Element" name="nesC:Literal">
```

```
                    <children xsi:type="xmi:Attribute" name="literal" value="pkt"/>
                  </children>
                  <children xsi:type="xmi:Element" name="nesC:Literal">
                    <children xsi:type="xmi:Attribute" name="literal" value="NULL"/>
                  </children>
                </children>
                <children xsi:type="xmi:Element" name="nesC:Body">
                  <children xsi:type="xmi:Element" name="nesC:Expression">
                    <children xsi:type="xmi:Attribute" name="operator" value="returnValue"/>
                    <children xsi:type="xmi:Element" name="nesC:Literal">
                      <children xsi:type="xmi:Attribute" name="literal" value=""/>
                    </children>
                  </children>
                </children>
              </children>
              <children xsi:type="xmi:Element" name="nesC:For">
                <children xsi:type="xmi:Element" name="nesC:Expression">
                  <children xsi:type="xmi:Attribute" name="operator" value="Assign"/>
                  <children xsi:type="xmi:Element" name="nesC:Literal">
                    <children xsi:type="xmi:Attribute" name="literal" value="0"/>
                  </children>
                </children>
                <children xsi:type="xmi:Element" name="nesC:Expression">
                  <children xsi:type="xmi:Attribute" name="operator" value="Lesser"/>
                </children>
                <children xsi:type="xmi:Element" name="nesC:Expression">
                  <children xsi:type="xmi:Attribute" name="operator" value="Plus"/>
                  <children xsi:type="xmi:Element" name="nesC:Expression">
                    <children xsi:type="xmi:Attribute" name="operator" value="Plus"/>
                  </children>
                </children>
                <children xsi:type="xmi:Element" name="nesC:Body">
                  <children xsi:type="xmi:Element" name="nesC:Expression">
                    <children xsi:type="xmi:Attribute" name="operator" value="Assign"/>
                    <children xsi:type="xmi:Element" name="nesC:Literal">
                      <children xsi:type="xmi:Attribute" name="literal" value="pkt->value[i]"/>
                    </children>
                    <children xsi:type="xmi:Element" name="nesC:Literal">
                      <children xsi:type="xmi:Attribute" name="literal" value="170"/>
                    </children>
                  </children>
                </children>
              </children>
              <children xsi:type="xmi:Element" name="nesC:If">
                <children xsi:type="xmi:Element" name="nesC:Expression">
                  <children xsi:type="xmi:Attribute" name="operator" value="Equal"/>
                  <children xsi:type="xmi:Element" name="nesC:Expression">
                    <children xsi:type="xmi:Attribute" name="operator" value="MethodCall"/>
                    <children xsi:type="xmi:Element" name="nesC:MethodRef">
                      <children xsi:type="xmi:Element" name="nesC:Command">
                        <children xsi:type="xmi:Attribute" name="name" value="send"/>
                        <children xsi:type="xmi:Attribute" name="nesC:InternalTypes" value="bool"/>
                      </children>
                      <children xsi:type="xmi:Element" name="nesC:Variable">
                        <children xsi:type="xmi:Attribute" name="name" value="AM_BROADCAST_ADDR"/>
                      </children>
                      <children xsi:type="xmi:Element" name="nesC:Variable">
                        <children xsi:type="xmi:Attribute" name="name" value="AM_pkt"/>
                        <children  xsi:type="xmi:Attribute"  name="nesC:InternalTypes"  value="refer-
ence"/>
                      </children>
                      <children xsi:type="xmi:Element" name="nesC:Variable">
                        <children  xsi:type="xmi:Attribute"  name="name"  value="sizeof(WirellessMes-
sage)"/>
                      </children>
                    </children>
                  </children>
                  <children xsi:type="xmi:Element" name="nesC:Literal">
                    <children xsi:type="xmi:Attribute" name="literal" value="SUCCESS"/>
                  </children>
                </children>
                <children xsi:type="xmi:Element" name="nesC:Body">
                  <children xsi:type="xmi:Element" name="nesC:Expression">
                    <children xsi:type="xmi:Attribute" name="operator" value="Assign"/>
                    <children xsi:type="xmi:Element" name="nesC:Literal">
                      <children xsi:type="xmi:Attribute" name="literal" value="TRUE"/>
                    </children>
                    <children xsi:type="xmi:Element" name="nesC:Variable">
                      <children xsi:type="xmi:Attribute" name="name" value="AMlocked"/>
                      <children xsi:type="xmi:Attribute" name="nesC:InternalTypes" value="bool"/>
                    </children>
                  </children>
                </children>
              </children>
```

