Dissertação de Mestrado

Inatel
Instituto Nacional de Telecomunicações

INTEGRAÇÃO DO SERVIÇO NOVAGENESIS EMBARCADO EM UM SISTEMA OPERACIONAL ORIENTADO A EVENTOS EM HARDWARE REAL

Gabriel Dias Scarpioni

Dezembro de 2017

Integração do serviço NovaGenesis embarcado em um sistema operacional orientado a eventos em hardware real.

GABRIEL DIAS SCARPIONI

Dissertação apresentada ao Instituto Nacional de Telecomunicações, como parte dos requisitos para obtenção do Título de Mestre em Telecomunicações.

ORIENTADOR: Prof. Dr. Antônio Marcos Alberti.

Santa Rita do Sapucaí
2017

Scarpioni, Gabriel Dias
S286i
Integração do serviço NovaGenesis embarcado em um sistema operacional orientado a eventos em hardware real. / Gabriel Dias Scarpioni. – Santa Rita do Sapucaí, 2017.
56 p.

Orientador: Prof. Dr. Antônio Marcos Alberti.
Dissertação de Mestrado em Telecomunicações – Instituto Nacional de Telecomunicações – INATEL.
Inclui bibliografia.

1. Internet do futuro 2. Internet das coisas 3. Programação orientada a eventos 4. Arquitetura orientada a serviço 5. Sistema operacional para embarcados 6. Mestrado em Telecomunicações. I. Alberti, Antônio Marcos. II. Instituto Nacional de Telecomunicações – INATEL. III. Título.

CDU 621.39

Ficha Elaborada pelo Bibliotecário da Instituição CRB6-2718

Inatel
Mestrado em Telecomunicações

FOLHA DE APROVAÇÃO

Dissertação defendida e aprovada em _____ /_____ /_____,
pela comissão julgadora:

Prof. Dr. Antônio Marcos Alberti
INATEL

Prof. Dr. Joel José Puga Coelho Rodrigues
INATEL

Prof. Dr. Plácido Rogério Pinheiro
UNIFOR

Coordenador do Curso de Mestrado
Prof. Dr. José Marcos Câmara Brito

“ Livros não mudam o mundo, quem muda o mundo são as pessoas. Os livros só mudam as pessoas.”

Mario Quintana

À minha família

Agradecimentos

Agradeço a Deus por ter me sustentado durante esta grande realização em minha vida, agradeço toda minha família pela paciência e compreensão, especialmente minha companheira Paulinha pelo apoio nos momentos mais difíceis dessa caminhada.

Agradeço a todos os amigos e colegas, em particular o parceiro de laboratório Vâner José Magalhães pela enorme contribuição no desenvolvimento deste trabalho, o Prof. Dr. Antônio Marcos Alberti por acreditar no potencial da contribuição e pela orientação realizada e o Prof. Dr. Edielson Prevatto Frigieri pelas orientações e conselhos na utilização do sistema operacional EventOS. À todos os professores do mestrado pelo auxilio e pelo aprendizado compartilhado.

Ao Inatel, ICT LAB e todos aqueles que contribuíram com a realização deste trabalho, Muito obrigado!

Índice

Lista de Figuras

Lista de Tabelas

Lista de Siglas

API - *Application Programming Interface*
CPU - *Central Processing Unit*
DHT - *Distributed Hash Table*
EPGS - *Embedded Proxy/Gateway Service*
FI - *Future Internet*
FIFO - *First In First Out*
FIoT - *Future Internet of things*
GIRS - *Generic Indirection Resolution Service*
HID - *Hardware Identification*
HTS - *Hash table service*
ICN - *Information-Centric Networking*
IoT - *Internet of Things*
IP - *Internet Protocol*
IPv6 - *Internet Protocol version 6*
ISR - *Interrupt Service Routine*
JSON - *JavaScript Object Notation*
kB - *KiloBytes*
LAN - *Local Area Network*
MAC - *Media Access Control*
NAT - *Network Address Translation*
NB - *Name Binding*
NG - NovaGenesis
NLN - *Natural Language Name*
NRNCS - *Name Resolution and Networking Cache Service*
OSID - *Operation System Identification*
PGCS - *Proxy/Gateway/Controller Service*
PID - *Process Identification*
PSS - *Publish/subscribe Service*
RAM - *Random Access Memory*
ROM - *Read-Only Memory*
RTOS - *Real-Time Operating System*
SCN - *Service-Centric Networking*
SDN - *Software Defined Networking*
shmID - *shared memory Identification*
SO - *Sistema Operacional*
SOA - *Service-Oriented Architecture*
SPI - *Serial Peripheral Interface*
SVN - *Self-Verifying Name*
UART - *Universal Asynchronous Receiver/Transmitter*
XaaS - *anything as a service*
XIA - *eXpressive Internet Architecture*

Publicações

ALBERTI, A.; SCARPIONI, G.; MAGALHAES, V.; CERQUEIRA, A.; RODRIGUES, J.; RIGHI, R. *Advancing NovaGenesis Architecture Towards Future Internet of Things, IEEE Internet of Things Journal,* 2017.

ALBERTI, A.; SCARPIONI, G.; MAGALHAES, V. *Rede IoT Colaborativa NovaGenesis,* in *XXXV Simpósio Brasileiro de Telecomunicações e Processamento de Sinais,* 2017.

Resumo

A Internet tornou-se uma importante ferramenta para a sociedade, aplicações e serviços são incorporados a todo momento. Conceitos como o da Internet das coisas, que pretende conectar qualquer tipo de dispositivo na Internet, aumentam a abrangência e escala da Internet para um patamar jamais imaginado em sua concepção. Porém a arquitetura atual da Internet tem sofrido para atender toda essa expansão, problemas como a escalabilidade, segurança e mobilidade fomentam o desenho de novas arquiteturas de Internet. Este trabalho analisa o desempenho de uma arquitetura futurista de Internet chamada NovaGenesis associada a aplicações de Internet das coisas. A NovaGenesis é um modelo revolucionário que defende a criação de uma nova Internet ”do zero” empregando conhecimentos atualizados na arquitetura. O serviço da NovaGenesis para Internet das coisas foi integrado a um sistema operacional orientado a eventos, e o conjunto foi embarcado em um *hardware* para realizar experimentos verificar o funcionamento do modelo proposto pela NovaGenesis.

Palavras-chave: Internet do futuro; Internet das coisas; programação orientada a eventos; arquitetura orientada a serviço; sistema operacional para embarcados.

Abstract

The Internet has become an important tool for society. Applications and services are incorporated at all times. Concepts such as the Internet of Things, which intends to connect any type of device to the Internet, increase the scope and scale of the Internet to a level never imagined in its original design. However, the current Internet architecture has suffered to meet all this expansion. Problems such as scalability, security and mobility foster the design of alternative Internet architectures. This work analyzes the performance of a futuristic Internet architecture called NovaGenesis, which is associated with the Internet of things concept. NovaGenesis proposes a revolutionary model with the creation of a new Internet from scratch, using state-of-the-art technologies. The NovaGenesis Internet of things service was integrated into an event-oriented operating system and embedded in hardware to conduct filed experiments. A proof-of-concept was experimentally validated, demonstrating NovaGenesis can be considered an alternative to the *status quo*.

Keywords: Future Internet; Internet of things; Event-driven programming; service-oriented architecture; Operating Systems for Embedded Systems

Capítulo 1

Introdução

O perfil das aplicações que fazem uso da Internet está sofrendo grandes mudanças em relação ao propósito inicial. Serviços como os de *streaming* de vídeo aumentaram substancialmente o volume de informações trafegadas pela rede [1]. A estrutura da Internet utilizada atualmente já sofreu várias alterações para suportar novos serviços ou para atender a escala de utilização da rede. Alterações como o endereço NAT e o IPv6 foram implementadas para adaptar a Internet as demandas que surgiram ao longo dos anos. [2].

Uma das grandes revoluções que vem acontecendo na utilização da Internet é o conceito de Internet das coisas (IoT ou *Internet of things*). IoT vem despertando o interesse da área industrial, serviços e acadêmica, seja na proposta de novas aplicações ou na tentativa de padronizar o serviço. O número de trabalhos relacionados a está tecnologia cresceu imensamente no últimos anos [3]. A IoT define a conexão de bilhões de dispositivos na rede. As chamadas "coisas" podem ser qualquer objeto que possa gerar medidas ou ações em um determinado contexto. Objetos que até agora eram inimagináveis de serem conectados na Internet, agora passam a gerar dados úteis para as aplicações [4]. Por outro lado, a Internet atual não está preparada para este crescimento exponencial do número de dispositivo. Problemas relacionados a escalabilidade, confiabilidade, gerenciamento, mobilidade e segurança começaram a ganhar ainda mais notoriedade com o surgimento da IoT [14].

Com a consciência destes problemas emergentes, muitas iniciativas surgiram com o propósito de remodelar a Internet, fazendo uso de tecnologias atuais para criar uma arquitetura mais adaptada as aplicações correntes e que ofereça melhores condições aos serviços futuros. Estas iniciativas são generalizadas como Internet do Futuro (FI) [5]. É neste contexto que esta contribuição está inserida: avaliar o comportamento de uma arquitetura de Internet do Futuro em aplicações de IoT. Os projetos de FI se dividem em duas linhas de pesquisa basicamente, a evolucionária e a revolucionária também conhecida como *clean-slate* (folha em branco) [13].

A linha evolucionária defende que novas soluções sejam adicionadas sobre os protocolos básicos do modelo atual para resolver os problemas da Internet. De certa forma, esta evolução já aconteceu várias vezes na arquitetura, como por exemplo a criação do IPv6 para resolver os problemas relacionados a falta de endereço IP. Nesse contexto, a chave é até quando o modelo atual suportará estas evoluções?

Os revolucionários propõe que uma nova arquitetura seja criada por completo sem qualquer compromisso com o modelo atual utilizando tecnologias e conceitos mais adequados para os casos de uso da Internet nos dias atuais e futuros. Aplicam-se em redesenhar com profundidade as limitações enfrentadas pela Internet hoje, tipicamente empregam alguns dos seguintes paradigmas para a reconstrução: *information-centric networking (ICN)* [7], [8], [9], *service-oriented architecture (SOA)* [8], [31], *software-defined networking (SDN)* [10], *service-centric networking (SCN)* [11], *self-verifying naming (SVN)* [26], [8] [12], dentre outros. Nesta linha podem ser citados os seguintes projetos: *eXpressive Internet Architecture (XIA)* [15] e a NovaGenesis (NG) [16].

Nesta dissertação será analisado o comportamento do modelo proposto pela NovaGenesis, arquitetura revolucionária de Internet do Futuro que considera em seu desenho serviços voltados para aplicações IoT. A NovaGenesis propõe o conceito de *Future Internet of things* (FIoT), ou seja, construir uma rede de Internet das coisas utilizando os paradigmas abordados nas novas arquiteturas, trazendo para o cenário IoT a evolução/revolução proposta por iniciativas de FI.

Além dos problemas já mencionados, uma rede IoT sofre com as restrições impostas pelos nós que a formam. Geralmente, um nó de uma rede IoT possui pouca capacidade computacional e limitações energéticas devido ao uso de baterias[6]. Por isso, ao projetar-se uma solução para FIoT deve-se ponderar as restrições energéticas do nó. Esta dissertação propõe a integração da NovaGenesis com um sistema operacional que ofereça mecanismo para controlar o processamento computacional do nó visando atingir a máxima eficiência energética do sistema.

Um sistema operacional (SO) é uma interface entre o software da aplicação e o hardware da plataforma onde o sistema está embarcado [17]. Faz a gerência do acesso aos recursos de hardware, uma vez que a aplicação pode possuir múltiplas tarefas, concorrendo simultaneamente pelos mesmos recursos. Outra característica do SO é a capacidade de compartilhar a CPU entre os vários processos ativos no sistema, dividindo o tempo de execução conforme a prioridade de cada tarefa. Este bloco do sistema operacional é chamado de *Scheduler* ou escalonador, que pode ser classificado como preemptivo ou não preemptivo [18].

O escalonador preemptivo permite que um processo seja interrompido durante a execução por um outro processo que possui maior prioridade na tabela de escalonamento do sistema. Por outro lado, o escalonador não preemptivo não permite a interrupção de nenhum processo, permitindo que uma nova tarefa seja executada somente quando a tarefa atual for finalizada.

Um SO também pode ser classificado como um sistema operacional de tempo real (*RTOS*). Neste caso, cada tarefa possui um tempo para ser atendida, e deve ser processada durante este período garantido uma reposta ao evento dentro de um limite máximo de tempo. Basicamente as características de tempo real podem ser separadas em *hard real-time* e *soft real-time*. No primeiro caso, o tempo limite deve ser rigorosamente respeitado e todos os atrasos inerentes ao processo de escalonamento, como o tempo para salvar um contexto durante a troca de tarefas ou até mesmo o tempo que o processador gasta para atender uma interrupção, devem ser considerados afim de atender o tempo estipulado. No *soft real-time*, o limite de tempo é um pouco mais maleável em relação ao *hard real-time* ocasionando pequenas variações no tempo de atendimento de alguma tarefa [19].

O sistema operacional escolhido para trabalhar em conjunto com a NG nesta contribuição foi o EventOS. SO desenvolvido para ser utilizado em aplicações IoT. Com o foco em eficiência energética e

a escalabilidade do sistema, o EventOS aplica em sua arquitetura o modelo publica-assina e conceitos de programação orientada a evento para a controle dos processos.

Um evento pode ser definido como uma ação independente que pode ocorrer de forma esporádica ou de forma cíclica em um sistema [20]. Nos sistemas embarcados, uma grande parte dos processos são dependentes dos eventos, seja por exemplo o recebimento de um pacote por uma interface de comunicação ou a detecção de uma variação em um pino digital. Estes eventos são utilizados como "gatilhos"para disparar um processo novo dentro do sistema embarcado. Esta dependência dos processos caracteriza um sistema *event-driven*, ou seja, um sistema orientado a eventos.

No formato tradicional, o controle do sistema embarcado aguarda de forma ociosa o acontecimento dos eventos em vários trechos da aplicação, causando um desperdício de recursos como a energia e o processamento, algo muito prejudicial em sistemas alimentados por baterias. Já em uma programação orientada por eventos existe uma inversão no controle do sistema, onde o próprio evento notifica seu acontecimento e invoca o controle do sistema para o processo, que pode ser configurado para o modo de baixo consumo durante o tempo de espera [21]. Desta forma, o EventOS baseia-se que todos os processos devem ser iniciados por um evento, mantendo o sistema o maior tempo possível em modo de baixo consumo

O modelo publica-assina é uma referência para a programação de sistemas distribuídos. Propõe que uma tarefa realize a assinatura de um evento do qual deseja receber notificações de ocorrência através de um agente intermediário chamado *broker*, sem necessariamente conhecer a origem do evento. Do outro lado, com a ocorrência de um novo evento, um publicador envia uma notificação ao *broker* que posteriormente notifica todas as tarefas que assinam o evento publicado. Neste modelo, existe um desacoplamento entre a origem e o destino de um evento sendo mais importante a própria informação. [22].

Na Figura 1.1 é apresentado o funcionamento básico de uma estrutura publica-assina. Uma tarefa pode assinar vários eventos, e um evento pode ser assinado por várias tarefas, sendo do *broker* a função de centralizar as informações que vinculam os eventos aos assinantes e por notificar cada assinante registrado sobre a ocorrência de um novo evento.

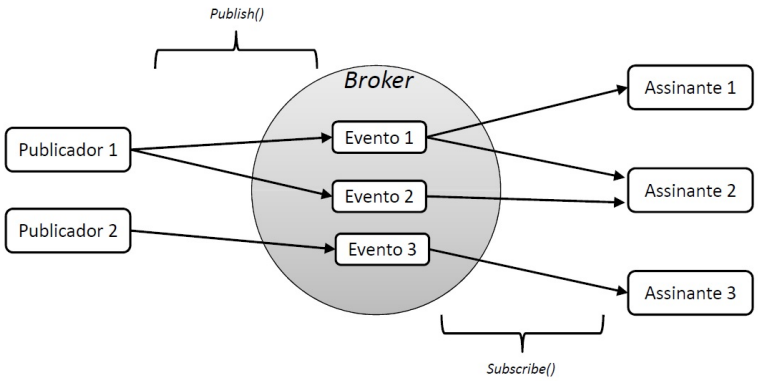

Figura 1.1: Sistema publica/assina.

