UNIVERSIDADE DA BEIRA INTERIOR

Engenharia

Desenvolvimento da Network Capable Application Processor. Norma ISO/IEC/IEEE 21451

Versão final após defesa

Gaspar Ferreira António

Dissertação para obtenção do Grau de Mestre em

Engenharia Eletrotécnica e de Computadores

(2º ciclo de estudos)

Orientador: Prof. Doutor António Eduardo Vitória do Espírito Santo

Co-Orientador: Prof. Doutor Bruno Ribeiro

Covilhã, julho de 2018

Dedicatória

Aos meus pais, António Gaspar e Maria F. Ferreira

Ao meu filho

A todos os sobrinhos, sobrinhas e restantes familiares

Agradecimentos

O início de um trajeto até pode ser fácil, mas, chegar ao final de “uma trajetória diferente de zero (0)” não é uma tarefa simples, no entanto a presença e a ajuda de outros pode diminuir esta dificuldade.

Começo por agradecer a Deus, pela vida e por todas as provisões feita para me manter vivo.

Ao meu Orientador e coorientador, Professor Dr. António Espírito Santo e Bruno Ribeiro, pela grande ajuda que me deu na forma de orientação e instrução.

À Rita Pinto, pelo esclarecimento de conteúdos, simpatia e amizade, o que contribuiu para o avanço do projeto, ao Pedro Serra pela liderança do laboratório.

Aos meus irmãos - os Ferreiras -, primos, sobrinhos e sobrinhas, mães e tios - os Franciscos - pelo apoio demonstrado nas ligações constantes, reflexo da sua preocupação com o meu bem-estar e adaptação nas terras de Camões.

Ao casal Amaral (Emanuel e Juana), ao casal Fernando Miranda e Cláudia Nambandwa, à Otávia Silva e à família Matos onde destaco a Cristina e a Raquel que contribuíram para a minha alegria e fácil adaptação a Portugal e a congregação de forma geral. A vossa generosidade, os vossos convívios, sorrisos e amizade deram-me a certeza de que nunca estive sozinho durante este percurso.

Ao melhor grupo de estudantes e colegas que já existiu na UBI: Ntunitangua René Pindi, Alfredo Luemba, Pedro Patrício, Arlindo Isaac, Costantino Dário, Analcísio Rodino, César, Sofia e o Bruno Silva. A nossa união enquanto colegas era o *“cooler”* que arrefecia o processador do nosso cérebro, tornando fáceis os exercícios e as matérias complicadas.

Ao Instituto Superior Politécnico do Huambo (ISPH) por me ter selecionado, aos meus colegas e amigos, Móser José, Fátima Dantas, Jusualdo Figueira e o Santos Mota pela companhia e coragem que me deram até ao final do curso.

Em último, mas não menos importante, agradeço a todos os meus amigos-irmão: Abel Domingos J. da Silva (ADJS), Patrício Monteiro, Figueira João, Joel Mendes, Sagrado Capoco, José Matiti (N. 4ever), ao casal Nicolau Matias e Almerinda Jornal, Eduardo João e Azarado Vaz, que apesar da distância estiveram sempre perto de mim. A todos o meu muito obrigado.

Este trabalho foi desenvolvido no âmbito do projeto Energy and Water Systems Integration and Management (EdgeWise), apoiado pela Fundação para a Ciência e Tecnologia (FCT), com a referência ERANETMED/0004/2014, enquadrado na iniciativa ERANETMET dos Estados Membros, Países Associados e Países Parceiros Mediterrâneos (Projeto com o ID eranetmed_nexus-14-044).

Resumo

O surgimento da eletrónica marca, sem dúvida, o início da era moderna, e, desde então, a eletrónica sempre esteve presente na base do desenvolvimento das sociedades em todo mundo. Com o surgimento da internet, a aplicabilidade da eletrónica toma um novo caminho, pois a união destes dois elementos deu origem à internet das coisas (IoT), onde mais coisas do que pessoas passaram a ligar-se à internet. Hoje a IoT é utilizada em várias áreas do saber e aplicada de várias formas, tudo para auxiliar o homem nas mais diversas tarefas. A área da instrumentação e da medida é uma das muitas áreas onde se sente o efeito positivo da existência da IoT, pois esta área utiliza sensores e atuadores para controlar fenómenos físicos. O desenvolvimento desta dissertação é focado no desenvolvimento de serviços que permitem à *Network Capable Application Processor* (NCAP), - que é traduzido para português como Processador de Aplicação Capacitador de Rede, que possui uma unidade de processamento e uma interface de comunicação, ser responsável pela interligação do Módulo de Interface de Transdutor Inteligente (TIM) a uma rede de utilizador. A TIM é conhecida por permitir a integração de sensores/atuadores, condicionadores de sinal e um *Transducer Electronic Data Sheet* (TEDS) - o que traduzido em português quer dizer Transdutor de dados eletrônicos. Esta ligação é feita por meio de um módulo de comunicação que pode ser ponto a ponto (P2PCom), comunicação via rede (NetCom), ou outros módulos. Nesta dissertação, estes serviços foram desenvolvidos por meio da linguagem de programação Python e executados numa ferramenta de hardware, *Raspberry* Pi. Esta tecnologia será uma mais-valia para diversas áreas, em particular na área de instrumentação e medida.

Palavras chaves

NCAP, TIM, módulo de comunicação, TEDS, IoT, instrumentação e medida

Abstract

The emergence of electronics is undoubtedly the beginning of the modern era and since then, electronics has always been at the heart of the development of societies around the world. With the advent of the Internet's applicability of taking electronic a new path, the union of these two elements has given rise to the Internet of Things (IoT), where more things that people have become connected to the Internet. Nowadays, IoT is used in various fields of knowledge and applied in various ways, all to help the man in the most diverse tasks. The field of instrumentation and measurement is one of many areas that feel the positive effect of the existence of IoT, after this area uses sensors and actuators to control physical phenomena. The development of this work focuses on the development of services that enable networks Capable Application Processor (NCAP), has a processing unit and a communication interface, and he is responsible for the Intelligent Interconnect Transducer Interface Module (TIM) to a User Network, where TIM is known to enable the integration of Sensors / Actuators, Signal Conditioner and the Electronic Data Sheet Transducer (TEDS). This connection is made via a communication module that can be peer-to-peer (P2PCom), Internet communication (NetCom), or other modules. In this thesis, these services were developed using the Python programming language and executed in a *Raspberry pi* hardware tool. This technology will add value to several areas, particularly in the field of instrumentation and measurement.

Key Words

NCAP, TIM, communication module, TEDS, IoT, instrumentation and measurement.

Índice

Lista de Figuras

Lista de Tabelas

Lista de Acrónimos

AIEE - American Institute of Electrical Engineers

ANSI - American Nacional Standard Institute

BSI - British Standard Institute

DIN - Deutsches Institut für Normung

DSI - Digital de Serviços Integrados

FPGA - Field Programmable Gate Array.

GPIB - General Purpose Interface Bus

GPU - Graphics Processing Unit

HP-IB - Hewlett-Packard Interface Bus

IBSG - Internet Business Solutions Group

IEC - International Electrotechnical Comission

IEEE - Institute of Electrical and Electronics Engineers

INMETRO - Instituto Nacional de Metrologia

IoT - internet of things

IRE - Institute of Radio Engineers

ISO - International Organization for Standardization

LLC - logical link control

MIT - Massachusetts Institute of Technology

NCAP - Network Capable Application Processor

NFC - Near Field Communication

NINT - N_IO_INTERRUPT Usado pela STIM para requisitar serviço à NCAP

NIOE - N_IO_ENABLE Habilita transporte de dados

NIST - National Institute of Standards and Technology

NIST - National Institute of Standards and Technology

NSDET - N_STIM_DETECT Reconhecimento *plug_and_play* de uma STIM

NTRIG - N_TRIGGER Desempenha função de disparo

OSI - Open Systems Interconnection

RFID - identificação de frequência de rádio

SAD - Serviços de Agregação de Dados

SCI - Serviços de Colaboração e Inteligência

SD - Secure Digital

SID - Serviço de Identificação

STIM - Smart Transducer Interface Module

SU. - Serviços de Ubiquidade

TCP/IP - Transmission Control Protocol /Internet Protocol.

TEDS - Transducer Eleletronic Data Shetets

TII - Transducer Independent Interface.

TIM - Transducer Interface Module

TSA -Transducer Services API

USB - Universal Serial Bus

WTIM - Wireless Transducer Interface Modul

WTIM - Wireless Transducer Interface Modul e

XDCR – Transdutor

Capítulo 1- Introdução

A ideia de ligar vários dispositivos através da internet deu *origem* ao conceito da *Internet of Things* (IoT), em português designado por internet das coisas, e marcou uma nova era no mundo tecnológico. Este conceito otimizou o tempo dos utilizadores, tornou mais fácil a integração do mundo real no mundo digital; esta integração possibilita a comunicação constante, não só entre pessoas, mas também entre pessoas e coisas, ou, entre dispositivos ligados em redes. Em áreas, tais como aplicações ambientais, agricultura, ambiente doméstico, medicina, automobilismo e muitas outras, é cada vez mais frequente observar a supervisão de sistemas e ambientes por meio de redes de transdutores inteligentes. A falta de normalização e uma vasta gama de tecnologia existente no mercado, fez com que surgissem muitos problemas relacionados com a construção e a configuração das redes de sensores inteligentes.

Uma forma encontrada para garantir a comunicação entre o mundo digital e o real, de forma padronizada, foi fazer uso da norma IEEE 1451, garantindo que as aplicações que a utilizem estejam de acordo com o estabelecido na norma. Esta apresenta recursos suficientes para garantir mais do que apenas uma comunicação. Permite também que se possa efetuar de forma digital o controlo e a gestão de dispositivos, definindo as propriedades para tal controlo. Como será apresentado neste trabalho, esta norma permite obter informações do meio envolvente. São especificados pela norma dois módulos conhecidos como *Network Capable Application Processor* (NCAP) e *Smart Transducer Interface Module* (STIM), interligadas por uma interface que permite a comunicação entre eles. A NCAP responsabiliza-se pelo processamento e a gerência da rede, enquanto que a TIM agrega um conjunto de até 255 transdutores ligados, uma memória não volátil que armazena as TEDS, e uma interface que permite a comunicação com a NCAP. Mais detalhes sobre os módulos fornecido pela norma IEEE 1451 serão apresentados no capítulo 2 deste trabalho.

1.1. Enquadramento e Motivação

O mundo da tecnologia tem verificado um grade crescimento nos últimos tempos, e, com isso, a utilização da IoT tem sido cada vez mais frequente, proporcionando desta forma grandes vantagens para as industrias, empresas e residências individuais, tanto a nível financeiro, como a nível de produção, elevando o nível de flexibilidade em poder adicionar novos módulos, o que possibilita a interligação de vários e diferentes blocos TIM, onde cada um se pode ligar utilizando interfaces de comunicação diferentes. Esta flexibilidade, segundo [1], foi possível devido a padrões, entre outros, como o IEEE 1451. Este padrão teve a sua conceção na década de 1960, e foi sendo desenvolvido com o tempo. Através dos artigos publicados evidencia-se que, por intermédio dele, se tornou mais fácil a aquisição e controlo de dados de sensores e atuadores, como a temperatura, humidade, a existência de gases que podem prejudicar o meio ambiente e tantas outras grandezas existentes. Uma das caraterísticas impressionantes observadas com o surgimento da norma foi o fato de facilitar a conexão e a remoção de

dispositivos ligados à NCAP, sem necessitar que a tarefa de reconfiguração tenha intervenção humana, isso por possuir a caraterística *plug-and-play*.

No presente trabalho pretende-se desenvolver a NCAP de tal forma que se possa conetar a TIM, independentemente do módulo de comunicação (IEEE 1451.X). O modulo de comunicação IEEE 1451.5 tem a vantagem de permitir a troca de informação entre módulos distantes um do outro. A utilização de TEDS e módulos de comunicação sem fio com base nas normas IEEE 802.15.4 e IEEE 1451.5 proporcionam a fácil interação entre os módulos NCAP e os módulos TIM.

1.2. Objetivos

Este trabalho tem como objetivo a construção de uma plataforma que suporte o desenvolvimento de sensores inteligentes que observem a norma ISO/IEC/IEEE 1451. Esta norma possui dois grandes elementos: a *Network Capable Application Processor* (NCAP) e a *Smart Transducer Interface Module* (TIM). A primeira é responsável por interligar a TIM à rede do utilizador. Este trabalho pretende desenvolver a NCAP utilizando para isso ferramentas de hardware facilmente disponíveis. O resultado deve ser capaz de se interligar com a TIM e fazer a interligação com a rede do utilizador.

A NCAP será desenvolvida em torno do *Raspberry Pi* utilizando a linguagem de programação Python. O resultado deve ser demonstrado em funcionamento.

1.3. Organização da tese

Com exceção do primeiro capítulo - que é uma introdução do trabalho-, e da conclusão, o presente trabalho encontra-se dividido em 5 capítulos que abordam diferentes temáticas relacionadas com o tema principal, o que permite um melhor entendimento do tema.

O capítulo 2 apresentará um breve historial da norma IEEE 1451, a sua fusão com outras entidades e descreve as normas que fazem parte da família IEEE 1451. Neste capítulo é ainda apresentado o módulo constituinte da NCAP e a sua interligação com a TIM, apresenta também detalhes da comunicação entre eles utilizando o módulo de comunicação IEEE 1451.2 bem como a descrição e a função das linhas que compõem a *Transducer Independent Interface (*TII).

O capítulo 3 tratará da apresentação da plataforma de desenvolvimento e a preparação de ferramentas utilizadas para se poder efetuar a ligação da placa *Raspberry* Pi a uma rede de utilizadores.

Os capítulos 4 e 5 fazem o estudo e a implementação da aplicação, apresentando as mensagens, os seus comandos e estruturas da API.

Capítulo 2 - Breve Histórico da Norma IEEE 1451

2.1. Necessidades de normalização

Este capítulo tratará de fornecer as definições e o significado de cada um dos termos relacionados com o tema. Esta abordagem irá permitir a compreensão da temática, para que o leitor, mesmo que não esteja familiarizado com o tema, não se sinta num mar sem rumo.

A norma resulta do trabalho de normalização, segundo [2], Esta atividade visa estabelecer, diante de um problema, a utilização de uma solução comum. Obtém-se, assim, um nível organizacional definido no contexto específico da norma.

As normas são documentos ou regras publicadas por organizações profissionais que se responsabilizam por estabelecer padrões que determinam as diversas e várias atividades, processos e dispositivos. [3]

O motivo da existência de normas é que se alcance uma padronização nas metodologias utilizadas na implementação de processos, e para que cada termo tenha o mesmo significado e seja entendido em diferentes situações, facilitando a comunicação, tanto entre as equipas internas de uma organização, quanto entre as diferentes organizações externas envolvidas.

A utilização das normas vem garantir que todas as organizações que realizam uma determinada ação o façam em conformidade com as demais, para que possam facilitar o uso das mesmas e para que não se observe incompatibilidades aquando da inserção, ou conexão, dos dispositivos fabricados por eles com os outros, fabricados por empresas diferentes. O uso de normas também facilita a obtenção de um determinado parâmetro de comparação para um determinado atributo.

A abordagem feita acima transmite a ideia de que a norma traz benefícios apenas nas áreas de desenvolvimentos de dispositivos eletrónicos. Na realidade não é simplesmente a área das tecnologias a única beneficiada pela aplicação das normas. Abaixo serão alistadas algumas de várias áreas que também são beneficiadas ao fazer o uso de normas como encontrado em [4].

- Na Economia, a normalização proporciona uma redução da crescente variedade de produtos e procedimentos, o que resulta numa melhor economia dos recursos disponíveis;
- Na segurança, a normalização estabelecida protege a saúde em geral e a vida humana;
- Na proteção ao consumidor, a normalização amplia a qualidade de produtos e serviços e defende os seus interesses;

- No desenvolvimento sustentável, a normalização permite desenvolver diversos métodos para avaliação de impactos ambientais, sociais e económicos, bem como as ferramentas para mitigação dos efeitos que causam esses impactos;
- Na eliminação de barreiras técnicas e comerciais, a normalização evita a existência de regulamentos conflitantes sobre produtos e serviços em diferentes países, facilitando o intercâmbio comercial;
- Na comunicação, a normalização visa promover meios mais eficientes para que haja troca de informações entre os fabricantes e os clientes, aumentando, assim, a confiabilidade das relações comerciais e de serviços. Nesta área a normalização permite ainda que possa existir comunicação entre dispositivos conectados entre si, bem como a partir de uma interface que se possa gerenciar dados existente noutra.

2.1.1. *Institute of Electrical and Electronics Engineers* (IEEE)

Em português o IEEE é conhecido como Instituto de Engenheiro Eletricista e Eletrónico é uma organização profissional sem fins lucrativos, empenha-se em contribuir para o avanço do conhecimento e aplicações que podem ser aplicadas em Engenharia Elétrica e Eletrónica. O IEEE tem como finalidade o avanço científico, tecnológico e educativo servindo também para difundir a teoria e a prática da engenharia eletrotécnica, das comunicações e o processamento da informação. [5]

Segundo [6], esta organização foi formada em 01 de janeiro 1963 com a união de Instituto de Engenheiros de rádio (IRE) e o Instituo Americano de Engenheiros Eletricistas (AIEE), ela encontra-se situada em várias partes do mundo e incorpora muitas filiais, fazem parte desta organização engenheiros eletrotécnicos como não deixaria de ser, engenheiros da computação, cientistas da computação e os profissionais de telecomunicações. Tinham como objetivo promover o conhecimento no campo da engenharia elétrica, eletrônica e computação. Com o passar do tempo os focos começaram a mudar, tanto é que atualmente dado a sua abrangência são incorporadas nela várias áreas como a micro e nanotecnologias, ultrassom, bioengenharia, robótica, materiais eletrônicos, e muitas outras. Apesar dessa incorporação, ainda assim, o mais importante para eles é estabelecer padrões para formatos de computadores e dispositivos menores, permitindo assim o aproveitamento das ideias que as associações possuíam antes da sua união. Pensando naquilo que era realmente importante, e o foco da organização bem como na sua abrangência, com as duas associações já unidas, decidiu dividir as áreas de controlo, permitindo assim que vários padrões fossem criados. Entre ele destaca-se alguns que são tidos como os mais notáveis.

1. IEEE 488, padrão de comunicação digital paralelo de 8-bits, ainda usado para ligar instrumentos de teste em rede. Também conhecido como GPIB e HP-IB.

2. IEEE 754, aritmética de ponto flutuante, possibilitando uma maior precisão em cálculos.
3. IEEE 802, os padrões IEEE 802 são referentes às redes locais e redes metropolitanas, e dentro desta família existem alguns sub-padrões conhecidos:

- IEEE 802.1, padrão que especifica a relação entre os padrões IEEE e a sua interação com os modelos OSI, bem como especifica de modo similar as questões de interconetividade e administração de redes;
- IEEE 802.2, controle lógico de ligação (LLC), que oferece serviços de conexão lógica a nível de camada 2;
- IEEE 802.3, Padrão que especifica as camadas de ligação de dados do modelo OSI para a interconexão de redes locais;
- IEEE 802.11, Padrão para redes sem fio.

Estes padrões ou normas são apenas alguns de tantos que existem na família IEEE.

A norma IEEE 1451 é composta por um conjunto de normas que visa definir e descrever as interfaces que tornam fácil ou permitem efetuar a conexão entre transdutores (atuadores e sensores), independentemente de rede de comunicação, microprocessadores, sistema de instrumentação e de controlo. Esta norma possui uma caraterística fundamental que visa atribuir uma definição ao *Transducer Eleletronic Data Shetets* (TEDS) que é uma estrutura de memória que auxilia o Transdutor no armazenamento de dados de identificação, calibração, correção de dados, medição e de informações relacionadas com a fabricação do próprio transdutor. A Norma IEEE 1451 tem como meta estabelecer a permissão de dados existente nos transdutores por meio de interfaces padronizadas se estes estiverem ligados a um sistema de rede com ou sem fios. [7]

2.1.2. *International Organization for Standardization* (ISO)

A *International Organization for Standardization*, que em Português significa Organização Internacional de Padronização, é um órgão não governamental criada na Suíça no ano de 1947 e sediada em Genebra. Tem como objetivo a normalização e atividades relacionadas com a intenção de facilitar o intercâmbio de serviços e bens. Pretende ainda promover a cooperação em diversos domínios como: Intelectual, científico, tecnológico e na atividade económica, 90 de seus membros são representantes das entidades máximas de normalização em alguns países como, por exemplo, *American Nacional Standard Institute* (ANSI), *British Standard Institute* (BSI), *Deutsches Institut für Normung (*DIN*)* e o Instituto Nacional de Metrologia (INMETRO). O trabalho técnico da ISO é conduzido por comitês técnicos (TC)

Visto que as normas ISO não são de carater imutáveis, elas são revistas no período de 5 anos, pelo menos uma vez. Fazem parte da ISO um conjunto de normas, e entre as mais importantes destacam-se a ISO 9000, responsabiliza-se na implantação e funcionamento do sistema de

qualidade das organizações; ISO 14000, que se responsabiliza na gestão ambiental e ISO 2600, com diretrizes sobre responsabilidade social.

A Comissão Eletrotécnica Internacional (IEC) conjuntamente com a ISO constituem o sistema especializado para padronização mundial. Os órgãos nacionais membros da ISO ou da IEC têm participado no desenvolvimento de normas internacionais através de comités técnicos estabelecidos pela respetiva organização para cuidar de áreas específicas de atividade técnica. Os comités técnicos ISO e IEC trabalham em colaboração em áreas em que ambas possuem um interesse mútuo. Outras organizações internacionais, governamentais e não-governamentais, em ligação com ISO e IEC, também participam do trabalho. No campo da tecnologia da informação, a ISO e a IEC estabeleceram um comité técnico conjunto, ISO / IEC JTC 1.

2.1.3. *International Electrotechnical Comission* – (IEC)

Em Português a Comissão Eletrotécnica internacional é uma organização de padronização de tecnologias elétricas, eletrônicas e relacionadas. Alguns dos seus padrões são desenvolvidos juntamente com a Organização Internacional para Padronização (ISO), a sua sede foi fundada igualmente em Genebra na Suíça em 1906. De modo similar à ISO, foram desenvolvidas pela IEC um conjunto de normas de forma individual e algumas de forma conjunta com ISO.

2.1.4. ISO / IEC / IEEE 21451

O padrão ISO/IEC/IEEE 21451.x é composto por vários padrões (alguns em revisão), que definem a NCAP, STIM, TEDS e tantas outras interfaces que podem, ou não, ser localizada entre a NCAP e STIM. O padrão ISO/IEC/IEEE 214510 lança as bases para futuras famílias do padrão ISO/ IEC/IEEE 21451.x para que sejam compatíveis tal como definidos nos padrões anteriores. Este padrão responsabiliza-se ainda por expandir a interoperabilidade sintática por meio de um formato padronizado de mensagens que nele é implementado usando atuadores e sensores inteligentes, permitindo deste modo que cada módulo da TIM possa cooperar com cada módulo da NCAP que opera como intermediário. Para facilitar a tarefa de implementação, este padrão define um conjunto comum de comandos separando-os em famílias diferentes, algumas destas famílias são os que serão mencionadas agora. [8]

2.1.5. ISO / IEC / IEEE 21451-1

O Comité Técnico de Tecnologia de Sensores do IEEE *Instrumentation and Measurement Society* deu início aos trabalhos, tendo mais tarde constituído uma *Joint Technical Committee* com a comissão técnica de normalização da *IEEE Industrial Electronics Society*. Esta norma define um modelo de objetos com a interface neutra usada para ligar os dispositivos inteligentes como sensores e atuadores. Contém blocos, serviços e componentes que especificam as interações entre os sensores e atuadores, constitui a base para implementar o código do aplicativo que é executado no processador. [9]

2.1.6. ISO/IEC/IEEE 21451-2

Este padrão define a relação comercial entre a NCAP e a STIM, que permite a troca de informação entre os seus módulos por meio de um barramento de 10 linhas. Os fabricantes dos transdutores se deparavam com problemas na hora de fazer a conexão com a rede, pelo que foi necessário efetuar uma normalização das interfaces a nível de conexão com hardware. Para que este problema fosse solucionado efetuou-se uma revisão do padrão e foi introduzida a ideia de se estudar a inclusão de interfaces mais e comuns como é o caso do RS232 e a *Universal Serial Bus* (USB).

