UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
DEPARTAMENTO DE AUTOMAÇÃO E SISTEMAS
ENGENHARIA DE CONTROLE E AUTOMAÇÃO

Lucas Henrique Alves Feitosa

Sistema de Gestão e Análise de Defeitos em Sistemas Ciber-físicos Conectados à Internet

Florianópolis
2019

Lucas Henrique Alves Feitosa

Sistema de Gestão e Análise de Defeitos em Sistemas Ciber-físicos Conectados à Internet

Relatório submetido à Universidade Federal de Santa Catarina como requisito para a aprovação na disciplina DAS 5511: Projeto de Fim de Curso do curso de Graduação em Engenharia de Controle e Automação.
Orientador(a): Prof. Antônio Augusto Medeiros Fröhlich, Dr.

Florianópolis
2019

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Feitosa, Lucas Henrique Alves
Sistema de gestão e análise de defeitos em sistemas ciber-físicos conectados à internet / Lucas Henrique Alves Feitosa ; orientador, Antônio Augusto Medeiros Fröhlich, 2019.
70 p.

Trabalho de Conclusão de Curso (graduação) - Universidade Federal de Santa Catarina, Centro Tecnológico, Graduação em Engenharia de Controle e Automação, Florianópolis, 2019.

Inclui referências.

1. Engenharia de Controle e Automação. 2. Sistemas ciber físicos. 3. Internet das coisas. 4. Sistemas de gerenciamento. 5. Root cause analysis. I. Fröhlich, Antônio Augusto Medeiros. II. Universidade Federal de Santa Catarina. Graduação em Engenharia de Controle e Automação. III. Título.

Lucas Henrique Alves Feitosa

Sistema de Gestão e Análise de Defeitos em Sistemas Ciber-físicos Conectados à Internet

Esta monografia foi julgada no contexto da disciplina DAS5511: Projeto de Fim de Curso e aprovada na sua forma final pelo Curso de Engenharia de Controle e Automação.

Florianópolis, 27 de novembro de 2019

Banca Examinadora:

Antônio Augusto Medeiros Fröhlich
Orientador na Empresa
LISHA

Prof. Antônio Augusto Medeiros Fröhlich, Dr.
Orientador no Curso
Universidade Federal de Santa Catarina

Prof. Leandro Buss Becker, Dr.
Avaliador
Universidade Federal de Santa Catarina

Jamal Rahman
Debatedor
Universidade Federal de Santa Catarina

Murilo Peruch Nunes
Debatedor
Universidade Federal de Santa Catarina

Agradecimentos

Agradeço aos meus pais pelo auxílio e encorajamento durante todas as etapas de minha formação.

À minha namorada Taynara por estar ao meu lado e ter me apoiado nessa fase da minha vida, te amo.

Ao LISHA por abrir suas portas para o desenvolvimento deste trabalho e para aquisição de conhecimentos. Em especial agradeço ao meu orientador, Guto, ao colega de equipe João e ao restante dos colegas de laboratório, sempre dispostos a discutir e auxiliar na solução dos problemas.

À UFSC e ao curso de Engenharia de Controle e Automação por proporcionarem minha formação acadêmica e me auxiliarem a chegar neste ponto. Aos muitos professores que tive durante minha vida, muito obrigado por seus ensinamentos.

E aos demais amigos e familiares que sempre estiveram presentes e ficam alegres por minhas conquistas.

Muito obrigado.

Resumo

O LISHA é um laboratório da UFSC com uma forte vertente em design de sistemas embarcados e que tem obtido sucesso na realização de pesquisas em segurança, confiança, design e eficiência energética de Rede de Sensores Sem Fio, no contexto de *IoT* (*Internet of Things*) e *CPS* (*Cyber-Physical Systems*). Para validar essas pesquisas, têm desenvolvido projetos e parcerias nos setores ambiental, fotovoltaico, de veículos elétricos e domótico, utilizando uma plataforma de desenvolvimento de soluções *IoT* concebida no laboratório. Essa, guia o desenvolvimento e a implementação do sistema *IoT* desde o sistema embarcado até a aplicação *Web* de monitoramento e controle. O processo de desenvolvimento desses projetos, no entanto, apresenta grande demanda de manutenção de dispositivos instalados exibindo defeitos. Uma falta de metodologia e estrutura de documentação e apresentação de dados tem dificultado o processo de análise desses defeitos, atrasando ou até impossibilitando a identificação das causas raiz dos problemas. Isso, consequentemente, dificulta um tratamento eficaz dos problemas, levando a manutenções paliativas e acúmulo desses problemas. Buscando estruturar, sistematizar e integrar a documentação e a apresentação de todo o processo de desenvolvimento de projetos *IoT*, este trabalho apresenta o desenvolvimento de um sistema *Web* de gerenciamento de projetos *IoT*, a validação do mesmo em um estudo de caso e uma análise dos dados de sintomas, relações de causa-efeito, causas raiz e tratamento de falhas, resultantes do mesmo.