A interação proposta no modelo publica-assina resulta em um descorrelação em relação ao tempo, pois as partes não precisam estar interagindo ao mesmo tempo. Com relação ao espaço, os envolvidos

não precisam se conhecer podendo inclusive estar em máquinas diferentes. Desacoplar as partes envolvidas na geração e no consumo de um evento aumenta a escalabilidade do sistema e reduz a necessidade de sincronismo entre as diversas entidades envolvidas em aplicações distribuídas que são assíncronas por natureza [23]. No EventOS o escalonador de tarefas opera como o *broker* do modelo, recebendo novos eventos publicados, e distribuindo as notificações para os assinantes responsáveis pelo tratamento.

Neste contexto, esta contribuição tem como objetivo demostrar o funcionamento do serviço proposto pela NovaGenesis (NG) em IoT, apresentar a viabilidade de utilizar a NG em cenários de FIoT e integrar o serviço da NG com um sistema operacional que proporcione controle sobre o *hardware* e forneça opções de baixo consumo de energia. As principais contribuições deste trabalho são:

- Demonstrar o funcionamento do serviço para IoT proposto pela NovaGenesis em condições reais de utilização;

- Adaptar o serviço da NovaGenesis para operar em conjunto com técnicas que ofereçam uma melhor eficiência energética;

- Propor uma solução que dispense qualquer tipo de configuração ao adicionar um novo dispositivo na rede NovaGenesis;

- Sugerir um modelo que tenha a possibilidade de operar com as características heterogêneas de uma rede IoT;

Nesse contexto, o Capítulo 2 detalha o sistema operacional EventOs e os recursos empregados para aumentar a eficiência do sistema embarcado. O Capítulo 3 apresenta a NovaGenesis, em especial os serviços destinados à aplicação IoT. O Capítulo 4 mostra toda a implementação realizada nesta dissertação, apresentando todas as fases de desenvolvimento deste trabalho. No Capítulo 5 demonstra-se os experimentos realizados na dissertação e os resultados obtidos. E, finalmente, no Capítulo 6 conclui-se esse trabalho.

Capítulo 2

EventOS

Geralmente, os nós IoT possuem *hardwares* dotados de poucos recursos computacionais e principalmente baixa capacidade energética. Estas características estimulam o desenvolvimento de novos sistemas operacionais para esta aplicação. O EventOS [24] é um projeto iniciado em 2015 pelo Prof. Dr. Edielson Prevato Frigieri com o objetivo de criar um sistema operacional escalável e energeticamente eficiente para redes de sensores IoT. Emprega o padrão publica-assina aliado a uma programação orientada a eventos para otimizar a utilização da CPU e o consumo de energia. Neste capítulo, serão apresentados alguns detalhes do EventOS, que foi o sistema operacional selecionado para este trabalho.

2.1 Escalonador de tarefas

Conforme apresentado na seção 2.2, o escalonador de um sistema operacional é o responsável por determinar a utilização da CPU entre os múltiplos processos ativos no sistema. No EventOS, o escalonador assume o papel do *broker* no modelo publica-assina, sendo responsável por armazenar os novos eventos em filas e notificar os processos assinantes conforme sua prioridade.

No EventOS, o escalonador pode assumir três estados diferentes como apresentados na Figura 2.1. No estado *sleeping*, o microcontrolador é configurado para o modo de baixo consumo (*Idle*) e a CPU não executa nenhum processo. Com a ocorrência de um novo evento (interrupção de *hardware*), a CPU retorna para o funcionamento padrão e o escalonador é alterado para *scheduling*. Neste estado, as filas são percorridas em busca de algum evento para ser processado. Se algum evento for encontrado, as tarefas dos assinantes são notificadas e o evento é retirado da fila, caracterizando o estado de *delivering*. Após o tratamento do evento pelos assinantes, o escalonador assume novamente o controle (*scheduling*) e se não houver mais nenhum evento em nenhuma fila, o sistema retorna ao estado de *sleeping* para aguardar um novo ciclo.

Os novos eventos são organizados nas filas conforme sua prioridade, onde os primeiros a entrarem, serão os primeiros a saírem de uma determinada fila (FIFO), respeitando sempre a prioridade de cada uma. Para receber as notificações, uma tarefa deve realizar a assinatura do evento, o que geralmente ocorre na inicialização do sistema. Para cada evento suportado, o sistema operacional cria uma lista de

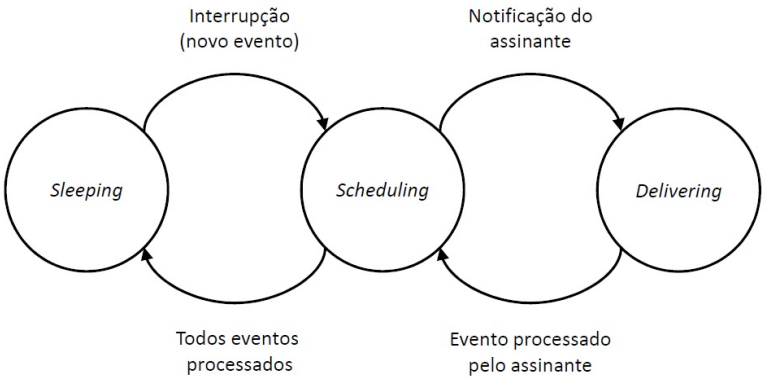

Figura 2.1: Possíveis estados do escalonador.

tarefas assinantes com as informações necessárias para o escalonador notificar a ocorrência. A ordem de notificação é baseada na sequência de assinatura. As primeiras tarefas a assinarem o evento serão as primeiras a serem notificadas.

A Figura 2.2 exemplifica o mecanismo de escalonamento do EventOS. Após a interrupção, são gerados dois eventos com diferentes prioridades. Para cada evento existe uma lista com as tarefas assinantes. O escalonador percorre as filas iniciando pela de maior prioridade. Ao localizar um evento realiza as notificações conforme a ordem de assinatura. Após a tratamento do ultimo evento da lista de menor prioridade o sistema retorna para o modo de baixo consumo.

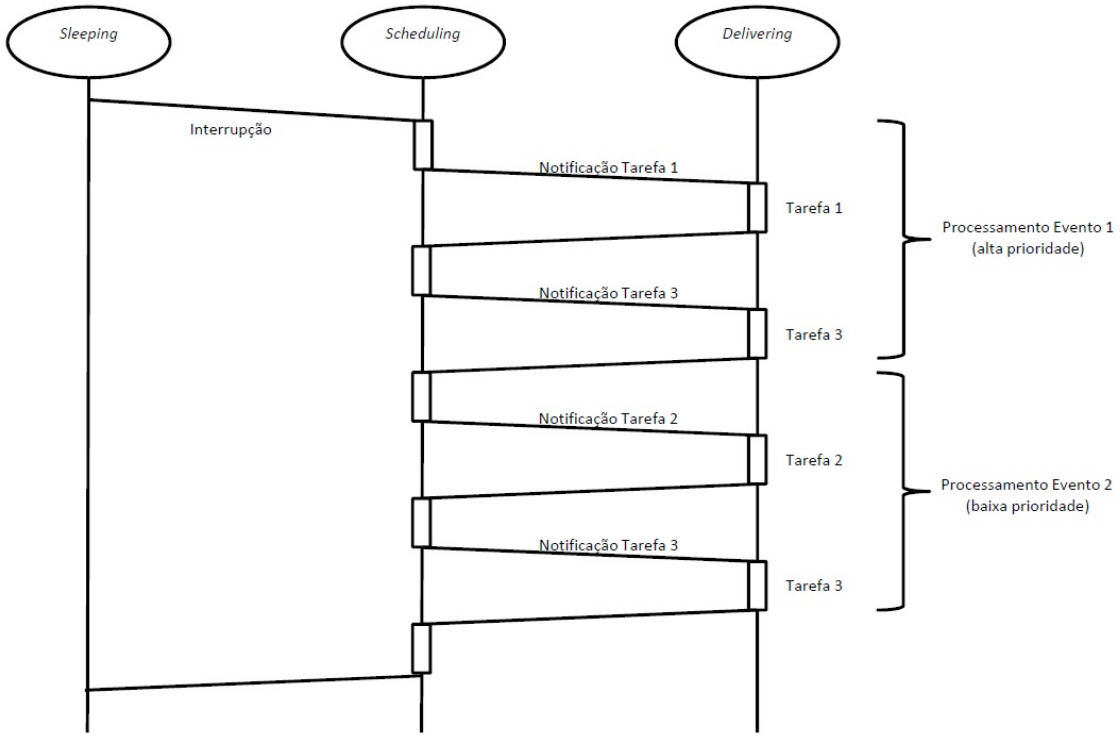

Figura 2.2: Dinâmica do escalonador.

Com o padrão publica-assina, os eventos foram desacoplados das tarefas de processamento, criando

uma independência em relação ao instante de tempo em que um evento é publicado, e o instante de tempo em que o evento é processado pelo assinante. Outra vantagem é que com o padrão publica-assina, eventos e assinantes podem ser inseridos e retirados do sistema de forma simples e rápida, possibilitando a escalabilidade do EventOS.

2.2 Tempo de Vida do Evento

Um possível problema para o modelo de filas prioritárias proposto inicialmente no EventOS é a inibição de processamento de eventos de menor prioridade quando os eventos de maior prioridade ocorrem numa frequência elevada. Neste cenário, a fila de maior prioridade recebe novos eventos a todo instante, impedindo o escalonador de percorrer as filas de menor prioridade e consequentemente o processamento dos eventos nelas presente.

A ideia adotada para resolver a inibição foi a criação de um sistema de ”contador de tempo de vida” que pode resultar na elevação da prioridade de um evento. Nesta solução, foram criados três tipos de variáveis de controle:

- Contador de referência: Este contador é incrementado a cada novo evento processado pelo escalonador, ou seja, a cada evento que é retirado de uma fila e enviado a algum assinante resulta no incremento deste contador;

- Tempo inicial do evento: Esta variável é vinculada a um evento e recebe o valor do contador de referência no momento em que o escalonador realiza a inserção na fila;

- Tempo de vida: É o tempo máximo que um evento pode esperar em uma fila. Valor constante e pré determinado;

A cada novo evento processado pelo escalonador é realizada uma comparação entre o contador de referência e o tempo inicial do primeiro evento de cada fila. Se a diferença entre as variáveis for igual ou superior a um valor de tempo de vida, o evento é retirado da fila em que se encontra e adicionado na fila com prioridade imediatamente maior com o valor inicial atualizado. Um detalhe importante nesta implementação é que na fila de maior prioridade não é realizado este controle, pois um evento que está nesta fila inevitavelmente será processado.

A Figura 2.3 ilustra o processo de atualização da prioridade de um evento. Em (a) são inseridos quatro novos eventos (EVENTO1, EVENTO2, EVENTO3 e EVENTO4) em duas prioridades diferentes. Em (b) e (c) o escalonador processa o EVENTO1 e o EVENT02, respectivamente, incrementado o contador de referência em duas unidades. Neste ponto a diferença entre o contador referência e o tempo inicial do EVENTO4 é igual ao valor definido como tempo de vida, resultando em uma atualização na prioridade do evento conforme se mostra em (d).

Esta alteração garante o processamento de qualquer evento, pois independentemente do número de elementos distribuídos nas filas ou da frequência em que novos são criados, um evento de baixa priori-

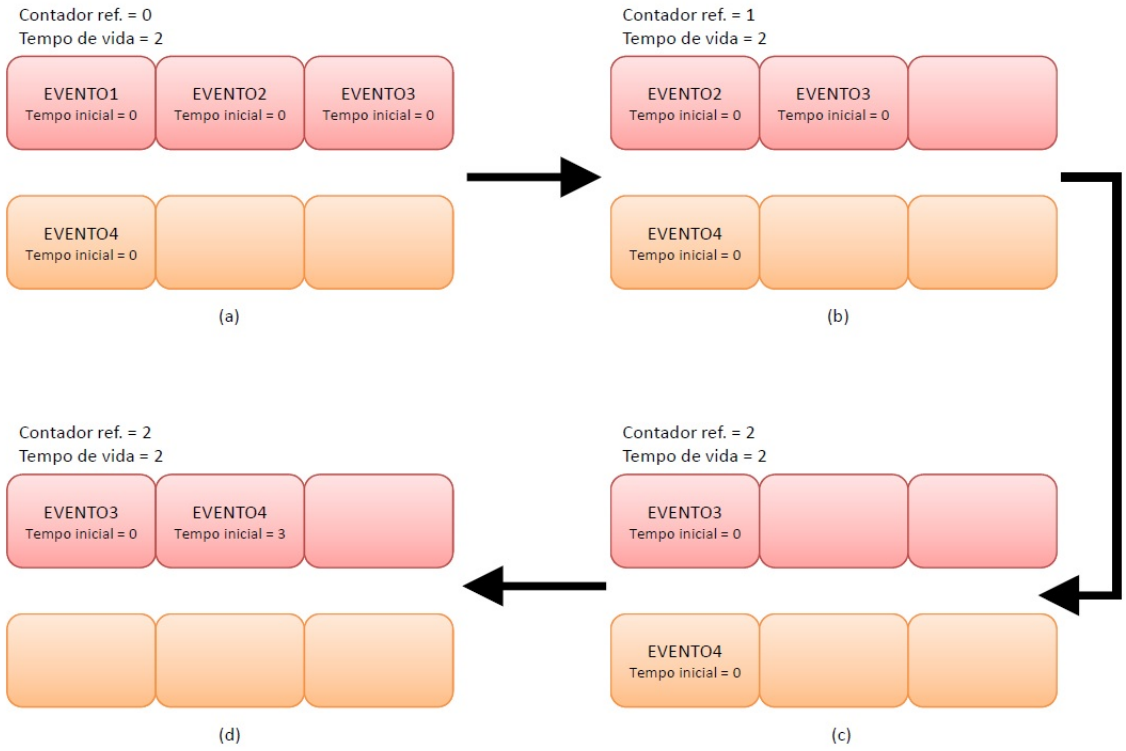

Figura 2.3: Exemplo de atualização do evento por tempo de vida.

dade poderá sofrer atualizações até que em certo momento atinja um nível de prioridade que possibilite o EventOS de processar.

2.3 *Low-Power*

O EventOS foi projetado para atender as restrições energéticas geralmente encontradas em sistemas IoT. O primeiro recurso empregado é a programação orientada a evento. Neste modelo acontece uma inversão no controle do sistema, onde o próprio evento gerado é utilizado para iniciar o processamento. Durante o tempo de espera, o sistema é configurado para um modo de baixo consumo e retorna para o modo ativo somente na ocorrência de um novo evento, ou seja, quando realmente é necessário.

Outra característica de *low-power* do EventOS é o *tail-chaining*, recurso diretamente relacionado com a arquitetura ARM^{TM}, que foi a arquitetura selecionada como referência para o desenvolvimento. O *Tail-chaining* é um processo econômico para a troca de contexto entre duas interrupções. Neste método, a CPU ignora o manuseio de oito registros na pilha que não afetam o processamento das interrupções, resultando em uma latência e um consumo menor para a alternar entre processos [25]. O EventOS aproveita deste recurso executando todos os processos inerentes ao tratamento de um evento através de uma sucessão de interrupções. A Figura 2.4 ilustra o mecanismo de interrupções empregado no sistema.