2.1.7. ISO/IEC/IEEE 21451-4

Define o protocolo e a interface que permitem que os transdutores analógicos partilhem informações digitais com um objeto ISO / IEC / IEEE 21451. Também define o formato da ficha de dados de segurança Eletrónicos do Transdutor (TEDS), que se baseia nos TEDS ISO/IEC/IEEE 21451-2. Este padrão tem como principal objetivo estabelecer a compatibilidade entre os transdutores existentes com os do modelo ISO/IEC/IEEE 21451x, focando-se principalmente na adição de capacidade de armazenar TEDS para sensores antigos.

2.1.8. ISO/IEC/IEEE 21451-5

Este padrão é o responsável por apresentar as especificações de comunicação entre a *Wireless Transducer Interface Module* (WTIM) com a NCAP, é flexível, pois suporta várias tecnologias que permitem a comunicação com outros dispositivos fazendo uso de tecnologias como 802.15.4, Bluetooth, ZigBee e Wi-fi. Faz ainda a descrição das funções e protocolos que os dois módulos (WTIM e NCAP) devem suportar.

2.1.9. ISO/IEC/IEEE 21451-7

Define uma interface e um protocolo que permite a comunicação entre os transdutores e sistemas RFID. Dado que este padrão fornece informações sobre como identificar produtos ou realizar monitoramento de estados, abre novas oportunidade aos fabricantes de sistemas RFID e sensores. Ele também define os novos formatos das fichas de dados de segurança eletrónicos transdutores (TEDS) com base na série na série de padrões ISO/IEC/IEEE 21451. [10]

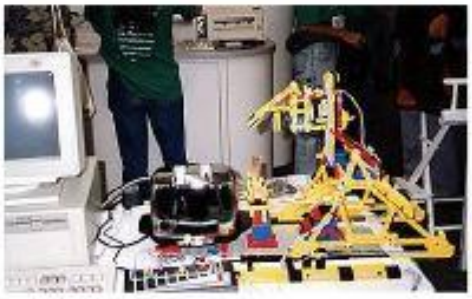

Figura 1: Torradeira de John Romkey

2.2. A necessidade de normas no contexto do IoT e dos sensores inteligentes

A *internet of things* (IoT), em português conhecido como a internet das coisas é a evolução da internet que tem causado mudanças em tudo e em todos, e, essas mudanças não têm apenas efeito nos objetos, afetam também os humanos. Devido ao impacto causado na comunicação, educação, nos negócios na ciência e em várias áreas da sociedade, por esse e tantos outros motivos muitos afirmam que a internet é a mais poderosa e importante criação na história humana. A IoT vem facilitar o quotidiano nas mais diversas atividades que anteriormente o homem desenvolvia com maior dificuldade. Com o surgimento da IoT mais facilmente se pode recolher, analisar e distribuir dados que podem ser transformados em informação, conhecimento e posteriormente em sabedoria. Este cenário é já uma realidade, já há diverso projetos da IoT desenvolvidos e em desenvolvimentos.

Hoje a internet das coisas é importante para a sociedade, já que tantas pessoas fazem uso desta tecnologia, com ou sem conhecimento, da origem da mesma. Para que se possa ver o quão importante é esta tecnologia e aumentar a nossa admiração por ela, bem como valorizar quem as fez chegar até ao dia de hoje, é necessário que se faça um rescaldo de como tudo começou. Tudo começou no ano 1990 quando John Romkey teve a genial ideia de criar a primeira torradeira que possuía componentes que permitiam a sua conexão ou manuseio (ligar e desligar) por meio de internet, esta criação foi considerada como primeiro dispositivo da internet das coisas, apresentada na conferência INTEROP. Lançado o desafio pelo presidente da INTEROP, John conectou a torradeira a um computador com rede TCP/IP e teve um resultado fabuloso, apenas com um pequeno reparo, o pão era posto à mão na torradeira. Não cruzando os braços diante desta crítica, John apresentou no ano seguinte o mesmo projeto. Desta vez acompanhada por um robô, controlado pela internet, igualmente como a torradeira, permitindo assim que o pão fosse posto na torradeira de forma automática, como mostra a figura 1.

Depois de John ter apresentado o seu inovador e apreciado trabalho, o estudo continuou e vários outros desenvolvimentos relacionados foram realizados por pessoas e equipas diferentes. Em setembro do ano 1999 o cofundador e diretor executivo do *Auto-ID Cetnter* Kevin Ashton,

foi preletor de uma palestra para *Procter e Gamb* apresentando assim uma nova ideia do sistema identificação de frequência de rádio (RFID) para se ter o controlo do produto na cadeia de fornecimento. Para chamar atenção dos presentes no título da apresentação colocou a expressão *Internet of things*. Kevin foi considerado o criador do termo *Internet of Things,* visto que a primeira vez que esse termo foi usado era exatamente no dia da sua apresentação. Com a nova ideia sobre a tecnologia RFID apresentado por Kevin a mesma se destacou nos anos posterior à apresentação. [11]

Assim, como vários outros conceitos novos tiveram um precedente, da mesma forma as raízes da IoT podem ser rastreadas até o *Massachusetts Institute of Technology* (MIT), do trabalho até o Auto-ID Center. Este grupo de investigação, para além do campo de RFID, também trabalhavam no campo de tecnologias e sensores emergentes. É composto por laboratórios em sete universidades de pesquisas espalhadas por quatro continentes, pois pelas pesquisas realizadas, as instituições foram escolhidas pelo Auto-ID Center para projetar a arquitetura da IoT. [12]

Passados cinco anos após a apresentação de Kevin, isto é, em janeiro de 2005, a *Wall Mart* e o departamento da defesa do Estados Unidos exigiram que os fornecedores de diversos produtos fizessem uso de etiquetas RFID, com o objetivo de facilitar o controlo de stocks. Este foi o marco do conceito de Internet das coisas.

Será então necessário apresentar uma definição da IoT. Segundo a *Cisco* e a *Internet Business Solutions Group* (IBSG), a IoT é um fenómeno que se verificou quando foram conectados à internet mais objetos do que pessoas. Levando em consideração esta definição apresentada, pode-se afirmar que até 2003 não existia IoT, já que existiam mais pessoas conectadas à internet do que objetos. Para se ter uma noção da realidade, em 2003 havia aproximadamente 6,3 bilhões de pessoas habitando o planeta terra, e apenas 500 milhões de dispositivos estavam ligados à internet. Ao determinar a média de dispositivos ligados, verifica-se que existia cerca de 0.08 dispositivos por pessoa, isto é menos de um dispositivo. Nos anos que sucederam a 2003 começou a verificar-se um crescimento exponencial da IoT, e o que estava na base deste desenvolvimento era o elevado número de smartphones e tablets produzidos. Tanto é que até 2010 foram ligados à internet cerca de 12,5 mil milhões de dispositivos, o que correspondia a 1,84 dispositivos por pessoa, isto é, já havia mais dispositivos ligados à internet do que pessoas como mostra a figura 2. Este facto levou a Cisco a prever que até 2050 pode vir a existir quase 50 mil milhões de dispositivos ligados à internet. [12]

Agora é hora de refletir um pouco no que foi escrito até aqui. Como seria o desenvolvimento da IoT se não seguisse um padrão específico? Será que se conseguiria verificar essa tão grande explosão? A resposta é clara e é um rotundo não. Tudo isto foi possível apenas porque os dispositivos interligados entre si seguiram um padrão de conexão ou de comunicação. Caso contrário, os fabricantes teriam imenso trabalho, e jamais conseguiriam fabricar dispositivos

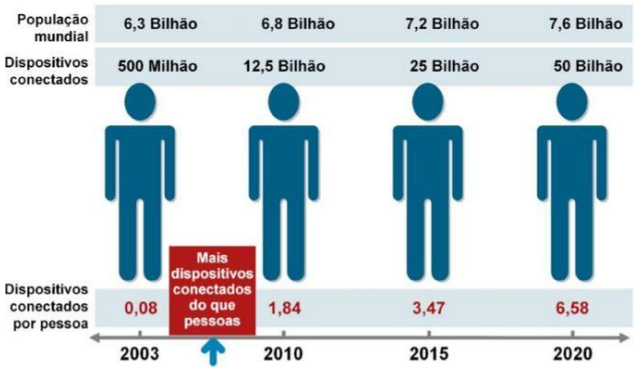

Figura 2: O surgimento da IoT

com caraterísticas diferentes e ainda assim permitir que diversos dispositivos se ligassem e efetuassem as devidas trocas de informações.

Pense um pouco no que acontece ou aconteceu com os seres humanos. A evolução dos seres humanos deu-se devido a comunicação, porque cada descoberta feita foi sendo compartilhada, facilitando a vida das gerações futuras, sem a necessidades de redescobrir a mesma coisa. Este princípio de compartilhamento de informação e aproveitamento das descobertas feitas anteriormente é mais evidente quando examinamos a forma como os humanos processam os dados que são adquiridos. Como verificado na figura 3, ao observar a figura no sentido da seta, cada camada inclui um elemento como: informação, conhecimento, e sabedoria, onde os dados são os resultados adquiridos e processados que dão origem a informação, as informações quando agrupadas dão origem ao conhecimento, este por sua vez quando unido à experiência resulta em sabedoria. Mesmo que se verifique uma mudança no conhecimento, a sabedoria não muda com o tempo. Visto que a sabedoria começa com uma simples recolha de dados. Devemos salientar que é necessário que se faça uso do IoT, pois contribui para uma recolha confiável de dados que podem dar lugar a sabedoria. Que relação existe entre a comparação realizada e a IoT? Esta deve ser a pergunta que se levanta.

A resposta é bastante simples, assim como se obteve o conhecimento e o mesmo foi transformado em sabedoria, esta sabedoria é imutável. O conhecimento até pode variar, mas a sabedoria faz com que os desenvolvimentos de dispositivos sigam um padrão que facilita a ligação e a partilha de informações entre eles. Isso porque a IoT possui um bloco básico de

Figura 3: Evolução dos dados

construção que se resume em dispositivos, redes de comunicação e sistemas de controle, como será mostrado na figura 4. Para que se efetue a interconexão das partes que formam o bloco é necessário que se siga um determinado padrão. O uso de conhecimento adquirido pelas descobertas feitas noutros momentos foi possível continuar a construção e desenvolvimentos de vários dispositivos que se pudessem interligar. Se não seguissem um padrão de construção seria quase impossível interligar incontáveis dispositivos e permitir que se efetuasse troca de informação entre eles.

A figura 4 é uma representação de bloco de construção da IoT, que pode ser vista como a ligação de várias tecnologias, que se complementam para facilitar a integração dos objetos físicos ao mundo virtual, cada elemento destes blocos desempenha uma determinada tarefa nesse processo como será descrito a seguir. [13]

- Identificação - responsável pela identificação dos objetos antes de serem ligados à internet uma vez que os objetos ao serem ou para serem ligados à internet, devem antes ser identificados para receber a permissão de ligação. Este bloco executa esta tarefa usando as Tecnologias como RFID, *Near Field Communication* (NFC) e endereçamento IP que podem ser empregados para identificar os objetos;
- Sensores/Atuadores - os sensores são os responsáveis pela recolha de dados de acordo com a sua programação no meio em que são inseridos, podendo ainda os encaminhar para um centro de armazenamento, ao passo que os atuadores se responsabilizam em manipular o ambiente ou apresentar uma reação de acordo com a leitura de dados;
- Comunicação - parte que faz o uso de diversas técnicas para ligar os objetos inteligentes, usa algumas tecnologias como WiFi, Bluetooth, IEEE 802.15.4 e RFID para desempenhar um papel importante no consumo de energia dos objetos;
- Computação - os dados recolhidos pelos sensores serão importantes se forem processados e fornecerem assim conhecimento à equipa responsável pelo tratamento de dados. A computação é a responsável por incluir diferentes e diversos tipos de unidade de processamento como: microcontroladores, processadores e FPGA, que são os responsáveis pela execução dos algoritmos que tornam inteligentes os objetos;

Serviços - não existe um único serviço a ser realizado no bloco de construção, existem diversas classes de serviço na IoT, onde cada classe é responsável por uma tarefa específica.

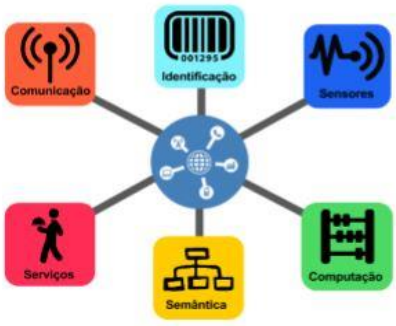

Figura 4: Blocos construtivos da IoT

Algumas classes de serviços existentes na IoT são: Serviço de Identificação (SID), Serviços de Agregação de Dados (SAD), Serviços de Colaboração e Inteligência (SCI) e Serviços de Ubiquidade (SU).

- O SID mapeia as entidades físicas em entidades virtuais, de acordo a sua programação, ou por outras palavras pode-se dizer que este serviço se destina a identificar um determinado valor específico;
- O SAD é responsável pela recolha e compilação dos dados obtidos através dos objetos inteligentes;
- Os SCI usam o SAD para tomar decisão e reagir de maneira adequada diante de um cenário;
- O SU promove sempre que for necessário o serviço de SCI.

Semântica é a habilidade de extração de conhecimento dos objetos na IoT. Trata da descoberta de conhecimento e uso eficiente dos recursos existentes na IoT, a partir dos dados existentes, com o objetivo de prover determinado serviço.

Devemos salientar que este bloco não tem uma topologia física padronizada, até porque alguns blocos são lógicos e não físicos. Apesar da sua invisibilidade, ainda assim desempenham um papel muito importante para os dias de hoje.

2.3. Módulos constituintes da NCAP (e interligação com a TIM)

2.3.1. O que é a NCAP?

Num universo onde a tecnologia evolui a cada dia que passa, verifica-se que, os conceitos, bem como a forma de pensar, variam e ganham lógica constantemente, portanto seria imprudente apresentar uma única definição do que se quer, pois existem muitas ideias que expressam corretamente a definição da NCAP, mas, segundo a Instituto Nacional de Padrões e Tecnologia, em inglês *National Institute of Standards and Technology* (NIST), o processador de aplicação capacitador de rede, em inglês *Network Capable Application Processor* (NCAP) é um dispositivo que se localiza entre a STIM e a rede, para facilitar a execução da comunicação na rede, comunicações STIM, funções de conversão de dados e funções de aplicativos. A NCAP geralmente contém um controlador e a interface para a rede mais ampla que pode suportar outros nós, o hardware da NCAP é aquele que consiste nos microprocessadores e nos seus suportes em que o seu hardware de implementação é realizado na camada física da rede por onde se encontram fixas as interfaces de I/O como citado em [14]. Uma NCAP poderia ser um microprocessador de 8-bits para uma rede de controlo *DeviceNet* ou um microprocessador de 32 bits para uma rede de controlo baseada em Ethernet. Uma vez lidas das TEDS relativas a uma TIM em concreto, a NCAP sabe o quão rápido ela se pode comunicar com essa STIM, quantos canais contém e o formato de dados dos transdutores existentes. Também sabe quais as unidades físicas que estão a ser medidas e como converter as leituras em unidades calibradas observando o Sistema Internacional (SI).

Uma NCAP deve ser capaz de realizar as duas funções principais pelas quais foi desenvolvida: gerenciar as atividades dos transdutores e comunicar com a rede do utilizador. O seu processador deve ser capaz de implementar atividades específicas como a ligação em qualquer tipo de rede, bem como controlar as interfaces IEEE1451. Com estas funções a serem desempenhadas de forma correta, a NCAP garante o cumprimento de, pelo menos, umas das três caraterísticas fundamentais: Interoperabilidade; Modelagem Orientada a Objeto; e driver da TII. Cada caraterística representa uma ou algumas funções da NCAP. Por exemplo, a Interoperabilidade é a caraterística através da qual a NCAP pode ser ligada a qualquer tipo de rede de controlo, tornando fácil a sua ligação de transdutores com diversos ambientes externos. A Modelagem Orientada a Objetos permite ou facilita a implementação de uma linguagem de programação orientada a objeto aberta e padronizada. A TII facilita o controle da STIM pela NCAP, esta caraterísticas, serão detalhadas mais abaixo. [15]

Segundo o padrão IEEE 1451, a NCAP é o elemento central de uma rede de transdutores inteligentes. A NCAP pode ser considerada como um nó de rede composto de duas partes, uma lógica e outra parte física.

A NCAP é um dispositivo desenvolvido em torno de um microcontrolador que realiza tarefas de processamento sobre os dados simples *onboard* recolhido da TIM. É da sua responsabilidade enviar comandos que a TIM executará, e, para isso, pode-se optar em fazer o uso de *Transducer Independent Interface* (TII).

O objetivo principal do módulo TIM é a realização de processamento em tempo real, visto que a NCAP se limita em realizar o processamento de dados simples, dando o privilégio ao utilizador final de realizar a análise com maior critério dos dados recolhidos da TIM. [16]

Como uma NCAP pode ser utilizada? A NCAP inicia uma medida, ou ação, por meio do disparo na STIM, ao que esta última responde com uma confirmação quando a função for concluída. A STIM pode interromper a NCAP se ocorrer uma exceção, como erro de hardware, falha de calibração ou falha no autoteste. Na secção 4 estas operações serão detalhadas com mais pormenor. A estrutura de uma NCAP é apresentada na figura 5.

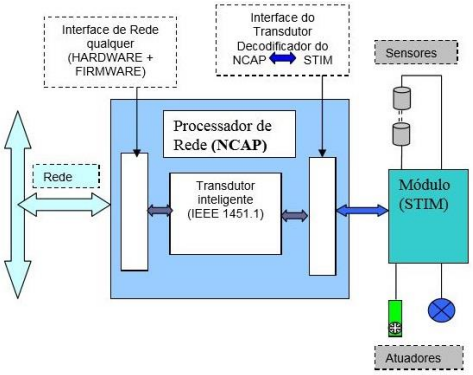

Figura 5: Ilustração da NCAP

2.3.2. Qual a funcionalidade da NCAP?

Uma NCAP pode ser utilizada como um controlador de nó numa rede de múltiplos elementos. Pode-se comunicar com outros nós NCAP, através da rede de utilizador, e com o utilizador final. Para isso faz uso de dois protocolos: protocolo cliente/servidor; ou protocolo de publicação/assinatura. Esta comunicação é chamada de comunicação de alto nível, e nesta comunicação as NCAP podem estar interligadas, utilizado a interface Ethernet, ou a comunicação wireless.

2.3.3. Comunicação entre NCAP

Para que seja possível estabelecer uma comunicação entre diversas NCAP é necessário que as mesmas façam parte de um conjunto de sensores inteligentes com a capacidade de serem inseridos numa rede de sensores. Assim, a comunicação entre sensores associados a uma NCAP faz mais sentido se os mesmos estiverem capacitados para medir uma determinada grandeza. Visto que os dados recolhidos pelos sensores devem ser enviados à central de observação dos mesmos, deve existir um sensor chamado *sink*, que em situação normal localiza-se próximo da central de observação, que requisita aos demais sensores disponíveis na rede, chamados de *sources*. Estes respondem com dados às requisições feitas pelo *sink*, caso possam efetuar a medida da grandeza requisitada.

Como mostra a figura 6, a disposição dos sensores não segue uma organização rígida e metodológica em termos de topologias. Quer isto dizer que eles podem estar organizados de uma ou de outra forma, o importante é que todos possam estar organizados e ser capazes de atender a uma solicitação de acordo com a requisição feita, o que se presume que essa rede não segue uma abordagem *multihop* para entrega de dados ao *sink*.

A NCAP contém a componente responsável pelo processamento de dados e comunicação dos sensores em rede com o utilizador final. Apesar de ser considerada a responsável por atribuir características de inteligência ao sensor, o trabalho de comunicação não é apenas da responsabilidade da NCAP. Deve-se também considerar a contribuição da TIM, pois a comunicação entre a NCAP e a TIM também designada de comunicação de baixo nível, esta comunicação é o foco deste trabalho. [17]

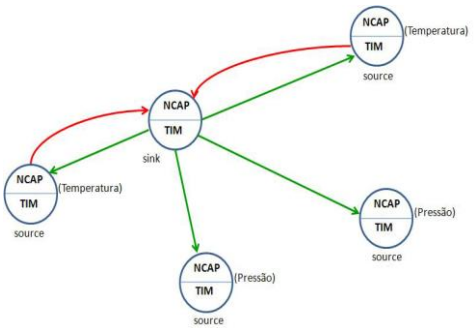

Figura 6: Ilustração de comunicação entre NCAP

Uma vez que a comunicação de baixo nível é efetuada por dois dispositivos diferentes, devemos aprofundar um pouco mais sobre a STIM. Assim, poder-se-á ter uma melhor perceção do que realmente é a comunicação entre estes dois módulos.

2.3.4. *Smart Transducer Interface Module STIM*

Assim como a NCAP é definida pela norma IEEE 1451, como intermediária entre a STIM e a rede de utilizador final, da mesma forma, a norma IEEE 1451 define uma interface que serve de ligação entre os transdutores e o sistema de aquisição de dados, tendo por base sistemas de instrumentação, microprocessadores, controlo e redes de campo. Assim, a STIM é a responsável pela produção de dados que servem de base à elaboração da informação a enviar aos transdutores. Ao sensor são disponibilizados os mecanismos necessários ao correto acondicionamento dos sinais a adquirir. Com a digitalização dos dados é possível implementar mecanismos de calibração e diagnóstico, seguindo-se o envio da informação através de um canal associado. Uma possibilidade disponível é suportada pela norma IEEE 1451.2 que define a comunicação série entre a NCAP e a TIM. [18]

A STIM é um módulo com interface que condiciona o sinal, efetua a conversão analógico-digital ou o inverso. A composição da TIM não é imutável, ela varia de acordo com a necessidade, pode ser composta desde um simples sensor ou atuador, permitindo a saída de dados em dois tipos (analógico ou digital), como também pode ser composta por uma unidade que contém inúmeros e diferentes transdutores. [19]

Quanto à sua arquitetura, a STIM é definida por um número de canais transdutores com capacidades para efetuar a medição de diferentes grandezas, como temperaturas, pressão, humidade e tantas outras de acordo com a necessidade do utilizador. Estes canais são os que permitem à NCAP definir que comando será enviado de acordo com os dados que se pretende analisar. [18]

O conjunto de blocos que aparecem na Figura 7 representam os elementos constituintes da STIM, e, cada um deles tem uma função específica a desenvolver. De salientar que apesar de existirem vários blocos XDCR (transdutores), apenas se considerar um nodo da rede, no sentido

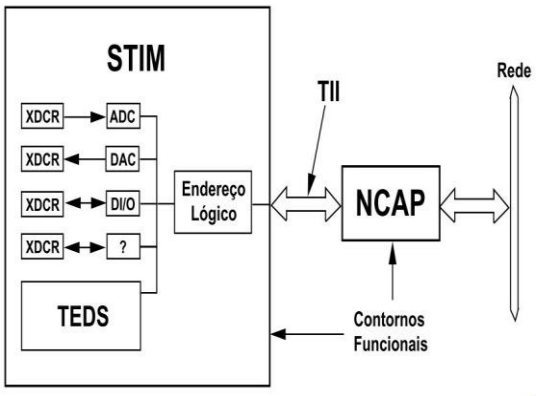

Figura 7: Ilustração de um STIM

de que todos eles desempenham a mesma função. De seguida será explicado ou apresentado a função de cada elemento ou bloco.