Palavras-chave:. sistemas ciber-físicos, internet das coisas, *Root Cause Analysis*, sistema de gerenciamento.

Abstract

LISHA is a UFSC laboratory with a strong embedded design approach and has been successfully conducting research on safety, reliability, design and energy efficiency of Wireless Sensor Network (WSN) in the context of IoT (Internet of Things) and CPS (Cyber-Physical Systems). To validate those researches, the laboratory has been developing projects and partnerships in the environmental, photovoltaic, electric vehicle and home automation sectors, using its IoT development platform. This guides the development and implementation of the IoT systems from the embedded system to the monitoring and control Web application. The development process of these projects, however, has a high maintenance demand on installed devices that exhibit defects. A lack of methodology and structure for documenting and presenting data has been hampering the process of analyzing those defects, delaying or even making it impossible to identify the problems root causes. This, consequently, hinders an effective treatment of the problems, leading to palliative maintenance and accumulation of those problems. Seeking to structure, systematize and integrate the documentation and presentation of the entire IoT project development process, this paper presents the development of an IoT project management web system, its validation in a case study and an analysis of the resulting data concerning symptoms, cause-effect relationship, root causes, and treatment of failures.

Keywords: cyber-physical systems, internet of things, root cause analysis, managment system.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

API - do inglês *Application Programming Interface*

CAN - do inglês *Controller Area Network*

CIA2 - Construindo Cidades Inteligentes: da Instrumentação dos Ambientes ao desenvolvimento de Aplicações

CMS - Sistema de Gerenciamento de Conteúdo, do inglês *Content Management System*

CRUD - do inglês *Create, Read, Update and Delete*

ECEF - do inglês *Earth-Centered Earth-Fixed*

EPOS - do inglês *Embedded Parallel Operating System*

ER - Entidade-Relação

HTTPS - do inglês *Hypertext Transfer Protocol Secure*

IA - Inteligência Artificial

IoT - Internet das coisas, do inglês *Internet of Things*

LISHA - Laboratório de Integração de Software e Hardware

PRAD - Projeto de Recuperação da Qualidade da Água dos Córregos do Campus Reitor João David Ferreira Lima

RCA - Análise de Causas Raiz, do inglês *Root cause analysis*

RF - Requisito Funcional

RSSF - Rede de Sensores Sem Fio

SBTVD - Sistema Brasileiro de Televisão Digital

SI - Sistema Internacional de Unidades

SSB - do inglês *Smart Solar Building*

SVN - do inglês *Apache Subversion*

TSTP - do inglês *Trustful Space-Time Protocol*

UFSC - Universidade Federal de Santa Catarina

UML - do inglês *Unified Modeling Language*

URL - do inglês *Uniform Resource Locator*

Sumário

1 Introdução

A internet das coisas (*IoT*) tem permitido a criação de sistemas de monitoramento e atuação inteligentes, interconectados e coordenados, utilizando sistemas embarcados, rede de sensores/atuadores sem fio, *Cloud* e *Edge Computing*, *Big Data* e Inteligência Artificial (IA). Esses sistemas, onde os componentes computacionais interagem e controlam aspectos do mundo físico, são denominados Sistemas Ciber-Físicos (*CPS*) e vêm sendo implementados em diversos domínios: mobilidade urbana, meio-ambiente, Indústria 4.0, distribuição de energia, domótica, etc.

Esse novo paradigma tem permitido um maior conhecimento dos processos, personalização para usuários e ações gerenciais baseados em dados (*Data-Driven*). Ao mesmo tempo, tem trazido desafios quanto a diversos aspectos, tais quais segurança, design, quantidade de dados, padronização, sustentabilidade e gerenciamento.

O Laboratório de Integração de Software e Hardware (LISHA), da Universidade Federal de Santa Catarina (UFSC), onde este trabalho foi desenvolvido, tem promovido pesquisas que abordam alguns desses desafios que emergem do *IoT*. Ao mesmo tempo, vem aplicando e validando essas pesquisas em projetos e parcerias nos setores Ambiental, Fotovoltaico, de Veículos Elétricos e Domótico.

Muitas dessas aplicações passam a apresentar problemas recorrentes após instaladas. Por diversas vezes, esses problemas acabam sendo tratados com manutenções paliativas (como o *reset* ou a troca de dispositivos), de forma a postergar e acumular problemas. Essa situação, unida à limitação da mão de obra do laboratório, se torna um problema ao passo que os projetos escalonam.