```
            <children xsi:type="xmi:Element" name="nesC:Variable">
              <children xsi:type="xmi:Attribute" name="name" value="i"/>
              <children xsi:type="xmi:Attribute" name="nesC:InternalTypes" value="int"/>
              <children xsi:type="xmi:Element" name="nesC:Expression">
                <children xsi:type="xmi:Attribute" name="operator" value="Init"/>
                <children xsi:type="xmi:Element" name="nesC:Literal">
                  <children xsi:type="xmi:Attribute" name="literal" value=""/>
                </children>
              </children>
            </children>
            <children xsi:type="xmi:Element" name="nesC:Struct">
              <children xsi:type="xmi:Attribute" name="nesC:InternalTypes" value="pointer"/>
              <children xsi:type="xmi:Attribute" name="name" value="pkt"/>
              <children xsi:type="xmi:Attribute" name="struct_name" value="WirellessMessage"/>
              <children xsi:type="xmi:Element" name="nesC:Expression">
                <children xsi:type="xmi:Attribute" name="operator" value="Init"/>
                <children xsi:type="xmi:Element" name="nesC:Expression">
                  <children xsi:type="xmi:Attribute" name="operator" value="Cast"/>
                  <children xsi:type="xmi:Element" name="nesC:Struct">
                    <children xsi:type="xmi:Attribute" name="nesC:InternalTypes" value="pointer"/>
                    <children xsi:type="xmi:Attribute" name="name" value=""/>
                    <children xsi:type="xmi:Attribute" name="struct_name" value="WirellessMessage"/>
                  </children>
                  <children xsi:type="xmi:Element" name="nesC:Expression">
                    <children xsi:type="xmi:Attribute" name="operator" value="MethodCall"/>
                    <children xsi:type="xmi:Element" name="nesC:MethodRef">
                      <children xsi:type="xmi:Element" name="nesC:Command">
                        <children xsi:type="xmi:Attribute" name="name" value="getPayload"/>
                      </children>
                      <children xsi:type="xmi:Element" name="nesC:Variable">
                        <children xsi:type="xmi:Attribute" name="name" value="AM_pkt"/>
                        <children  xsi:type="xmi:Attribute"  name="nesC:InternalTypes"  value="refer-
ence"/>
                      </children>
                      <children xsi:type="xmi:Element" name="nesC:Variable">
                        <children  xsi:type="xmi:Attribute"  name="name"  value="sizeof(WirellessMes-
sage)"/>
                      </children>
                    </children>
                  </children>
                </children>
              </children>
            </children>
          </children>
        </children>
      </children>
      <children xsi:type="xmi:Element" name="nesC:Struct">
        <children xsi:type="xmi:Attribute" name="name" value="AM_pkt"/>
        <children xsi:type="xmi:Attribute" name="struct_name" value="message_t"/>
      </children>
      <children xsi:type="xmi:Element" name="nesC:Header">
        <children xsi:type="xmi:Attribute" name="name" value="lib"/>
        <children xsi:type="xmi:Element" name="nesC:Enum">
          <children xsi:type="xmi:Attribute" name="name" value=""/>
          <children xsi:type="xmi:Attribute" name="enum_name" value=""/>
          <children xsi:type="xmi:Element" name="nesC:Variable">
            <children xsi:type="xmi:Attribute" name="name" value="WIRELESS_MSG_ID"/>
            <children xsi:type="xmi:Element" name="nesC:Expression">
              <children xsi:type="xmi:Attribute" name="operator" value="Assign"/>
              <children xsi:type="xmi:Element" name="nesC:Literal">
                <children xsi:type="xmi:Attribute" name="literal" value="99"/>
              </children>
            </children>
          </children>
          <children xsi:type="xmi:Element" name="nesC:Variable">
            <children xsi:type="xmi:Attribute" name="name" value="TimePeriod"/>
            <children xsi:type="xmi:Element" name="nesC:Expression">
              <children xsi:type="xmi:Attribute" name="operator" value="Assign"/>
              <children xsi:type="xmi:Element" name="nesC:Literal">
                <children xsi:type="xmi:Attribute" name="literal" value="100"/>
              </children>
            </children>
          </children>
          <children xsi:type="xmi:Element" name="nesC:Variable">
            <children xsi:type="xmi:Attribute" name="name" value="MAX_PAYLOAD"/>
            <children xsi:type="xmi:Element" name="nesC:Expression">
              <children xsi:type="xmi:Attribute" name="operator" value="Assign"/>
              <children xsi:type="xmi:Element" name="nesC:Literal">
                <children xsi:type="xmi:Attribute" name="literal" value="28"/>
              </children>
            </children>
          </children>
        </children>
        <children xsi:type="xmi:Element" name="nesC:Struct">
```