- Bloco XDCR - este bloco é a representação de um transdutor controlado e monitorizado por um microcontrolador que executa as funções de endereçamento, escrita, e leitura, conjuntamente com a gestão das TEDS. Segundo [20], o bloco é definido como um parâmetro escalar, onde é guardado a leitura de nível, é ainda descrito como a instancia de uma classe que descreve o canal de dados associado à entrada analógica;
- Bloco ADC, DAC - segundo o descrito em [21] e [22], estes dois blocos são conversores que permitem a leitura de dados ou grandezas físicas fornecidas pelo transdutores, sendo que este converte uma grandeza física em uma grandeza elétrica (tensão ou corrente). Logo, o ADC é responsável por converter os dados elétricos em valores digitais reconhecido pelo computador (0/1). Este último será responsável pelo processamento, filtragem e armazenamento de dados. O DAC é responsável por converter os dados digitais em analógicos. Desta feita, pode-se afirmar que estes dois conversores são utilizados para permitir a troca de informação entre dois sistemas, como mencionado em [21], ou seja, permitem a conversão e a transferência de dados do ambiente, mundo analógico para o mundo digital, de forma que os dados possam ser monitorizados;
- Bloco "?" - representa outros tipos de I/O não convencionais, diferentes do DAC/ADC e DI/O digital. Onde o bloco DI/O é o bloco que representa a entrada e saída de dados digitais.

2.3.5. A comunicação entre a NCAP e TIM

A comunicação entre a NCAP e a TIM, também conhecida como comunicação de baixo nível, não é tao simples como se imagina, pois precisa-se de apoio de outros elementos que contribuem grandemente para que esta comunicação se torne possível. Entre tantos elementos destaca-se a TII, como uma interface periférica serie, definida pelo padrão IEEE1451, também conhecida como a interface de comunicação digital entre a NCAP e a TIM como é ilustrado na figura 8. Isto não quer dizer que esta seja a única interface que permite a comunicação da NCAP, pode ainda utilizar diferentes interfaces, quer sejam elas seriais ou paralelas, isso fará com que a NCAP possua a compatibilidade com diversos módulos TIM que possam existir, e para tornar possível esta comunicação a IEEE 1451 aprovou um padrão IEEE1451.5 que permitiu uma padronização de comunicação wireless entre NCAP e a TIM.

Tal como já foi dito, a NCAP é responsável pelo processamento de dados recolhido da TIM, para efetuar este processo existem dois tipos de memórias para a NCAP, uma é a memoria de dados, neste são armazenados todos os dados extraídos da TIM a ser processado pela NCAP, este processo é realizado devido a um programa de execução da NCAP que é armazenado em uma outra memória conhecida como memória programável. [16]

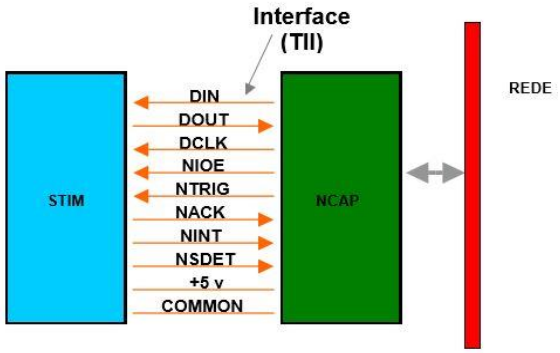

Figura 8: Ilustração da comunicação entre a NCAP e a STIM segundo a norma IEEE 1451.2

Em muitos casos apresenta-se a TII como parte da TIM, apesar deste trabalho ter como o foco o desenvolvimento da NCAP, não se pode deixar de falar deste elemento tão importante que serve de ligação ou de canal e, que facilita em grande parte a comunicação entre a NCAP e TIM, ele é ainda o responsável pela definição de protocolos, especificações elétricas e temporizações que são utilizadas para assegurar o transporte de dados robustos extraídos das diversas combinações de NCAP e STIM que circulam pelas 10 linhas da TII. Oito destas linhas destinam-se à implementação das funcionalidades definidas pelo padrão. As duas linhas restantes são utilizadas pela NCAP para fornecer energia à STIM. Estas mesmas linhas são agrupadas em quatro grupos distintos como encontrado em [23]: dados; disparo; suporte; e interrupção.

Uma das questões levantada muitas vezes é: porque são necessárias várias linhas para se efetuar uma comunicação bilateral?

esta pergunta é levantada porque muitos estão acostumados com o conceito de que a comunicação se resume apenas no envio e na receção de mensagens (texto ou áudio), mas para o caso da comunicação da NCAP e a STIM é muito mais do que isso, a comunicação para eles é envio e receção de qualquer sinal (dados), cada linha da TII desempenha uma função especifica, e a particularidade que estas mesmas linhas possuem, é que elas não são bidirecionais. A tabela 1 mostra as caraterísticas de cada uma das linhas, bem como a sua função.

É necessário ressaltar que na coluna “Guiado por” da tabela 1, depende da perspetiva que se vê, sobre tudo para a STIM, isso para não se ter a ideia de que muitos deles são controlado pela STIM, apenas NINT é totalmente controlado pela STIM.

Tabela 1: Caraterísticas das linhas de TII

Linha	Lógica	Guiado por	Função da linha
NSDET	Ativo baixo	STIM	Usado pela NCAP para detetar a presença de um STIM
POWER	N/A	NCAP	Fonte de potência nominal
NTRIG	Borda negativa	NCAP	Desempenha funções de disparo
NACK	Borda negativa	STIM	Reconhecimento de disparo Reconhecimento de transporte de dados
DIN	Positiva	NCAP	Transporte de dados e endereço da NCAP para o STIM
DOUT	Positiva	STIM	Transporte de dados do STIM para a NCAP
DCLK	Borda Positiva	NCAP	A troca de dados é feita na borda positiva sobre a DIN e DOUT
NIOE	Ativo baixo	NCAP	Ativação de sinais de transportes de dados e delimitação dos moldes dos transportes.
NINT	Borda negativa	STIM	Usado pra STIM para requisitar serviços da NCAP
COMMON	N/A	NCAP	Sinal comum ou terra

Tabela 2: Atribuição de pinos e cores, estabelecida pelo padrão - baseado em A. Amaral

Pinos	Sinal	Cor	Direção para a NCAP	Direção para o STIM
1	DCLCK	Castanho	OUT	IN
2	DIN	Vermelho	OUT	IN
3	DOUT	Laranja	IN	OUT
4	NACK	Amarelo	IN	OUT
5	COMMON	Verde	POWER	POWER
6	NIOE	Azul	OUT	IN
7	NINT	Violeta	IN	OUT
8	NTRIG	Cinza	OUT	IN
9	POWER	Branco	POWER	POWER
10	NSDET	Preto	IN	OUT

Como foi dito no parágrafo anterior, a tabela 1 espelha a configuração dos pinos de acordo os quatros grupo de linha existentes na TII, bem como a função ou a especificação de cada uma das linhas, mas isto não quer dizer que a tabela espelha a organização das linhas de forma organizada, o padrão especifica a ordem bem como a cor de cada pino para que se possa facilitar a conexão da NCAP com qualquer STIM, a tabela 2 mostra essa organização.

Com as duas tabelas apresentadas, pode-se agora mostrar com mais detalhe como é feita a comunicação entre os dois modelos do padrão.

A comunicação começa justamente do lado da NCAP, uma vez que é por muitos destacados como o chefe, esperando a resposta do STIM como empregado, e isto começa com as linhas ou pinos que formam o grupo Suporte (NSDET e POWER):

A) Ocorrência de processo de inserção;
B) A ativação da potência da STIM por NCAP;
C) Ocorrência de um evento de extração;
D) A NCAP retira a potência antes fornecida para STIM.

A sequência apresentada pode não ocorrer ao mesmo tempo, pois, pode-se verificar um atraso ou não na execução de uma ação para outra, primeiro vai ilustrar-se o controlo de potência do STIM com atraso (figura 9), este controlo é chamado de controlo ativo, será mostrado de seguida o que acontece realmente em cada ponto antes mencionado.

A) A NCAP envia um sinal que permite detetar a existência de uma STIM com o qual poderá se comunicar;

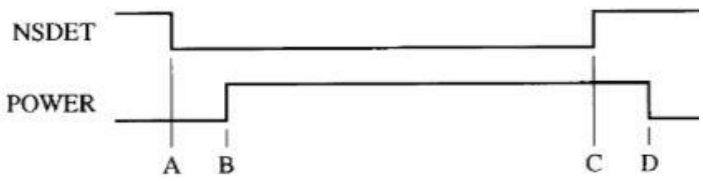

Figura 9: Controlo de potência do STIM com atraso

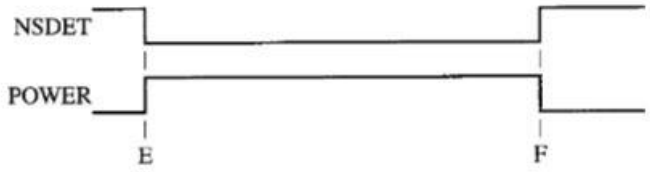

Figura 10: Controlo de potência do STIM sem atraso

B) O sinal enviado pela NCAP retorna informando que existe uma determinada STIM, a NCAP pode atrasar algum tempo para reconhecer este sinal, mas, apesar do atraso, ela envia ou inicia a alimentação da STIM. O sinal NSDET verificar continuamente a existência de uma STIM;
C) Se em algum momento não for detetado uma STIM o NSDET informa a NCAP;
D) Quando a NCAP percebe que já não existe uma STIM, ele suspende o fornecimento de energia.

2.3.6. Controle ativo da potência no STIM enviado pela NCAP

No caso em que exista um controlo da energia fornecida à STIM sem atraso, chamado de controlo não ativo, verificam-se as mesmas ocorrências, só que já não ocorrem de forma sequencial tal como se viu no controlo ativo. Para este caso, as ocorrências são ativadas em paralelos, ou seja, as inicializações dos dois primeiros eventos ocorrem em simultâneo assim como o término deles.

E) A NCAP envia um sinal que permite detetar a existência de uma STIM com o qual poderá comunicar e em simultâneo ativa a alimentação do circuito.
F) Verifica-se a inexistência da STIM e, em simultâneo, a NCAP descontinua a alimentação dos circuitos da STIM, cancelando o envio de energia à STIM, como se verifica na figura 10. [23]

2.3.7. Controle não ativo da potência no STIM enviado pela NCAP

O resto dos elementos que constituem a TII são ativados se o processo especificado nas alinhas anteriores (A-F) se cumprirem. Quando verificada a existência de uma STIM e se efetuar a ativação de energia para alimentar os circuitos de STIM, a NCAP ativa a linha NTRIG da TII que suporta a função de disparo estabelecendo o momento em que determinado comando será enviado pela NCAP para o STIM. O sinal enviado da NCAP para a STIM é chamado de sinal de disparo, assim que a STIM receber o sinal de disparo ativa o NACK, pois uma das funções de NACK é reconhecer o disparo efetuado pela NCAP. De seguida, a STIM usa a mesma linha para

informar a NCAP de que pode ser efetuado o envio de dados, quando a NCAP recebe a resposta vinda da STIM ativa a linha NIOE, inicializando a entrada e saída de dados, sinalizando a ativação do transporte e delimitando a estrutura de dados. É neste período que as linhas DIN e DOUT são ativadas, pois elas são encarregues de dar a entrada e saída de dados da STIM para a NCAP. A transferência de dados pelas linhas DIN e DOUT é controlada pela linha de relógio DCLK. A cada transição do relógio os dados são armazenados. A linha cujo controlo é realizado totalmente pela STIM é a MINT pois é através desta linha que o STIM requer um determinado serviço à NCAP. [23]

Feita esta abordagem, cabe agora selecionar a plataforma de desenvolvimento, ou seja, as ferramentas que possibilitarão a realização da comunicação entre a NCAP e a TIM. O capítulo 3 tratará de descrever a plataforma de desenvolvimento para esta comunicação.

Capítulo 3 - Plataforma de Desenvolvimento

3.1. Introdução à plataforma *Raspberry Pi*

Sendo um trabalho que necessita ser implementado, não será novidade dizer que se fará uso de uma ferramenta que possibilitará alcançar o objetivo do trabalho, para tal optou-se pela ferramenta *Raspberry* Pi 3 modelo B.

O objetivo deste capítulo não é falar tudo sobre o *Raspberry* Pi, mas sim trazer um conhecimento básico para que se tenha ideia da versatilidade deste pequeno dispositivo.

Sem falar da etimologia da palavra, o *Raspberry* Pi é uma placa com um tamanho aproximado a um cartão de crédito, é considerado um computador completo pois quando ligado a um monitor, teclado e rato pode fazer a maioria das coisas que um *Desktop* faz. É considerado um dos lançamentos mais extraordinário dos últimos anos.

A ideia de se ter um computador de baixo custo e de fácil portabilidade, surgiu em 2006 na universidade de Cambridge, onde a maior preocupação era facilitar o ensino de informática nos colégios e que cada estudante pudesse possuir um computador barato o suficiente, para que junto com outros materiais pudessem efetuar as suas práticas em casas.

Foi no final de 2011 que Pete Lomas o criador do Hardware do *Raspberry* Pi e cofundador da *Raspberry fundation* com a sua equipa fizeram o lançamento do *Raspberry* Pi.

Uma das coisas que mais impressionou no *Raspberry* Pi, é o facto de se efetuar o *reset* com a maior simplicidade, uma vez que era manuseado inicialmente por pessoas menos experientes. Para não se perder o pequeno e poderoso computador Pi, a sua configuração permite que se possa recuperar a configuração original apenas com um simples *reset*, por mais alterações que o utilizador faça.

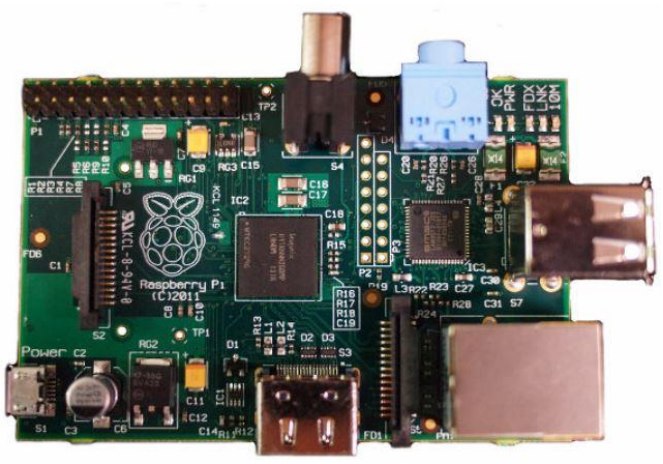

Figura 11: Raspberry Pi lançado em 2011

Com as caraterísticas apresentadas pela placa *Raspberry* Pi, foram superadas a espectativas, ultrapassando em grande escala o objetivo estabelecido. O *Raspberry* Pi tornou-se famoso entre as pessoas ligadas à tecnologia, além de ser usado apenas por estudantes que tinham o desejo de aprender a informática ou programação básica, o *Raspberry* Pi passou a ser usado até mesmo por pessoas comuns, ou seja, por pessoas que já tinham muita informação sobre a programação, o que fez com que passasse a ser usado para desenvolvimento de vários projetos que contribuem em grande parte no desenvolvimento tecnológico. Talvez até aqui ainda não se perceba o motivo de tanta euforia ao se falar do *Raspberry* Pi, então observe as caraterísticas do *Raspberry* Pi.

O *Raspberry* Pi tem uma dimensão de 8.5 x 5.3 cm, o que facilita a sua portabilidade. Possui um *chip* integrado *Broadcom* BCM283x (variando de acordo o modelo), este *chip* contém um processador com inúmeras frequências de funcionamento. Possui um processador gráfico *VideoCore* IV, possui uma quantidade de memória que varia de acordo o modelo, entre os 256MB e o 1 GB. Faz parte das suas caraterísticas uma saída de vídeo e áudio através de uma ficha HDMI (saída de vídeo composto), bem como uma saída simples de áudio através de um *minijack*. Com exceção dos modelos A e A+, outros modelos possuem uma ligação *etherneth* 10/100.

Desde o primeiro dispositivo lançado em 2011 até ao momento, já foram criados 8 modelos diferentes do *Raspberry* pi, sendo o mais usado no momento o *Raspberry* Pi 3 Modelo B que é ilustrado na figura 12, por causa das inovações que apresenta. De seguida será exibido uma figura que mostrará a descrição das partes que compõe o *Raspberry* Pi 3 Modelo B.

Como se verifica na figura 12, cada número representa um elemento cuja descrição será mostrada a seguir.

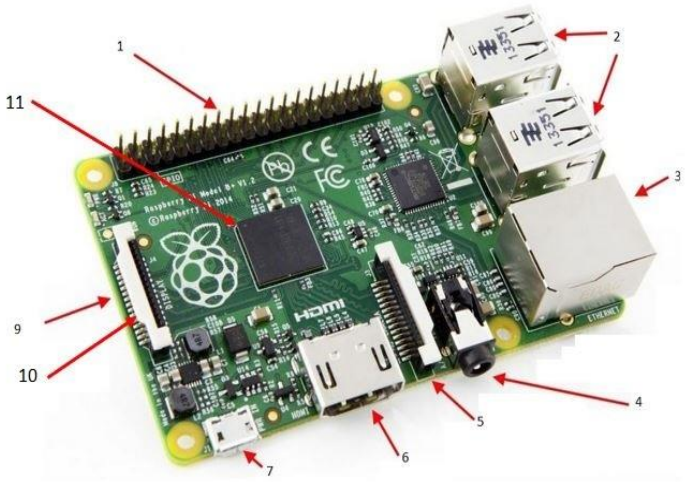

Figura 12: Elementos que compõe o Raspberry Pi 3 Modelo B

3.1.1. Composição do *Raspberry Pi* 3

1. Conjunto de 40 pinos que formam o GPIO. Permite a interação com outros tipos de hardwares, como Leds, Motores, Arduíno.
2. 4 portas USB que possibilitam a ligação de teclado, rato, pendrives bem como outros dispositivos de entrada e saída.
3. Porta Ethernet com ficha RJ45 que possibilita a ligação da placa à internet6 por meio de um cabo de rede.
4. Conetor de áudio analógico que permita a saída de áudio da placa por meio de fones de ouvido ou alto-falante.
5. ficha para camara CSI.
6. Porta HDMI que permite a saída composta de Vídeo e áudio.
7. 5V Micro USB permite alimentar a placa com 5V.
8. LED status que sinaliza o funcionamento da placa
9. *Slot* para cartão Micro SD (*Secure Digital*) onde é inserido o cartão SD que armazena tudo.
10. ficha de exibição DSI.
11. Chip BCM2837 que incorpora o processador. [24]

É importante salientar que, embora digamos que o *Raspberry* Pi é um microcomputador, não se pode ter a ideia que ele possa realizar todas e qualquer tarefas realizadas com um *Desktop*, por causa de algumas limitações existentes. Por exemplo: de modo a minimizar o custo, os desenvolvedores decidiram incluir um único *chip* de memória LPDDR de 256 MB, o que de certa forma limita o desempenho e o uso de aplicações *descktop*. No que concerne ainda à memória, esta divide os 256 MB por duas partes, uma para o processador e a outra para a GPU, permitindo desta forma que a GPU fique com 186 MB, enquanto que a restante parte é reservada para o sistema. O *Raspberry* Pi não possui uma memória de armazenamento interno HD, o armazenamento é feito no cartão SD que é introduzido na ranhura que se encontra na parte inferior da placa. O mesmo acontece com a alimentação, o *Raspberry* Pi não possui uma fonte de alimentação capaz de armazenar energia e fazer uso dela caso não esteja ligado à corrente. O *Raspberry* Pi necessita de estar continuamente alimentado enquanto estiver em uso. A sua alimentação é feita por meio de uma porta micro USB, que foi escolhido para permitir que o custo se mantenha baixo pela facilidade de poder utilizar qualquer carregador de celular que permite que os 5V recebidos sejam enviados aqueles componentes que necessitam de 5V para o seu funcionamento, como é o caso de dispositivos ligados a portas USB, bem como a porta HDMI. Outro detalhe, não menos importante, que faz com que o *Raspberry* seja diferente de um *Desktop*, é o fato de que o *Raspberry* não possui a BIOS ou Setup, as suas configurações que se relacionam com o Hardware e ao processo de arranque do sistema, encontram-se em um arquivo de texto que podem ser localizados na raiz do cartão, o "*config.text*", esta raiz pode conter as diversas opções que em um PC podem encontrar-se no Setup.

Não existe um computador sem um sistema operativo, visto que se afirmou que o *Raspberry* Pi é um microcomputador, o normal é que possuísse um Sistema operativo (SO), apesar de não vir com um SO, o *Raspberry* é compatível com várias distribuições de Linux, como Debiam, Arch Linux e Fedora, mas de lembrar que os SO que correm em *Raspberry* pi são diferentes daqueles que correm na plataforma PC, uma vez que a plataforma não tem a BIOS, também por não ter uma imagem única para dispositivos ARM, enumeração de dispositivos *plug-and-play*, faz com que seja difícil para o sistema identificar o Hardware automaticamente no decorrer do processo de arranque. Felizmente muitas pessoas se voluntariaram para dar solução a este problema, desenvolvendo várias imagens que podem ser descarregada e guardá-las no cartão SD que será colocado no *Raspberry* pi. Só assim se tornou possível ligar dispositivos a *Raspberry* pi e usá-la como um computador como ilustrado na figura 13.

Pela razão apresentada na figura 13, fez com que se escolhesse a placa *Raspberry* Pi como sistema de desenvolvimento a ser utilizado neste trabalho, porque facilita o contacto com os resultados que se obterão sem ser necessário a presença de um computador de alto nível, mas como não deixaria de ser, serão necessários várias ferramentas que possibilitarão a obtenção dos resultados que se espera, e, estes, serão apresentados no capitulo seguinte.

Várias vezes foi dito que a placa *Raspberry* Pi funciona como um computador, e, como se sabe, um computador necessita de um sistema operativo para o seu funcionamento. Este caso não foge à regra, tendo sido selecionado o sistema operativo (SO) o Debian.

Segundo [25] e [26] Debian é o nome de um projeto que ao mesmo tempo dá o nome ao sistema operativo que é a junção dos nomes de um dos fundadores Ian Murdock e a sua esposa e Debora. É um SO inteiramente composto por Software livre, e, mantido por meio de doações. Assim, qualquer pessoa pode utilizar, efetuar alterações e distribuir, desde que as alterações estejam devidamente documentadas. Tal como em outros sistemas operativos, no núcleo do Debian está o Kernel que permite que a placa possa executar outros programas.