O restante deste capítulo visa aprofundar a análise do problema apresentado, fundamentando a motivação do trabalho, bem como delinear os objetivos do trabalho frente à problemática, alinhar o escopo de trabalho no curso de Engenharia de Controle e Automação e descrever a estrutura do documento.

1.1 Motivação

Inserido no desenvolvimento de projetos *IoT* no LISHA (descrito em detalhes na seção 2.1), notou-se uma grande demanda de manutenção em dispositivos instalados apresentando falhas. Os dispositivos em questão são tanto ligados a projetos já entregues, que passaram pelos testes e pelo período de validação, quanto a projetos em desenvolvimento.

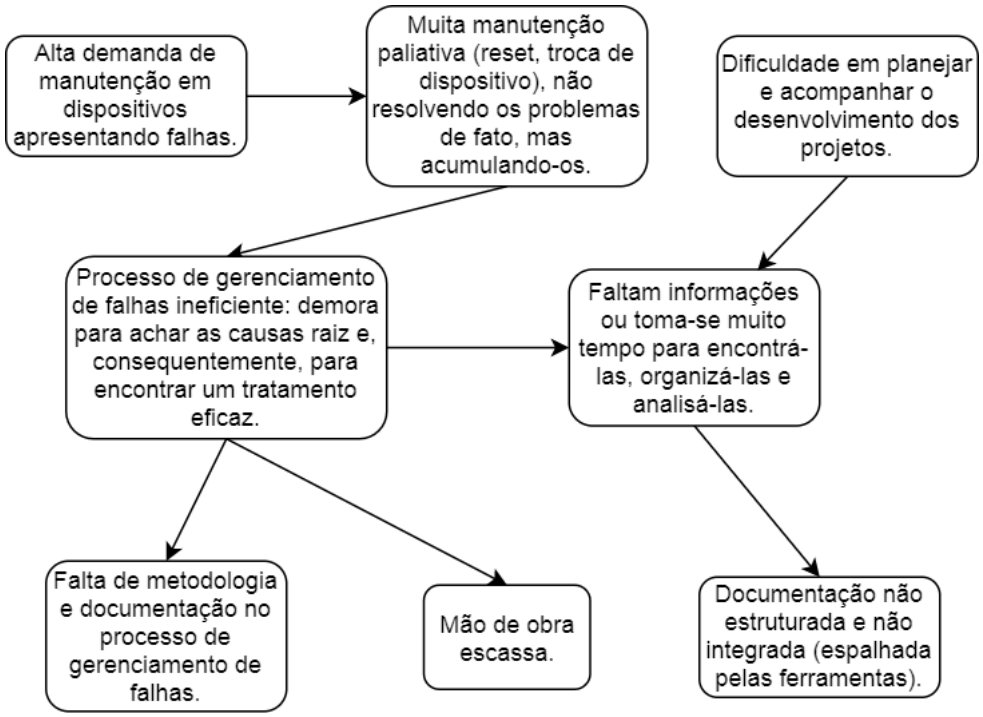
Analisando o processo de desenvolvimento desses dispositivos, verificou-se uma ineficiência na etapa de gerenciamento de falhas, tomando-se muito tempo na identificação

das causas raiz dos problemas e atrasando um tratamento eficaz dos mesmos. Com isso, há ocorrência de muitas manutenções paliativas, como o *reset* ou a troca de dispositivos, fazendo com que os problemas, por não serem resolvidos de fato, se acumulem e voltem a ocorrer com frequência.

Isso levou a um estado, no LISHA, em que diversas estações instaladas estavam apresentando problemas. *Gaps* de dados monitorados e estações travadas são exemplos dos sintomas desses problemas. Consequentemente emerge disso, uma constante necessidade de manutenções, deixando a escassa mão de obra do laboratório refém desse processo.

Analisando a ineficiência dos processos citados, verificou-se que muitas informações relevantes são perdidas, ou que toma-se muito tempo para encontrá-las, estruturá-las e organizá-las. Isso se deve a uma falta de estrutura e metodologia perante os processos, principalmente o de gerenciamento de falhas. Essa lacuna, unida à não integração de informações entre as plataformas de documentação e de dados do *LISHA* e à limitação da mão de obra no laboratório, leva a uma dificuldade em manter, estruturar, organizar e apresentar as informações relevantes para a análise de falhas e para o planejamento e acompanhamento de etapas do ciclo de desenvolvimento e de vida das soluções *IoT*. Deste modo, a motivação deste trabalho pode ser resumida no diagrama causa-efeito da figura 1.