```
          <children xsi:type="xmi:Attribute" name="typedef" value="true"/>
          <children xsi:type="xmi:Attribute" name="nesC:ExternalTypes" value="nx_struct"/>
          <children xsi:type="xmi:Attribute" name="name" value="WirellessMessage"/>
          <children xsi:type="xmi:Attribute" name="struct_name" value="WirellessMessage"/>
          <children xsi:type="xmi:Element" name="nesC:Array">
            <children xsi:type="xmi:Attribute" name="nesC:ExternalTypes" value="nx_uint8_t"/>
            <children xsi:type="xmi:Attribute" name="name" value="value"/>
            <children xsi:type="xmi:Element" name="nesC:Variable">
              <children xsi:type="xmi:Attribute" name="name" value="NumberOfBytes"/>
              <children xsi:type="xmi:Element" name="nesC:Expression">
                <children xsi:type="xmi:Attribute" name="operator" value="Assign"/>
                <children xsi:type="xmi:Element" name="nesC:Literal">
                  <children xsi:type="xmi:Attribute" name="literal" value="28"/>
                </children>
              </children>
            </children>
          </children>
        </children>
        <children xsi:type="xmi:Element" name="nesC:Header">
          <children xsi:type="xmi:Attribute" name="name" value="Timer"/>
          <children xsi:type="xmi:Element" name="nesC:Variable">
            <children xsi:type="xmi:Attribute" name="name" value="TMilli"/>
          </children>
        </children>
      </children>
    </children>
    <children xsi:type="xmi:Element" name="nesC:Component">
      <children xsi:type="xmi:Attribute" name="name" value="MainC"/>
    </children>
    <children xsi:type="xmi:Element" name="nesC:Component">
      <children xsi:type="xmi:Attribute" name="name" value="LedsC"/>
    </children>
    <children xsi:type="xmi:Element" name="nesC:Component">
      <children xsi:type="xmi:Attribute" name="name" value="ActiveMessageC"/>
    </children>
    <children xsi:type="xmi:Element" name="nesC:Component">
      <children xsi:type="xmi:Attribute" name="name" value="AMSenderC"/>
      <children xsi:type="xmi:Attribute" name="type" value="Generic"/>
    </children>
    <children xsi:type="xmi:Element" name="nesC:Component">
      <children xsi:type="xmi:Attribute" name="name" value="TimerMilliC"/>
      <children xsi:type="xmi:Attribute" name="type" value="Generic"/>
    </children>
    <children xsi:type="xmi:Element" name="nesC:Wiring">
      <children xsi:type="xmi:Attribute" name="user" value="App.Boot"/>
      <children xsi:type="xmi:Attribute" name="provider" value="MainC"/>
    </children>
    <children xsi:type="xmi:Element" name="nesC:Wiring">
      <children xsi:type="xmi:Attribute" name="user" value="App.Leds"/>
      <children xsi:type="xmi:Attribute" name="provider" value="LedsC"/>
    </children>
    <children xsi:type="xmi:Element" name="nesC:Wiring">
      <children xsi:type="xmi:Attribute" name="user" value="App.AMControl"/>
      <children xsi:type="xmi:Attribute" name="provider" value="Wireless"/>
    </children>
    <children xsi:type="xmi:Element" name="nesC:Wiring">
      <children xsi:type="xmi:Attribute" name="user" value="App.AMSend"/>
      <children xsi:type="xmi:Attribute" name="provider" value="AMSend"/>
    </children>
    <children xsi:type="xmi:Element" name="nesC:Wiring">
      <children xsi:type="xmi:Attribute" name="user" value="App.Timer"/>
      <children xsi:type="xmi:Attribute" name="provider" value="Timer"/>
    </children>
  </children>
</xmi:Root>
```

“SendMsgXbytes_XML.xml” – Representação XML