O Debian utiliza o kernel do Linux, e para auxiliar o kernel, o Debian tem como utilitários e ferramentas do GNU o que faz com que seja chamado muitas vezes de Debian GNU/Linux. O Debian é ainda conhecido por fazer o uso de um conjunto de pacotes APT do GNU/Linux para fazer a gestão dos seus pacotes, e isto serve de uma vantagem porque permite-o realizar

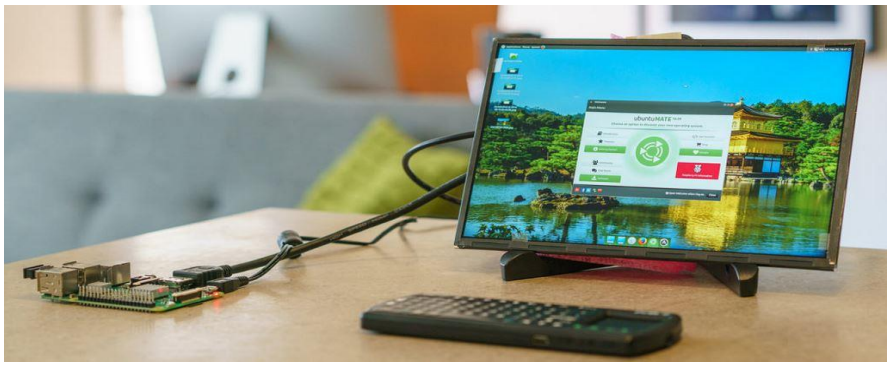

Figura 13: Exemplo do Raspberry como PC

atualizações de forma fácil a partir de versões antigas, efetuando ainda instalações de novos pacotes e a remoção dos anteriores, o que permite manter sempre os pacotes mais estáveis.

3.2 Preparação das ferramentas de desenvolvimento

Para este trabalho foram utilizadas algumas ferramentas como *Python, Eclipse*, PyDev, *Remote Debug*, nesta secção serão apresentados, as definições de cada ferramenta, assim como o processo de instalação.

Python é uma linguagem de programação que tem como criador Guido Van Rossum em 1991, foi um projeto que tinha como objetivo a produtividade e legibilidade. Considerada como uma linguagem de programação interpretada de código-fonte aberto e disponível para correr em diversos sistemas operativos. Desta forma, o python não precisa ser compilado, apenas precisa ser interpretado. Ao ser escrito um programa nesta linguagem, para que a máquina o possa executar, é lido por um segundo programa, que serve como interpretador, traduzindo para a máquina aquilo que o código quer dizer. O interpretador para a linguagem Python é interativo, disponibilizando assim uma interface por onde são inseridos todos os possíveis comandos de acordo com a necessidade do programador, podendo observar o efeito de cada comando. [27]

A escolha desta linguagem está relacionada em parte com o facto de a placa escolhida ter sido projetada inicialmente para ser programada em python, tanto é que o *Raspberry* Pi que vem depois da palavra *Raspberry* é a representação da linguagem de programação que seria usada, o outro motivo é, devido as caraterísticas que a mesma linguagem apresenta.

- Pouco uso de carateres especiais, e isso faz com que a linguagem se torne mais próxima do pseudocódigo executável;
- Não uso frequentes de palavras reservada voltada para o sistema, o que torna a linguagem mais próxima da linguagem natural.

Eclipse é a denominação de uma comunidade de código aberto criada pela IBM em 2001 em que os seus projetos são concentrados na criação de plataforma de desenvolvimento aberta. Como ferramenta são apresentadas várias definições, mas muitas destas a tornam conhecida como uma plataforma de software livre baseado em java, para desenvolvimento integrado, ou de forma simples pode-se dizer que o Eclipse é um IDE para desenvolvimento Java que suporta várias outras linguagens de programação como por exemplo, PHP, Java, C/C++, PHP, ColdFusion, Python, e plataforma Android. [28]

Além das ferramentas citadas acima foram usadas também as ferramentas como, *Pydev, Remote System Explorer* (RSE).

O Pydev permite utilizar a linguagem Python e desenvolver códigos a partir do Eclipse, ao passo que RSE permite ter acesso remoto à placa por meio do PC, desta forma pode-se ter acesso a placa sem ter de ligar a ela um monitor. Apesar de todas esta ferramentas serem consideradas como aplicativos, ainda assim nem todas elas são instaladas como tal, tudo porque estas devem

ser incorporadas no Eclipse, desta feita, foi necessário seguir alguns passos para se ter o ambiente de desenvolvimento disponível, como será mostrado de seguida.

3.2.1 Instalação de ferramentas

1. Baixou-se e efetuou-se a Instalação do Python 2.7. Como um programa normal;
2. Considerando a possibilidade de que já existe na máquina (PC) *Java Runtime Environment* (JRE), seguiu-se os passos que serão alistados de seguida. A existência do JRE é fundamental, pois ele permitiu que se pudesse executar o Eclipse no PC;
3. Fez-se o download do Eclipse para o sistema operacional em uso (Windows);
4. Descompactou-se o Eclipse e efetuou-se a instalação do mesmo;
5. Efetuou-se a instalação do Pydev, como será mostrado abaixo.

Para se efetuar a instalação do Pydev, iniciou-se o Eclipse clicando duas vezes no *icon* do Eclipse criado no ambiente de trabalho depois da sua instalação, apresenta o caminho do *Worckspace* como é mostrado na figura 14.

Ao clicar *Launch*, apresenta uma outra janela que é a janela de boas vindas do Eclipse como mostra a figura 15.

Clicando em Ajuda (*Help)*, abre-se uma caixa de texto com várias opções, clicando em Eclipse *Marcketplace*, abre-se uma janela, na caixa de pesquisa, com um *click* em palavra Pydev e em seguida em ir (Go), apresentando assim o aplicativo ou plugin Pydev, que é instalado como mostra 16.

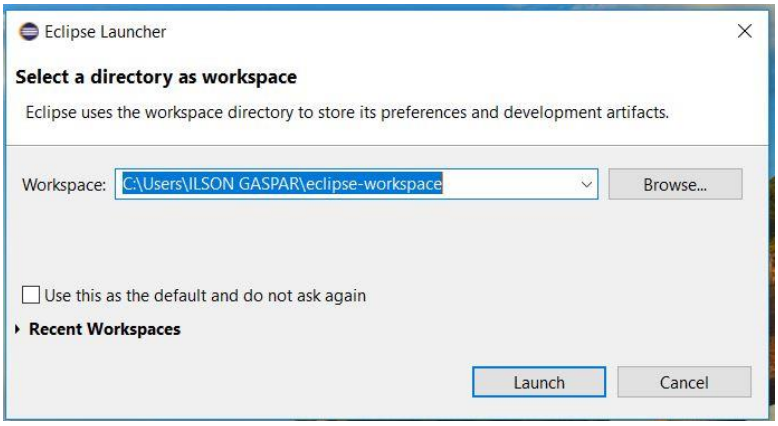

Figura 14: Caminho do *Worckspace*

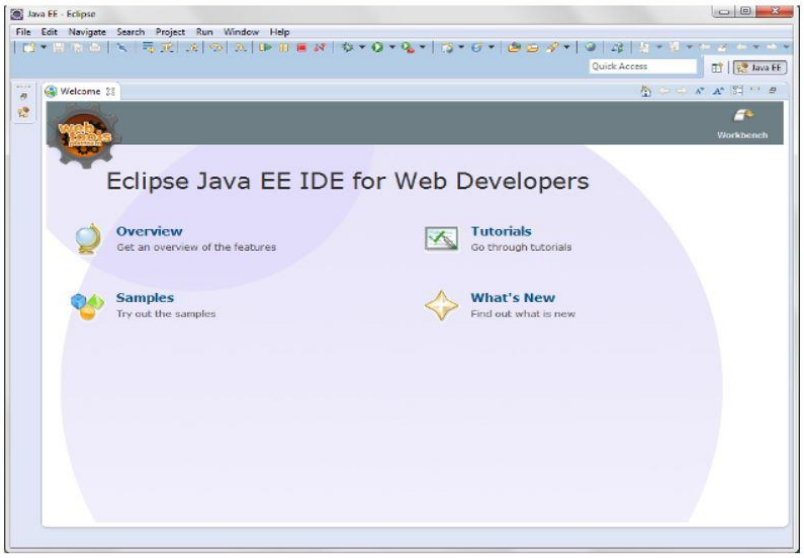

Figura 16*: Janela de boas vindas do Eclipse*

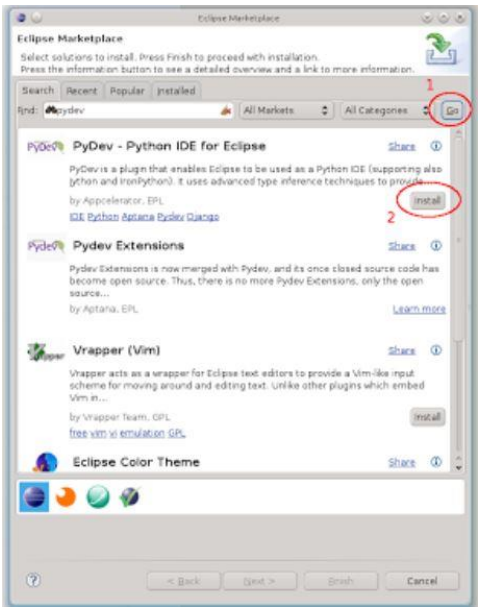

Figura 15: Janela de instalação de Pydev

6. Instalação de RSE, o processo de instalação do RSE, é semelhante ao do Pydev, com a particularidade de o RSE não se encontrar em *Marcketpalce* logo, para efetuar a instalação do SER seguiu-se os seguintes passos.

   Clicando em ajuda (*help*), das várias opções clicou-se em instale um Novo Software (*Instale New Software*), abrindo uma janela nova, no campo "trabalhar com", escolheu-se a opção todos os sites disponíveis (--*All Available Sites*--), e, lista um conjunto de ferramenta, para facilitar a busca, na caixa de filtro colocou-se a palavra "*Remote*" e selecionou-se as duas ferramentas como como mostrará a figura 17.

Para verificar se realmente as ferramentas foram instaladas com sucesso, clica-se em Window (no botão do menu), *perspective*, *open perspective* e outro (*other*), os *plugins* instalados aparecem todos em uma única janela, mostrando assim que foram instalados com êxitos. Como se pode ver nas figuras 18 e 19.

Com o processo de instalação de ferramentas concluído com sucesso, pode-se realizar a ligação da placa *Raspberry Pi* a uma rede utilizadores, como será descrito na secção seguinte.

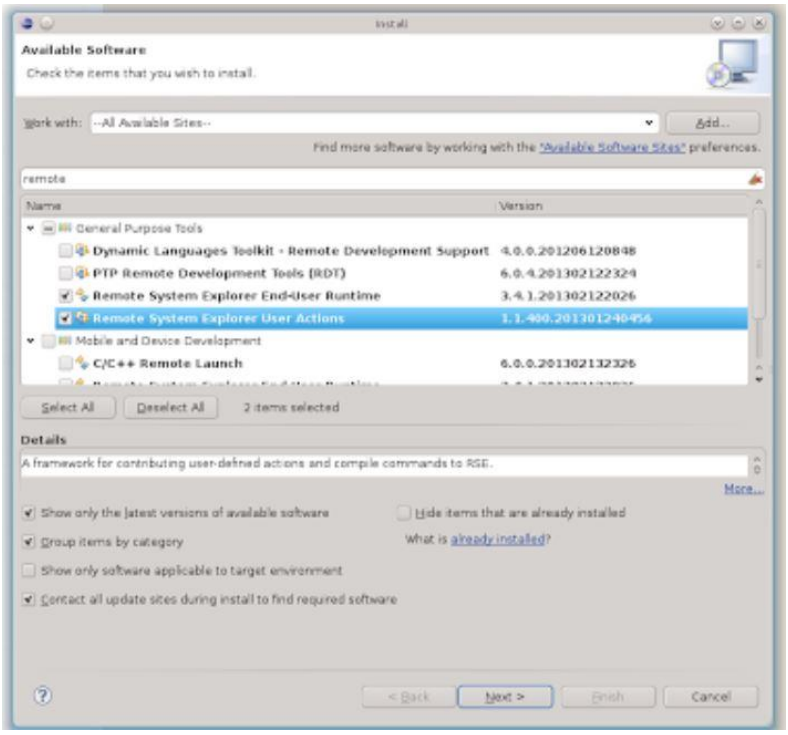

Figura 17: Instalação de RSE

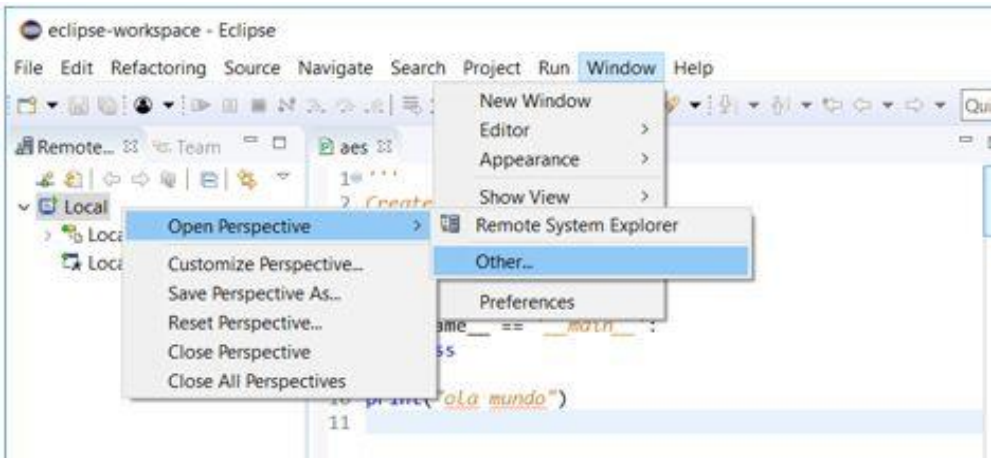

Figura 18: Passos para chegar a lista de Plugins

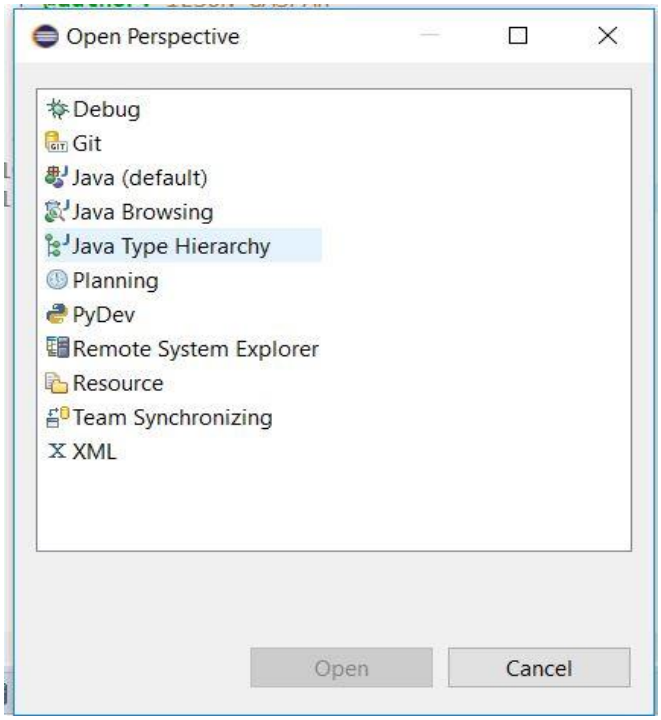

Figura 19: Lista de plugins

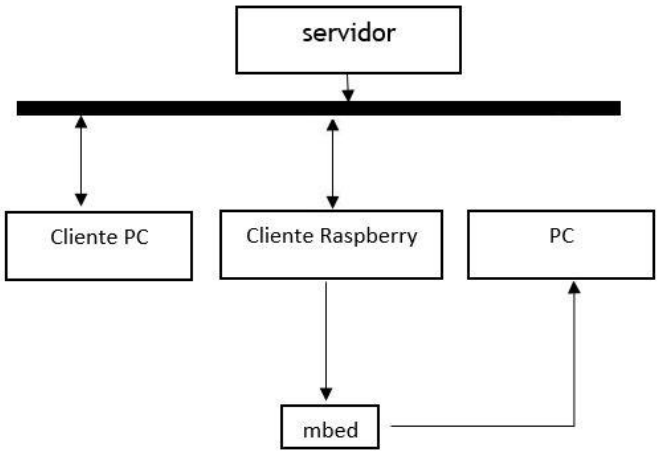

Figura 20: Estrutura de redes de utilizadores

3.3. Ligação do *Raspberry Pi* a uma rede de utilizador

A figura 20 representa a rede montada para a realizar a comunicação entre os utilizadores, uma rede pequena com apenas três (3) utilizadores, o que não quer dizer que é o número limite, podem ser acrescentados mais utilizadores caso seja necessário, por se tratar de apenas uma experiência, foi construída a rede com apenas três utilizadores, o Servidor, Cliente PC e o *Raspberry* Pi. Os outros elementos que aparecem na estrutura da rede, não fazem parte dela, são apenas elementos intermediários, ou seja, o elemento "PC" é um computador que permite o acesso remoto da placa *Raspberry* via SSH, o que quer com isto dizer que a placa *Raspberry* se conecta a servidor por meio do PC, já o elemento mbed é uma placa como mostra a figura 21, utilizada para permite a entrada de dados via teclado na placa *Raspberry*, segundo [29]

Figura 21: Placa mbed

mostra que a mbed não foi fabricada apenas para este propósito, mas para este projeto este é o único objetivo, o *Raspberry* é conectado com o mbed via UART e por sua vez esta é conectada ao PC via USB.

Assim sendo não será necessário conectar um monitor nem qualquer dispositivo de entrada na placa para que ela se comunique com os outros elementos da rede, a informação enviada para a *Raspberry* será visualizada no PC ao passo que os dados são introduzidos meio do teclado que está ligado ao PC.

Para se cumprir com o objetivo desta secção, é necessário que os dispositivos façam parte de uma rede de utilizadores ligados entre eles, isso permitirá que se possa ter os dados existente em diferentes dispositivos, ou pelo menos, ter acesso aos dados disponibilizados por eles.

Nesta secção será feito uma demostração de comunicação existente entre os dispositivos, ou seja, os utilizadores da rede. Foram usados dois computadores por onde se foram inicializados vários utilizadores virtuais. Foi necessária uma placa mbed NXP LPC1768 com o objetivo disponibilizar uma porta serial virtual através da interface USB, permitindo a comunicação com um terminal de PC. Desta forma, o *Raspberry* torna-se um dos utilizadores, podendo-se comunicar com os outros utilizadores por meio de uma porta COM que possibilita a entrada de dados por meio de teclado conectados ao computador.

De seguida será mostrado o processo de ligação da *Raspberry* à rede de utilizadores. Fazendo uso do PuTTy, fez-se a configuração para permitir que o computador reconhecesse a mbed, que possibilitará dar entrada de dados a *Raspberry* por meio do teclado, lembrando ainda que a ligação é feita utilizando a ligação serial em uma porta específica, estabelecida pelo computador. A figura 22 mostra a interface de configuração da mbed por meio do PuTTY.

A figura 23, mostra a configuração de acesso remoto ao *Raspberry*. Assim como a configuração do mbed, o processo de configuração para o acesso remoto à placa *Raspberry* Pi é o mesmo, com a particularidade de: Para o *Raspberry* a conexão é SSH, sendo esta ligação, permite que a placa seja acedida de forma remota, com a placa ligada a internet obtém-se o seu IP, que é

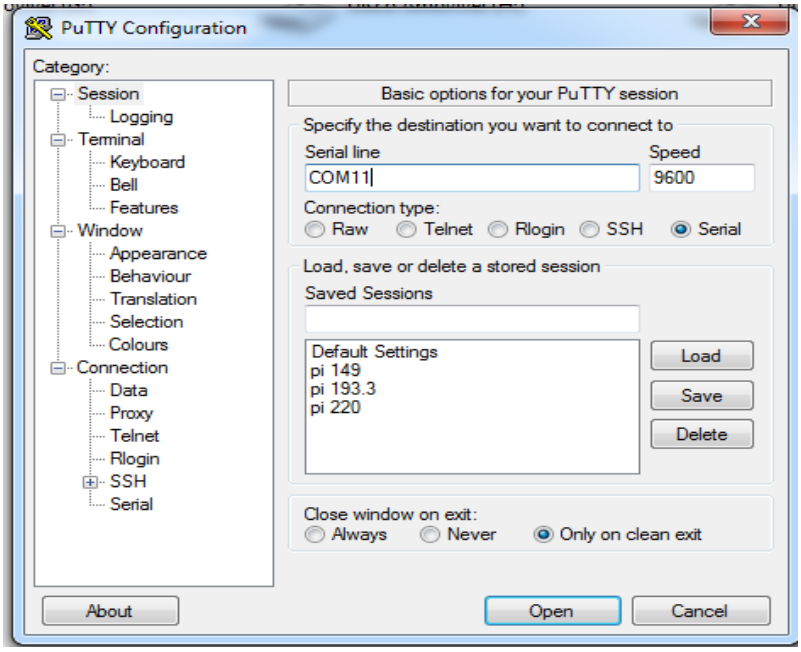

Figura 22: Configuração da mbed com o PuTTy

colocado na caixa de texto "Host name (*IP address*)", os outros campos são preenchidos de forma automática. Tendo concluído este processo a placa está pronta a ser acedida a partir do computador configurado.

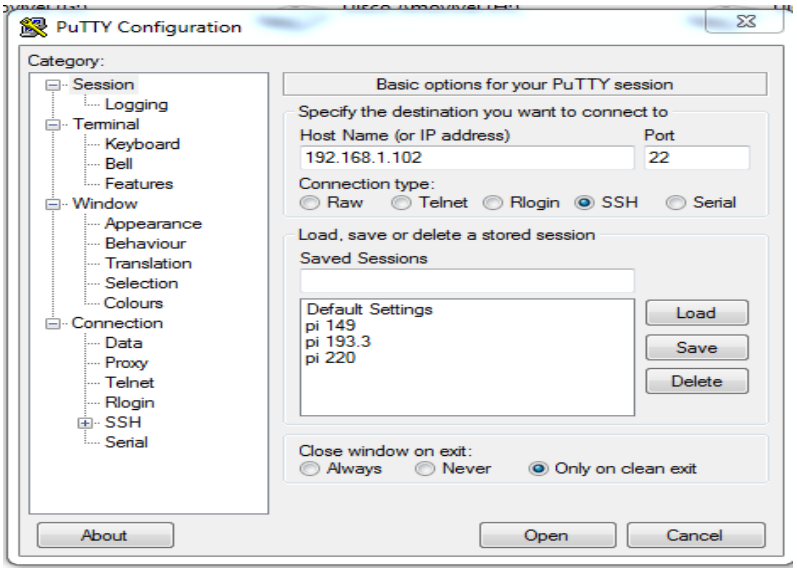

Figura 23: Configuração de acesso remoto ao Raspberry

Por defeito, a placa *Raspberry* Pi possui um *user name* e uma *password*, que é necessário para se ter acesso a placa por meio de um computador, basta inserir como *user name* "pi" e como *password* "*Raspberry*". Após o *user name* e a *password* terem sido inseridos corretamente, pode-se ter acesso aos ficheiros remotamente como mostram as figuras 24 e 25. Lembrando que o acesso aos ficheiros da placa não é realizado por uma interface gráfica, uma vez que se

está a trabalhar com o *prompt* do PuTTy. O acesso é feito por meio de comandos básicos de Linux como será mostrado nas figuras já citadas.

Com o login efetuado, pode-se visualizar as pastas da placa *Raspberry* a partir do computador configurado, como mostra a figura 25.