Figura 1 – Diagrama de causa-efeito dos problemas que motivaram o desenvolvimento deste trabalho



Fonte: Autor

Alguns exemplos de informações que seriam relevantes para o aprimoramento do trabalho realizado no LISHA e que estão ausentes são:

- Dispositivos instalados: manual, datasheet, versão, inventório, localização, versão do firmware, etc;
- Dados da aplicação: dados dos sensores, logs da aplicação, relacionamento entre nodos;
- Informações de projetos: descrição, objetivos, metas, prazos, etc;
- Planejamentos, tarefas, relatos, acontecimentos e falhas passadas;
- Etapa do ciclo de desenvolvimento.

1.2 Objetivos

Analisando os problemas expostos, percebem-se três frentes de ataque viáveis:

a) Estruturar, sistematizar e integrar a documentação de todo o processo de desenvolvimento de projetos *IoT* para atacar a ineficiência nos processos de gerenciamento de falhas e de planejamento e, indiretamente, a alta demanda de manutenção;

b) Aprimorar o processo de design e implementação dos projetos para minimizar o acontecimento e o efeito de falhas, como por exemplo a aplicação de técnicas de design tolerante a falhas;

c) Aperfeiçoar o processo de testes (testes automatizados, simulações, *hardware in the loop*) para maximizar a detecção de problemas de design antes da instalação, minimizando a demanda de manutenção.

Dessas três frentes, considera-se a primeira como sendo a mais urgente e que terá um impacto mais amplo e mais rápido. Portanto, apenas a primeira frente será abordada nesse trabalho, deixando as outras duas como possíveis - e necessários - trabalhos futuros.

Considerando os problemas expostos, o presente trabalho busca atingir os seguintes objetivos gerais:

1. Aprimorar o processo de gerenciamento de falhas, realizando análises mais profundas sobre o tema, buscando promover maior agilidade na solução dessas falhas e tratar as causas raiz dos problemas identificados.
2. Aprimorar os processos de planejamento e acompanhamento do desenvolvimento dos projetos *IoT*.

Para implementar os objetivos gerais acima citados, são propostos os objetivos específicos:

1. Estruturar, sistematizar e integrar a documentação de todo o processo de desenvolvimento de projetos *IoT* através de um sistema de gerenciamento *Web*.

2. Introduzir o sistema implementado ao processo de desenvolvimento de projetos *IoT*, envolvendo-o em um estudo de caso para validá-lo e para gerar um banco de dados de sintomas, análises de causa-efeito, causas raiz e tratamentos de falhas.

1.3 Contexto do trabalho no curso de Engenharia de Controle e Automação

Durante o desenvolvimento do trabalho, diversas áreas do conhecimento foram necessárias para atingir o objetivo final, em sua maioria tendo relação com o curso de Introdução Engenharia de Controle e Automação. De maneira geral, o desenvolvimento de projetos *IoT* requer conhecimentos de elétrica e eletrônica, programação de sistemas embarcados, redes e protocolos de comunicação, instrumentação de sensores e atuadores, modelagem de processos e, dependendo do caso, sistemas de controle. Além disso, o trabalho necessitou de conceitos de modelagem de *software* e programação *Web*, por conter o desenvolvimento de um sistema *Web*.

1.4 Estrutura do relatório

O restante deste documento está organizado em seis capítulos. No capítulo 2 apresenta-se a empresa onde o projeto foi desenvolvido, abordando sua história, área de atuação e processos. No capítulo 3 são discutidas as propostas, o embasamento e a metodologia do projeto. Em sequência, no capítulo 4, são discutidos os conceitos e ferramentas utilizados no desenvolvimento do trabalho. No capítulo 5 é apresentado o desenvolvimento do projeto, descrevendo cada uma das etapas realizadas. Por fim, nos capítulos 6 e 7, são apresentados os resultados e os trabalhos futuros e as considerações finais, respectivamente.

2 LISHA

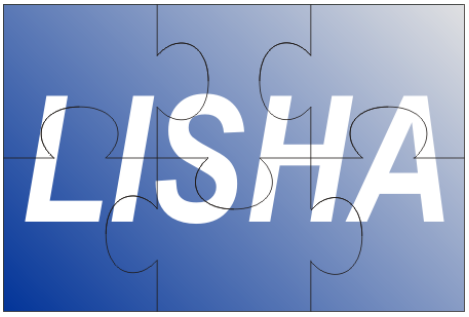
O LISHA foi fundado em 1985 no campus Florianópolis da UFSC, com o intuito de promover pesquisas nas áreas de arquitetura de computadores, sistemas operacionais e rede de computadores e suas aplicações. Em 2013, expandiu para o campus Joinville da UFSC.

Desde 2001 mantém e faz pesquisa em cima do *EPOS* (*Embedded Parallel Operating System*) [1], uma plataforma *open-source* de desenvolvimento de sistemas embarcados, desenvolvida na tese de doutorado do Professor Antônio Augusto Fröhlich, hoje coordenador do laboratório. O projeto se baseia no método *Application-Driven Embedded System Design* [2], que guia o desenvolvimento de componentes de *software* e de *hardware* automaticamente adaptados perante as especificações de uma aplicação, permitindo ao desenvolvedor focar seu trabalho nas aplicações.