Quando ocorre um evento externo, a CPU é retirada do estado de baixo consumo e o processamento é deslocado para a interrupção (ISR) de alta prioridade referente ao evento. Então o escalonador é notificado e uma interrupção de baixa prioridade é gerada. Ao finalizar a interrupção de alta prioridade,

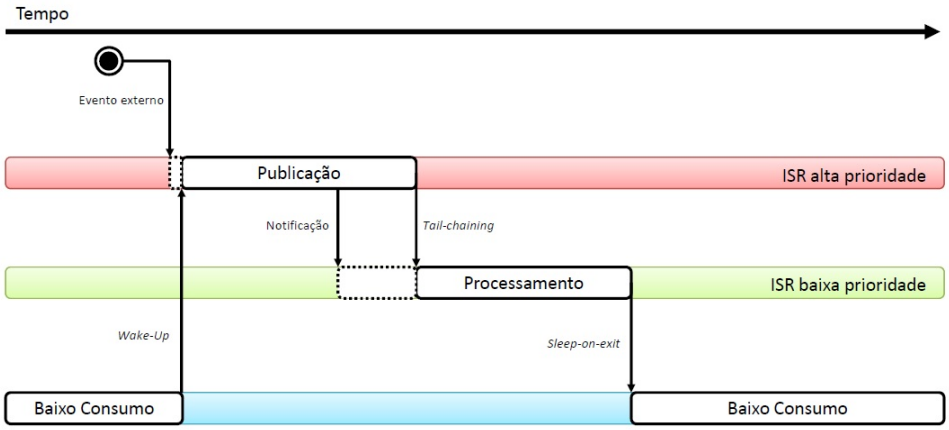

Figura 2.4: Representação do *Tail-Chaining*.

acontece o *tail-chaining* sendo que o processamento é deslocado para a interrupção de baixa prioridade que ainda esta pendente. É neste momento que o escalonador de fato notifica os assinantes e o evento é processado pela aplicação. Todo o processamento do EventOS é realizada dentro de interrupções, e a troca de contexto é realizada somente entre interrupções, justificando o uso do *tail-chaining*.

E por fim, o EventOS emprega um recurso chamado *Sleep-on-Exit* que reconfigura de forma automática a CPU para o modo baixo consumo imediatamente após finalizar o processamento da última interrupção ativa no sistema, sem a necessidade de algum comando adicional [25]. Em resumo, o EventOs emprega um modelo orientado a eventos aliado a técnicas encontradas na arquitetura, para executar o mínimo de processamento possível para a aplicação, resultando em um sistema mais eficiente.

Capítulo 3

NovaGenesis

A NovaGenesis é um projeto de Internet do Futuro iniciado em 2008 pelo Prof. Dr Antônio Marcos Alberti. Segue a linha das pesquisas revolucionárias, que propõe o redesenho completo da Internet empregando tecnologias mais apropriadas e modernas para suprir as demandas atuais e futuras da Internet [26], [27]. A Figura 3.1 apresenta a arquitetura da proposta pela NovaGenesis.

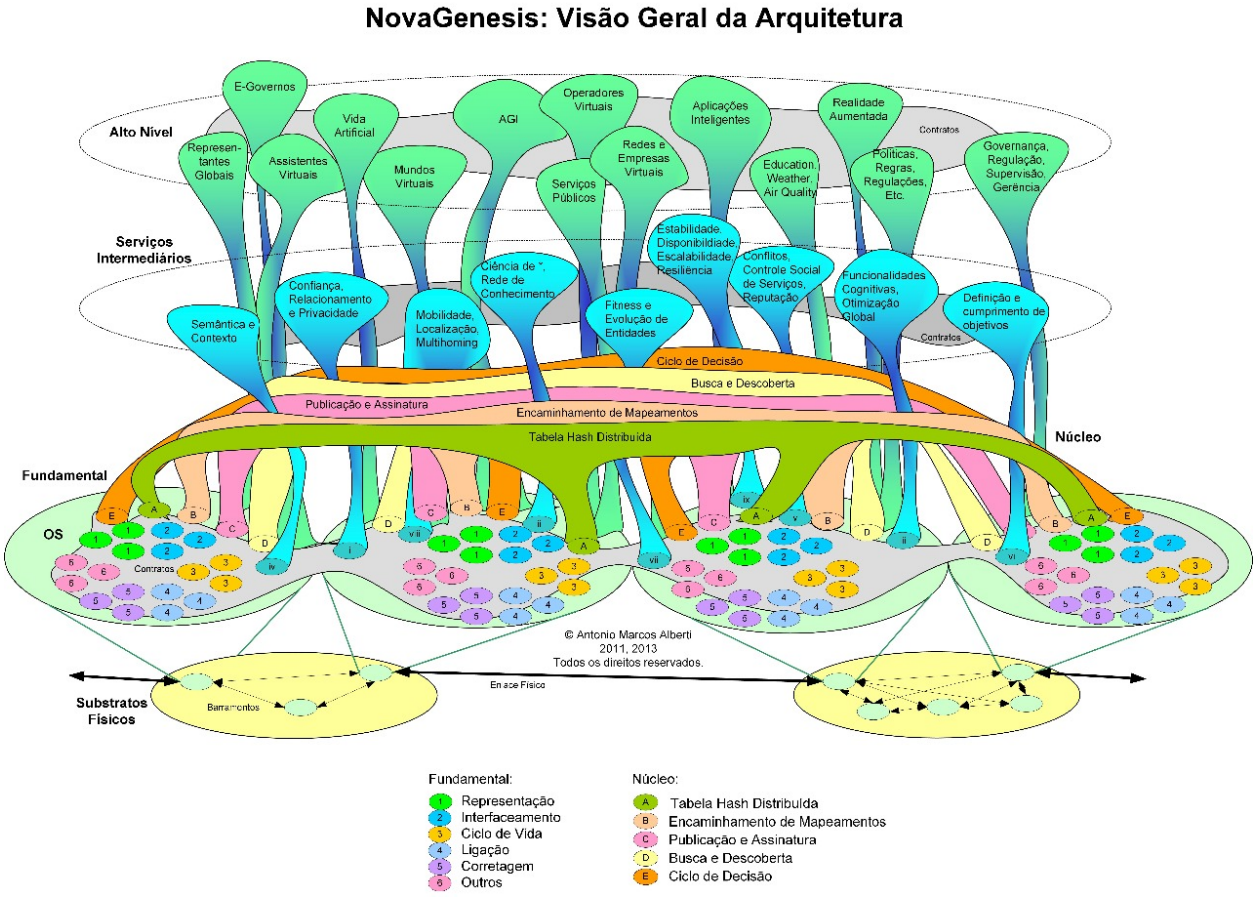

Figura 3.1: Arquitetura [16].

A camada de ”Substratos Físicos” representa os enlaces físicos utilizados para conectar redes de computadores, como por exemplo, *links* de rádio ou fibras ópticas. Na sequência exite a camada chamada de ”Fundamental”, que representa a execução dos processos da NovaGenesis pelo sistema operacional do computador. A camada ”Núcleo” inclui os serviços básicos da NG, como resolução de nomes, descoberta de serviços, armazenamento de nomes, dentre outros. A camada de "Serviços Intermediários"abrange uma diversidade de serviços que visam melhorar dificuldades enfrentadas atualmente, como a mobilidade e a segurança da rede. E na camada ”Alto Nível” estão as principais aplicações clientes que poderão fazer uso da NovaGenesis.

3.1 Visão Geral

Um dos conceitos fundamentais empregados na arquitetura é a nomeação dos diferentes elementos que compõem a arquitetura NovaGenesis. Um nome é uma forma de designar uma determinada existência. É utilizado para definir uma pessoa, um objeto, um animal, enfim, tudo o que precisa de uma definição recebe um nome.

Quando o nome carrega algum significado é chamado de *Natural Language Name* (NLN), por exemplo,”Lápis” é um nome facilmente reconhecido utilizado para definir um objeto. Existe ainda o conceito de Nome Autoverificável ou *Self-Verifying Name* (SVN), que são nomes calculados a partir de funções *Hash* que embaralham NLNes criando um novo nome sem o menor significado para as pessoas. Na NovaGenesis todas as operações dependem dos nomes, são eles que endereçam e localizam os recursos na rede. Os NLNes podem ser qualquer nome que designe um elemento, o endereço MAC de uma placa de rede, o arquivo fonte de um programa de computador ou o nome do arquivo de uma simples foto. Os SVNes são obtidos através de um algoritmo *Hash* chamado de *MurMurHash3* [28], a Figura 3.2 ilustra o mecanismo utilizado para gerar os SVNes na NG.

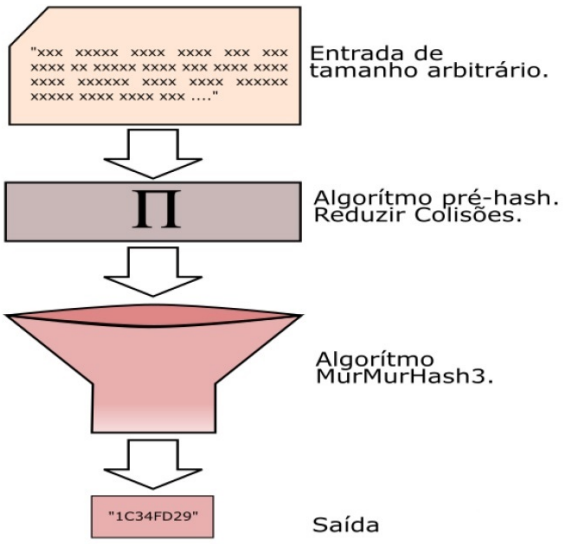

Figura 3.2: Geração de SVNes [30].

Um *name binding* (NB) é uma associação entre dois ou mais nomes relacionados, por exemplo, um numero telefônico pode ser relacionado ao nome de uma pessoa, que pode ser associada um ende-

reço residencial. Esta associação permite localizar todas as informação de um contexto a partir de um único nome, além de criar uma circunstância única para o contexto descrito [29]. Na NovaGenesis o *name binding* é utilizado para proporcionar um método genérico e escalonável para relacionar os nomes dos elementos que compõem um serviço, gerando um identificador único dentro de um escopo. Os identificadores são armazenados em tabelas distribuídas pela rede, permitindo a arquitetura localizar e comunicar com qualquer serviço a partir da combinação de nomes criada.

A NovaGenesis define como um serviço qualquer entidade capaz de processar, armazenar ou trocar informações, assim qualquer processo computacional desenvolvido na arquitetura pode ser considerado um serviço. Com o surgimento da virtualização, conceitos como redes definidas por *software* (SDN - *Software Defined Networking*) fomentam a ideia de considerar todas as coisas como um serviço (*anything as a service - XaaS*) [31], onde circuitos eletrônicos convencionais serão substituídos por códigos computacionais denominados de maneira geral como serviços, imaginar uma arquitetura de Internet que consiga interagir com esses novos conceitos de serviços é fundamental para proporcionar uma solução eficiente e escalável.

Na NG todos os processos computacionais criados na arquitetura são considerados serviços e as fases previstas para um serviço inclui a exposição de recursos para a rede, a negociação de contratos com outros serviços, o monitoramento da qualidade do serviço prestado e a liberação dos recursos contratados. Os contratos entre os serviços especificam as responsabilidades e as limitações de cada atividade, bem como os critérios de qualidade que devem ser observados durante sua vigência. Desta maneira, a NovaGenesis permite que serviços menores sejam associados para criar serviços maiores, possibilitando a interação entre os diversos serviços existentes em uma arquitetura de Internet independente da esfera em que se enquadram. Para uma melhor compreensão, serão apresentados na sequência alguns serviços implementados na NG envolvidos nesta contribuição

O *Name Resolution and Networking Cache Service* (NRNCS) é o serviço responsável por implementar, armazenar e gerenciar os *name bindings* e conteúdos associados de um serviço e suas permissões. Ao publicar seus NBs para o NRNCS, um serviço publica as permissões necessárias para que um terceiro serviço os assinem, ou seja, somente um serviço autorizado pode assinar este publicador. O NRNCS foi implementado utilizando outros três serviços fundamentais: o PSS, o GIRS e o HTS.

O *Publish/subscribe service* (PSS) é serviço que promove o encontro entre serviços publicadores e serviços assinantes na NG. A arquitetura atual da NovaGenesis implementa o modelo publica/assina para realizar a troca de informações entre os diferentes serviços da rede. Quando iniciado, um serviço não possui nenhum conhecimento dos possíveis parceiros que possam assiná-lo e os serviços já ativos desconhecem as características deste novo serviço inviabilizando sua assinatura, ou seja, é necessário um intermediador para apresentar e possibilitar a troca de informações entre os serviços, que no caso da NovaGenesis é o PSS.

O *Generic indirection resolution service* (GIRS) é um serviço que atua entre o PSS e o *Hash Table Service* (HTS). Quando o GIRS recebe do PSS os NBs de um novo serviço disponível, é de sua responsabilidade decidir em qual tabela *hash* serão salvos os NBs recebidos. Ou seja, este serviço realiza o balanceamento entre as tabelas *hash* distribuídas disponíveis no domínio.

O HTS implementa as tabelas *hash* para armazenar os NBs e as informações referentes aos serviços presentes em um domínio.Por exemplo, quando o NRNCS recebe os *name bindings* de um novo ser-

viço, é o HTS que armazena as novas informações e disponibiliza métodos de busca simplificados para consultas posteriores. O HTS de um domínio pode comunicar-se com o HTS de outro domínio, criando um ambiente hierárquico e distribuído para o armazenamento e a consulta das informações.

Outro serviço envolvido nesta contribuição é o *Proxy/gateway/controller service* (PGCS) que implementa o roteamento e o encaminhamento das mensagens NovaGenesis, ou seja, é responsabilidade deste serviço adaptar e endereçar as mensagens NG conforme a tecnologia utilizada na camada de enlace. Possui uma tabela local para armazenar os NBs e os endereços a nível de enlace dos serviços conhecidos, por exemplo, o endereço MAC que identifica o dispositivo onde o serviço está hospedado em uma rede Wi-Fi. A Figura 3.3 apresenta a abstração proporcionada pelo PGCS em relação a camada de enlace utilizada pelo serviço.

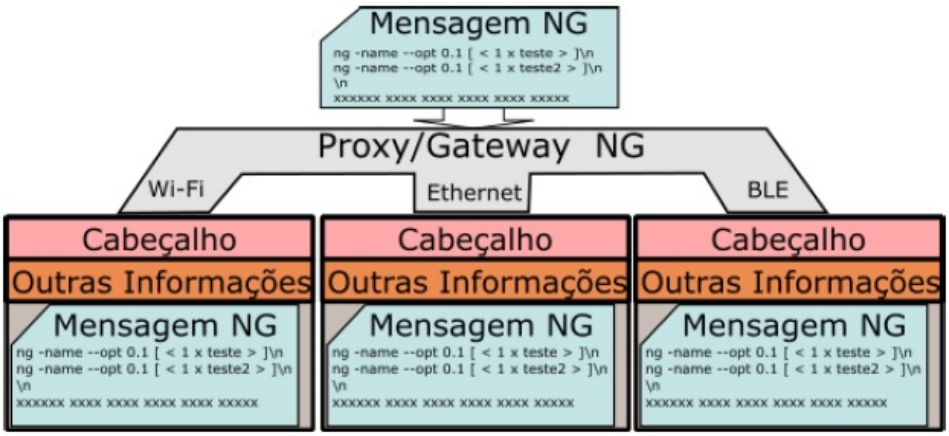

Figura 3.3: Abstração da camada de enlace [30].

Em resumo, a NovaGenesis é uma arquitetura que baseia-se em serviço, onde pequenos serviços genéricos podem ser combinados para criar serviços maiores com as particularidades que são necessárias, criando uma rede modular e escalonável [31].

3.2 NovaGenesis para IoT

Para funcionar em sistemas embarcados e permitir a interação da NG com dispositivos de IoT, foi necessário criar e adaptar alguns serviços na arquitetura. Esta seção apresenta os serviços criados, as alterações implementadas e detalha os processos que ocorrem durante a comunicação dos serviços envolvidos em aplicações IoT.

3.2.1 *Embedded Proxy/Gateway Service (EPGS)*

O EPGS é o serviço para sistemas embarcados da NovaGenesis. Dada as limitações geralmente encontradas nos nós IoT, o EPGS implementa o mínimo necessário para o funcionamento do serviço, incluindo o algoritmo de geração dos SVNes, a tabela *Hash* para armazenar os nomes dos parceiros e obviamente a capacidade de processar as mensagens NG desenvolvidas para a FIoT.

Os SVNes são utilizados para distinguir e consequentemente endereçar um serviço na NovaGenesis. A identificação do EPGS é formada pelo nome do *hardware* onde a solução está embarcada (HID), pelo nome do sistema operacional utilizado (OSID) e pelo nome do processo disponível no serviço (PID). A partir destes nomes, os SVNes são calculados e a combinação entre eles definem o EPGS na rede.