```
<?xml version = '1.0' encoding = 'ISO-8859-1' ?>
<nesC:Model name="Network Embedded System C" version="2.0">
  <nesC:Configuration name="SendMsgXbytesAppC">
    <nesC:Module name="SendMsgXbytesC" identifier="App">
      <nesC:Interface name="Boot">
        <nesC:Event name="booted" nesC:InternalTypes="void">
          <nesC:Body>
            <nesC:Expression operator="Assign">
              <nesC:Literal literal="FALSE"/>
            </nesC:Expression>
            <nesC:Expression operator="MethodCall">
              <nesC:Literal literal=""/>
              <nesC:MethodRef/>
```

```
      </nesC:Expression>
    </nesC:Body>
  </nesC:Event>
</nesC:Interface>
<nesC:Interface name="Leds"/>
<nesC:Interface name="SplitControl" identifier="AMControl">
  <nesC:Event name="startDone" nesC:InternalTypes="void">
    <nesC:Struct name="error" struct_name="error_t"/>
    <nesC:Body>
      <nesC:If>
        <nesC:Expression operator="NotEqual">
          <nesC:Literal literal="error"/>
        </nesC:Expression>
        <nesC:Body>
          <nesC:Expression operator="MethodCall">
            <nesC:Literal literal=""/>
            <nesC:MethodRef>
              <nesC:Command name="start" nesC:InternalTypes="void"/>
            </nesC:MethodRef>
          </nesC:Expression>
        </nesC:Body>
        <nesC:Body>
          <nesC:Expression operator="MethodCall">
            <nesC:Literal literal=""/>
            <nesC:MethodRef>
              <nesC:Command name="startPeriodic" nesC:InternalTypes="void"/>
            </nesC:MethodRef>
          </nesC:Expression>
        </nesC:Body>
      </nesC:If>
    </nesC:Body>
  </nesC:Event>
  <nesC:Event name="stopDone" nesC:InternalTypes="void">
    <nesC:Struct name="error" struct_name="error_t"/>
  </nesC:Event>
</nesC:Interface>
<nesC:Interface name="AMSend">
  <nesC:Event name="sendDone" nesC:InternalTypes="void">
    <nesC:Struct nesC:InternalTypes="pointer" name="msg" struct_name="message_t"/>
    <nesC:Struct name="error" struct_name="error_t"/>
    <nesC:Body>
      <nesC:If>
        <nesC:Expression operator="Equal">
          <nesC:Variable name="AM_pkt" nesC:InternalTypes="reference"/>
          <nesC:Literal literal="msg"/>
        </nesC:Expression>
        <nesC:Body>
          <nesC:Expression operator="Assign">
            <nesC:Literal literal="FALSE"/>
          </nesC:Expression>
          <nesC:Expression operator="MethodCall">
            <nesC:Literal literal=""/>
            <nesC:MethodRef>
              <nesC:Command name="led0Toggle" nesC:InternalTypes="void"/>
            </nesC:MethodRef>
          </nesC:Expression>
        </nesC:Body>
      </nesC:If>
    </nesC:Body>
  </nesC:Event>
</nesC:Interface>
<nesC:Interface name="Timer" identifier="Timer">
  <nesC:Event name="fired" nesC:InternalTypes="void">
    <nesC:Body>
      <nesC:If>
        <nesC:Expression operator="Equal">
          <nesC:Literal literal="pkt"/>
          <nesC:Literal literal="NULL"/>
        </nesC:Expression>
        <nesC:Body>
          <nesC:Expression operator="returnValue">
            <nesC:Literal literal=""/>
          </nesC:Expression>
        </nesC:Body>
      </nesC:If>
      <nesC:For>
        <nesC:Expression operator="Assign">
          <nesC:Literal literal="0"/>
        </nesC:Expression>
        <nesC:Expression operator="Lesser"/>
        <nesC:Expression operator="Plus">
          <nesC:Expression operator="Plus"/>
        </nesC:Expression>
        <nesC:Body>
```