Ao longo dos anos, participou de projetos da esfera pública e privada, e em diferentes domínios de aplicação, como por exemplo o Sistema Brasileiro de Televisão Digital (SBTVD) e o CIA^2 (acrônimo para Construindo Cidades Inteligentes: da Instrumentação dos Ambientes ao desenvolvimento de Aplicações). No primeiro, fez parte, entre 2005 e 2006, do consórcio de instituições acadêmicas que estudaram a proposta da definição do SBTVD, tendo contribuído diretamente com os estudos da camada de transporte e na codificação de áudio e vídeo. No CIA^2, do qual fez parte entre 2011 e 2013, foi representante da UFSC no projeto envolvendo 18 universidades brasileiras que cooperaram para construir os fundamentos de uma cidade inteligente, tendo sido responsável, juntamente com a Universidade de Brasília, por desenvolver e validar uma infraestrutura para aplicações IoT.

Naturalmente, devido ao contexto da ciência e tecnologia, a pesquisa e desenvolvimento no laboratório passou a abordar os desafios trazidos pela internet das coisas como

Figura 2 – Logo do LISHA



Fonte: LISHA

segurança, design inteligente, eficiência energética, dentre outros. Dessa vertente, surgiram importantes pesquisas, como:

a) *EPOSMote* [3]: uma plataforma *open-source* de *hardware*, com os componentes necessários para o desenvolvimento de Rede de Sensores Sem Fio (RSSF) e *CPS*.

b) *SmartData* [4]: uma *Application Programming Interface* (*API*) de alto nível para redes de sensores que visa fornecer uma abstração de dados sensoriais auto-contida em termos de semântica, localização, temporização e confiança, para o sensoriamento;

c) *TSTP* (*Trustful Space-Time Protocol*) [5]: um protocolo de comunicação *cross-layer* para redes de sensores sem fio, que entrega mensagens autenticadas, criptografadas, temporizadas, geo-referenciadas e contendo dados de acordo com o Sistema Internacional de Unidades (SI).

A forte vertente em desenvolvimento de sistemas embarcados, unido ao sucesso das recentes pesquisas em desafios do *IoT* e à constante realização de parcerias e projetos, que permite a implementação e validação desses conceitos, culminou na criação de uma plataforma completa de desenvolvimento de soluções *IoT*. Esse e outros aspectos do desenvolvimento de projetos *IoT* no LISHA serão abordados à seguir.

2.1 Desenvolvimento de projetos *IoT*

Os professores coordenadores do LISHA (Florianópolis e Joinville) captam projetos das esferas pública e privada, arrecadando recursos para o laboratório e permitindo a implementação e validação das pesquisas realizadas. A maioria desses projetos, atualmente, são focados em *IoT*.

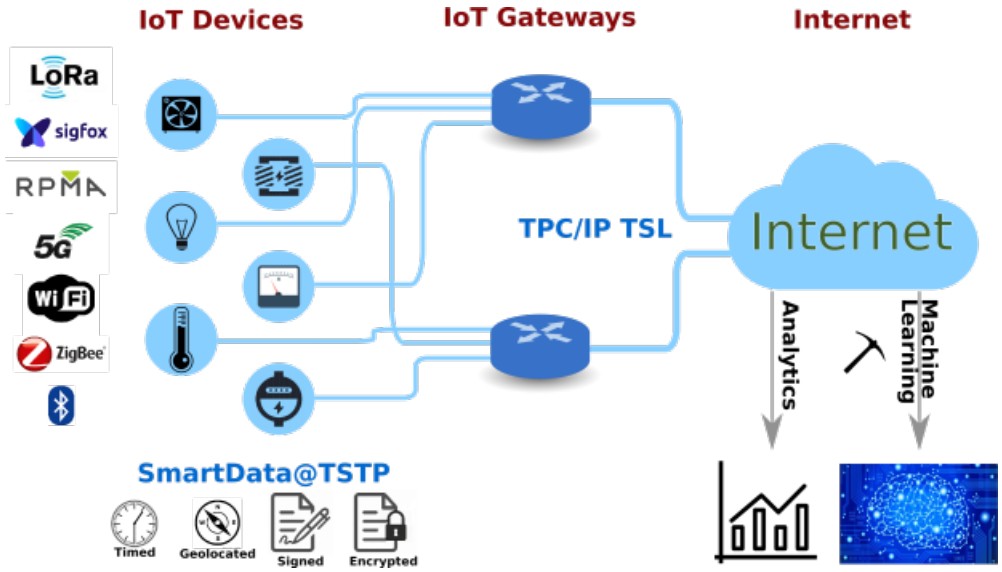
Os projetos *IoT* necessitam de uma infraestrutura com diversos componentes se comunicando e trabalhando coordenadamente para entregar serviços (conforme descrito na seção 4.3); rede de sensores e atuadores *self-aware* conectados à nuvem (internet) através de *gateways IP* seguros; uma *API* para mediar essa conexão com servidores de *Big Data* e com IA (partes da nuvem); e aplicativos *Web* para apresentação de dados, compõe essa infraestrutura.