Outra implementação necessária foi a criação da tabela *Hash* para armazenar as identificações dos serviços parceiros. O EPGS necessita manter os SVNes do serviço que o representa na rede, para que possa encaminhar as mensagens e estabelecer contratos com outros serviços. Estas informações são armazenadas na DHT, tornando sua implementação indispensável.

E finalmente, o EPGS tem a capacidade de processar e responder todas as mensagens projetadas para o serviço IoT através de uma interface Wi-Fi, sendo que, a adaptação das mensagens para a camada de enlace faz parte da implementação.

3.2.2 *Proxy/gateway/controller service* (PGCS) para IoT

O serviço de PGCS sofreu algumas modificações para as aplicações IoT propostas pela NovaGenesis. O PGCS foi alterado para atuar como um *gateway* da rede IoT para o ”mundo” externo. Basicamente, o PGCS foi alterado para disponibilizar para EPGS SVNes de um serviço PSS disponível. Desta forma, o EPGS consegue utilizar as funcionalidades previstas no PSS para divulgar suas informações. Possibilitando que serviços externos a rede IoT descubram as características de cada nó EPGS e estabeleçam novos contratos.

3.2.3 Cliente IoT

Para possibilitar os testes do serviço de IoT da NG, foi desenvolvido um cliente (*IoTTestApp*) para consumir os dados gerados pelo EPGS. Está aplicação utiliza os serviços de um PSS para assinar no NRNCS os dados gerados por um ou mais EPGSes presente no domínio. Desta forma, quando o EPGS publicar alguma informação o cliente assinante será notificado, possibilitando a utilização dos dados publicados para qualquer fim que uma aplicação possa ter.

3.2.4 Princípios de funcionamento

Os novos serviços propostos especificam algumas etapas de funcionamento. Na sequencia são apresentados os processos desenvolvidos e as informações compartilhadas entre cada serviço nas diferentes fases do modelo proposto.

Uma das primeira ações executadas pelo EPGS na fase de *Initialization* é o cálculo de seus SVNes, nesta operação são utilizados informações (NLNes) obtidos do *hardware* e do sistema operacional onde o EPGS está embarcado. Em seguida, o EPGS envia uma mensagem de *Hello* para o PGCS com seus SVNes, NLNes e o endereço MAC. Como neste ponto o EPGS ainda não conhece o PGCS, a mensagem é enviada em *broadcast* pelo enlace Wi-Fi. O PGCS recebendo o *Hello*, armazena em uma tabela *hash* local os NBs do EPGS e responde a mensagem com uma confirmação de recebimento. Paralelo ao

processo apresentado, o PGCS possui uma rotina independente para publicar mensagens de *Hello* para o EPGS. Este comando carrega os SVNes, os NLNes e o endereço MAC do PGCS, além dos SVNes do serviço de PSS disponível para o uso do EPGS. Então, nesta fase de inicialização, ocorre uma troca de NBs entre o EPGS e o PGCS, onde o representante e o representado se identificam, além da troca do endereço MAC utilizado para a comunicação Wi-Fi.

Na *Initialization* também ocorre a inicialização do aplicativo cliente. Os SVNes são cálculados e uma mensagem de *Hello* é enviada. A diferença neste processo em relação a inicialização do EPGS é que não existe a utilização de enlace nesta troca de mensagem, os processos ocorrem dentro da mesma entidade física, compartilhando apenas recursos de memória para a troca de mensagens. Por isso, o parâmetro *shmID* (*shared memory ID)* é enviado no *Hello* da aplicação. A Figura 3.4 apresenta a fase de *Initialization* prevista na NovaGenesis.

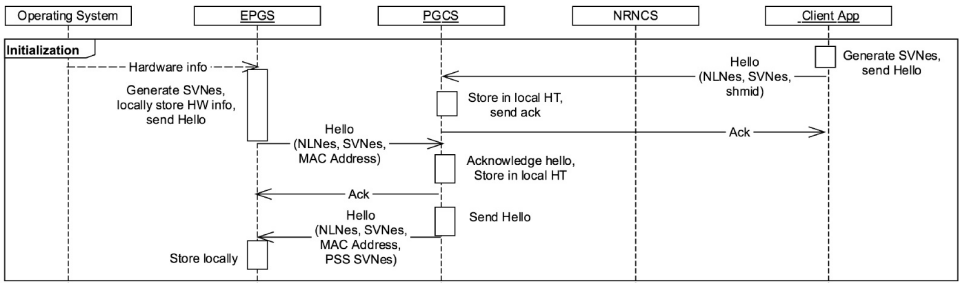

Figura 3.4: *Initialization* [14].

Na Figura 3.5 ilustra a fase *Exposition & Discovery*. Esta fase é responsável pela troca dos SVNes do PSS e como resultado a descoberta de potenciais serviços parceiros na rede. O processo é iniciado com a aplicação cliente perguntando para PGCS os SVNes de sua API de PSS, o PGCS envia os SVNes para o cliente que armazena em em uma tabela local. Com esta troca, a aplicação cliente está hábil de publicar as palavras chaves que o representam para o NRNCS, possibilitando que os outros serviços o conheçam e o assinem. Da mesma forma, a aplicação cliente descobre através do NRNCS potenciais parceiros na rede, por exemplo, suponha que a aplicação seja defina em suas palavras chaves como ”cliente IoT consumidor de dado” e que o PGCS publique que suas palavras chaves são ”representante de serviços publicadores IoT”. Analisando a semântica das informações publicadas, o PGCS identifica um potencial parceiro para consumir os dados gerados pelo EPGS, e a aplicação identifica um possível fornecedor de medidas, então existe um processo de assinatura e os SVNes e NLNes dos serviços são trocados. Com este processo, a aplicação cliente e o representante da rede IoT foram apresentados como potenciais parceiros, dependo da oferta de serviço do EPGS, um contrato poderá ser estabelecido e a aplicação cliente irá receber medidas de um nó IoT.

Em seguida ocorre o *Service Offer*. Nesta fase, o EPGS publica uma oferta de serviço para o PGCS, que encaminha para o NRNCS para armazenar e divulgar a oferta. Uma oferta de serviço carrega as capacidades do serviço prestado pelo EPGS, ou seja, expõe os recursos oferecidos por aquele nó IoT. O cliente IoT realiza basicamente o mesmo processo, publica sua oferta que atinge o PGCS através do serviço de NRNCS. Neste ponto do cenário apresentado, o PGCS possui duas ofertas, cabendo a ele analisar e estabelecer um contrato entre os serviços, esta analise é realizado no *Service Acceptance*. Nesta fase, o PGCS analisa as duas ofertas, e se decidir aceita-las, publica as notificações e estabelece um contrato entre os serviços ofertantes. A Figura 3.6 apresenta o *Service Offer* e o *Service Acceptance*.

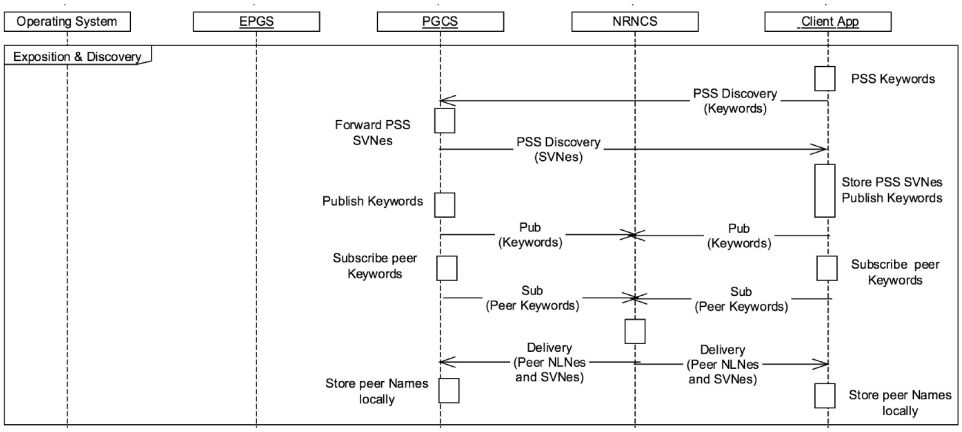

Figura 3.5: *Exposition & Discovery* [14].

A Figura 3.7 mostra a última etapa do processo. O (*Publishing data*) é a publicação das medidas realizadas pelo nó IoT e a assinatura pelo cliente consumidor. Ao receber uma nova leitura do *hardware* o EPGS pulica uma mensagem com os identificadores do serviço assinante e um arquivo JSON com as medidas coletadas. A mensagem atinge o NRNCS que armazena em uma memória *cache* local até que o assinante seja notificado e receba os dados gerados pelo EPGS, finalizando os processos necessários para estabelecer a troca de informações entre um nó e uma aplicação IoT previstos no modelo da NovaGenesis.

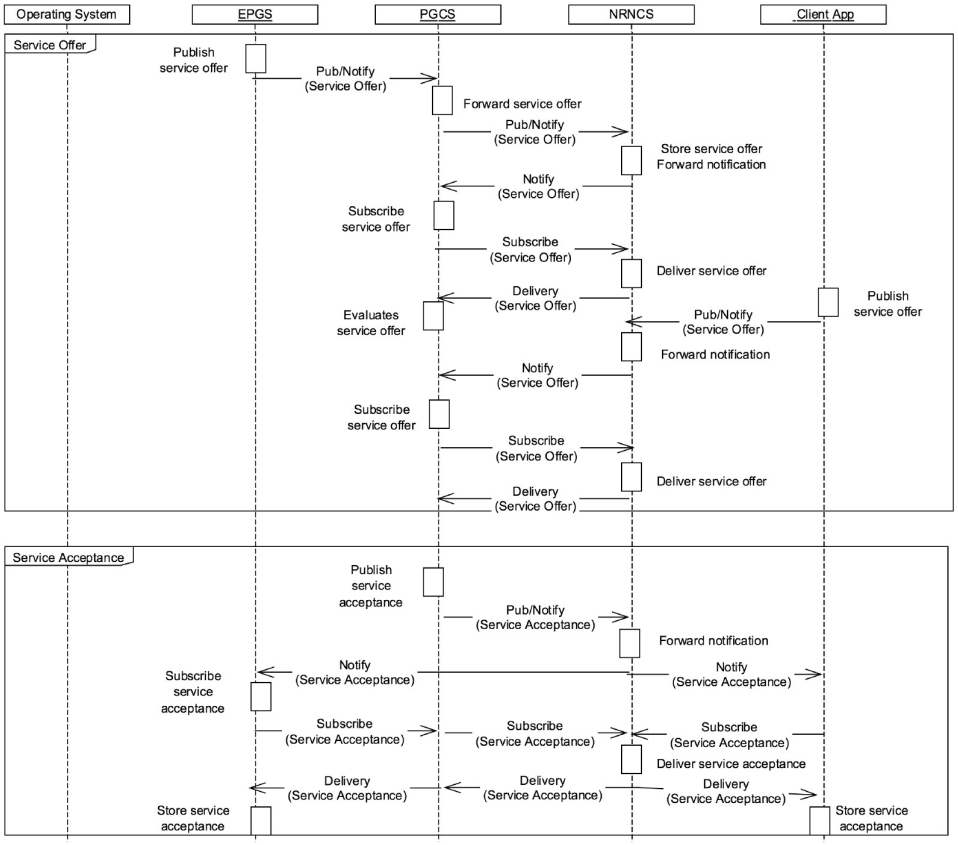

Figura 3.6: *Service Offer* & *Service Acceptance* [14].

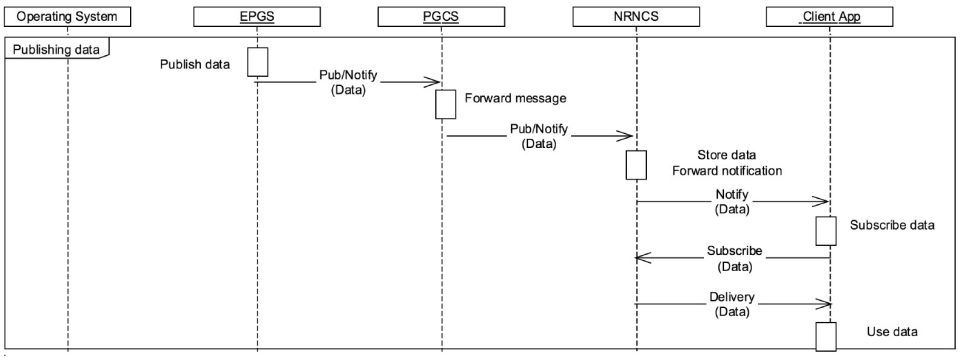

Figura 3.7: *Publishing data* [14].

Capítulo 4

Implementações

A contribuição deste trabalho esta em observar o comportamento do modelo proposto pela NovaGenesis, implementando uma solução que associe o bloco EPGS a uma aplicação básica de IoT, criando assim um cenário real de FIoT. Para tal, foram realizadas várias adaptações para acomodar o modelo proposto em um sistema embarcado e o desenvolvimento de uma aplicação. Foi integrado também na solução, o sistema operacional EventOS, com o objetivo de adicionar os recursos de baixo consumo destacados anteriormente, além de oferecer a priorização de tarefas e o controle dos processos executados pela CPU de forma estruturada. A Figura 4.1 apresenta em linhas gerais a estrutura em blocos da solução proposta.

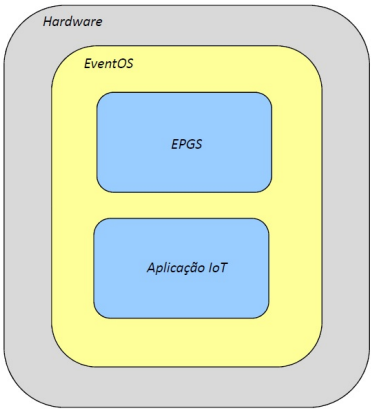

Figura 4.1: Visão geral da contribuição.

O bloco do *hardware* engloba o sistema embarcado onde a solução foi hospedada (microcontrolador) e todos os periféricos utilizados como os sensores e dispositivos para conexão Wi-Fi. Sobre o *hardware* está o bloco do SO (EventOS) gerenciando os recursos de processamento da solução, e finalmente o EPGS e a aplicação desenvolvida. Na sequência do capítulo são apresentadas todas as atividades realizadas para a concepção da contribuição.

4.1 Definição e integração do *Hardware*

Nesta seção, é detalhado a definição do *hardware*, os requisitos observados e a integração entre os diferentes blocos para compor a solução.

4.1.1 Microcontrolador

A definição do microcontrolador foi um importante passo neste trabalho, somente após esta escolha foi possível planejar e dimensionar as atividades conseguintes. O microcontrolador está intimamente relacionado com o sistema operacional, que por sua vez, rege o funcionamento dos blocos de aplicação. Ou seja, a determinação deste bloco causa um impacto em todas as camadas da solução, e por isso foi a primeira atividade executada.

A análise realizada foi em relação a compatibilidade com o EventOS, a *performance* entregue pelo microcontrolador e a capacidade de estabelecer conexões Wi-Fi. Nesta linha, foi definido que o microcontrolador selecionado seria o LPC1769 [32] da NXP™. Um dos principais motivos foi por ser a referência utilizada no desenvolvimento do EventOS, possuindo todas as implementações necessárias para rodar o sistema operacional nesta plataforma, eliminando a necessidade de realizar qualquer adaptação no EventOS.

O LPC1769 utiliza um processador ARM Cortex-M3, trabalha com a frequência de *clock* em até 120 MHz, possui uma memória *flash* de programa de 512 kB e 64 kB de memória RAM. Oferece os protocolos de comunicação serial, o que permite a integração com *hardwares* externos, interface Ethernet MAC, facilitando a implementação de soluções *Ethernet*, além de conversores analógico digital que podem ser utilizados para a leitura de uma variedade de sensores externos.

Outra característica interessante para a aplicação foi o fato do LPC1769 possuir um numero único de identificação. Como o modelo da NovaGenesis é baseado em nomeação, e um dos nomes propostos no EPGS é o HID, que define o nome do *hardware*, este número único oferecido pelo microcontrolador torna-se uma ótima alternativa para gerar este nome.