Para que a placa *Raspberry* seja ligada a uma rede de utilizadores, foi necessário que se produzissem códigos que permitiram a conexão ou a inserção dos dispositivos nesta mesma rede. Produziu-se dois códigos em python, um para o servidor que permitirá que os utilizadores possam ter acesso a um ID e posteriormente trocar informação entre utilizadores (cliente-cliente), bem como permitir a troca de informação entre cliente e o servidor (cliente-servidor). Começando pela atribuição de um ID, como mostra a figura 26 e 27. O funcionamento da comunicação feita em rede com o *Raspberry* será mostrado na secção 3.4.

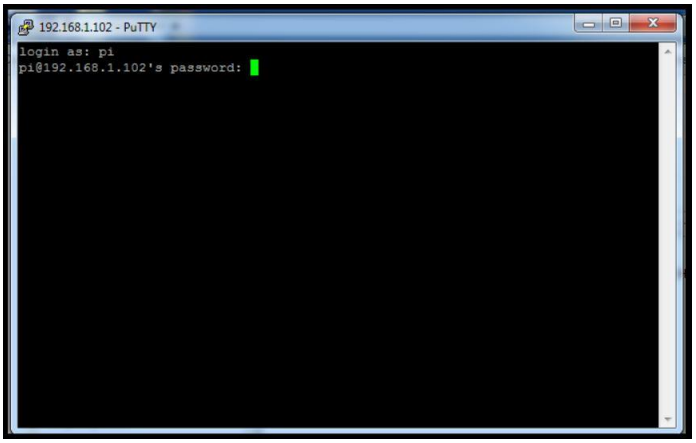

Figura 24: Login do Raspberry Pi remotamente por meio do PuTTy

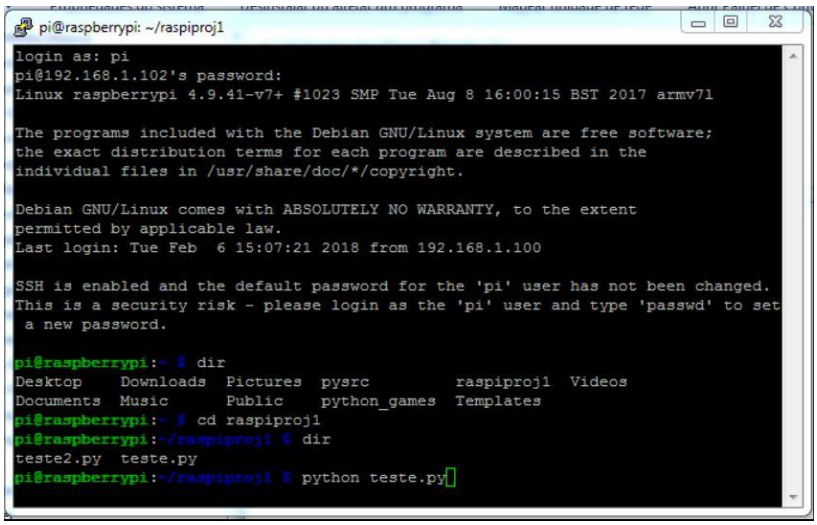

Figura 25: Interface do Raspberry Pi vista do computador

3.4. Exemplo demonstrador de operacionalidade

Existe uma sequência de passos a serem dados para que a comunicação aconteça dentro da rede instalada. Criou-se uma rede com dois utilizadores, onde um deles é a placa *Raspberry* Pi e o outro um *Desktop*. Tendo sido criado a codificação para comunicação na placa e no computador, ambos devem estar padronizados, isto é, ligados ao mesmo servidor para permitir a comunicação entre eles, isso é feito por identificar em seus códigos de programação, o IP do servidor na qual estarão ligados.

O processo de comunicação começa com o arranque do servidor, que também é um código de programação rodando em uma máquina ligado em rede com os utilizadores, de seguida coloca-se a rodar os códigos dos utilizadores. Tão logo que começa a rodar os códigos dos utilizadores o servidor faz a atribuição dos ID aos utilizadores de acordo a ordem de ligação, começando o 1, a posição zero (0) é reservada para o servidor. As figuras abaixo exemplificam a execução da mensagem. O arranque do servidor não causa nenhum impacto visual, pois este não abre nenhuma interface gráfica. Neste caso, as figuras que serão mostradas serão apenas dos utilizadores, porque o arranque dos seus programas, modifica as suas interfaces.

A figura 26 é praticamente a continuação da figura 25. Da mesma forma, a primeira linha de comando da figura 26 mostra o arranque do programa na placa, a segunda linha mostra a resposta do sistema, informando a conexão do utilizador ao servidor, e de seguida é enviado o seu ID, que é o "2", o servidor envia um *broadcast* a todos os utilizadores. A figura 27 na sua primeira linha ("REGISTED WITH ID – 1"), à placa é atribuído o ID 1, por ser o primeiro utilizador a se ligar na rede, como já foi explicado no parágrafo anterior. A atribuição do ID não é feita de forma aleatória, mas sim, de acordo com a ordem de ligação dos utilizadores. Assim, como acontece com o utilizador *Raspberry*, também acontece com o utilizador *Desktop*. A figura 28 mostrará o resultado da comunicação entre a placa e o utilizador *Desktop*.

Ao observar a figura 26, verifica-se que, a conexão do utilizador *Desktop* e a resposta do sistema são apresentado na mesma interface, ao passo que do utilizador *Raspberry* são apresentados em duas interfaces, isto acontece devido os *prompts* para cada utilizador, e a explicação está na secção 3.3.

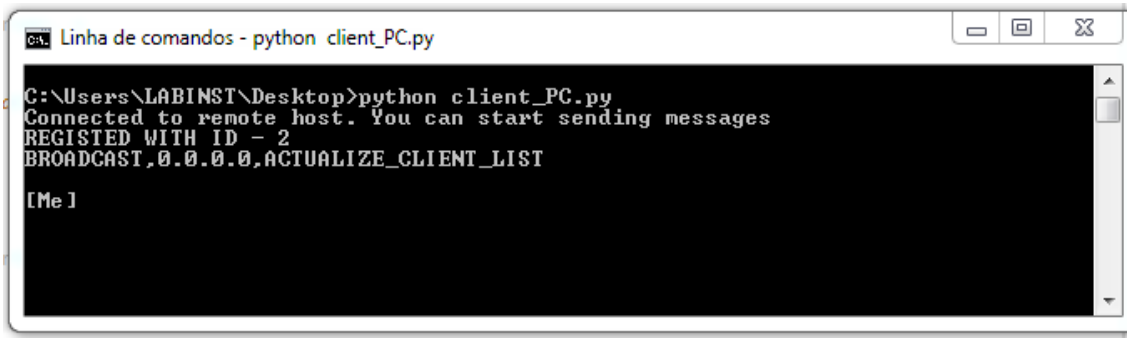

Figura 26: ID do utilizador Desktop atribuído pelo servidor

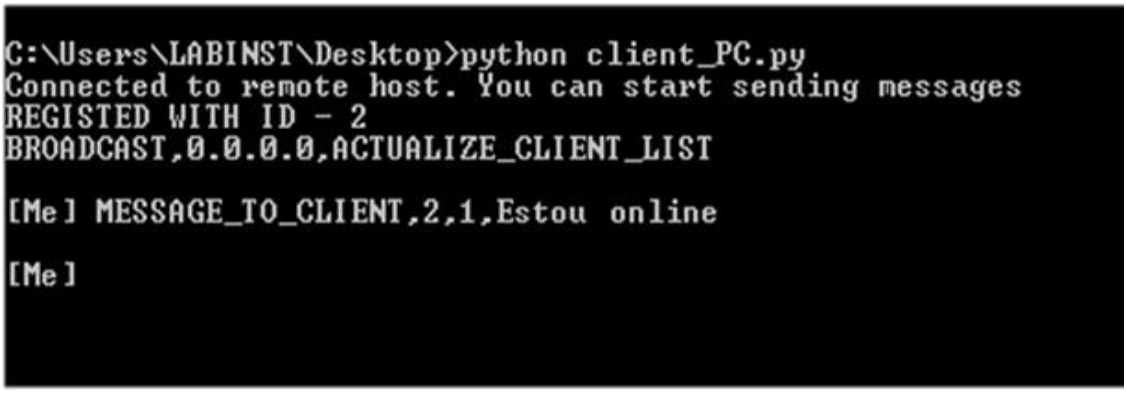

Figura 27: Envio de mensagem entre utilizadores

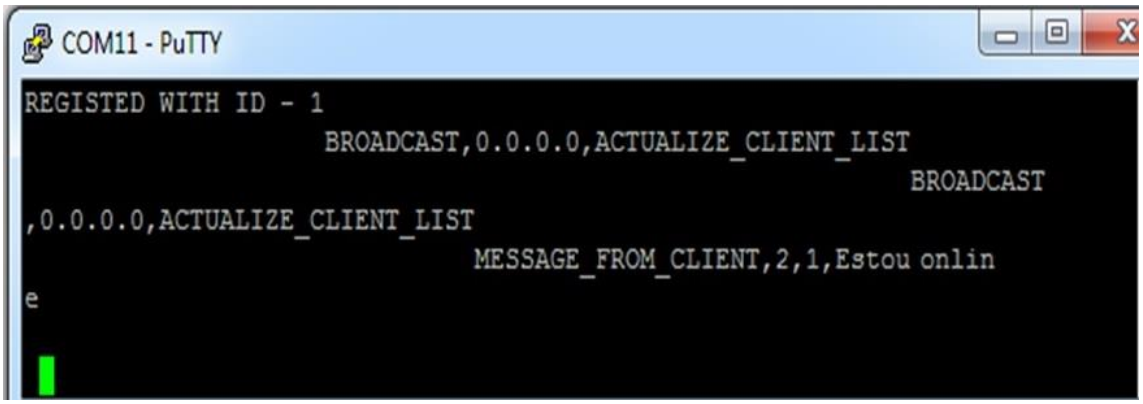

Figura 28: Receção de mensagem entre utilizadores

As figuras 27 e 28 mostram o resultado da comunicação executada, na figura 27 aparece o envio de mensagem do *Desktop* para *Raspberry*, este envio acontece na penúltima linha. "MESSAGE_TO_CLIENT" é um comando que o utilizador envia ao servidor indicando que deseja comunicar-se com um outro utilizador, o "2" é o ID do utilizador que solicita a comunicação, o "1" é o ID do utilizador com que o utilizador 2 deseja se comunicar, e, "estou online" é a mensagem enviada para o utilizador 1. Já a figura 28 mostra a receção da mensagem pelo utilizador 1, "MESSAGE_FROM_CLIENT" é o comando enviado pelo servidor ao utilizador destinatário informando a origem da mensagem que lhe foi enviada.

A pesar das figuras mostrarem apenas a mensagem do *Desktop* para o *Raspberry Pi*, isso não quer dizer que as mensagens são unidirecionais, pois elas podem ocorrer também na ordem inversa, o processo é o mesmo, a única diferença será a ordem dos ID, o mesmo acontece com a mensagem entre utilizadores e servidores, modificando apenas o comando, ao invés de ser enviado o comando "MESSAGE_TO_CLIENT", envia-se o comando "MESSAGE_TO_SERVER".

Para alguém com habilidades em trabalhar com a linha do comando pode ter a ideia de que se possa criar os scripts a partir de comandos do *prompt*, mas foi mais do que isso, por isso de seguida será apresentado o código que permitiu a realização da comunicação entre os utilizadores da rede criada.

O objetivo de mostrar o *script* das figuras 29 e 30, não é explicar linha por linha de comando utilizado, mas sim mostrar de uma forma geral o tipo de programação usada para permitir a comunicação entre utilizadores da rede, ou seja, Cliente/Cliente e Cliente/Servidor e vice-versa. Utilizando o Python incorporado no eclipse fez-se a programação de *Socket* que segundo [27], [30] e [31] é uma ligação entre nodos, ou seja, é uma das maneiras de ligar dois nodos em uma rede, para permitir a comunicação entre eles. Quando um *socket* (o mesmo que dizer um nó) estiver escutando em uma porta privada (IP), o outro *socket* localiza um diferente para formar a ligação. É da responsabilidade do servidor formar o *socket* ouvinte, à medida que o cliente encontra o servidor, pode-se assim dizer que existe um servidor e um cliente. A programação do *socket* é iniciada com a importação da sua biblioteca, criando um *socket* simples onde são instanciados dois parâmetros, o primeiro é o AF_INET referindo-se ao endereço ipv4, e o SOCK_STREAM ao TCP orientado para ligação.

```python
import sys
import socket
import select

HOST = ''
SOCKET_LIST = []
RECV_BUFFER = 4096
PORT = 9009

def chat_server():

    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    server_socket.bind((HOST, PORT))
    server_socket.listen(10)

    # add server socket object to the list of readable connections
    SOCKET_LIST.append(server_socket)

    print "Chat server started on port " + str(PORT)

    while 1:

        # get the list sockets which are ready to be read through select
        # 4th arg, time_out  = 0 : poll and never block
        ready_to_read,ready_to_write,in_error = select.select(SOCKET_LIST,[],[],0)

        for sock in ready_to_read:
            # a new connection request recieved
            if sock == server_socket:
                sockfd, addr = server_socket.accept()
                SOCKET_LIST.append(sockfd)
                print "Client (%s, %s) connected" % addr

                broadcast(server_socket, sockfd, "[%s:%s] entered our chatting room\n" % addr)

            # a message from a client, not a new connection
            else:
                # process data recieved from client,
                try:
                    # receiving data from the socket.
                    data = sock.recv(RECV_BUFFER)
                    if data:
                        # there is something in the socket
                        broadcast(server_socket, sock, "\r" + '[' + str(sock.getpeername()) + '] ' + data)
                    else:
                        # remove the socket that's broken
                        if sock in SOCKET_LIST:
                            SOCKET_LIST.remove(sock)

                        # at this stage, no data means probably the connection has been broken
                        broadcast(server_socket, sock, "Client (%s, %s) is offline\n" % addr)

                # exception

                except:
                    broadcast(server_socket, sock, "Client (%s, %s) is offline\n" % addr)
                    continue

    server_socket.close()

# broadcast chat messages to all connected clients
def broadcast (server_socket, sock, message):
    for socket in SOCKET_LIST:
        # send the message only to peer
        if socket != server_socket and socket != sock :
            try :
                socket.send(message)
            except :
                # broken socket connection
                socket.close()
                # broken socket, remove it
                if socket in SOCKET_LIST:
                    SOCKET_LIST.remove(socket)

if __name__ == "__main__":

    sys.exit(chat_server())
```

Figura 29: Código de programação para o servidor

```python
# chat_client.py

import sys
import socket
import select

def chat_client():
    host = "192.168.1.100"
    port = 9009

    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.settimeout(2)

    # connect to remote host
    try :
        s.connect((host, port))
    except :
        print 'Unable to connect'
        sys.exit()

    print 'Connected to remote host. You can start sending messages'

    while 1:
        #socket_list = [sys.stdin, s]
        socket_list = [s]

        # Get the list sockets which are readable
        ready_to_read,ready_to_write,in_error = select.select(socket_list , [], [])

        for sock in ready_to_read + [sys.stdin]:
            if sock == s:
                # incoming message from remote server, s
                data = sock.recv(4096)
                if not data :
                    print '\nDisconnected from chat server'
                    sys.exit()
                else :
                    #print data
                    sys.stdout.write(data)
                    sys.stdout.write('\n')
                    sys.stdout.write('[Me] '); sys.stdout.flush()

            else :
                # user entered a message
                msg = sys.stdin.readline()
                s.send(msg)
                sys.stdout.write('\n')
                sys.stdout.write('[Me] '); sys.stdout.flush()

            sys.stdout.flush()

if __name__ == "__main__":

    sys.exit(chat_client())
```

Figura 30: Código de programação para o servidor

Como foi referido, os códigos apresentados nas figuras 29 e 30 devem estar em uma máquina ou em várias, para que eles possam ser executados. A forma de execução varia, o código servidor pode ser executado a partir do eclipse como um interpretador, ou a partir do *prompt*, o importante é que o código servidor deve ser executado antes que o código cliente, para que o ID do cliente possa ser atribuído. De seguida, o código cliente deve ser executado no *prompt*, pois é por lá que ocorre a comunicação, de salientar que não fará sentido executar o código cliente a partir do eclipse porque não tem uma interface de comunicação.

4. Capítulo 4 – A NCAP Segundo a Norma IEEE 1451

Esta parte é dedicada especificamente para explicar os serviços e funcionalidades que serão implementados na NCAP e que, de certa forma, facilitará ou permitirá a comunicação entre a NCAP e a TIM, desta forma é conveniente recordar a estrutura da NCAP. Como se especificou no capítulo anterior, a NCAP permite a comunicação entre o utilizador da rede com a TIM, e esta comunicação não ocorre de forma automática, mas sim por meio de serviços. Serviços estes que ocorrem em camadas específicas, como mostrado na figura 31, as camadas que serão descritas estão especificadas com as letras A, B e C respetivamente e a seta D.

A camada com a letra A, é a camada que permite à NCAP enviar informações para a rede de utilizador, ou seja, toda informação que a NCAP colhe da TIM, esta é enviada para a rede de utilizador para que a mesma seja analisada e utilizada.

A camada com a letra B, representa a própria NCAP, ou seja, é neste espaço onde são implementados todos os serviços necessários que permitirá a interação com as outras camadas.

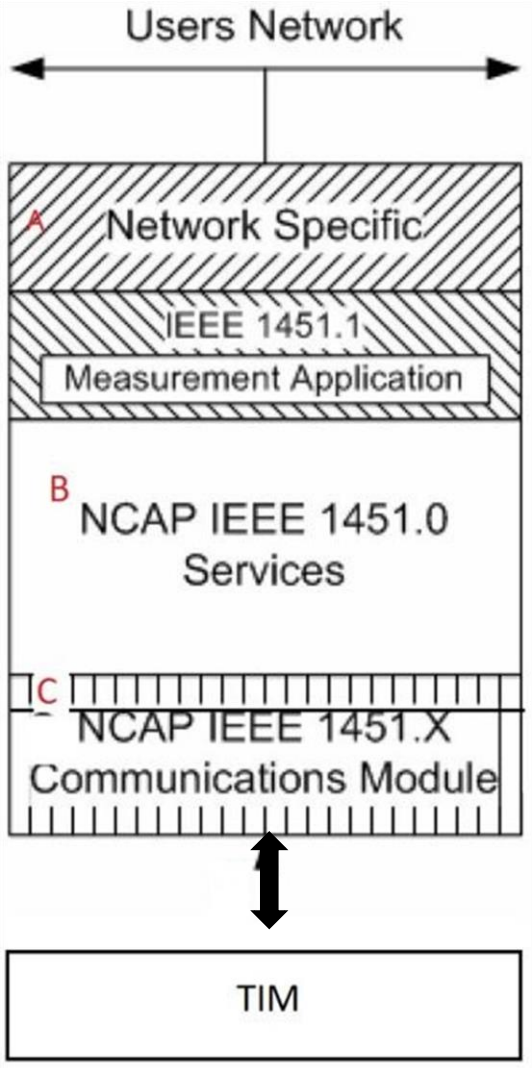

Figura 31: Camadas da NCAP

A camada C, é a camada que permite à NCAP comunicar-se com a TIM, conhecido também como módulo interface de comunicação, é conhecido como a camada que serve de intermediária entre os serviços NCAP IEEE 1451.0 e qualquer outro serviço definido pela norma, representado como NCAP IEEE 1451.X. Representa uma interface física, ou não, que fica entre a NCAP e a TIM, ou por outras palavras pode-se dizer que é o meio pela qual se efetua a ligação delas.

As funcionalidades implementadas na parte B da figura 31 deve permitir que a NCAP possa ter contacto com as TEDS que se destinam em armazenar as informações dos sensores, como também deve permitir trabalhar em *plug-and-play*, ou seja, a interação entre a NCAP e a TIM não pode ser interrompida em caso de se adicionar ou se realizar mudança de TIM, deve realizar-se a atualização automática a cada nova conexão de uma nova TIM. Como encontrado em [32],[33] *plug and play* é a capacidade que um dispositivo tem de se conectar ou desconetar com o outro de modo automático. A NCAP deve estar projetada no sentido de não ser particularizada, deve ser compatível com qualquer dispositivo da família da norma, ou conjunto de dispositivos.

Como mostra o parágrafo anterior, para que se possa realizar a comunicação entre a NCAP e a TIM e se obter dados corretos, os serviços ou funcionalidades implementadas devem incluir o endereçamento, que vai permitir efetuar a distribuição de dados de uma forma correta.

4.3. Endereçamento

O endereçamento é um serviço utilizado pela NCAP definido pela norma IEEE 1451.0, é composto por 16 bits, para indicar o destino de um determinado sinal que ele poderá enviar, o resultado eficaz de um sinal enviado, depende muito do endereçamento feito, quando o endereçamento é bem feito, ou seja, quando é selecionado o destino correto do sinal, o retorno virá da fonte certa e, se não houver impedimento, o resultado será o que se espera. Este endereçamento ocorre em dois níveis ou grupos, o primeiro nível é para definir a camada física de implementação e é realizada para definir o meio de comunicação, permitindo assim o envio e receção de mensagem entre a NCAP e a TIM ou entre duas TIM. O segundo nível está relacionado com o *Transducer Channel numberI*(TC), *em* português número do canal transdutor, uma vez que, se a NCAP tem a capacidade de se conectar a diversos TIM, esta camada serve para especificar dentro da TIM que tipo de mensagem será manipulada bem como avisar a NCAP da origem de uma determinada mensagem. Este segundo nível deve cumprir com várias outras regras de endereçamentos, essas regras definem um nome específico, e ocupam um espaço específico num octeto, que permite que ele seja utilizado em todo o padrão sem conflitos. Todo o endereçamento realizado ocorre com base no espaço existente ou disponível, o endereçamento pode ser realizado para a TIM ou para um determinado canal do transdutor dependendo do bit ocupado no octeto, cada bit de um determinado octeto foi reservado para uma tarefa ou função. Desta forma, para cada valor atribuído a um bit, é indicada uma informação existente. Como todos os serviços são realizados através da norma, qualquer dispositivo que seja desenvolvido levando em conta as regras ou requisitos fornecidos pela

norma, poderá de forma aceitável comunicar com os demais dispositivos conectados à mesma rede. Deste modo possibilita uma correta interpretação e reconhecimento na rede. Esta organização permite efetuar um endereçamento de modos a ocupar os 2^{16} bits existentes, dividido em quatro (4) grupos: TIM, TC, grupos de TCs e global, como será mostrado na tabela 3, cuja legenda será mostrada de seguida. Cada letra representará a legenda de um determinado grupo de modo sequencial, tal como aparece na tabela.

A: Representa o endereçamento para uma TIM específica.

B: Representa o endereçamento de um grupo específico de TCs da TIM distingue-se dos demais por ter no seu bit mais significado o valor zero (0)

C: Representa o endereçamento de um grupo de TCs existente na TIM, a sua caraterística principal é a existência do valor 1 no bit mais significativo.

D: Representa o endereçamento a todos os TCs existente na TIM.

Existem casos particulares na divisão do endereçamento, apesar de parecerem apresentar uma contradição nas caraterísticas apresentadas em cada grupo, assim, para acabar com esta confusão o grupo "Grupo TCs" é separado em dois como mostra a tabela 4.

De modo similar, as letras D e E, têm ambas a mesma representação que a letra C da tabela 3, a diferença é que, nesta tabela, a caraterística é diferente. No caso da letra D precisa de mais que 2 bits, onde os dois bits mais significativos possuem o valore zero (0) e um (1) respetivamente, indicando que se trata de um endereçamento de grupos, os bits restantes possuem o valore zero (0) indicando o(s) grupo(s). Já a letra 'E' possui nos 2 bits mais significativos o valor um (1), os restantes bits em representação binária apontam apenas para um grupo da TIM.