Para implementar tudo isso, o LISHA utiliza sua plataforma de desenvolvimento de soluções *IoT* - ou simplesmente plataforma *IoT*. Essa é inserida num processo de desenvolvimento com as etapas de planejamento, design e implementação, testes, gerenciamento de falhas, instalação e monitoramento, manutenção e validação, para o desenvolver os projetos *IoT* captados. Esses conceitos, bem como exemplos de projetos recentes, são aprofundados nas subseções a seguir.

2.1.1 Plataforma *IoT*

O LISHA dispõe de uma plataforma de desenvolvimento de soluções *IoT* que fornece *frameworks* para o desenvolvimento de hardware e software, da rede de sensores e atuadores, e soluções de armazenamento, interface, análise e apresentação de dados na nuvem, padronizando e facilitando o desenvolvimento e a manutenção desses projetos. A figura 3 resume a arquitetura de uma solução *IoT* desenvolvida com base na plataforma *IoT* do LISHA.

Figura 3 – Arquitetura da solução *IoT* através da Plataforma *IoT* do LISHA



Fonte: LISHA

Na plataforma *IoT*, o desenvolvimento dos sistemas embarcados que compõe a rede de sensores/atuadores e os *gateways*, é feito majoritariamente através das plataformas *EPOS* [1] e *EPOSMote* [3]. O *EPOS* fornece o *framework* para o desenvolvimento do *firmware*. Neste, o desenvolvedor utiliza a interface *SmartData* para criar a aplicação dos nodos e dos gateways de forma abstrata, sem lidar diretamente com *drivers* e protocolos de comunicação. O *SmartData*, por sua vez, define e lida com a interação com os sensores/atuadores, o modelo de dados geral e a comunicação entre os nodos e com a nuvem. Já o *EPOSMote*, fornece a plataforma de hardware para implementar os sistemas embarcados; com placas de expansão (*shields*) complementando suas funcionalidades e expandindo as capacidades de sensoriamento e atuação, os meios de alimentação energética e os meios de comunicação.

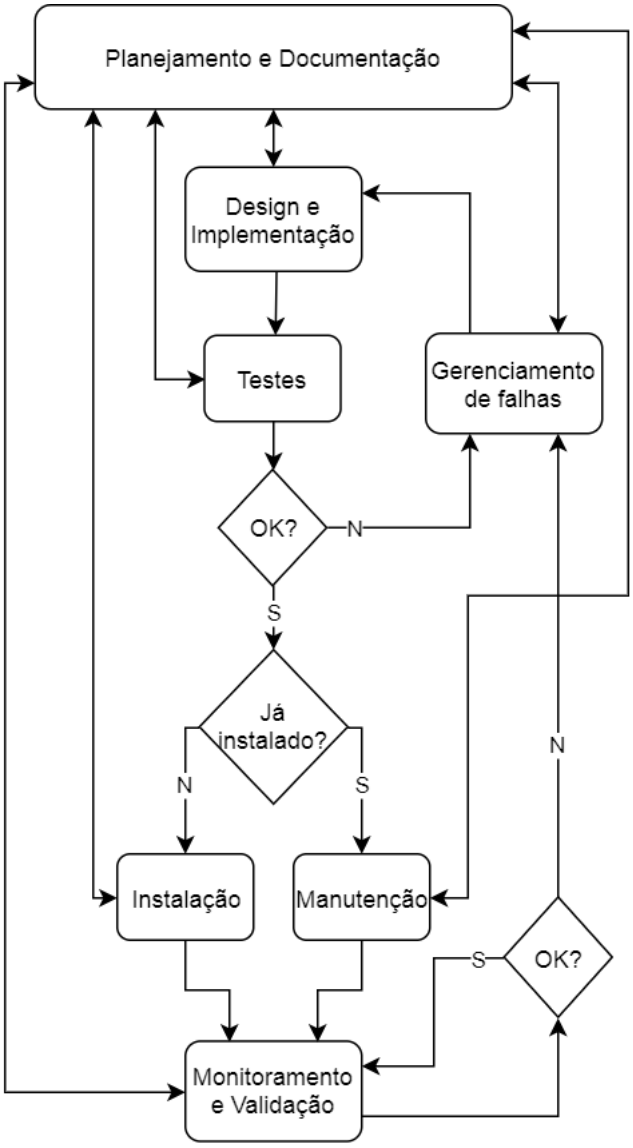
Além disso, a plataforma também fornece uma solução em nuvem para armazenar, analisar e apresentar os dados dos dispositivos *IoT*. Um servidor *HTTPS* (*Hypertext Transfer Protocol Secure*) abre conexões seguras com clientes, como *gateways* e aplicações Web e Mobile, e interage com os mesmos através de uma *HTTP API*, armazenando ou lendo informações de um banco de dados. Esse servidor aceita ainda a criação de *scripts*