O próximo passo foi especificar um *kit* de desenvolvimento que utiliza o microcontrolador definido. Para o solução proposta foi escolhido o kit *LPC1769 LPCXPRESSO BOARD* [33] do fabricante *Embedded Artists™* como ferramenta de desenvolvimento, apresentado na Figura 4.2. A opção em utilizar esta placa foi pautada nos seguintes itens:

- Circuito mínimo: Todo o circuito mínimo necessário para o funcionamento do LPC1769 como oscilador e alimentação já está validado na placa;
- Gravador e depurador: A placa escolhida já possui todos os recursos necessários para a gravação e a depuração do microcontrolador embutidos, portanto não necessita de nenhuma ferramenta adicional;
- Acesso aos pinos do microcontrolador: Os principais pinos do microcontrolador estão disponibilizados em conectores de expansão, facilitando a integração com outros *hardwares* necessários;

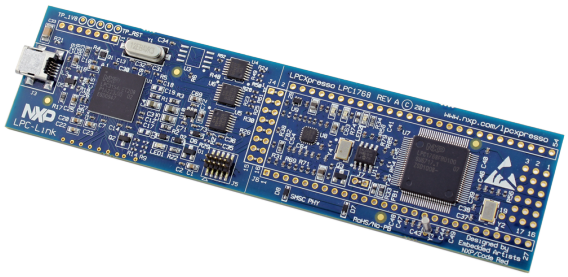

Figura 4.2: *LPC1769 LPCXPRESSO BOARD* [33].

O único quesito que a placa não atendeu foi em oferecer a possibilidade de conexão Wi-Fi, porém com a disponibilidade dos pinos do microcontrolador, esta deficiência foi superada integrando um módulo Wi-Fi externo.

4.1.2 Integrar módulo Wi-Fi

O módulo escolhido para a solução foi o RS9110-N-11-22 [34] da *Redpine*^{TM}. Oferece uma solução completa para Wi-Fi sendo compatível com os padrões 802.11 b/g/n. Possui endereço MAC integrado e pode ser conectado em um dispositivo *host* através de um barramento de comunicação do tipo *Serial Peripheral Interface* (SPI) ou através de uma interface *Universal Asynchronous Receiver/Transmitter* (UART). O módulo possui ainda um pino de saída chamado INTR, que pode ser conectado ao *host* para gerar uma interrupção (evento) sempre que um novo pacote de dados é recebido pela Wi-Fi, característica totalmente alinhada com a ideia de *event-driven* proposta neste trabalho.

A Figura 4.3 apresenta um diagrama em blocos do módulo e as possibilidades de integração com o *host*. O aspecto mais relevante do módulo para esta contribuição é a capacidade de acessar diretamente a camada de adaptação Wi-Fi MAC, ou seja, o módulo nesta condição se limita em controlar apenas os *bytes* de preâmbulo e o CRC do quadro *Ethernet*. Todas as demais informações devem ser enviadas integralmente pelo dispositivo *host*. Essa característica é vital para este trabalho, uma vez que a proposta central é avaliar um novo modelo de Internet em um cenário IoT, então qualquer dependência do módulo Wi-Fi com o modelo atual acima da camada de enlace (TCP/IP) seria prejudicial ao trabalho.

4.2 Estabelecer conexão Wi-Fi

Com o *hardware* definido, os esforços foram em criar uma conexão Wi-Fi entre um computador e o nó IoT em desenvolvimento. Esta etapa foi resumida em configurar um roteador para criar uma rede Wi-Fi para o projeto, conectar um computador e o sistema embarcado à rede criada, e enviar um pacote em *broadcast* do nó IoT para a rede.

Nesta fase, teve inicio o desenvolvimento do *firmware* da solução, sendo as primeiras atividades ligadas a configuração do módulo Wi-Fi. O software de desenvolvimento utilizado durante toda a elaboração do *firmware* desta contribuição foi o LPCXpresso^{TM} v7.7.2 379.

O fabricante do módulo disponibiliza pacotes de *firmware* para controle e configuração do módulo,

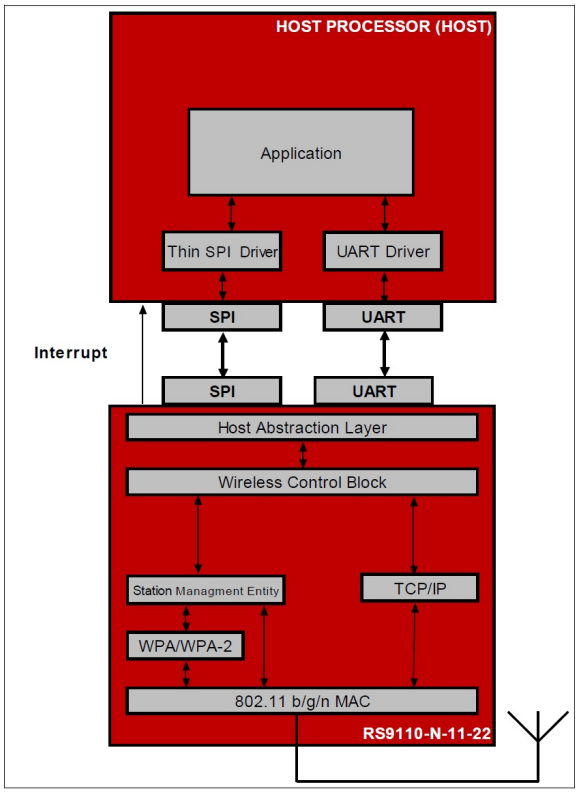

Figura 4.3: Diagrama em blocos de uma solução com o RS9110-N-11-22 [34].

cabendo ao desenvolvedor tarefas de adaptação para a realidade da aplicação em desenvolvimento, as atividades realizadas neste contexto foram:

- Inicializar a interface de comunicação SPI do microcontrolador utilizada para a integração com o módulo;
- Configurar microcontrolador para gerar interrupção quando houver algum evento no pino ISR do módulo;
- Embarcar os pacotes de *firmware*;

Na sequência, foi criado um processo para conexão do módulo na rede. Este processo trata-se de uma máquina de estados responsável pelas configurações iniciais no módulo visando autenticar e conectar na rede selecionada. Também realiza a leitura do endereço MAC do módulo e disponibiliza para a aplicação utilizar conforme a necessidade. O roteador utilizado na rede possui um recurso que apresenta o endereço MAC de todos os dispositivos conectados. A Figura 4.4 apresenta a lista de endereços visualizada no roteador após a máquina de estado realizar todos os processos de inicialização e conexão. Em destaque, o endereço MAC do módulo.

Com a conexão estabelecida, esta etapa foi finalizada com o envio de mensagens em *broadcast* do nó IoT. A Figura 4.5 apresenta um pacote capturado pelo Wireshark no computador conectado a rede. O campo de destino está endereçado como *broadcast* (FF-FF-FF-FF-FF-FF), no campo de origem

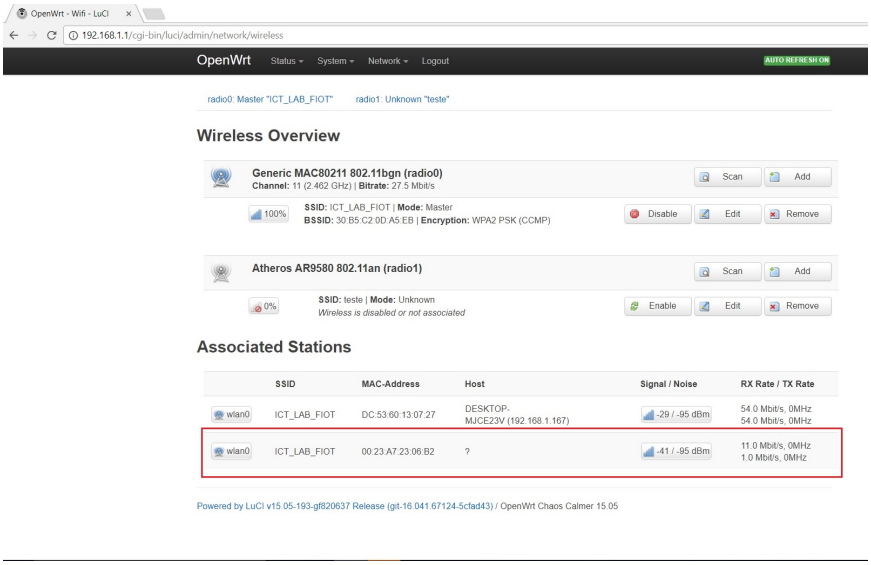

Figura 4.4: Dispositivos conectados ao roteador.

está com um endereço MAC fornecido pelo módulo Wi-Fi. O tipo do *payload* esta com um numero qualquer (1234) e no *payload* foi colocado a mensagem *Hello World!*. Desta forma foi validada que a estrutura construída até então suporta enviar qualquer tipo de informação no *payload* de um quadro Wi-Fi, inclusive mensagens de um novo modelo de Internet.

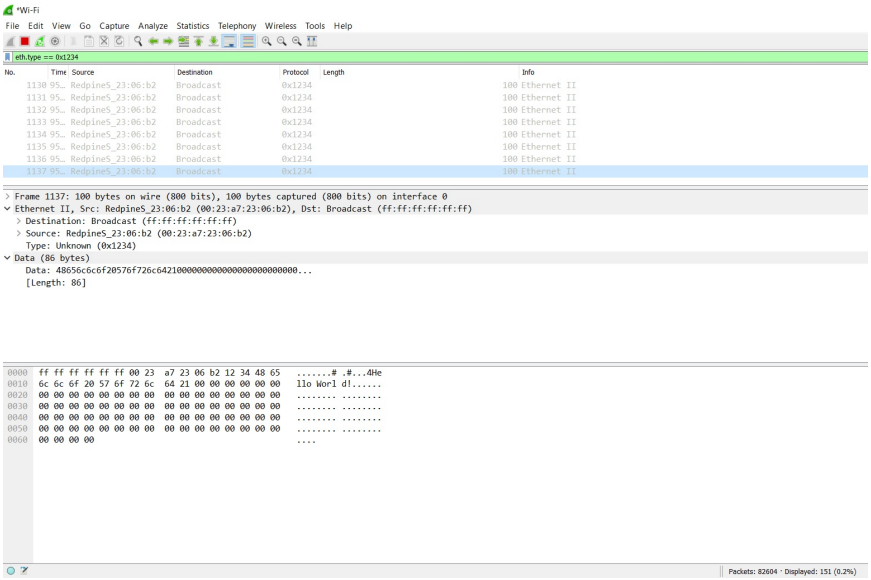

Figura 4.5: Pacote *Hello World!* capturado pelo Wireshark.

4.3 Integrar EventOS e EPGS ao *hardware*

Inicialmente, o EPGS não foi idealizado para trabalhar em um sistema *event driven*, apesar de todos os processos possuírem relações direta com eventos gerados pelo sistema. Por exemplo, após a inicialização do EPGS, a primeira etapa para estabelecer a comunicação com o PGCS é o envio de mensagens

Hello em *broadcast*. Este envio está condicionado a um intervalo de tempo previamente estabelecido, sendo repetido até receber uma mensagem do PGCS. Ou seja, o envio do *Hello* esta associado diretamente com dois eventos distintos, a finalização de um intervalo de tempo e a recepção de um pacote específico pela rede Wi-Fi. Outro exemplo é durante o processo de publicação de medidas. Os dados são coletados e enviados pelo EPGS respeitando um intervalo de tempo, novamente evidenciando a dependência do serviço em relação aos eventos. As atividades desenvolvidas nesta etapa foram direcionadas em criar uma camada de adaptação entre o EPGS e o EventOS chamada de *EventEPGS*. A sequência desta seção detalha as ações realizada por esta nova camada e os recursos utilizados.

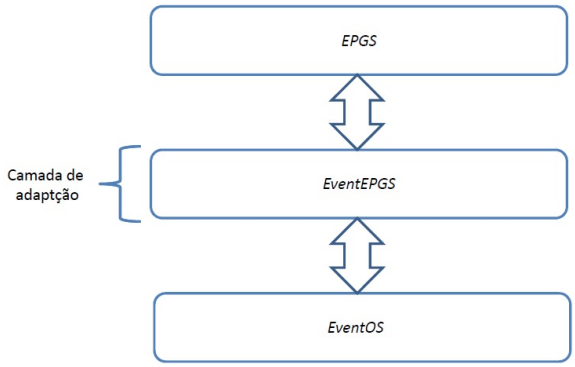

Figura 4.6: Camada de adaptação *EventEPGS*.

4.3.1 Identificadores para o EPGS

No capítulo 3, foi apresentado que o EPGS é endereçado na rede pela combinação de três nomes distintos: o nome do *hardware* onde o EPGS está embarcado (HID), o nome do sistema operacional(OSID) e o nome do processo (PID).

Nesta contribuição, o HID foi associado a um numero serial único encontrado no LPC1769. O número é formado por 128 bits e é indicado pelo fabricante do microcontrolador para ser utilizado como um identificador do dispositivo dada sua singularidade. Utilizando este número, foi evitada a repetição dos nomes dos dispositivos na rede FIoT garantindo que pelo menos um SVN seja único na rede.

O OSID refere-se ao sistema operacional, neste caso o EventOS, utilizado para controlar os processos do EPGS e os recursos de *hardware*. Foi criado no SO uma *string* constante, definida como "*EventOS*" para ser utilizada como o nome do sistema operacional, ou seja, todas soluções que utilizarem o EventOS terão o SVN relativo ao sistema operacional originado a partir desta *string* criada.

O PID é relativo ao processo executado pelo EPGS, sendo que, este nome já está definido internamente no serviço. Portanto, não resultou em nenhum esforço de adaptação ao sistema proposto. A Figura 4.7 apresenta o exemplo de dois nós da rede FIoT com as mesmas características e seus respectivos nomes.

Apesar dos dois nós utilizarem o mesmo serviço (EPGS), o mesmo SO (EventOS) e o mesmo *hardware* (LPC1769), a singularidade do HID garante que o *name binding* final dos dois nós sejam diferentes.

Figura 4.7: Nomeação do EPGS.

4.3.2 Inicialização do EPGS

Na inicialização do EPGS deve ser realizada a alocação das variáveis de controle, o registro dos nomes utilizados para gerar os SVNs e o registro das características que descrevem a aplicação. O EPGS disponibiliza quatro funções de *firmware* para as configurações iniciais:

- *initEPGS()*: Este método é responsável por alocar memória para a estrutura de variáveis utilizadas pelo EPGS;

- *setHWConfigurations()*: Configura os identificadores do *hardware* onde o EPGS está embarcado;

- *addKeyWords()* e *addHwSensorFeature()*: Utilizadas para criar descritivos da aplicação IoT. O método *addKeyWords()* denota uma característica geral da aplicação, por exemplo, pode ser utilizada a *string* "Temperatura" para indicar que a tarefa principal do nó é realizar medidas relacionadas a temperatura do ambiente. Já o método *addHwSensorFeature()* adiciona descrições mais específicas do serviço oferecido pelo nó, como por exemplo temperatura máxima e mínima que o nó é capaz de medir, ou ainda, a resolução da medida.

Com este conhecimento, foi implementado na camada *EventEPGS* a função de inicialização chamada de *init_EventEPGS()*. A Figura 4.8 apresenta algumas linhas de código da função.

Obviamente, a primeira ação realizada pela função foi alocar as variáveis de controle do EPGS utilizando o método *initEPGS()*. Em seguida, as variáveis *pHID*, *pSOID* e *pMAC* receberam do sistema os valores de HID, OSID e do endereço MAC, respectivamente, para realizar a configuração dos identificadores de *hardware*. A Figura 4.8 mostra que as variáveis citadas foram utilizadas como argumento da função *setHWConfigurations()*. Desta forma, a contribuição que aqui está sendo apresentada, fornece para o EPGS os nomes necessários para a geração dos SVNes, além do endereço MAC do módulo Wi-Fi lido durante a inicialização.