Tabela 4: Classe de Endereço

Classe de Endereço	Valor	Legenda
TIM	0	A
TC	$1 \leq E \leq 32767$	B
Endereço de Grupo TCs (AddressGroup)	$32768 < E \leq 65534$	C
Global	65535	D

Tabela 3: Endereço de Grupo TCs

Classe de Endereço	Valor	Legenda
Bit Mapeado	$32768 < E \leq 49151$	D
Binário	$49152 \leq 65534$	E

4.4. Estados e modos de operação

Esta subsecção não explicará detalhadamente o funcionamento da TIM, mas, explica os diferentes estados e modos de operação da TIM e TCs de uma forma sintetizada, segundo [34], [35] estes estados são conhecidos como amostra (*sampling)* e o modo de transmissão (*transmission mode*). Existe uma diferença entre os estados da TIM de forma geral e os estados dos TCs. Para os TCs depois do estado de inicialização ser completo existem dois estados, Transdutor Inativo (*Transducer Idle*) e o Transdutor Operando *(Transducer Operating)* como mostrado na Fig 32. Já a TIM não possui estados particulares no momento que completar a inicialização, possui 3 estados diretos: TIM Inicialização (Inicialization), TIM ativo (ative), e o estado da TIM desativado (*Sleep mode*), como se mostra na figura 33.

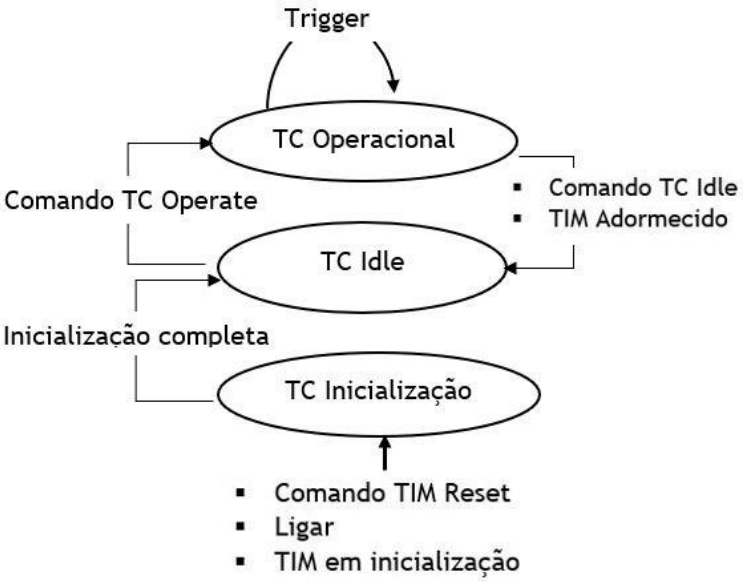

Figura 32: Estados e modos de operação do TCs

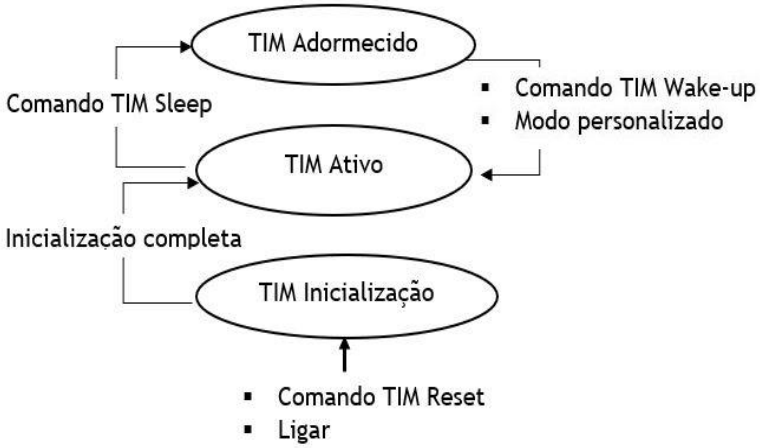

Figura 33: Estados e modos de operação da TIM

No caso da TIM, ao completar a inicialização passa automaticamente de um estado para outro, da TIM inicialização para TIM Ativa, onde estão disponíveis todas funções, até que decide enviar o comando TIM *Sleep*, para adormecer a TIM, resultando desta forma em uma economia de energia, por possuir todas as funcionalidades no modo desativadas ou inoperante. Neste estado, o único comando reconhecido pela TIM é o comando *wake-up*, que faz a TIM passar do modo adormecido para o modo Ativo, a menos que a TIM detete a ocorrência de um *reset* ou que acabou de ser ligado. O mesmo comportamento é verificado pelo TC, após finalizar a inicialização entra logo no estado inativo, sem funções no estado de operação, até receber o comando *TC Operate* que ativa o estado TC operacional, voltando ao estado anterior inativo de duas formas, a TIM em que este TC pertence volta ao estado adormecido, ou se o TC receber o comando *TC Idle* segundo a referência [34].

Segundo a referência [23], só quando o TC se encontra no estado operacional, é que se descreve o comportamento do TC em relação a amostra. O modo como o TC adquire a amostra e a transmissão de dados, assim como a forma em que este dado é enviado à NCAP é definido pelo modo de operação. Esta operação depende em parte da presença ou ausência do *trigger*. Segundo [36] existe uma relação diferente entre o *trigger* e a amostragem, por esta razão um sensor ou atuador pode operar em qualquer dos 5 modos de amostragem que a norma fornece, sendo *trigger-initiated* e *free-running sampling* os modos independentes ou exclusivos, ao passo que, os três modos restantes dependem dos dois já mencionados.

Trigger-initiated: Segundo [34] permite o sensor adquirir dados logo que deteta a receção do comando de início, o atuador faz o uso do mesmo comando para dar início a suas funcionalidades.

Free running without pre-tigger: O [37] diz que este modo de amostragem permite um sensor de forma automática e contínua obter, converter e armazenar parâmetros físicos, durante estes três processos o sensor não efetua a leitura até que o processo seja completo, ao receber um comando trigger, envia os dados obtido, e recomeça o processo para aquisição de novos dados.

Free running with pre-tigger: de modo similar ao anterior, este também de forma automática e contínua converte e armazena as leituras feitas até o limite ou até receber um trigger. Este mesmo modo especifica duas formas diferentes de amostragem, com buffer ativado (*with buffers enabled*) ou sem buffer ativado (*without buffers enabled*). No primeiro caso as informações são continuamente armazenadas no buffer do sensor que possui uma memória disponível, até não existir mais *buffer* vazio, os dados lidos neste período são descartados até que sejam disponibilizados mais buffers. Já no segundo caso vai seguindo o mesmo procedimento, so que neste, os dados lidos são descartados até o TC receber um trigger, para recomeçar.

Continuous sampling mode: começa a realizar o seu serviço de recolha e armazenamento logo que recebe um evento. É semelhante com *o free-running without pre-trigger",* o que os difere um do outro, é que, este modo não para o serviço mesmo que possuir os *buffers* cheios, quando

assim for o *buffer* mais antigo é descartado mesmo que este não tenha sido lido para continuar com o processo.

Immediate operation: neste modo o TC transmite imediatamente à NCAP os dados recolhidos dos sensores quando recebe um comando de leitura. Sendo o comando recebido bem com os dados são utilizados pelo atuador.

4.5. Mensagens e seus comandos

Uma mensagem, é todo e qualquer sinal enviado da NCAP para TIM ou o inverso, com direito ou não de uma resposta, como por exemplo executar uma tarefa que pode ser: ler um dado na TIM; escrever, atualizar ou um pedido de inicialização da TIM, através de um meio comunicação definido pela norma. Por se tratar de uma definição, não apresenta uma idea única, quer com isto dizer que podem ser encontrada muitas outras definições de Mensagem entre a NCAP e a TIM, todas elas embora com expressões diferente, estarão centradas em uma única ideia, como encontrado em [38] uma mensagem, é um sinal ou conjunto de sinais, fornecido pelo mecanismo criado pelo módulo de comunicação de interface definido pela norma, permite a troca de informação ou envio de uma instrução entre a NCAP e a TIM. Da mesma forma que é fornecida uma definição para mensagem pela norma, também é fornecida uma estrutura para as mensagens. A estrutura da mensagem, é a forma como as mensagens são construídas ou organizada de forma a chegar ao destino certo por meio de interface do módulo de comunicação. Existe uma ordem de transmissão, de cabeçalho e dados, esta ordem não depende dos valores existente nos bits, ela segue uma transmissão sequencial na ordem da numeração dos octetos, ou seja, os bits mais e menos significativos não ditam a ordem de envio de dados, a tabela 5 apresenta a organização de um conjunto de octetos.

A TIM e os TCs são monitorizado e controlados atravéz da NCAP, estas ações são possíveis por meio de uma estrutura de mensagens, e por meio destas acções, a NCAP pode enviar varios sinais estruturados na TIM e TEDs, ou, definido estados a ser operado da TIM e TCs, assim como a TIM pode iniciar uma mensagem para a NCAP, quer seja uma mensagem uni/bidirecional ela deve possuir sempre uma estrutura. A referência [38], mostra que existe uma estrutura de mensagem para NCAP, *Comand Message* (mensagem de comando), uma estrutura de mensagem para a TIM, *Message Reply* (resposta de mensagem) e segundo [39] existe uma estrutura para iniciar a mensagem na TIM, *TIM initiated message* (mensagem de início da TIM).

Tabela 5: Organização e a ordem de envio dos octetos

octeto								**octeto**								**octeto**								**octeto**							
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
1								2								3								4							
5								6								7								…							

4.5.1. Estrutura da mensagem de comando

Comandos estruturados através do qual a NCAP pode monitorizar e controlar os TIMs e o TCs, permitindo de modo igual a realização de algumas funções como leitura/escrita de dados que o sistema utiliza e, leitura /escrita de TEDS. Esta estrutura é composta por 5 campos, onde cada um dos campos ocupa um número específico de octetos como mostra a tabela 6, as legendas dos campos são mostradas na tabela 7.

Na tabela 6, mostra-se a distribuição dos octetos por cada campo, e, verifica-se que dois dos campos apresentados como o nº do TC destino e o tamanho possuem ou ocupam dois octetos, para salientar que os dois octetos não são ocupados de forma aleatória, um dos octetos é o mais significativo e o outro octeto é o menos significativo. Como mencionado no 4.3 o destino da mensagem depende do valor existente no bit mais significativo do conjunto de 16 bits, visto que o Nº TC de destino ocupa dois octetos, se o valor for um (1) no bit 16, a mensagem é enviada para um TC específico, se o valor for zero (0) no bit 16, a mensagem é enviada para TIM de modo geral [40]. Já o tamanho está relacionado com os octetos dependente do comando.

A subsecção 4.2, fez referência ao tipo de mensagem existente entre a NCAP e a TIM, podendo ser uni ou bidirecional dependendo do comando enviado, no caso o comando ter a característica de uma mensagem bidirecional, a TIM tem a obrigação de responder a mensagem recebida com ou sem sucesso, desde que todos os requisitos estejam em conformidade.

Tabela 6: Quantidade de octeto por campo da estrutura de mensagem de comando

Nome do campo	Nº TC de destino	Classe do comando	Função do comando	Tamanho	Octetos dependentes do comando
Nº de octetos	2	1	1	2	N

Tabela 7: Campos e legenda de mensagem de comando

Campo	legenda
Nº TC de destino	Campo com 16 bits e que e que indica o destino da mensagem
Classe do comando	campo com 8 bits e que define a categoria do comando
Função do comando	Determina a função do comando
Tamanho	Define o numero de octeto que formam o campo
Octetos dependente do comando	É constituido pela informação especifica de cada comando, depende da clase e da função do comando

4.5.2. Estrutura da mensagem de resposta

Toda e qualquer mensagem que a TIM envia à NCAP como resposta a uma mensagem de comando, é chamado de mensagem de resposta. Assim como a mensagem de comando, a mensagem de resposta também tem uma estrutura própria. Ao contrário da mensagem de comando, esta é formada por três campos, como se vê na tabela 8 com a quantidade de bytes respetivo por campo, já a tabela 9 apresenta as legendas destes campos.

Tabela 8: Quantidade de octeto por campo da estrutura de mensagem de resposta

Nome do Campo	Indicador de sucesso /falha	Tamanho	Octetos dependentes da resposta
Nº de octetos	1	2	N

Tabela 9: Campos e sua legenda de mensagem de resposta

Campo	Definição
Indicador de sucesso ou falha	Este campo informa o sucesso ou falha da mensagem com base no valor do deu retorno, se o valor for 1 ou diferente de 0, indica o sucesso da mensagem, se o valor for zero (0) indica a falha do comando, levando o sistema a verificar a causa da falha do comando.
tamanho	Este campo indica o número de octeto dependente da mensagem de resposta, indicando o bit de erro ao descartar a mensagem se o comprimento da mensagem não coincidir com o tamanho da mensagem recebida.
Octeto dependente da resposta	É o campo que depende da resposta por cada mensagem de comando.

4.5.3. Estrutura das mensagens iniciadas pela TIM

As tabelas 6-9 apresentadas até ao momento, apenas mostram a estrutura das mensagens iniciadas pela NCAP, mesmo as respostas dadas pela TIM, é resultado do sinal recebido da NCAP. A TIM também pode começar uma mensagem, esta mensagem só ocorre assim que se liga uma TIM à NCAP, permitindo desta forma à TIM informar as suas caraterísticas à NCAP, para que NCAP a reconheça em modo *plug-and-play*, as restantes mensagens são realizadas através dos comandos, *command* e *Command Reply*. Desta feita é necessário que se tenha igualmente uma estrutura para esse tipo de mensagem (iniciada pela TIM). A estrutura desta mensagem é semelhante à estrutura da mensagem inicializada pela NCAP (mensagem de comando), a única diferença está no primeiro campo como apresentado na Tabela 10.

Tabela 10: Campos e sua legenda de mensagem iniciada pela TIM

Campo	legenda
Número do Canal do Transdutor de Origem	Ao contrário do Nº de TCs, neste campo é fornecido o Nº TC não do destino, mais da origem.
Classe do comando	A legenda para estes campos é semelhante à legenda da tabela 7, no campo com o mesmo nome.
Função do comando	
Tamanho	
Octetos dependente do comando	

A distribuição dos octetos ocupados pelos campos também é semelhante ao processo inverso, por esta rezão não se apresenta uma tabela com a distribuição de octetos para os campos. A mensagem trocada entre a NCAP e a TIM depende de uma ordem, para ditar o início, o fim e a sequência do envio das mensagens, essa ordem é dada por meio de comandos.

4.6. Comandos

Nesta subsecção, serão tratados assuntos relacionados com os comandos definidos pela norma IEEE 1451.0, esta encontra-se dividida em duas partes: Campo Classe e Campo Função agrupados em 2 bytes, sendo o mais significativo ocupado pelo campo classe e o menos significativo ocupado pelo campo função, são normalmente enviados da NCAP para a TIM, são

Tabela 11: Classe dos comandos

Id	classe de comando	Categoria
0	*Reserved*	Reserva do identificador
1	*CommonCmd*	Comando comum entre TIM/TC
2	*XdcrIdle*	Comando específico para TC em estado de operação inativo
3	*XdcrOperate*	Id_2, no estado operação “operacional”
4	*XdcrEither*	Id_2 e Id_3
5	*TIMsleep*	Comando para TIM no estado de operação adormecido
6	*TIMActive*	Id5 no estado operacional ativo
7	*AnyState*	Id_5 e Id_6
8-127	*ReservedClass*	Reserva do identificador

de grande importância porque permitem que se faça a leitura e escrita de TEDS, permitem a leitura e escrita de dados que são utilizados pelo sistema, permitem ainda configurar e controlar os módulos transdutores. A tabela 11 mostra o conjunto de comandos existentes na norma.

Tabela 11, levando em conta o seu destino e os estados de operação das TIM e dos TC em que eles são enviados, um comando válido para a TIM pode não ser válido para o TC, por esta razão para cada um existe um comando específico em função do que se pretende. O único comando aceitável quer pela TIM como pelo TC, é o comando *CommonCmd,* por este motivo é chamado de comando que define os comandos comum entre a TIM e a TC, fazem parte deste grupo os comandos apresentados na Tabela 12, os restantes são comandos específicos. A diversidade de comandos singulares da TIM e do TC é vasta e predefinidos pela norma, o que quer com isso dizer que são imutáveis ao ponto de vista do programador. Não serão apresentados neste documento, mas podem ser encontrados em [35] e [36] todos os comandos possíveis. Ainda sobre os comandos comuns, como foi dito na secção 4.1, é o endereçamento que vai determinar se o comando será executado por uma TIM ou por um TC. O campo *cmdFuntionId* contém o ID do comando comum, e, indica o comando comum a ser executado, para esta fase do projeto serão necessários os campos 1-4, em função das interfaces e métodos implementados. O campo TCs/TIM indica o endereçamento do comando, ou seja, para onde o comando pode ser enviado, por se tratar de comandos comuns, podem ser enviados tanto para os TC como para as TIM. Estes comandos são aceites apenas se os TC ou os TIM estiverem em estado ativo. Como se vê no campo estado, dos comandos selecionados, apenas o *Write* TEDS *segment* não requer uma resposta.

Tabela 12: Comandos comuns

Cmd_Função_Id	comandos	TC/TIM	Estado	Resposta esperada
0	Reserved	Any	active	
1	Query TEDS	Any	Active	Yes
2	Read TEDS segment	Any	Active	No
3	Write TEDS segment	Any	active	Yes
4	Update TEDS segment	Any	active	No
5	Run self-test	Idle	active	No
6	Write servisse request mask	Any	active	Yes
7	Read servisse request mask	Any	active	Yes
8	Read status-event register	Any	active	Yes
9	Read status-condition register	Any	active	No
10	Clear status-event register	Idle	active	No
11	Write status-event protocol state	Any	active	Yes
12	Read status-event protocol state	Any	active	Yes
13-127	Reserved		active	
128-255	Open for manufacturers		active	

Tabela 13: Código de acesso as TEDS (*TEDS access code)*

TEDS Código_acesso	Nome - TEDS	TEDS	requerido opcional
0		Reservad	
1	MetaTEDS	Meta-TEDS[1]	Requerido
2	MetaIdTEDS	Meta-identification TEDS[2]	Opcional
3	ChanTEDS	TransducerChannel TEDS[1]	Requerido
4	ChanIdTEDS	TransducerChannel Identification TEDS[2]	Opcional
5	CalTEDS	Calibration TEDS[1]	Opcional
6	CalIdTEDS	Calibration identification TEDS[2]	Opcional
7	EUASTEDS	End users' application-specific TEDS[3]	Requerido
8	FreqRespTEDS	Frequency response TEDS[1]	Opcional
9	TransferTEDS	Transfer function TEDS[1]	Opcional
10	CommandTEDS	Commands TEDS[2]	Opcional
11	TitleTEDS	Location and title TEDS[2]	Opcional
12	XdcrName	User's transducer name TEDS[3]	Requerido
13	PHYTEDS	PHY TEDS[1]	Requerido
14	GeoLocTEDS	Geographic location TEDS[2]	Opcional
15	UnitsExtention	Units extention TEDS[2]	Opcional
16-127		Reserved	Requerido
128-255		Manufacturer-defined TEDS	Opcional

Tabela 14: Comandos do estado operacional do transdutor (XdcrOperate)

cmd_Função_Id	Comando	Ad.TC	proxy	Grupo	Requerido Opcional
0	Reserved	__			
1	Read TransducerChannel data-set segmen	Sim	Sim	Não	Sim
2	Write TransducerChannel data set segment	Sim	Sim	Não	Não
3	Trigger command	Sim	Sim	Sim	Não
4	Abort Trigger	Sim	Sim	Sim	Não
5-127	Reserved	__			
128-255	Open for manufacturers	__			

Comando *Query* permite à NCAP solicitar informações sobre como ler ou escrever as TEDS, e é necessário especificar o tipo de TEDS, bem como o retorno da informação, esta especificação é enviada em um argumento de 1 byte chamado *TEDSAcessCode*, cada valor enviado nesse argumento indica um determinado tipo de TEDS. A tabela 13 mostra os *TEDSAcessCode* especificado na norma. A norma em [42] apresenta mais tabelas relacionadas com os comandos, e muitas delas ajudam na interpretação e organização do código, como é o caso do tamanho da mensagem.

Assim como existe um grupo de elementos para comandos comuns, também existe um grupo de elementos para os outros comandos. A Tabela 14 apresenta os comandos da classe 3. Essa tabela

é importante porque apresenta o *Funtion_Id* dos comandos e serviços utilizados nos exemplos 4 e 5 que serão mostrados na secção 5, deste grupo foram implementados os comandos com o *cmd*_Função_Id 1 e 2

4.7. *Application Programming Interface* API

Tal como referido em [43], a API é uma sigla para *Application Programming Interface* o que em Português significa Interface de Programação de Aplicativos. É definido como num conjunto de instruções de programação que permite construir e aceder a um aplicativo ou a uma plataforma baseada na web de uma forma a simplificar a tarefa do utilizador. A API permite que dois ou mais aplicativos se comuniquem sem a ação do utilizador, isto é possível devido aos diversos códigos que definem um objeto um determinado comportamento, de forma sintetizada se pode dizer que a API é composta de funções que são acedidas apenas por meio de programação.

Esta não é a única definição de API que se pode encontrar, existem inúmeras definições para API, mesmo assim, todas elas partilham da mesma ideia que a definição apresentada no parágrafo anterior, por exemplo: em [37] e [35] e tantas outras bibliografias já mencionadas neste documento definem a API como um mecanismo que facilita a construção de módulos permitindo ainda que os outros desenvolvedores possam incrementar funcionalidades de acordo com a norma e ainda se integram de forma perfeita, tornando mais fácil a interação a NCAP e a TIM. Segundo a referência [34], a API define uma lista que permite efetuar a leitura e escrita nos TC e nas TEDS, essa lista permite ainda enviar informações de configuração e controle da TIM. São definidas pela norma IEEE 1451.0 duas API: i) Módulo de Comunicação (MC) apresentado na tabela 15 e ii) Serviços do Transdutor (ST) apresentado na tabela 16.

A API ST é exclusivamente implementada na NCAP. A medição e o controle de acesso são feitos pela NCAP uma vez que os aplicativos da norma IEEE 1451 fazem o uso desta API. Os métodos nele contidos permitem efetuar a leitura e escrita ao TC e TEDS bem como as outras operações

Tabela 15: Interfaces da API Módulo de Comunicação

Interfaces	Descrição
Comm	Mecanismo de comando do "ciclo de vida" de um módulo IEEE 1451.X
P2PComm	Suporta as operações de comunicação ponto a ponto
NetComm	Suporta a operação de comunicação em rede
Registration	Registo de módulo IEEE1451.X com a camada IEEE1451.0
P2PRegistration	Registo de TIMs específico com a camada IEEE1451.0
NetRegistration	Registo de TIMs específico e grupo de TIMs com a camada IEEE1451.0
Receive	Sem métodos genéricos. Reservados para os trabalhos futuros
P2PRegistration	Métodos para anunciar a mensagem ponto á ponto recebida a camada IEEE 1451.0
NetReceiv	Método para anunciar mensagens de rede recebida a camada IEEE 1451.0

Tabela 16: Interfaces da API Serviços do Transdutor

Interfaces	Descrição
TimDiscovery	Possui métodos que permitem descobrir módulos de comunicação entre as diversas aplicações disponíveis da norma IEEE 1451, TIMs, TCs.
TransducerAcess	Possui métodos que permitem aceder a transdutores
TransducerManager	Possui métodos para controle de acesso a TIMs
TedsManager	Controla a leitura, a escrita e efetua o gerenciamento de informação existente das TEDS na NCAP
CommManager	Possui métodos de controlo acesso e ao Módulo de comunicação
AppCallback	Possuem métodos para funcionalidades especiais.

já mencionadas no parágrafo acima, bem como as outras operações já mencionadas no parágrafo acima, coma também enviar outros comandos para a TIM.