para a análise de dados, possibilitando a utilização de IA. Por fim, a apresentação dos dados é feita pela ferramenta *Grafana*, que acessa o banco de dados e, a partir disso, produz gráficos com os resultados obtidos.

2.1.2 Processo de desenvolvimento

Para a realização dos projetos, os professores coordenadores fazem o papel de gerentes: coletam requisitos dos clientes, definem prazos gerais e delegam tarefas. Os mestrandos, doutorandos, estagiários e bolsistas implementam, instalam e dão manutenção à maioria dos projetos. Nesse contexto, o processo de desenvolvimento de projetos *IoT* no LISHA pode ser resumido pelo fluxograma da figura 4.

Figura 4 – Fluxograma do processo de desenvolvimento de projetos *IoT* no LISHA



Fonte: Autor

O processo começa com a chegada de especificações de um projeto *IoT* (informações

básicas, requisitos, prazos, etc) à equipe delegada para desenvolvê-lo. Essa, por sua vez, monta um planejamento inicial contendo: especificação e divisão de tarefas; definição de marcos (*milestones*), como protótipos, por exemplo; calendário de prazos; entre outros. Atualmente, não há uma metodologia de desenvolvimento e planejamento de projetos definida (metodologias ágeis como o *Scrum*, por exemplo), cada equipe define sua metodologia de trabalho. Há, porém, consenso em documentar tudo na *Wiki* (site do LISHA baseado no gerenciador de conteúdo *TikiWiki*) e versionar todo código com *Apache Subversion* (*SVN*) no repositório remoto do LISHA. Para o planejamento e acompanhamento de projetos, é comum a utilização de ferramentas como *Trello* ou *Web2Project*.

Em seguida ocorre a etapa de Design e Implementação, onde a infraestrutura e os serviços da solução *IoT* são modelados e implementados, tomando como base as especificações do projeto. Na modelagem, utilizam-se: mapas e plantas para modelar a topologia da rede de sensores e atuadores; lista de materiais para descrever as estações que abrigarão os dispositivos; e *UML* (*Unified Modeling Language*), automatos, fluxogramas, entre outros, para modelar os serviços. O design, então, é implementado através da plataforma *IoT* do LISHA, na forma de sistemas embarcados e seus *firmwares*, *scripts* para análise de dados na nuvem e *dashboards* de monitoramento e apresentação dos dados.

Na etapa de testes, os componentes desenvolvidos ou em desenvolvimento passam por rotinas de testes unitários e integrados e testes em laboratório e em campo, para encontrar falhas de design e implementação. Caso comportamentos indesejados sejam identificados, passa-se para a etapa de gerenciamento de falhas. Caso passe nos testes e ainda não esteja instalado, vai para a etapa de instalação. Caso já tenha sido instalado, significa que veio da etapa de gerenciamento de falhas, então vai para a etapa de Manutenção.

Na etapa de instalação, as estações que abrigarão os dispositivos são construídas e instaladas nos locais do modelo da infraestrutura, provendo-as de uma fonte de energia (rede elétrica com bateria de *back-up*, bateria alimentada por painel solar, etc). Sistemas embarcados, antenas, sensores e atuadores são instalados e conectados nessas estações.

A etapa de manutenção consiste em atualizar o *firmware* dos dispositivos, trocar dispositivos, realizar reparos mecânicos e elétricos, recalibrar sensores, dentro outros. Posteriormente a essa etapa, passa-se para a de Monitoramento e Validação.

Na etapa de Monitoramento e Validação, o funcionamento das soluções *IoT* instaladas é monitorado através do *Grafana*. Caso detecte-se alguma anomalia, vai para o Gerenciamento de falhas. Após algum tempo de monitoramento sem a detecção de anomalias, considera-se a solução como validada.

A etapa de gerenciamento de falhas, de um ponto de vista de processo, recebe como entrada os sintomas observados de um problema e produz na saída a detecção das causas raiz e sugestões de tratamento para as mesmas. O processo interno para tal, consiste de

uma extensa coleta e organização de dados pertinentes e de uma, também, extensa análise de causa-efeito. Esse processo, caso realizado com afinco, buscando-se sistematicamente as causas raiz, denomina-se *Root Cause Analysis*.