E por fim, foram registrados os descritivos da aplicação. Os descritivos estão completamente ligados ao tipo da aplicação embarcada, por exemplo, se a aplicação for de medida de luminosidade, os descri-

```
void init_EventEPGS(void)
{
    char* pHID;
    char* pSOID;
    char* pMAC;
    char* pKey;
    char* pType;
    char* pValue;
    char n,i;


    /*Init epgs*/
    initEPGS(&tagNgEPGS, 1);


    /* Config names */
    pHID = system_getSerialNumber();
    pSOID = EventOS_getName();
    pMAC = Wifi_getMac();
    setHwConfigurations(&tagNgEPGS, pHID, pSOID, "Wi-Fi", "eth1", pMAC);


    /* Config App keys */
    pKey = get_keyApp();
    addKeyWords(&tagNgEPGS, pKey);

    /* Config App features */
    n = get_numberFeaturesApp();
    for(i = 0; i < n; i++)
    {
        pType = get_featureTypeApp(i);
        pValue = get_featureValueApp(i);
        addHwSensorFeature(&tagNgEPGS, pType, pValue);
    }
```

Figura 4.8: Bloco de inicialização do EPGS.

tivos estarão relacionados com esse tipo de medida. Se a aplicação envolver medida de temperatura, os descritivos serão relativos a esta tarefa. Ou seja, idealmente a aplicação deve fornecer os descritivos que deseja registar, cabendo ao *EventEPGS* apenas transferir estas informações para o EPGS. A Figura 4.8 ilustra que esta condição foi respeitada nesta implementação, os descritivos são recebidos da aplicação e enviados para o EPGS.

4.3.3 Mapeamento e configuração dos eventos

Após as rotinas de inicialização, o EventOS é configurado para o estado *sleeping*, a CPU paralisa o processamento das instruções e o sistema entra em um modo de baixo consumo aguardando uma nova interrupção de *hardware*. Com o acontecimento da interrupção, a CPU retorna para o modo ativo e todo o processamento é deslocado para o tratamento da interrupção detectada, resultando na publicação de um ou mais eventos referentes a interrupção processada. Na sequência, o escalonador do EventOS inicia a notificação das tarefas assinantes (*scheduling*) para o processamento do evento publicado (*delivering*).

O EPGS padroniza algumas etapas para estabelecer a comunicação com o cliente responsável em consumir os dados gerados pela aplicação IoT embarcada na solução. Entender como estas etapas podem ser associadas a eventos de interrupção geradas pelo *hardware* do microcontrolador foi vital

para realizar a integração entre o EPGS e o EventOS.

O primeiro bloco analisado refere-se as etapas de *Initialization*, *Exposition & Discovery* e *Service Offer* descritos no capítulo 3. A primeira ação do EPGS após a inicialização é o envio de mensagens *Hello* para a rede. As mensagens devem ser enviadas de forma periódica respeitando uma janela de tempo, ou seja, o disparo da mensagem pode ser associada a um evento gerado ao final de uma contagem de tempo. O EPGS deve enviar repetidamente a mensagem *Hello* até que uma mensagem *Hello* do PGCS seja recebida. Portanto, a recepção de um pacote específico na interface Wi-Fi pode ser utilizado para encerrar o envio de mensagens *Hello* e executar a próxima etapa do EPGS. Na etapa de *Exposition & Discovery* não existe nenhuma ação do EPGS sendo desprezada nesta atividade. A etapa *Service Offer* inicia com o EPGS realizando um envio da mensagem de oferta de serviço para o PGCS, considerando apenas as ações do EPGS. O envio da mensagem de oferta é a ação executada após a recepção do pacote *Hello* enviado pelo PGCS, ou seja, o evento utilizado pelo EPGS para finalizar o envio de mensagens *Hello* pode ser o mesmo que dispara o envio da oferta de serviço.

No bloco *Service Acceptance*, o EPGS deve aguardar a mensagem de *Notify* (*Service Acceptance*) para disparar um *Subscribe* (*Service Acceptance*), ou seja, novamente a recepção de um comando pela interface Wi-Fi pode ser utilizada para gerar o evento necessário para a troca de mensagens descritas. O último estágio desta etapa está em receber uma mensagem de *Delivery* (*Service Acceptance*) para armazenar os identificadores do cliente interessado nas medidas da aplicação. Portanto, o evento de recepção pode ser aproveitado para encerrar esta etapa e iniciar o processo de envio de medidas.

A última etapa projetada para o EPGS é o envio de medidas de forma periódica obedecendo um intervalo de tempo, algo bem parecido com o cenário do envio de *Hello*. Assim sendo, o evento citado para controle de tempo pode ser empregado nesta etapa também. A Figura 4.9 apresenta o diagrama de estados correlacionando os diferentes estágios do EPGS e os eventos necessários para a execução.

Duas características devem ser observadas neste diagrama: a primeira é que com apenas dois eventos de *hardware*, um relacionado a controle de tempo e outro com a recepção de mensagens no Wi-Fi, é possível executar todas as etapas definidas pelo EPGS. A segunda observação é que em todas as etapas onde existe a necessidade de esperar algum tipo evento, a CPU esta configurada para o modo de baixo consumo. Característica totalmente alinhada com a proposta deste trabalho.

Feito este mapeamento, foram configurados no microcontrolador os recursos de *hardware* para gerar os dois eventos definidos. Para controle do tempo foram utilizados os recursos de *Timer* presentes no LPC1769. Basicamente, o *Timer* é um contador baseado na frequência do oscilador utilizado pelo sistema embarcado, gera uma interrupção sempre que o contador atinge um valor previamente configurado. Para o evento relacionado com a recepção de pacotes, foi aproveitada a configuração realizada na fase de integração do Wi-Fi, que resulta em uma interrupção no microcontrolador sempre que houver alterações no pino ISR do módulo, sinalizando a recepção de um novo pacote na interface.

4.3.4 Criar o assinante EventEPGS

A última atividade da integração entre EPGS e EventOS foi criar na camada *EventEPGS* uma tarefa para assinar os eventos configurados no sistema e executar as ações correspondentes. Com a ocorrência de um novo evento, o escalonador do EventOS notifica a tarefa assinante criada, que por sua vez, deve

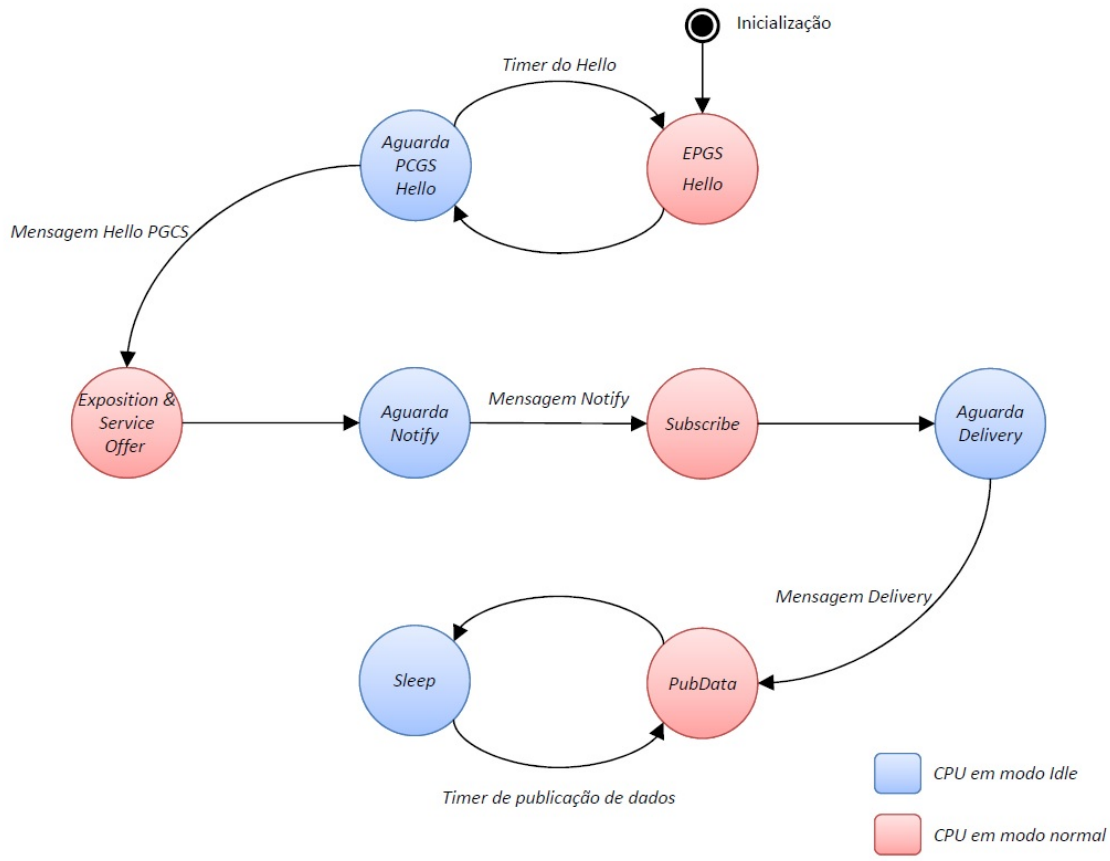

Figura 4.9: Estados do EPGS *versus* Eventos.

chamar as funções do EPGS correspondentes ao estágio da comunicação com o PGCS. Portanto, é papel deste assinante controlar as diferentes etapas propostas no FIoT. Alguns exemplos de funções disponíveis no EPGS para executar as ações necessárias em cada etapa:

- *RunHello()*: Método criado no EPGS para formatar e enviar uma mensagem *Hello*;
- *RunPubServiceOffer()*: Envia uma mensagem de oferta de serviço para o PGCS;
- *RunPublishData()*: Cria uma mensagem com os dados coletados pelo nó endereçada para o cliente IoT que contratou o serviço;

A tarefa assinante desenvolvida foi chamada de *process_EventEPGS()* e sua construção foi baseada em uma máquina de estados. Com a ocorrência de um novo evento, o process_EventEPGS() identifica em qual estado a comunicação está, chama os métodos correspondentes do EPGS e atualiza o estado da máquina conforme a natureza do evento. A Figura 4.10 apresenta uma visão geral da tarefa desenvolvida.

Com a criação desta estrutura de controle, foi possível também otimizar a geração de eventos no *hardware*. Por exemplo, após a finalização do envio de mensagens *Hello*, o sistema proposto não precisa

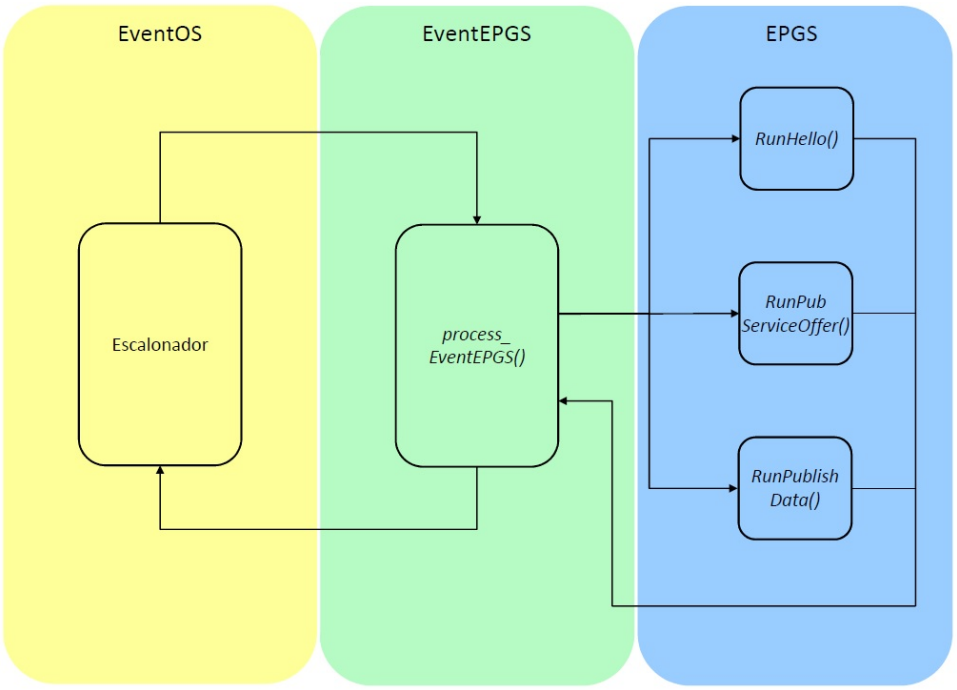

Figura 4.10: Camada de adaptação *EventEPGS*.

de nenhum evento relacionado a *Timer* até que a etapa de envio de medidas para o cliente seja iniciado. Foi desabilitado no process_EventEPGS() o *Timer* do microcontrolador quando não aplicado, levando a uma diminuição no número de eventos gerados no sistema. A CPU foi retirada do modo de baixo consumo somente com a ocorrência de um evento realmente válido para o estado vigente da aplicação, contribuindo para a redução no consumo de energia do nó de uma forma geral.

Portanto, com a criação deste assinante foi finalizado a integração do EPGS com o sistema. Os nomes para gerar os SVNes foram obtidos, a inicialização do sistema foi realizada, e principalmente, os eventos necessários para possibilitar a utilização do EventOS foram criados e foram assinados por uma tarefa que permitiu executar todas as etapas exigidas pelo modelo proposto na NovaGenesis.

4.4 Criar aplicação IoT

E, finalmente, encerrando as implementações desta contribuição, foi criada uma aplicação para gerar medidas de temperatura para serem publicadas pelo EPGS após um novo contrato de prestação de serviço. Olhando para o contexto de uma rede IoT, esta aplicação é de fato a principal atividade desempenhada pelo nó. A partir deste bloco que são geradas as medidas que são utilizadas no cenário de Internet das coisas.

O desenvolvimento desta aplicação gerou algumas funções para serem integradas no sistema implementado até então:

- *init_App()*: Inicia as configurações do *hardware* do microcontrolador para realizar as leituras de temperatura. Esta função foi adicionado na inicialização do sistema;

- *get_keyApp()*: Como apresentado na inicialização do EPGS, algumas características gerais da aplicação deve ser fornecida para descrever o serviço na rede e possibilitar novos contratos. Esta função foi projetada para retornar as *strings* referentes a estas características;

- *get_numberfeaturesApp()*: Retorna o número de descritivos que a aplicação possui para detalhar o serviço;

- *get_featuresTypeApp()* e *get_featuresValueApp()*: Assim como no *get_keyApp()*, o EPGS necessita de um detalhamento da aplicação para oferecer para a rede. Estas funções foram idealizadas para retornar para o sistema as *strings* que detalham o serviço;

- *get_measureApp()*: Este método é responsável por realizar a medida de temperatura. Esta função foi utilizada durante a etapa de publicação de dados do EPGS;

Desta maneira, foi integrado ao cenário desenvolvido, uma aplicação para gerar medidas que permitiu o teste de todas as fases de comunicação do modelo, desde a inicialização com o envio das mensagens *Hello* até a entrega de medidas lidas para um cliente IoT.

4.5 Dificuldades encontradas nas implementações

Durante o desenvolvimento das implementações, foram enfrentadas algumas dificuldades relacionadas a especificação do *hardware*, e, principalmente, relacionadas a adaptação do EPGS para uma plataforma embarcada. Esta seção tem como objetivo apresentar estas dificuldades e as soluções adotadas em cada caso.

A fase de especificação do *hardware* foi composta de duas etapas: a definição do microcontrolador e a definição de um módulo Wi-Fi. A primeira etapa foi, de certa forma, simples. Sabendo que, até então, o EventOS possue compatibilidade apenas com o LPC1769, escolher este microcontrolador acabou sendo natural. Por outro lado, a especificação do módulo Wi-Fi foi uma tarefa bastante complicada. Considerando que o módulo selecionado deveria permitir o acesso direto a camada de adaptação do enlace, a busca tornou-se muito específica. A principal dificuldade foi encontrar nas documentações dos fabricantes de módulos Wi-Fi, informações que asseguravam as características exigidas neste trabalho, despendendo muito tempo em analise de documento e teste.