A A*PI* MC é diferente da anterior, e é executada na NCAP e na TIM, esta é utilizada para permitir a comunicação entre as funções definidas pela norma IEEE 1451.0 coma as outras funções definidas pela norma IEEE 1451.X.

As API são desenvolvidas para permitirem que vários objetivos sejam alcançados, os objetivos não fogem das tarefas que já foram mencionadas nos parágrafos acima. De forma resumida e direta, pode-se dizer que as API têm como objetivo permitir a fácil integração e interação de funcionalidades criadas, fornecendo *interfaces* e tecnologias adaptada de acordo com as necessidades do sistema, para permitir de forma simples a comunicação entre aplicações desenvolvidas, desde que estas cumpram com os requisitos definidos pela norma. Na referência [19], são apresentadas de forma mais formal os objetivos das API, os dois tipos de API definidos pela norma IEEE 1451 encontram-se formados por interface de forma individual como é mostrado nas Tabela 15 e Tabela 16, o que permite cumprir com os objetivos descritos no parágrafo.

Segundo [44] os elementos apresentados na Tabela 16 são o conjunto de cinco interfaces da API Serviços do Transdutor para a NCAP, o que permitem alcançar os objetivos traçados pela norma, mas apenas as quatro primeiras são implementadas pela IEEE 1451.0, estas interfaces são conhecidas como aplicativos de medição. No caso de se necessitar um método específico, é chamada a interface AppCallback pelo padrão IEEE1451.0. De modo similar, a API Módulo de Comunicação também descreve um grupo de interfaces para a comunicação entre a NCAP e a TIM como apresentado na Tabela 15, são importantes para a camada IEEE 1451.X pois estas facilitam a comunicação ponto a ponto ou em rede.

Capítulo 5 - Especificação e Implementação dos Serviços da NCAP IEEE 1451.0

Esta secção será dedicada à implementação da NCAP, que permitirá cumprir com o estabelecido na subsecção 1.2. A implementação para este projeto será realizada em três níveis, o primeiro nível será responsável pela programação do aplicativo que vai solicitar os serviços, ou de outra forma, pode ser dito que é a programação que permitirá à NCAP ligar-se a uma rede de utilizadores, e, por meio do qual poderá receber comandos que depois de modificados são enviados à TIM. A ligação da NCAP à rede de utilizadores foi especificada neste trabalho na secção 3.3. O segundo nível é o responsável pela implementação de serviços na NCAP. De acordo com a norma IEEE 1451.0, os serviços a serem implementados são os que compõem o TRansducerServices API (TSA), que se responsabiliza em executar as interfaces da NCAP. Em detalhe, os serviços são: TimDiscovery, *TransducerAcess*, TransducerManager, TedManager e CommManager como encontrado na tabela 16. De seguida será efetuada a descrição dos serviços de forma individual. Para mais informação sobre os serviços que aqui serão mencionados, ver [42]. Esta secção como um todo é baseado na norma encontrada na última citação feita.

5.1. Interface *TimDiscovery*

É uma interface do TSA, que tem como responsabilidade descobrir as TIM e os TC disponíveis que podem ser associados ao padrão IEEE 1451.0, esta interface possui três funções ou métodos que serão listados de seguida

5.1.1. *reportCommModule* ()

Esta é uma função ou método sem argumento da *TimDiscovery*, que tem como função reportar os módulos de comunicação registada no padrão. Os módulos reportados podem ser únicos, de uma tecnologia específica IEEE 1451.5, podem ser da mesma tecnologia como por exemplo IEEE 1451.5 e IEEE 1451.3.

5.1.2. *reportTims*()

Esta é uma função ou método inteiro de 16 bits pertencente à *TimDiscovery*, é utilizado para reportar todas as TIM conhecidas num módulo de comunicação específico, possui dois parâmetros, um de entrada e um de saída com 8 e 16-bits, respetivamente, com a representação mostrada de seguida.

`UInt16` *reportTims* (parâmetros de entrada)

`in Args: UInt8` este é o parâmetro de entrada que indica o modulo de comunicação em que se deseja verificar as TIM existentes

`out Args: UInt16Array timi_d`: este é o argumento de saída que é uma lista ou conjunto de TIM existentes no modulo de comunicação especificado no parâmetro de entrada de uma NCAP.

5.1.3. *reportChannels* ()

Esta é uma função ou método inteiro de 16 bits da *TimDiscovery*, reporta todos os TC ligados a uma determinada TIM. Tem um parâmetro de entrada de 8-bits e dois de saída onde, um é inteiro de 16-bits e o outro é um *array* do tipo *string* como será mostrado de seguida:

`UInt16` *reportChannels*(parâmetro)

`in Args: UInt16 timId`: este é o argumento de entrada que indica qual é a TIM em que se pretende verificar os seus canais.

`out Args: UInt16Array channelIds:` este é o parâmetro de saída e contém todos os TCs existente na TIM especificado no parâmetro de entrada.

`out Args: StringArray names:` de modo similar ao anterior, este é um parâmetro de saída que contém os nomes dos canais existentes.

5.2. *Interface TransducerAcess*

Esta interface fornece métodos que são utilizados quando a aplicação necessita ter acesso ao TC, permite executar operações de leitura e gravação da TIM. As funções para esta interface são listadas de seguida.

5.2.1. *Open* ()

Tendo os elementos quer seja uma TIM ou uma TC devidamente registado seguindo a norma, esta função dá início à troca de dados. É uma função de tipo inteiro de 16-bits com dois parâmetros de entrada e um de saída sendo todos eles de 16-bits como será mostrada de seguida.

`Args:` UInt16 *open* ()

`in Args:` UInt16 *timId:* este é o parâmetro de saída que indica o ID da TIM quando se quer dar início à troca de dados.

`in Args`: UInt16 *channelId*: este parâmetro, de modo similar ao anterior, é o que indica o canal desejado, uma vez que uma determinada TIM pode conter vários canais.

`out Args`: UInt16 *transCommId*: este é o parâmetro de saída que indica o ID de comunicação de transporte. E este é o comando que será chamado na eventualidade de se desejar recuperar um dado de um canal específico.

5.2.2. *readData* ()

Esta função permite efetuar uma leitura de bloqueio de um TC específico, é uma função do tipo inteiro de 16-bits com três parâmetros de entradas e um de saída.

UInt16 *readData* (parâmetros)

in Args: UInt16 transCommId: é o ID de comunicação de transporte.

in Args: *TimeDuration timeout*: é o tempo limite que se deseja esperar para se obter a resposta

in Args: *UInt8 SamplingMode*: o tipo de amostra desejado.

out Args: ArgumentArray result: é o resultado retornado.

5.2.3. *writeData* ()

Esta função permite efetuar a gravação de bloqueio de um TC específico, é uma função do tipo inteiro de 16 bits, possui 4 parâmetros de entrada com as especificações semelhantes aos parâmetros da função anterior, como mostrado de seguida:

UInt16 *writeData* (parâmetros)

in Args: UInt16 *transCommId*.

in Args: *TimeDuration timeout*.

in Args: UInt8 *SamplingMode*.

in Args: *ArgumentArray value*.

5.2.4. *close* ()

Esta é uma função simétrica ou complementar à função "open", converte o parâmetro "transCommId" como inválido para realizar a comunicação, fechando dessa forma a comunicação do transdutor, possui apensa um parâmetro de entrada que indica o transCommId que será invertido.

UInt16 *close* (parâmetros).

in Args: UInt16 *transCommId*: O ID de comunicação de transporte indica qual o canal a utilizar.

5.2.5. *cancel* ()

Este método realiza o cancelamento da leitura e gravação, de bloquei, o ou um fluxo de medição, possui um argumento que indicará a operação a ser cancelada.

UInt16 *cancel* (parâmetro)

Args: UInt16 operation_id: especifica a operação para cancelar.

5.3. *Interface TransducerManager*

À semelhança das outras interfaces, a Interface *Transducermanager* é chamada pelo aplicativo com o objetivo de fornecer acesso aos recursos mais avançados, é utlizado para não subcarregar

a Interface descrito na secção 5.2, para que mais aplicativos possam executar operações de leitura e gravação dos TC. Esta interface esta formada por várias funções, e, estas passarão a ser mencionadas de seguida.

5.3.1. *lock* ()

Função da Interface 5.3 utilizada para bloqueio de TIM/TC impendido desta forma que outros aplicativos tenham acesso aos recursos. Permite vários bloqueios, mas todos acontecem na ordem sequencial, ou seja, na mesma ordem que os recursos forem acordados, é nessa ordem que se deve dar o bloqueio. É uma função com dois parâmetros de entrada, se o parâmetro *TransComId* se referir a um grupo, todos as TIM/TC devem ser bloqueados de forma sequencial. Em função do tipo de dado de cada elemento, segue a representação abaixo

`Args: UInt16` *lock*(parâmetro)

`in Args: UInt16` *transCommId*: especificação da comunicação do transdutor desejada.

`in Args`: TimeDuration timeout: o tempo de espera para adquirir o bloqueio.

5.3.2. *unlock*()

Esta função desempenha o mesmo papel que a função *lock*, mas no sentido contrário, ou seja, esta permite desbloquear as TIM/TC, e deve ser chamada na mesma proporção em que foi chamada a função *lock*. Possui somente um parâmetro, com o mesmo detalhe do primeiro parâmetro especificado na função *lock*, como se mostra de seguida.

`Args: UInt16` unlock(parâmetro)

`in Args: UInt16` transCommId

5.3.3. *reportLock*()

Função utilizada para reportar todos os *TransComId*, é uma função simples sem parâmetros de entrada, mas com um parâmetro de saída representando um conjunto ou uma lista de *TransComId* retornadas.

`out Args`: UInt16Array *transCommIds:*

5.3.4. *brackLock* ()

Esta função permite quebrar um bloqueio se um fluxo de gravação, de leitura ou de medição sem bloqueio estiver em andamento. À semelhança do anterior, possui de igual modo um parâmetro de entrada, com as mesmas especificações do descrito no ponto anterior. Como mostra a representação abaixo.

`Args: UInt16Array` *transCommIds:*

5.3.5. *sendComand*()

Esta é uma função que é chamada sempre que se deseja enviar um comando em qualquer altura que se deseja comandos novos. Possui 5 parâmetros de entrada e um de saída. Os tipos de

dados utilizados nos parâmetros, devem ser corretos, por esta razão serão listados os parâmetros de seguida com os seus respetivos tipos de dados.

`Args: UInt16` *sendCommand*(paramêtro)

`in Args: UInt16` transCommId: especificação da comunicação do transdutor desejada

`in Args:` *TimeDuration timeout*: indica o tempo de espera antes de um erro de tempo limite

`in Args: UInt8` cmdClassId: o código de classe de comando desejado.

`in Args: UInt8` cmdFunctionId: o código de função de comando desejado.

`in Args:` ArgumentArray inArgs: indica argumentos de entrada na forma array

`out Args:` ArgumentArray outArgs: indica o retorno em forma de array.

5.3.6. *startCommand*()

Esta função marca o início de uma operação sem bloqueio. Possui 7 parâmetros de entrada dos quais 5 são semelhantes aos descritos na função 5.3.5. Possui também parâmetros de saída que não existiam na função anterior, por este motivo apenas serão apresentados os que não faziam parte da função anterior.

`Args: UInt16` *startCommand*(parâmetro)

`in Args:` *triggerTime*: é a especificação de quando comecerá a operação.

`In args` *AppCallback callback*: especifica a interface de retorno.

`out Args:` UInt16 operationId: apresenta um conjunto de ID retornado da operação iniciada.

5.3.7. *configureAttributes*()

Esta função, como o próprio nome indica, é utilizada para configurar um *transCommId* para operação de leitura ou fluxo de medição. Possui dois parâmetros de entrada como especificado de seguida.

`Args: UInt16` *configureAttributes*(parâmetro)

`in Args: UInt16` *transCommId:* especificação da comunicação do transdutor desejada.

`in Args:` StringArray attributeNames: especifica os nomes dos atributos desejados.

5.3.8. *Trigger*()

Esta função permite executar um disparo num determinado *transCommId*. Possui 4 parâmetros de entrada.

`In Args: UInt16` *trigger* (parâmetro)

`in Args: UInt16` transCommId: especificação da comunicação do transdutor desejada.

`in Args`: *triggerTime*: é a especificação de quando começará a operação de leitura.

`in Args`: *timeout*: especificação do tempo necessário de espera para realizar a leitura.

`in Args`: UInt16 SamplingMode: especifica o mecanismo de acionamento.

5.3.9. *Clear ()*

Permite executar a limpeza num *transCommId* específico. Possui três parâmetros de entrada como será representado de seguida.

`Args: UInt16` *clear*(parâmetro).

`in Args: UInt16` *transCommId*: especificação da comunicação do transdutor desejada.

`in Args`: *TimeDuration timeout*: tempo limite de espera antes de um erro.

`in Args`: UInt8 *clearMode*: especifica o modo de limpeza, este modos são representados por números como encontrado na tabela 87 da [42].

5.4. Interface *TedsManager*

Assim como todos os serviços, Interface *TedsManager* é também uma *interface* do TSA predefinida pela norma, faz parte da camada IEEE 1451.0, quando chamada pelo aplicativo, permite o acesso às TEDS. É composto por diversos métodos, como os que serão mencionados abaixo.

5.4.1. *readTEDS*()

Esta função permite efetuar a leitura de uma determina TEDS caso esta esteja disponível na *cache*, do contrário efetua a leitura da TIM. Possui três parâmetros de entrada e um de saída, como se mostra a abaixo.

`Args: UInt16` *readTeds*(parâmetro)

`in Args: UInt16` transCommId.

`in Args`: *timeout*.

`in Args: UInt8` *tedsType*: especifica o que TEDS deve retornar.

`out Args`: ArgumentArray teds: conjunto de informações que contém no TEDS.

5.4.2. *writeTEDS* ()

Esta função permite escrever o bloco TEDS desejado para a TIM, caso a gravação seja realizada com sucesso é atualizada a *cache* TEDS, proporciona uma informação codificada num *array* que é convertido internamente na forma correta e transferida para a TIM num *octetoArray*, possui mais um parâmetro de entrada em relação à função anterior, o que totaliza 4, como será mostrado de seguida.

Args: UInt16 *writeTeds* (parâmetro).

in Args: UInt16 *transCommId*.

in Args: *timeout:* especifica a duração a aguardar antes de retornar um erro de tempo limite se nenhuma resposta for recebida.

in Args: UInt8 tedsType: especifica o que TEDS deve retornar.

in Args: ArgumentArray teds: ArgumentArray contém as informações do TEDS.

5.4.3. *writeRawTEDS()*

Desempenha o mesmo papel do *writeTEDS* e possui os mesmos parâmetros, só que esta função ignora o cache TEDS.

5.4.4. *updateTedsCache* ()

Esta função permite atualizar a cache da TEDS. Possui os mesmos parâmetros que a função *readTEDS* e as mesmas especificações.

5.5. *Inteface CommManeger*

É uma interface do TSA chamado pelo aplicativo para fornecer um mecanismo comum para gerenciar as comunicações disponíveis na NCAP. Possui apenas um método que lhe permite cumprir com o objetivo estabelecido pela norma.

5.5.1. *getCommaManager()*

Args: UInt16 *getCommModule* (parâmetro).

in Args: UInt8 *moduleId*.

out *ModuleCommunication*: comm commObject.

out Args: UInt8 type.

out Args: UInt8 technologyId).

5.6. Interface *AppCallback*

Esta é uma interface com caraterísticas particulares, porque a norma não define nenhuma função para ela. O seu desenvolvimento é da responsabilidade do programador e do fabricante.

Os serviços apresentados até agora, são todos os serviços existentes no padrão, nesta fase não foram implementados todos os serviços. Foram implementadas três interfaces, e, para cada interface foram implementados três métodos. Da interface *TIM Discovery*, foram implementados todos os métodos encontrados na norma, para a interface *TransducerAcess* foram implementados os métodos descrito nas secções 5.2.1, 5.2.2, 5.2.3 e para a interface TEDManager foram implementados os métodos descritos nas secções 5.4.1, 5.4.2, 5.4.4.

Para realizar a implantação levou-se em conta que o comando é recebido da aplicação em forma de *array* ou lista, contendo todas as informações possíveis, que são utilizadas para se construir a estrutura de comando de mensagem, que é enviado à TIM também como no formato de um *array* através do módulo de comunicação. O comando recebido da aplicação varia de acordo com a necessidade, quer isto dizer que, os elementos existentes no *array* que chegam à NCAP 1451.0 não são fixos. As duas posições iniciais do *array* são ocupadas pela interface (serviço da API) e o método desta interface respetivamente, em *"string",* as outras posições são reservadas para os argumentos deste método. De seguida será mostrada a construção da estrutura de comando de mensagem para alguns dos serviços implementados.

`Ex1: Comando_recebido_Aplicação =` ["*TransducerAcess*","open",1,2]

Ao receber este comando, através do código criado em *python*, incorporado no eclipse, como descrito na secção 3.2, consegue-se saber o serviço ("*TransducerAcess*") a que se pretende ter acesso, bem como o método ("open") que se quer executar. Para esse caso em concreto, pretende-se abrir uma comunicação do transdutor, de outra forma pode-se dizer que se pretende iniciar uma comunicação para uma TIM e um canal específico como mostra a secção 2.3.5, o número um (1) e o dois (2) encontrados na última posição do *array* são os argumentos de entrada do método, estabelecidos pela norma. Ao executar o método, é retornado à aplicação, a comunicação do transdutor (*TransComm*), correspondente à TIM e o TC especificado no argumento.

`Ex2: Comando_recebido_Aplicação =` ["*TedsManager*","read_teds",6,3,1,2]

Este comando é diferente do anterior, mas, a sintaxe é a mesma, tendo em conta a estrutura de mensagem da secção 4.5.1, a NCAP IEEE 1451.0 possui eventos que constroem uma mensagem, seguindo e respeitado a norma como se vê na Tabela 6. A estrutura de comando a ser enviado na TIM deve ter a seguinte estrutura:

`Comando_enviar_TIM =` [channel_Id, classe_Id, Funtion_Id, ted_acess_cod, ted_offset]

No comando recebido tem: 6 – é o argumento de entrada *trans*_com este número permite obter o *canal*_Id e a TIM por meio de um conjunto predefinido durante a codificação, uma vez que não se efetuou a ligação com a TIM para se obter o ID de forma automática.

`TransCom =` [[1,1,0],[2,1,1],[3,1,2],[4,2,0],[5,2,1],[6,2,2]]

Neste caso o canal encontrado tem o channel_Id = 2 e TIM_Id = 2. **1** – é o argumento de entrada ted_type é o ted_acess_code, que pode ser qualquer valor encontrado na tabela 13

O método read_teds mostra-nos a classe e a função a que o comando pertence, na Tabela 12 aparece o Read_TEDS_segment como sendo o comando que em que se permite ler as TEDS,

tem como cmdFuntion 2, e, faz parte dos comandos comuns, essa informação leva-nos à tabela 11 e dá-nos a cmdClassId que está relacionada com o commonCmd, e, este é 1.

O tamanho, segundo a norma, é extraído do octeto dependente do comando que, para este caso em específico, segundo a tabela 21 da norma [42], encontrado na página 63 mostra que, depende do *TEDSAcessCode* argumento de entrada que é um inteiro de 1 byte e do *TEDSOffset* que é um inteiro de 4-bytes, logo o tamanho resulta da soma desses bytes. (tamanho=5 bytes). Desta forma a estrutura de mensagem construída para este comando toma a seguinte forma:

- `Comando_enviar_TIM` = [2, 1, 2,5,1,2]

Ao receber este comando, a TIM entende que o comando é para o canal 2, deve executar o comando comum com os seguintes valores: *classe_* Id 1, *funtion_*id 2, do tipo de TEDS com *ted_acess code* 1 (MetaTEDS) e que deve começar a leitura na linha 2, indicado pelo *TEDSOffset*. A TIM deve do mesmo modo elaborar uma estrutura de mensagem para responder a requisição feita pela NCAP, Este por sua vez, transforma o código proveniente da TIM e envia para NCAP.

Para os outros serviços é feito a mesma anologia, mudando os comandos e provalmente a quantidade de elementos, de acordo a necessidade como se já disse, o exemplo seguinte mostra é uma amostra do que se acabou de dizer.

Ex_3: `Comando_recebido_Aplicação` =["*TedsManager*","write_teds",6,3,4," *ArgumentoArray* "] para este comando o existe um elemento a mais que o anterior, e, este é o *ArgumentoArray*, que contém a informação que se pretende escrever na TEDS, este elemento altera o tamanho em função da informação a enviar, desta forma o tamanho resulta da soma dos bites de *TEDSAcessCode* e *TEDSOffset* que da 5, adicionado o tamanho da informação a ser escrita no TEDs contida no argumento *array*.

T =5 + len (*ArgumentoArray*), a estrutura da mensagem de comando a ser enviado fica

- `Comando_enviar_TIM` = [2, 1, 3,T,1,2, *ArgumentoArray*].

Ex_4: Comando_recebido_Aplicação = ["*TedsManager*","update_teds",6,3,4]

Para o exemplo 5, seguindo a mesma analogia o comando a enviar para TIM fica:

- `Comando_enviar_TIM` = [2, 1, 4, 1, 4]

Ex_5: Comando_recebido_Aplicação=["*TransducerAcess*","read_data",1,1,"sampling_mode","T""]

T= *ArgumentoArray*

- `Comando_enviar_TIM` = [0, 3, 1, 4, 'T']

5.7. Resultados esperados

Nesta subsecção serão apresentados os resultados que se obtiveram na execução dos serviços implementados, serão utilizados para a comprovação do que se escreveu no paragrafo anterior,

os exemplos enumerados anteriormente. De salientar que os resultados obtidos não serão valores que resultaram de uma medição feita pela TIM nem por qualquer outro sensor, visto que o projeto da TIM e da NCAP se fez em separado, não se efetuou a ligação entre elas. Para se poder obter os resultados, deste modo, os resultados obtidos são apenas a construção da estrutura do comando de mensagem, dos trabalhos selecionados e implementados. Foi preparado uma variável que receberá os comandos da aplicação, comando este que tem de ser modificado para permitir criar a estrutura de mensagem que é enviado para a TIM. Quer-se com isto dizer que, tão logo que a outra parte do projeto estiver pronta, desde que esta tenha sido implementada seguindo e respeitando as exigências estabelecidas pela norma, não terá nenhum impedimento na comunicação entre a NCAP e a TIM. A codificação dos métodos escolhidos tal como está na norma, será apresentada na parte dos anexos.