```
                <nesC:Expression operator="Assign">
                  <nesC:Literal literal="pkt-&gt;value[i]"/>
                  <nesC:Literal literal="170"/>
                </nesC:Expression>
              </nesC:Body>
            </nesC:For>
            <nesC:If>
              <nesC:Expression operator="Equal">
                <nesC:Expression operator="MethodCall">
                  <nesC:MethodRef>
                    <nesC:Command name="send" nesC:InternalTypes="bool"/>
                    <nesC:Variable name="AM_BROADCAST_ADDR"/>
                    <nesC:Variable name="AM_pkt" nesC:InternalTypes="reference"/>
                    <nesC:Variable name="sizeof(WirellessMessage)"/>
                  </nesC:MethodRef>
                </nesC:Expression>
                <nesC:Literal literal="SUCCESS"/>
              </nesC:Expression>
              <nesC:Body>
                <nesC:Expression operator="Assign">
                  <nesC:Literal literal="TRUE"/>
                  <nesC:Variable name="AMlocked" nesC:InternalTypes="bool"/>
                </nesC:Expression>
              </nesC:Body>
            </nesC:If>
            <nesC:Variable name="i" nesC:InternalTypes="int">
              <nesC:Expression operator="Init">
                <nesC:Literal literal=""/>
              </nesC:Expression>
            </nesC:Variable>
            <nesC:Struct nesC:InternalTypes="pointer" name="pkt" struct_name="WirellessMessage">
              <nesC:Expression operator="Init">
                <nesC:Expression operator="Cast">
                  <nesC:Struct nesC:InternalTypes="pointer" name="" struct_name="WirellessMessage"/>
                  <nesC:Expression operator="MethodCall">
                    <nesC:MethodRef>
                      <nesC:Command name="getPayload"/>
                      <nesC:Variable name="AM_pkt" nesC:InternalTypes="reference"/>
                      <nesC:Variable name="sizeof(WirellessMessage)"/>
                    </nesC:MethodRef>
                  </nesC:Expression>
                </nesC:Expression>
              </nesC:Expression>
            </nesC:Struct>
          </nesC:Body>
        </nesC:Event>
      </nesC:Interface>
      <nesC:Struct name="AM_pkt" struct_name="message_t"/>
      <nesC:Header name="lib">
        <nesC:Enum name="" enum_name="">
          <nesC:Variable name="WIRELESS_MSG_ID">
            <nesC:Expression operator="Assign">
              <nesC:Literal literal="99"/>
            </nesC:Expression>
          </nesC:Variable>
          <nesC:Variable name="TimePeriod">
            <nesC:Expression operator="Assign">
              <nesC:Literal literal="100"/>
            </nesC:Expression>
          </nesC:Variable>
          <nesC:Variable name="MAX_PAYLOAD">
            <nesC:Expression operator="Assign">
              <nesC:Literal literal="28"/>
            </nesC:Expression>
          </nesC:Variable>
        </nesC:Enum>
        <nesC:Struct typedef="true" nesC:ExternalTypes="nx_struct" name="WirellessMessage"
struct_name="WirellessMessage">
          <nesC:Array nesC:ExternalTypes="nx_uint8_t" name="value">
            <nesC:Variable name="NumberOfBytes">
              <nesC:Expression operator="Assign">
                <nesC:Literal literal="28"/>
              </nesC:Expression>
            </nesC:Variable>
          </nesC:Array>
        </nesC:Struct>
        <nesC:Header name="Timer">
          <nesC:Variable name="TMilli"/>
        </nesC:Header>
      </nesC:Header>
    </nesC:Module>
    <nesC:Component name="MainC"/>
    <nesC:Component name="LedsC"/>
    <nesC:Component name="ActiveMessageC"/>
```