Na fase de adaptação do EPGS, as principais dificuldades encontradas estavam relacionadas as limitações imposta pelo sistema embarcado. Até o inicio de deste trabalho, o EPGS nunca havia sido embarcado em microcontrolador real. Todos os testes foram realizados em ambiente simulado. Porém, ao embarcar em um sistema real, houve algumas divergências no funcionamento do serviço. Onde, o principal problema foi relacionado ao gerenciamento de memória. O EPGS foi construído baseado em alocação dinâmica de memória, conforme novos processos são iniciados no serviço, a memória de dados é alocada ou liberada conforme a necessidade. No entanto, o mecanismo de gerenciamento de memória não apresentou um funcionamento muito eficiente. Ocasionando um problema conhecido como *leak* de

memória, quando a memória alocada para um processo não é liberada, apesar de não ser mais necessária. A solução foi uma varredura completa no EPGS para identificar e corrigir os pontos de alocação e liberação de memória;

Capítulo 5

Experimentos

Este capítulo descreve os experimentos realizados para validar as implementações apresentadas no Capítulo 4. Os primeiros testes foram concentrados em verificar as diferentes fases da comunicação do EPGS, desde a inicialização até a fase de envio de medidas para o cliente IoT. Foi avaliada a integridade dos comandos gerados pelo EPGS e analisado o comportamento da camada EventEPGS. O segundo experimento avaliou a robustez do sistema submetendo o cenário à várias horas de teste. E, finalmente, foram analisados os gastos de memória e processamento para executar cada fase do EPGS. A Figura 5.1 apresenta uma foto do cenário criado para os teste.

Figura 5.1: Cenário do experimento.

No primeiro laptop (*i*) foi executado o cliente IoT (*IoTTestApp*) e o *core* da NovaGenesis, incluindo os serviços PGCS e NRNCS. O outro laptop (*ii*) depurou as diferentes fases de funcionamento do nó FIoT (*iii*) utilizando os recurso de geração e visualização de *logs* disponibilizados pelo LPCXpresso^{TM}. Foi utilizado ainda um roteador (*iv*) para criar uma rede Wi-Fi para conectar os serviços da NG.

5.1 Inicialização

A primeira situação analisada foi a inicialização da camada EventEPGS realizada através da função *init_EventEPGS()*. Basicamente esta função deve iniciar o EPGS com os parâmetros obtidos do sistema, configurar o *hardware* para criar os eventos necessários para o funcionamento do EventOS e assinar os eventos. A Figura 5.2 apresenta os *logs* capturados no LPCXpresso™ após a inicialização do EventEPGS.

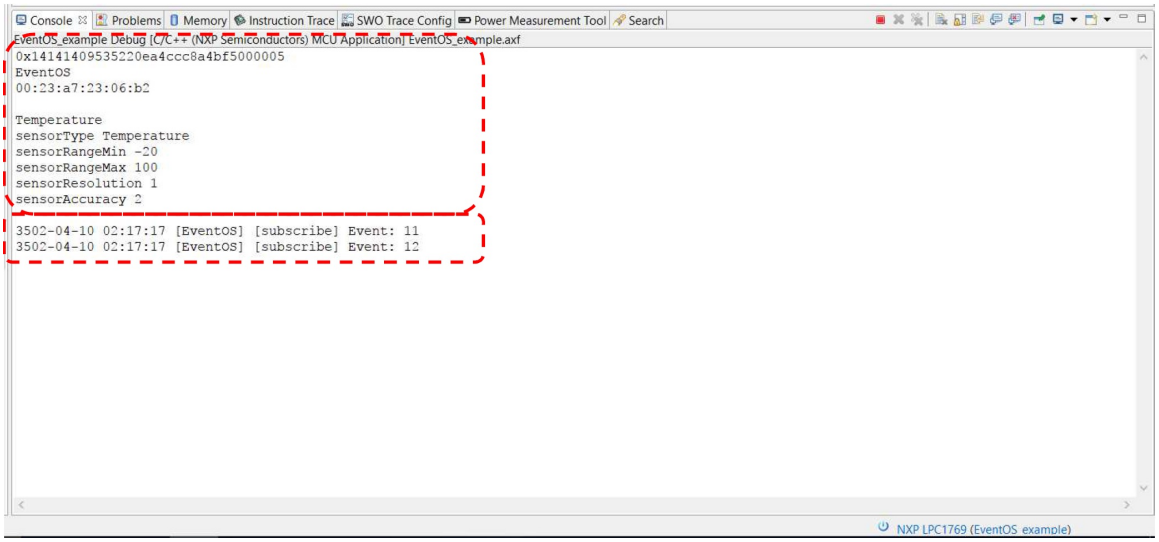

Figura 5.2: *Logs* de inicialização.

O primeiro bloco destacado apresenta os valores obtidos após a leitura do número serial e do nome do sistema operacional, utilizados posteriormente para o cálculo do HID e do OSID, e o endereço MAC obtido durante a inicialização do módulo Wi-Fi integrado ao *hardware* da solução. Na sequência são apresentados os descritivos fornecidos pela aplicação IoT, utilizados pelo EventEPGS para caracterizar o serviço prestado pelo nó. Já o segundo bloco aponta os *logs* referente a assinatura dos eventos pelo *process_EventEPGS*.

Com este teste foi validada a inicialização do sistema, pois o EPGS foi configurado com parâmetros obtidos da partir da leitura do *hardware* e dos descritivos da aplicação, conforme especificado na fase de implementação. Além disso, os eventos criados no EventOS foram assinados pela função *process_EventEPGS()* garantido que a ocorrência de qualquer evento configurado no sistema resulte na notificação desta função.

Este experimento comprova que o modelo proposto neste trabalho fornece uma solução capaz de operar com qualquer tipo de serviço IoT. Uma vez que, é a própria aplicação IoT que fornece para o EPGS seus descritores de funcionamento, novas aplicações possuirão novos descritores, criando novos serviços IoT na rede. Porém, no modelo desenhado neste trabalho, estes descritores são transparentes, sendo que, para criar novas aplicações IoT de diferentes naturezas, basta simplesmente acomodar esta nova aplicação na camada EventEPGS.

5.2 *process_EventEPGS()*

Neste experimento, foi verificada a dinâmica das atividades executadas no EventEPGS. O método *process_EventEPGS* foi projetado para ser notificado pelo escalonador do EventOS na ocorrência de novos eventos e executar os processos baseado nas circunstâncias do EPGS. A Figura 5.3 apresenta os *logs* do LPCXpresso™ capturados durante toda a execução *process_EventEPGS()*.

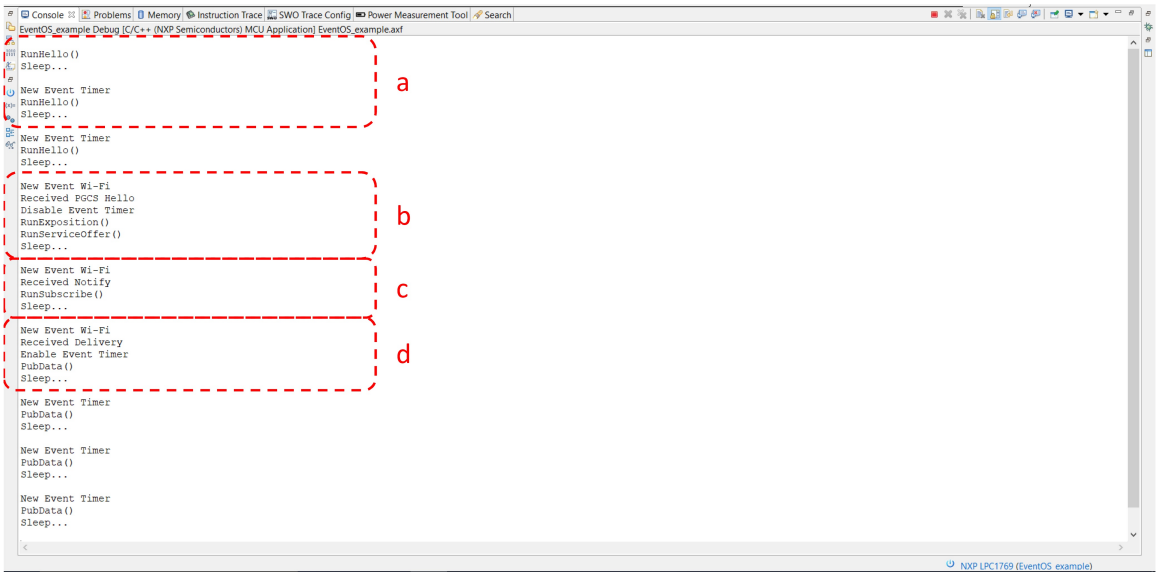

Figura 5.3: *Logs* do *process_EventEPGS*.

O bloco *(a)* mostra os *logs* capturados durante o envio de mensagens *Hello* do EPGS ocorrido no inicio da comunicação. Logo após a inicialização, o EventEPGS disparou uma mensagem de *Hello* e entrou em modo *sleep*, permanecendo até a ocorrência de um evento do tipo *Timer*, acarretando em uma nova mensagem *Hello* e posteriormente o retorno ao modo de baixo consumo. Este bloco valida as implementações do envio de mensagens *Hello*, pois estão condicionadas a um evento de *Timer* e o sistema aguarda a ocorrência em modo de baixo consumo.

No bloco *(b)* são apresentados os *logs* resultantes da chegada de uma mensagem *Hello* do PGCS, do envio das mensagens de *Exposition*, do envio do *Service Offer* e do retorno para modo *sleep*. Com o recebimento de uma mensagem *Hello* do PGCS, o EventEPGS suspende o envio do *Hello* periódico, dispara as mensagens de exposição de recursos, de oferta de serviço e entra em modo de baixo consumo para aguardar o recebimento de um *Notify*. Um detalhe apresentado entre o *Hello* do PGCS e o *Exposition* é a suspensão do evento de *Timer* que não será utilizado até a fase de publicação de medidas para o cliente IoT.

Em *(c)* ocorreu um evento de Wi-Fi com a chegada de uma notificação, ocasionando a mensagem de *Subscribe* e a *posteriori* o modo de baixo consumo. Estes *logs* mostram que o sistema permaneceu no modo *sleep* até receber uma mensagem de *Notify* do PGCS. Quando o EventEPGS foi notificado com o evento, respondeu com uma mensagem de *Subscribe*. Ao final, o sistema retornou para o estado inativo para aguardar o *Delivery* do PGCS.

O último bloco *(d)* inicia com o evento de recepção do *Delivery*, seguido pela reconfiguração dos

eventos de *Timer* e pela publicação de uma medida de temperatura, novamente o sistema entra no modo *sleep* até ocorrer um evento de *Timer* provocando uma nova publicação de temperatura. Com o recebimento do *Delivery* é dever do EPGS enviar periodicamente as medidas coletadas para o cliente IoT, por isso o evento de *Timer* volta a acontecer para sincronizar a publicação de medidas de temperatura.

Este experimento mostrou que o bloco EventEPGS executou todas as fases necessárias para o EPGS estabelecer um contrato e fornecer medidas para um cliente IoT. Apresentou também que todas as fases que necessitam aguardar algum evento, o sistema foi mantido em modo de baixo consumo, conforme prevê a programação orientada a eventos. E finalmente o modelo desenhado possibilitou reconfigurar o *hardware* durante as diferentes etapas de funcionamento para gerar somente os eventos necessários para cada fase, otimizando ainda mais o sistema.

Este experimento comprova que o modelo proposto neste trabalho, fornece uma solução que dispensa qualquer tipo de configuração para iniciar um novo serviço na rede NG. Comprova também, que o modelo desenvolvido esta alinhado com os ideais de uma programação orientada a evento, proporcionando uma melhor eficiência energética do nó IoT.

5.3 Experimentação das medidas de temperatura

Neste ensaio foram criadas algumas situações para verificar a qualidade do serviço de publicação de medidas de temperatura oferecido pelo EPGS. O objetivo deste experimento foi avaliar a estabilidade do serviço e a qualidade das medidas enviadas.

Para este teste foi utilizado um segundo nó IoT na rede. Desta forma, a aplicação cliente estabeleceu contrato com dois serviços de EPGS, recebendo medidas de temperatura de dois sensores distintos. A Figura 5.4 apresenta uma mensagem de publicação de temperatura capturada durante o teste.

```
ng -m --cl 0.1 [ < 1 s 28FD4420 > < 4 s 4C7CF9B2 5F472DA7 1A53F830 NULL > < 4 s 0BD95286 ED12F3ED 8E8B52EC 7EA46815 > ]
ng -p --notify 0.1 [ < 1 s 18 > < 1 s 94BF74F8 > < 1 s Measures.json > < 5 s pub 0BD95286 ED12F3ED 372ED1C1 EFCD1EC5 > ]
ng -info --payload 0.1 [ < 1 s Measures.json > ]
ng -message --type 0.1 [ < 1 s 1 > ]
ng -message --seq 0.1 [ < 1 s 2 > ]
ng -scn --seq 0.1 [ < 1 s CC34CC24 > ]

{Temperature: 26}
```

Figura 5.4: Mensagem de temperatura recebida pelo cliente IoT.

Estabelecido o contrato entre os nós IoT e o cliente, o cenário de teste foi executado por aproximadamente 189,6 horas, com cada sensor gerando uma publicação por segundo, totalizando aproximadamente 1,296 milhões de medidas no período amostrado. A Figura 5.5 apresenta os resultados amostrados no cenário mostrado na Figura 5.6.

Com este teste, o EPGS demonstrou ser um serviço robusto, pois funcionou por muitas horas seguidas, gerando uma quantidade de dados relativamente alta, e mesmo assim, continuou entregando as medidas conforme o esperado. De certa forma, o sistema também provou ser escalável, pois quando foi ligado um segundo nó na rede, um novo serviço foi criado, cabendo ao cliente estabelecer um novo contrato para receber suas medidas. Este comportamento permite que o cliente contrate ou revogue recursos conforme a necessidade da aplicação, ou seja, a NG possibilita um sistema escalável e eficiente em relação a utilização dos recursos disponíveis da rede.

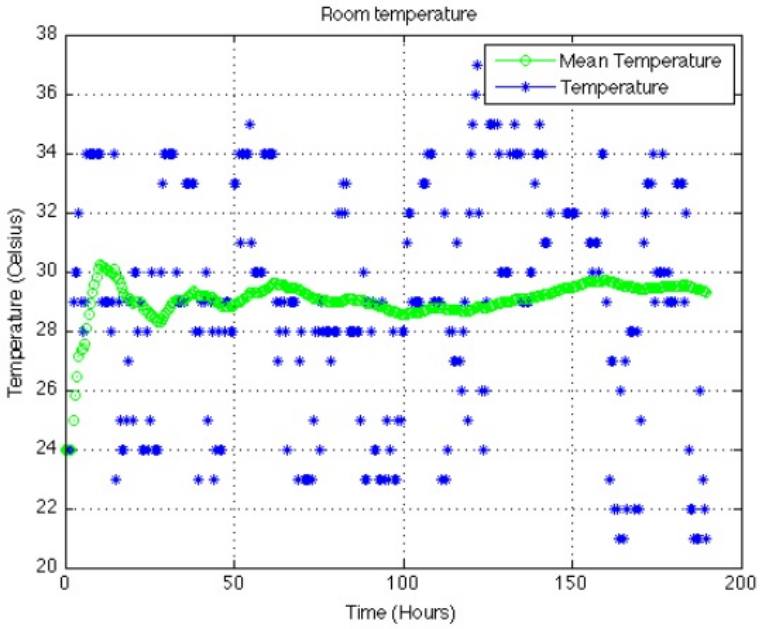

Figura 5.5: Gráfico das temperaturas amostradas [14].

Figura 5.6: Cenário com dois nós FIoT para medida de temperatura.

5.4 Consumo de memória e recursos computacionais

O último experimento realizado nesta dissertação foi mensurar os gastos de memória e de processamento para executar o EPGS no LPC1769. O consumo de memória ROM e RAM foi estimado pelo compilador do LPCXpresso^{TM} e duas situações distintas foram consideradas, somente a *stack* EPGS e o sistema completo incluindo EventOS, *stack* Wi-Fi, *drivers* de *hardware* do LPC1769 e o próprio EPGS.