Algorítmo 1: interface *TransducerAcess e* método open.

```
if(services[0]=="TransducerAcess"):
    t=c=0
    trasncom= 'channel id inexistente'
    if (services[1]=="open"):
        print services[1]
        for i in range(6):
            if TransCom[i][1]==services[2]:
                t=1
                if TransCom[i][2]==services[3]:
                    c=1
                    transcom=TransCom[i][0]
                    break
        if(t==0):
            transcom='tim_Id inexistente'
        else:
            if(c==0):
              transcom= 'channel id inexistente'
        return transcom
```

```
Python 2.7.14 (v2.7.14:84471935ed, Sep 16 2017, 20:25:58) [MSC v.1500 64 bit (A
D64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
================ RESTART: C:/Python27/TransducerServices.py ================
inser o comando ["TransducerAcess","open",1,2]
open
3
```

Figura 34: Interface resultado do TransducerAcess e método open

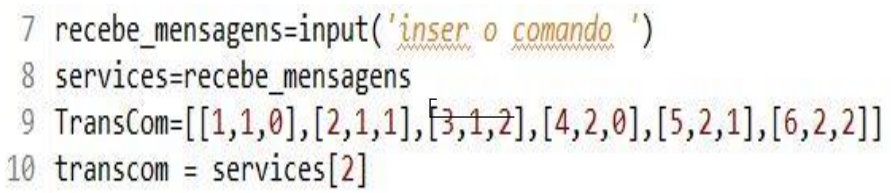

Figura 35: Cabeçalho da implementação

Para todos os serviços foi necessário levar em consideração o cabeçalho do código, por causa da falta de implementação dos outros serviços que serão implementados futuramente. O objetivo deste serviço é apresentado no algoritmo 1 cujo resultado é apresentado na Figura 34, esta mostra a comunicação de transporte (*TranscComm_ID*) de uma TIM e canal específico, para este caso quer se mostrar a comunicação de transporte da TIM 1 e canal 2, que de acordo a tabela predefinida é o 3, como se vê na Figura 19

Algorítmo 2: interface *TEDSMannager e* método *read_teds*.

```
if(services[0]=="TedsManager"):

    if (services[1]=="read_teds"):
        print services[1]
        if transcom<=6:
            for i in range(6):
                if TransCom[i][0]==transcom:
                    tim =TransCom[i][1]
                    channel = TransCom[i][2]
            tim_id=tim
            channel_id=channel
            cmdclass_id=1
            cmdfuntion_id=2
            tedcod_acess=services[4]
            leng = 5
            ted_offset=services[5]
            print "tim          =",tim_id
            print "ncnnal_id    =",channel_id
            print "cmdclass_id  =",cmdclass_id
            print "cmdfuntion_id=",cmdfuntion_id
            print "tamnho =      ",leng
            print "ted_cod_acess=",tedcod_acess
            command_message_estructure
=[channel_id,cmdclass_id,cmdfuntion_id,leng,tedcod_acess,ted_offset]
            return command_message_estructure
```

Figura 36: Resultado do método read_teds

Algorítmo 3: interface *TEDSMannager e* método *write_teds*.

```
if(services[0]=="TedsManager"):

    elif (services[1]=="write_teds"):
        print recebe_mensagens
        print services[1]
        if transcom<=6:

            for i in range(6):
                if TransCom[i][0]==transcom:
                    tim =TransCom[i][1]
                    channel = TransCom[i][2]

            tim_id=tim
            channel_id=channel
            cmdclass_id=1
            cmdfuntion_id=3
            tedcod_acess=services[4]
            if services[5]=="":
               write_cont="none"
            else:
                write_cont=services[5]
            leng = 5+ len(write_cont)
            ted_offset=3
            ##
            print "tim          =",tim_id
            print "ncnnal_id    =",channel_id
            print "cmdclass_id  =",cmdclass_id
            print "cmdfuntion_id=",cmdfuntion_id
            print "tamnho =      ",leng
            print "ted_cod_acess=",tedcod_acess
            print "o contéudo a escrever ",write_cont
            ##
            command_message_estructure
=[channel_id,cmdclass_id,cmdfuntion_id,leng,tedcod_acess,ted_offset,write_cont]
            return command_message_estructure
```

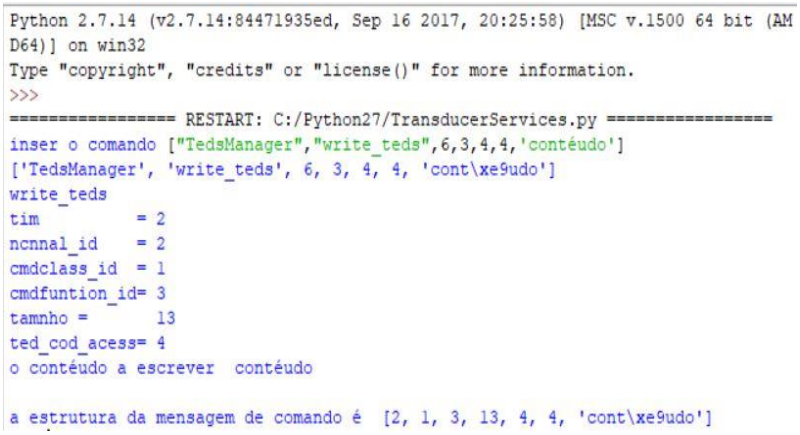

Figura 37: Resultado da interface TEDSMannager e método write_teds.

Algorítmo 4: interface *TEDSMannager e* método update_teds.

```
if(services[0]=="TedsManager"):

  if (services[1]=="update_teds"):
        print services[1]
        if transcom<=6:
            for i in range(6):
                if TransCom[i][0]==transcom:
                    tim =TransCom[i][1]
                    channel = TransCom[i][2]
            tim_id=tim
            channel_id=channel
            cmdclass_id=1
            cmdfuntion_id=4
            leng = 1
            tedcod_acess=services[4]
            print "tim           =",tim_id
            print "canal_id     =",channel_id
            print "cmdclass_id  =",cmdclass_id
            print "cmdfuntion_id=",cmdfuntion_id
            print "tamnho =       ",leng
            print "ted_cod_acess=",tedcod_acess
            command_message_estructure
=[channel_id,cmdclass_id,cmdfuntion_id,leng,tedcod_acess]
            return command_message_estructure
```

```
Python 2.7.14 (v2.7.14:84471935ed, Sep 16 2017, 20:25:58) [MSC v.1500 64 bit (AM
D64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
================ RESTART: C:/Python27/TransducerServices.py ================
inser o comando ["TedsManager","update_teds",6,3,4]
update_teds
tim           = 2
ncnnal_id     = 2
cmdclass_id   = 1
cmdfuntion_id= 4
tamnho =       1
ted_cod_acess= 4
a estrutura de comando é :  [2, 1, 4, 1, 4]
```

Figura 38: Resultado do método *update_teds*

Os algoritmos 2, 3 e 4 são a representação dos métodos *read_Teds*, *write_Teds* e *update_Teds*, selecionados da interface *TEDSMannager*, onde os seus resultados podem ser observados nas figuras 36, 37 e 38 respetivamente. A obtenção destes resultados deu-se pela execução do comando que a NCAP recebe da aplicação. Para o método *write_teds*, representado pelo algoritmo 3, o seu resultado é semelhante ao *read_Teds*, a diferença consiste na existência de um novo elemento na posição 6 que contém a informação que se pertente escrever na TEDS.

O método *update_teds*, representado no algoritmo 4, é o que contém a informação mais reduzida, quer seja da informação que recebe da aplicação, bem como a estrutura de mensagem criada, como se observa na figura 38.

Foram apresentadas até ao momento os métodos implementados da interface *TEDSMannager*, com a exceção do método "*open*" que faz parte da *TransducerAcess*. De seguida serão apresentados os outros métodos que pertencem à interface *TransducerAcess*, diferente dos métodos anteriores, desta vez apenas se fará a apresentação da construção do comando de mensagens dos dois códigos como se vê na figura 39. O código que originou este resultado, será colocado na parte dos anexos.

```
Python 2.7.14 (v2.7.14:84471935ed, Sep 16 2017, 20:25:58) [MSC v.1500 64 bit (AM
D64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
================ RESTART: C:/Python27/TransducerServices.py ================
inser o comando ["TransducerAcess","read_data",1,1,3]
read_data
tim          = 1
ncnnal_id    = 0
cmdclass_id  = 3
cmdfuntion_id= 1
tamnho       = 4
sampling_mode= 3

a estrutura da mensagem de comando é  [0, 3, 1, 4, 3]
>>>
================ RESTART: C:/Python27/TransducerServices.py ================
inser o comando ["TransducerAcess","write_data",1,1,3,"gsgsg"]
write_data
tim          = 1
ncnnal_id    = 0
cmdclass_id  = 3
cmdfuntion_id= 2
tamnho       = 9
sampling_mode= 3
write conteudo gsgsg

a estrutura da mensagem de comando é  [0, 3, 2, 9, 3, 'gsgsg']
```

Figura 39: Resultados para os métodos read_data e write_data

Capítulo 6 – Conclusões

O desenvolvimento desta dissertação foi feito em torno do seu objetivo principal que foi a construção de uma plataforma que suporte o desenvolvimento de sensores inteligentes e que observem a norma ISO/IEC/IEEE 1451, fulcralmente, na implementação dos serviços que permitiriam a ligação entre o Processador de Aplicação Capacitador de Rede (NCAP) e um Módulo de Interface de Transdutor Inteligente (TIM), esta dissertação centrou-se no lado da NCAP, o objetivo foi alcançado, pois no final conseguiu-se construir a estrutura de mensagem baseado na norma IEEE 1451, e, enviada à TIM.

Uma vez que os comandos são enviados de uma aplicação por meio de uma rede, foram criadas as condições suficientes para se receber este comando, fez-se a simulação da inserção do *Raspberry* Pi numa rede de três utilizadores, e, a mesma funcionou sem inconvenientes, o que se pressupõe que não existirá constrangimentos ao enviar comandos da aplicação para a NCAP. Para que tudo se consumasse no final foi necessário a utilização de ferramentas que facilitaram o desenvolvimento deste projeto, ferramentas como eclipse, que permitem incorporar o python e que foi utilizado como a linguagem de programação, que por meio do qual foram implementados os serviços da NCAP IEEE 1451.0.

O eclipse permitiu ainda que se efetuasse a simulação de ligação do *Raspberry P*i a uma rede de utilizadores, bem como o gerenciamento do *Raspberry Pi* de forma remota, uma vez que a NCAP é colocada nesta placa. No final foram analisado e experimentados 4 interfaces das 6 existentes no serviço de acesso aos transdutores *(Transducer Acess Services)* dentre os quais se obteve resultados de 10 métodos destas interfaces, o *report_comm_module*, o *report_tims* e o *report-channel* para a interface *TIMDiscovery*, os métodos *read_teds, write_teds, query_teds* e *update_teds* da interface *TEDSMannager,* e os métodos *open, read_data* e *write_data* da *inetrface TransducerAcess,* os resultados foram obtido tal como se esperava, por cada execução se obteve a construção da estrutura de mensagem de comando correspondente, tendo em conta os requisitos exidos pela norma, onde alguns comandos estão preparados para serem enviados à TIM e outros ao modulo de comunicação.

Pode-se dizer com certeza que a implementação do projeto completo será uma mais valia e trará grandes benefícios às pessoas que estão diretas ou indiretamente ligadas ao uso de tecnologia para exploração de medidas de grandezas ambientais e não só. Uma vez que a cada dia que nasce a presença de IoT é muito evidente, tem se verificado mais coisas a serem ligados à internet do que pessoas, sendo assim, vale apenas prestar atenção neste projeto.

6.1. Evolução e trabalhos futuros

Atendendo à grandiosidade e importância deste projeto, pretende-se continuar a pesquisa para evoluir e tornar mais prático a implementação de modo a facilitar o uso da aplicação.

Em princípio, os trabalhos futuros que se deve implementar para terminar, são os mencionados na norma, trabalhos estes que tornarão completo o projeto. Assim, espera-se que futuramente se possa implementar primariamente a parte do módulo de comunicação observando a norma.

Até aqui os resultados obtidos foram adquiridos por meio de comandos inserido de forma manual, o que pode ser facilmente escrito de forma errada, desta feita, espera-se igualmente que se possa: criar uma interface gráfica que permitirá ao utilizador efetuar a inserção dos comandos de forma automática, onde os comandos devem ser predefinidos durante a implementação, dando a possibilidade ao utilizador de escolher os elementos da estrutura de comando de acordo com a tarefa que pretende executar, ao invés de escrever de forma manual.

Finalmente, para concluir com o projeto, espera-se que se possa: efetuar a ligação entre a NCAP e a TIM, onde nesta última poderá existir alguns sensores e atuadores de diferentes grandezas, podendo assim ser efetuado as medições destas grandezas e enviar os resultados ao utilizador por meio da NCAP, podendo desta forma sentir a beleza e o poder da IoT.

Referência Bibliográfica

[1] J. Velez, R. Trafford, M. Pierce, B. Thomson, E. Jastrzebski, and B. Lau, "IEEE 1451-1-6 : Providing Common Network Services over MQTT," p. 12, 2018.

[2] F. FEIMEC and R. Fragoso, "24-28 A B R I L 2018," pp. 1–32, 2018.

[3] IPQ (Instituto Português da Qualidade), "Módulo 1 - O que são Normas e o que é a Normalização ?," p. 16, 2015.

[4] G. Vargas, "ESTUDOS BÁSICOS SOBRE NORMALIZAÇÃO: origem, conceitos e organismos reguladores.," p. 9, 2006.

[5] T. B. R. De Almeida, "O IMPACTO DO IEEE NA FORMAÇÃO PROFISSIONAL DOS ESTUDANTES DE ENGENHARIA ELÉTRICA POR MEIO DA," no. Região 7, p. 3, 2005.

[6] L. C. Nunes and L. F. Galvão, "Cálculo Numérico," *Unicesumar*, p. 208, 2016.

[7] L. B. Torri, "A Norma IEEE 1451 Aplicada a Redes Heterogêneas de Sensores sem Fios," p. 77, 2008.

[8] J. Higuera, W. Hertog, M. Perálvarez, J. Polo, and J. Carreras, "Smart lighting system ISO/IEC/IEEE 21451 compatible," *IEEE Sens. J.*, vol. 15, no. 5, pp. 2595–2602, 2015.

[9] N. Capable and A. Processor, "INTERNATIONAL STANDARD ISO / IEC / IEEE," vol. 2010, 2010.

[10] J. A. Guevara Rosas, "Arquitectura de nodo inteligente para redes de sensores inalámbricas y escalables: aplicaciones en monitorización ambiental," 2016.

[11] Mônica Mancini, "nternet das Coisas: História, Conceitos, Aplicações e Desafios," p. 9, 2017.

[12] D. Evans, "A Internet das Coisas Como a próxima evolução da Internet está mudando tudo," *Cisco*, pp. 1–13, 2011.

[13] B. P. Santos, L. A. M. Silva, C. S. F. S. Celes, and J. B. Borges, "Internet das Coisas: da Teoria à Prática.," *Homepages.Dcc.Ufmg.Br*, p. 52, 2016.

[14] I. Standard, *INTERNATIONAL STANDARD ISO / IEC / IEEE transducer interface for sensors and communication protocols and formats*, vol. 2010. 2010.

[15] R. M. D. E. Jesus, "' MATIW 1451 – Monitoramento e Acionamento de Transdutores Inteligentes atrav és da Web (Padr ão IEEE 1451) ,'" Universidade Estadual Paulista, Ilha Solteira, 2007.

[16] A. Gomes, "Arquitetura Hardware / Software de um Núcleo NCAP Segundo o Padrão IEEE 1451 . 1 : Uma prova de conceito Arquitetura Hardware / Software de um Núcleo NCAP Segundo o Padrão IEEE 1451 . 1 : Uma prova de conceito," pp. 16–18, 2010.

[17] A. Gomes, “Arquitetura Hardware / Software de um Núcleo NCAP Segundo o Padrão IEEE 1451 . 1 : Uma prova de conceito Arquitetura Hardware / Software de um Núcleo NCAP Segundo o Padrão IEEE 1451 . 1 : Uma prova de conceito,” p. 2, 2010.

[18] A. Sreejithlal, A. Jose, A. Shooja, and B. M. Kumar, “IEEE 1451.2 based smart sensor system using ADuc847,” *Proc. - 2015 Int. Conf. Commun. Inf. Comput. Technol. ICCICT 2015*, 2015.

[19] LEANDRO PRYTULA SENSOR, “SENSOR INTELIGENTE,” UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL ESCOLA DE ENGENHARIA, 2011.

[20] A. C. Supervisor, D. A. Jos, P. Figueiredo, S. Co-supervisor, D. M. Alexandre, and O. L. Thesis, “Universidade de lisboa instituto superior técnico,” no. November, 2016.

[21] V. Brusamarello, “Introdução de transdutores,” *Univ. Fed. do Rio Gd. do Sul*, 2009.

[22] C. Antonio, A. Dee, and C. Iii, “Eletrônica Básica / ELE 0937 Eletrônica Básica,” pp. 1–41, 2014.

[23] A. Amaral, “baseadas em Transdutores Inteligentes : Padrões IEEE Livros Grátis,” pp. 21–21, 2010.

[24] M. Richardson and S. Wallace, “Primeiros passos com o raspberry pi,” p. 192, 2013.

[25] J. Ramos, “O Sistema Operacional Universal,” p. 2, 2014.

[26] D. Le Breton, “Uma breve história da adolescência,” vol. 22, no. June, p. 160, 2017.

[27] U. F. Fluminense and E. De Engenharia, “Tutorial de Introdução ao Python,” 2009.

[28] C. de O. L. e outros Alexandre V. S. Lage, “tutorial de eclipse,” pp. 119–136, 2012.

[29] ARM, “mbed - Product Summary mbed - Product Details,” p. 1, 2014.

[30] “Socket Programming in What is a Socket ?,” pp. 1–12, 2017.

[31] C. Carbajal and I. Cem, “Network Programming,” 2013.

[32] F. Boem, S. Riverso, G. Ferrari-Trecate, and T. Parisini, “Plug-and-Play Fault Detection and Isolation for Large-Scale Nonlinear Systems with Stochastic Uncertainties,” *IEEE Trans. Automat. Contr.*, vol. 9286, no. c, 2018.

[33] E. Fantechi, “Table of of contents,” *Kernel-Mode Driv. Archit. Des. Guid.*, pp. 1–135, 2017.

[34] WESLEY BECARI, “Rede de sensores para engenharia biomédica utilizando o protocolo ieee1451,” ESCOLA POLITÉCNICA DA UNIVERSIDADE DE SÃO PAULO DEPARTAMENTO DE ENGENHARIA DE SISTEMAS ELETRÔNICOS, 2012.

[35] A. R. Rossi, “Anderson rodrigo rossi uma nova técnica de comunicação e alimentação de

transdutores inteligentes utilizando apenas um fio baseada no padrão ieee 1451," 2013.

[36] ISO, "International Standard Iso," *IEEE Stand. a Smart Transducer Interface Sensors Actuators— Common Funct. Commun. Protoc. Transducer Electron. Data Sheet Formats TM 1451.0*, vol. 2004, p. 238, 2004.

[37] M. de O. Namba, "Modelagem e Especificação de um Middleware para Redes de Sensores Sem Fio Aplicado à Saúde," *Portal.Inf.Ufg.Br*, p. 108, 2011.

[38] I. T. Popovic and N. M. Rajovic, "Design of IEEE 1451 smart transducer communication module for embedded application," *5th Eur. Conf. Circuits Syst. Commun. (ECCSC 2010)*, pp. 236–239, 2010.

[39] L. M. Rufino, R. V. Steiner, R. V. Raimundo, and A. A. Fröhlich, "Improvements in wireless communication and support for WMSN in IEEE 1451," *Eur. Signal Process. Conf.*, pp. 109–114, 2014.

[40] M. de O. Namba, "Modelagem e Especificação de um Middleware para Redes de Sensores Sem Fio Aplicado à Saúde," UNIVERSIDADE FEDERAL DE GOIÁS INSTITUTO DE INFORMÁTICA, 2011.

[41] IEEE Instrumentation and Measurement Society, *IEEE Standard for a Smart Transducer Interface for Sensors and Actuators Wireless Communication Protocols and Transducer Electronic Data Sheet (TEDS) Formats*, no. September. 2007.

[42] L. Raichel *et al.*, "International Standard Iso / Iec," vol. 2010, 2005.

[43] DOUGLAS CIRIACO, "O que é API? - TecMundo," 2009. [Online]. Available: https://www.tecmundo.com.br/programacao/1807-o-que-e-api-.htm. [Accessed: 03-May-2018].

[44] J. Río, M. Martínez, D. M. Toma, A. Mànuel, and H. G. Ramos, "IEEE 1451 HTTP Server Implementation for Marine Data," no. 6, p. 8800, 2011.

Anexo

Algorítmo Interface TimDiscovery

```
class TimDiscovery(object):
    '''
    classdocs
    '''
    def __init__(self):
        '''
        Constructor
        '''
    def  report_comm_module ( self ):

            error_code  =  0
            module_ids  = []
            return  { 'error_code' :  error_code ,  'module_ids' :
module_ids }

    def  report_tims ( self ,  module_id ):

        error_code  =  0
        tim_ids  =  []
        return  { 'error_code' :  error_code ,  'tim_ids' :  tim_ids }

    def  report_channels ( self ,  tim_id ):

        self.tim_id = tim_id
        error_code  =  0
        channel_ids  =  []
        names  =  []
        result =  { 'error_code' :  error_code, 'channel_ids' :
channel_ids,'channel_names' :  names }
        return result
```

Algorítmo Interface TedsManager

```
class TedsManager(object):
    '''
    classdocs
    '''
    def read_teds(self, trans_comm_id, timeout, teds_type):

        error_code = 0
        teds = []
        return {'error_code': error_code, 'teds': teds}

    def write_teds(self, trans_comm_id, timeout, teds_type,
arg_array_teds):
        error_code = 0
        return {'error_code': error_code}

    def read_raw_teds(self, trans_comm_id, timeout, teds_type):

        error_code = 0
        raw_teds = ()
        return {'error_code': error_code, 'raw_teds': raw_teds}

    def write_raw_teds(self, trans_comm_id, timeout, teds_type,
raw_teds):

        error_code = 0
        return {'error_code': error_code}

    def update_teds_cache(self, trans_comm_id, timeout, teds_type):

        error_code = 0
        return {'error_code': error_code}
```

Algorítmo Interface TedsManager

```
class TransducerAcess(object):
    '''
    classdocs
    '''
    def open(self, tim_id, channel_id): # implentar
        error_code = 0
        trans_comm_id = 0
        return {'error_code': error_code, 'trans_comm_id': trans_comm_id}

    def open_qos(self, tim_id, channel_id, qos_params):

        error_code = 0
        qos_params = ()
        trans_comm_id = 0
        result = {'error_code': error_code,'qos_params':
qos_params,'trans_comm_id': trans_comm_id}

        return result

    def open_group(self, tim_ids, channel_ids):

        error_code = 0
        trans_comm_id = 0
        return {'error_code': error_code, 'trans_comm_id': trans_comm_id}

    def open_group_qos(self, tim_ids, channel_ids, qos_params):

        error_code = 0
        qos_params = ()
        trans_comm_id = 0
        result = {'error_code': error_code,'qos_params':
qos_params,'trans_comm_id': trans_comm_id}

        return result

    def close(self, trans_comm_id): #implentar

        error_code = 0
        return {'error_code': error_code}

    def read_data(self, trans_comm_id, timeout, sampling_mode):
#implentar

        error_code = 0
        result = ()     #são os valores retornados.
        return {'error_code': error_code, 'result': result}

    def write_data(self, trans_comm_id, timeout, sampling_mode, value): #
value: é o ArgumentArray contendo o valor a ser enviado

#implentar
        error_code = 0
        return {'error_code': error_code}

    def start_read_data(self, trans_comm_id, trigger_time,
timeout,sampling_mode, callback):
        error_code = 0
```

```
        operation_id = 0
        return {'error_code': error_code, 'operation_id': operation_id}

    def start_write_data(self, trans_comm_id, trigger_time, timeout,
sampling_mode, value, callback):

        error_code = 0
        operation_id = 0
        return {'error_code': error_code, 'operation_id': operation_id}

    def start_stream(self, trans_comm_id, callback, operation_id):

        error_code = 0
        operation_id = 0
        return {'error_code': error_code, 'operation_id': operation_id}

    def cancel(self, operation_id):
        error_code = 0
        return {'error_code': error_code}
```