```
    <nesC:Component name="AMSenderC" type="Generic"/>
    <nesC:Component name="TimerMilliC" type="Generic"/>
    <nesC:Wiring user="App.Boot" provider="MainC"/>
    <nesC:Wiring user="App.Leds" provider="LedsC"/>
    <nesC:Wiring user="App.AMControl" provider="Wireless"/>
    <nesC:Wiring user="App.AMSend" provider="AMSend"/>
    <nesC:Wiring user="App.Timer" provider="Timer"/>
  </nesC:Configuration>
</nesC:Model>
```

"SendMsgXbytes" – Código nesC Gerado

Configuração: "SendMsgXbytesAppC"

```
configuration SendMsgXbytesAppC{
}
implementation{
        components SendMsgXbytesC as App;
        components MainC;
        components LedsC;
        components ActiveMessageC as Wireless;
        components new AMSenderC(WIRELESS_MSG_ID) as AMSend;
        components new TimerMilliC() as Timer;
        App.Boot -> MainC;
        App.Leds -> LedsC;
        App.AMControl -> Wireless;
        App.AMSend -> AMSend;
        App.Timer -> Timer;
}
```

Módulo: "SendMsgXbytesC"

```
#include "lib.h"

module SendMsgXbytesC @safe() {
        uses {
                interface Boot;
                interface Leds;
                interface SplitControl as AMControl;
                interface AMSend;
                interface Timer<TMilli> as Timer;
        }
}
implementation{
        message_t  AM_pkt;
        bool AMlocked;

        event void Boot.booted(){
                AMlocked = FALSE;
                call AMControl.start();
        }
        event void AMControl.startDone(error_t  error){
                if (error != SUCCESS){
                        call AMControl.start();
                }else{
                        call Timer.startPeriodic(TimePeriod);
                }
        }
        event void AMControl.stopDone(error_t  error){}
        event void AMSend.sendDone(message_t * msg, error_t  error){
                if (&AM_pkt == msg){
                        AMlocked = FALSE;
                        call Leds.led0Toggle();
                }
        }

        event void Timer.fired(){
                int i = 0;
                WirellessMessage * pkt = (WirellessMessage* )(call AMSend.getPayload(&AM_pkt, sizeof(WirellessMessage)));

                if (pkt == NULL){
                        return;
                }
                for(i = 0 ; i < NumberOfBytes ; i ++){
                        pkt->value[i] = 170;
                }
                if (call AMSend.send(AM_BROADCAST_ADDR, &AM_pkt, sizeof(WirellessMessage)) == SUCCESS){
                        AMlocked = TRUE;
```

```
                        }
                }
}
```

Cabeçalho: “lib.h”

```
#ifndef LIB_H
#define LIB_H

#include "Timer.h"
enum {
        WIRELESS_MSG_ID = 99,
        TimePeriod = 100,
        NumberOfBytes = 28,
        MAX_PAYLOAD = 28
};

typedef nx_struct WirellessMessage {
        nx_uint8_t value[NumberOfBytes];
} WirellessMessage;

#endif /* LIB_H */
```