Tabela 5.1: Consumo de memória experimentado

	ROM (bytes)	**RAM (bytes)**
EPGS *Standalone*	25.328	2.108
EPGS + EventOS + *drivers* + Wi-Fi	42.440	8.720
CCN-Lite	16.628	5.112
RPL + 6LoWPAN	53.412	27.739

A Tabela 5.1 mostra o consumo de memória da NG obtidos durante o experimento. Apresenta tam-

bém os valores de outras duas referências obtidas em [35]. O modelo proposto pela NG consome um total de 8.720 bytes de memória RAM, enquanto a solução RPL + 6LoWPAN necessita de 27.739 bytes para funcionar em conjunto com o RIOT e o *hardware* MSBA2, ou seja, a NovaGenesis proporciona uma economia de aproximadamente 69% na utilização da memória RAM do dispositivo embarcado. Outra vantagem experimentada da NG em relação ao RPL + 6LoWPAN está na utilização da memória ROM, enquanto a RPL + 6LoWPAN necessita de 53.412 bytes, a NG utiliza 42.440 bytes, uma redução de aproximadamente 20%. Em comparação ao CCN-Lite, o consumo de RAM da NG foi aproximadamente o mesmo, porém em relação a ROM houve acréscimo de 39% na utilização.

Na sequência foi estimado o numero de ciclos de máquina necessários para executar cada processo dentro do EPGS. Foi utilizado um recurso presente no LPCXpresso™ chamado de *cycle_delta* que basicamente conta o numero de ciclos da CPU gastos para executar um trecho de código do *firmware*.

Tabela 5.2: Comparação dos ciclos de CPU gastos na NG e no CCN-Lite

	Função	**Ciclos de CPU**
NG	*RunHello*	1.898.091
	RunExposition	3.833.896
	RunServiceOffer	3.375.442
	RunServiceAcceptance	2.348.335
	PubData	2.491.954
CCN-Lite	*memcmp ssse3*	14.002.814
	ccnl nonce find or append	7.525.050
	ccnl i prefixof c	4.062.659
	dehead	1.462.304
	ccnlcoreRXiorc	956.238
	ccnl extract prefix nonce ppkd	895.590
	memcpy ssse3	845.042

A Tabela 5.2 apresenta os valores experimentados para cada fase na NovaGenesis e os valores obtidos em [35] para o CCN-Lite. Como as operações não são as mesmas entre os dois modelos, não é possível comparar, porém, analisando a ordem de grandeza dos valores listados, o modelo proposto pela NG pode ser considerado uma boa alternativa.

5.5 Lições aprendidas

Esta seção tem como objetivo de alertar novas pesquisas correlatas sobre algumas lições aprendidas durante o desenvolvimento e teste deste trabalho. As lições aprendidas foram:

- Evitar a utilização de alocação dinâmica de memória em microcontrolador: Apesar de ser um recurso muito comum na construção de *softwares*, a alocação dinâmica de memória em aplicações microcontroladas ainda deve ser realizada com cuidado. As ferramentas utilizadas no desenvolvimento de aplicações embarcadas ainda não oferecem muitos recursos para avaliar o gerenciamento de memória. Portanto, identificar e corrigir problemas desta natureza torna-se complicado;

- Documentação é vital: Durante a especificação do módulo Wi-Fi foi gasto um tempo grande em teste. Tempo que poderia ter sido poupado caso houvesse documentos claros sobre as capacidades

dos módulos analisados. Então, construir um documento faça uma boa especificação do trabalho, é vital para a conclusão do mesmo;

- A simulação é muito importante, porém a prática também é necessária: As ferramentas de simulação são extremamente importante no desenvolvimento de novos trabalhos. Seja para simplesmente provar o funcionamento, ou ainda, emular condições de teste, a simulação apresenta cada vez mais resultados fidedignos para as aplicações. Porém, a prática revela algumas realidades não previstas em teste, ocasionando problemas não detectados. Por isso, se possível, a implementação prática de um trabalho deve ser levado em consideração.

Capítulo 6

Conclusão

Este trabalho teve como objetivo embarcar o EPGS em um *hardware* com características similares aos nós comerciais para analisar o funcionamento do modelo proposto pela NovaGenesis. Foi definida a utilização de um sistema operacional orientado a eventos para gerenciar a dinâmica dos processos na CPU e oferecer um sistema organizado e alinhado com diretrizes de baixo consumo de energia. As implementações envolvidas nesta dissertação foram: adaptar o EPGS para operar em conjunto com o sistema operacional definido, embarcar a solução em *hardware* e desenvolver uma aplicação para ser utilizada como referência para os testes e experimentações.

No Capítulo 1 foi apresentada uma introdução do cenário aonde essa contribuição se enquadra e os problemas enfrentados nos modelos atuais. O Capítulo 2 descreveu os princípios de funcionamento do EventOS, apresentando de forma resumida as principais características e os benefícios em utilizar este modelo associado a uma arquitetura ARM^{TM}. No Capítulo 3 foi apresentada uma visão geral dos principais serviços e conceitos da NovaGenesis, em especial, o serviço EPGS principal elemento desta contribuição.

No Capítulo 4 foi detalhada as implementações realizadas no trabalho, o processo de definição do *hardware* utilizado, e principalmente os esforços em adaptar o EPGS para trabalhar orientado a eventos. No Capítulo 5 foram apresentadas resultados experimentados no cenário construído ao longo deste trabalho, mostrando que o EPGS trabalhou em perfeita harmonia com um sistema orientado a eventos.

Este trabalho conclui que o EPGS mostrou-se um serviço robusto e confiável em ambiente laboratorial, mesmo após horas de teste. O *IoTTestApp* continuou recebendo as medidas de temperatura na mesma taxa configurada inicialmente. Desta forma, esta contribuição abre uma possibilidade de um trabalho futuro em analisar o desempenho do EPGS em soluções comerciais, considerando perda de pacotes por interferências do meio, problemas relacionados a latência de rede, enfim problemas enfrentados nas aplicações comerciais não tratados nesta dissertação.

A combinação do EventOS e EPGS concebeu um sistema engajado aos ideais de uma programação orientada a eventos. Todas as etapas definidas no EPGS aplicaram o conceito principal da metologia de programação orientada por eventos que é a inversão no controle do sistema. As ações do EPGS foram disparadas por um evento, sendo que a CPU sempre foi mantida em modo de baixo consumo de

energia enquanto aguardava. Outra característica implementada no sistema, foi a personalização dos eventos conforme o contexto da aplicação. Estas funcionalidades conferem ao modelo desenvolvido características de eficiência energética, mantendo a CPU em modo ativo somente quando necessário, essencial para sistemas alimentados por baterias.

Esta contribuição também concluiu que ao utilizar um número serial único do *hardware* é possível garantir a singularidade do serviço na rede, ainda que o novo nó utilize o mesmo sistema operacional e execute as mesmas atividades de outros nós já presentes na rede, envolver um número único para nomear o serviço, assegura uma nova combinação de SVNes, portanto, um novo serviço. Assim, novos serviços podem ser criados de forma simples e eficiente na rede IoT da NovaGenesis, garantindo um sistema escalável onde o serviço cliente pode contratar e revogar recursos conforme sua necessidade. Neste ponto surge outra possibilidade de trabalho futuro, que é escalar ainda mais o cenário proposto nesta dissertação utilizando nós simulados para analisar o comportamento da rede NG com o aumento do número de serviços IoT.

Um trabalho futuro a esta contribuição pode ser analisar o cenário experimentado em outras camadas de enlace, como por exemplo LoRa e Bluetooth, sendo necessário adaptar o EPGS e o PGCS para trabalhar com estas novas camadas. A partir destas modificações, podem ser levantadas métricas de rede comparando o desempenho da rede NG nas diferentes camadas de enlace.

E finalmente, a solução demonstrou ser uma boa alternativa em termo de consumo de memória e processamento se comparada com as soluções apresentadas em [35]. Os consumos referente a memória foram compatíveis com a realidade dos nós IoT e o numero de ciclos gastos para executar as diferentes etapas do EPGS esta na mesma ordem de grandeza de outras referências, portanto, viabiliza a utilização do modelo proposto pela NG.

Referências Bibliográficas

[1] LEIGHTON, T. *Improving performance on the internet, Commun. ACM,* vol. 52, no. 2, pp. 44–51, 2009.

[2] JAIN, R. *Internet 3.0: ten problems with current internet architecture and solutions for the next generation,* in *Military Communications Conference, 2006. MILCOM 2006. IEEE,* IEE, p. 1-9, 2006

[3] LIU, X.; BAIOCCHI, O. *A comparison of the definitions for smart sensors, smart objects and Things in IoT,* in *2016 IEEE 7th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON),* 2016

[4] ORTIZ, A.; HUSSEIN, D.; PARK, S.; HAN, S.; CRESPI, N.; *The Cluster Between Internet of Things and Social Networks: Review and Research Challenges,* in *IEEE Internet of Things Journal,* vol. 1, no. 3, pp. 206-215, 2014.

[5] ALBERTI, A. *A conceptual-driven survey on future internet requirements, technologies, and challenges, Journal of the Brazilian Computer Society,* vol. 19, no. 3, pp. 291–311, 2013.

[6] KIBRIA, M.; JARWAR, M.; ALI, S.; KUMAR, S.; CHONG, I. *Web objects based energy efficiency for smart home IoT service provisioning, 2017 Ninth International Conference on Ubiquitous and Future Networks (ICUFN),* 2017.

[7] AHLGREN, B.; ARANDA, P.; CHEMOUIL, P.; OUESLATI, S.; CORREIA, L.; KARL, H.; SÖLLNER, M.; WELIN, A. *Content, connectivity, and cloud: ingredients for the network of the future,* IEEE Communications Magazine, vol. 49, no. 7, pp. 62-70, 2011.

[8] HAN, D.; ANAND, A.; DOGAR, F.; LI, B.; LIM, H.; MACHADO, M.; MUKUNDAN, A.; WU, W.; AKELLA, A.; ANDERSEN, D. G.; BYERS, J. W.; SESHAN, S.; STEENKISTE, P. *Xia: Efficient support for evolvable internetworking,* in *Proc. of the 9th USENIX Conf. on Networked Systems Design and Implementation,* ser. NSDI'12. USENIX, 2012, pp. 23–23.

[9] DANNEWITZ, C.; KUTSCHER, D.; OHLMAN, B.; FARRELL, S.; AHLGREN, B.; KARL, H. *Network of information (netinf) - an information-centric networking architecture, Comput. Commun.,* vol. 36, no. 7, pp. 721–735, Apr. 2013.

[10] MCKEOWN, N.; ANDERSON, T.; BALAKRISHNAN, H.; PARULKAR, G.; PETERSON, L.; REXFORD, J.; SHENKER, S.; TURNER, J. *Openflow: Enabling innovation in campus networks, SIGCOMM Comput. Commun. Rev.,* vol. 38, no. 2, pp. 69–74, Mar. 2008.

[11] NORDSTRÖM, E.; SHUE, D.; GOPALAN, P.; KIEFER, R.; ARYE, M.; KO, S.; REXFORD, J.; FREEDMAN, M. J. *Serval: An end-host stack for service-centric networking,* in *NSDI, S. D. Gribble and D. Katabi, Eds. USENIX Association,* 2012, pp. 85–98.

[12] GHODSI, A.; KOPONEN, T.; RAJAHALME, J.; SAROLAHTI, P.; SHENKER, S. *Naming in content-oriented architectures,* in *Proceedings of the ACM SIGCOMM Workshop on Information-centric Networking,* ser. ICN '11. New York, NY, USA: ACM, 2011, pp. 1–6.

[13] REXFORD, J.; DOVROLIS, C. *Future Internet Architecture: Clean-Slate Versus Evolutionary Research,* Communications of the ACM, vol. 53, no. 9, p. 36, 2010.

[14] ALBERTI, A.; SCARPIONI, G.; MAGALHAES, V.; CERQUEIRA, A.; RODRIGUES, J.; RIGHI, R. *Advancing NovaGenesis Architecture Towards Future Internet of Things, IEEE Internet of Things Journal,* 2017.

[15] XIA Project. [Online] https://www.cs.cmu.edu/~xia/. Acessado em 01 de outubro de 2017.

[16] NovaGenesis Project. [Online] http://www.inatel.br/novagenesis. Acessado em 15 de setembro 2017.

[17] LI, Y.; WILSON, P. *Partos-11: an efficient real-time operating system for lowcost microcontrollers,Proceedings First IEEE International Workshop on Electronic Design, Test and Applications '2002,* 2002.

[18] MOHAMADI, T. *Real time operating system for avr microcontrollers, 2011 9th East-West Design & Test Symposium (EWDTS),* 2011.

[19] SHEN, J.; WU, Q.; LI, X.; ZHANG, Y. *Research of the Real-Time Performance of Operating System, 2009 5th International Conference on Wireless Communications, Networking and Mobile Computing,* 2009.

[20] ZHOU, H.; WU, F.; HOU, K. *An Event-driven Multi-threading Real-time Operating System Dedicated to Wireless Sensor Networks, 2008 International Conference on Embedded Software and Systems,* 2008.

[21] M. Samek, *Practical UML Statecharts in C/C++: Event-Driven Programming for Embedded Systems,* 2nd ed. CRC Press, 2008.

[22] LOYALL, J.; GILLEN, M.; HAIGH, K.; WALSH, R.; PARTRIDGE, C.; LAUER, G.; STRAYER, T. *A concept for publish-subscribe information dissemination and networking, 2012 IEEE International Conference on Communications (ICC),* 2012.

[23] EUGSTER, P.; FELBER, P.; GUERRAOUI, R.; KERMARREC, A. *The many faces of publish/subscribe, ACM Computing Surveys,* vol. 35, no. 2, pp. 114-131, 2003.

[24] EventOS Repository. [Online] https://github.com/edielsonpf/EventOS. Acessado em 15 de setembro 2017.

[25] Cortex-M3 Technical Reference Manual. [Online] http://infocenter.arm.com/help/topic/com.arm.doc.ddi0337e/DDI0337E_cortex_m3_r1p1_trm.pdf. Acessado em 01 de outubro 2017.

[26] ALBERTI, A. M.; CASAROLI, M. A. F.; SINGH, D.; RIGHI, R. R. *Naming and name resolution in the future internet: Introducing the NovaGenesis approach, Future Generation Computer Systems,*, vol. 67, pp. 163 – 179, 2017.

[27] ALBERTI, A. M.; MAZZER, D.; BONTEMPO, M. M.; RIGHI, R. R.; OLIVEIRA, L. H. de; SODRÉ JR., A. C. *Cognitive radio in the context of internet of things using a novel future internet architecture called NovaGenesis, Computers & Electrical Engineering,*, 2016.

[28] YAMAGUCHI, F.; NISHI, H. *Hardware-based hash functions for network applications,* in *IEEE Int. Conf. on Networks (ICON),* 2013, pp. 1–6.

[29] RAMIREZ, W.; MASIP-BRUIN, x.; YANNUZZI, M.; SERRAL-GRACIA, R.; MARTINEZ, A.; SIDDIQUI, M. *A survey and taxonomy of id/locator split architectures, Computer Networks,* vol. 60, pp. 13 – 33, 2014.

[30] ALBERTI, A.; SCARPIONI, G.; MAGALHAES, V. *Rede IoT Colaborativa NovaGenesis,* in *XXXV Simpósio Brasileiro de Telecomunicações e Processamento de Sinais,* 2017.

[31] SUN, L.; LI, Y.; MEMON, R. *An open IoT framework based on microservices architecture, China Communications,* vol. 14, no. 2, pp. 154-162, 2017.

[32] LPC1769. [Online] https://www.nxp.com/products/microcontrollers-and-processors/arm-based-processors-and-mcus/lpc-cortex-m-mcus/lpc1700-cortex-m3/512kb-flash-64kb-sram-ethernet-usb-lqfp100-package:LPC1769FBD100. Acessado em 15 de setembro 2017.

[33] LPC1769 LPCXPRESSO BOARD. [Online] https://www.embeddedartists.com/products/lpcxpresso/lpc1769_xpr.php. Acessado em 18 de setembro 2017.

[34] RS9110-N-11-02. [Online] http://www.redpinesignals.com/Modules/Internet_of_Things/Connection_Family/RS-9110-N-11-22.php. Acessado em 18 de setembro 2017.

[35] BACELLI, E.; MEHLIS, C.; HAHM, O.; SCHMIDT, T. C.; WÄHLISCH, M. *Information centric networking in the iot: Experiments with ndn in the wild,* in Proc. *of the 1st ACM Conf. on Information-Centric Networking,* ser. ACM-ICN '14. New York, NY, USA: ACM, 2014, pp. 77–86.