2.1.3 Projetos *IoT* recentes

Nessa seção, tomou-se como exemplo alguns projetos *IoT* em atividade no LISHA para exemplificar os diferentes domínios de aplicação nos quais as soluções e, consequentemente, pesquisas do laboratório se encontram.

2.1.3.1 *SSB*

O *Smart Solar Building* (*SSB*), inaugurado em 2015 no *Sapiens Park* sob o esforço dos grupos Fotovoltaica, LISHA e Instituto de Eletrônica de Potência, é um laboratório voltado para a realização de experimentos nas áreas de sustentabilidade, energias renováveis e automação. Para tal, o LISHA fornece sua plataforma *IoT* para o monitoramento e controle de alguns processos cotidianos, como luminosidade e temperatura das salas, consumo de água e consumo de energia, bem como o monitoramento da geração de energia fotovoltaica. A figura 5 mostra o *SSB*.

Figura 5 – *Smart Solar Building*

Fonte: Fotovoltaica

2.1.3.2 *eBus*

O *eBus* (figura 6) é um projeto coordenado pelo Prof. Ricardo Rüther em parceria com as empresas WEG, Marcopolo, Mercedes, Eletra e com o Ministério da Ciência, Tecnologia e Inovação, para a criação de um ônibus elétrico alimentado por energia solar que realiza o transporte entre o Campus Trindade da UFSC e o Sapiens Parque de forma limpa, confortável e produtiva. O *LISHA* utiliza sua plataforma de *IoT* para monitorar e disponibilizar dados como a localização do ônibus, corrente, tensão e estado de carga do banco de baterias, velocidade e aceleração, estado dos componentes (portas abertas, modo carga, etc), falhas, dentre outros. A aquisição desses dados (com exceção da localização, que é adquirida através de um módulo GPS) é feita lendo-se e interpretando os dados do barramento *CAN* (*Controller Area Network*). Esses dados são úteis para pesquisa, manutenção do ônibus e, no caso da localização, para a comunidade.

Figura 6 – *eBus*

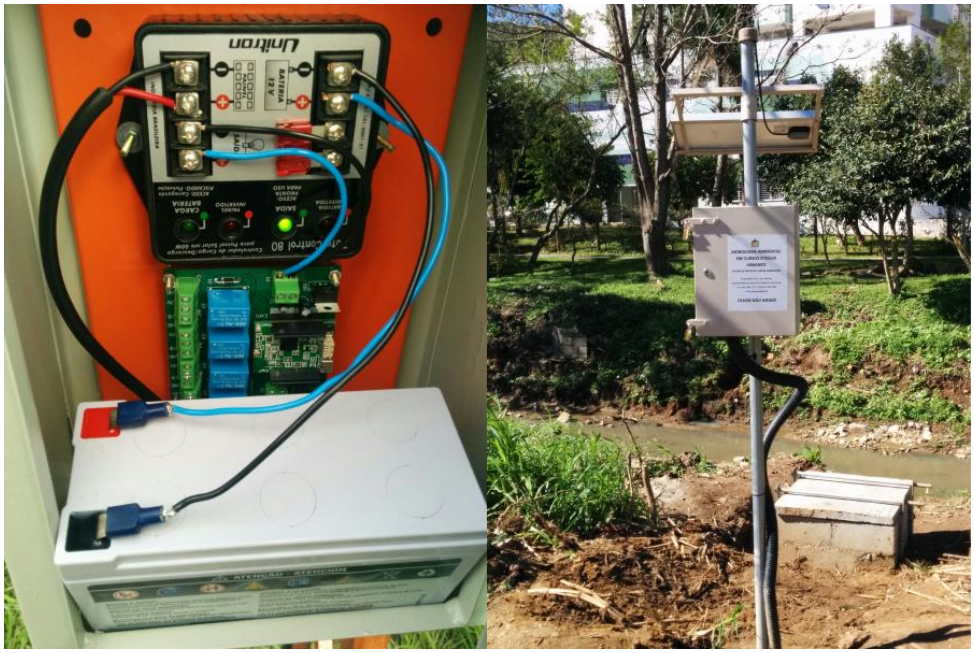
Fonte: Fotovoltaica

2.1.3.3 PRAD

O Projeto de Recuperação da Qualidade da Água dos Córregos do Campus Reitor João David Ferreira Lima (PRAD) [6], visa diagnosticar e recuperar remotamente a qualidade dos cursos da Microbacia do Campus Trindade da UFSC, também conhecida como Bacia do Rio do Meio (parte componente da superfície vertente e da rede de drenagem da Bacia Hidrográfica do Itacorubi), através da realização de um Projeto de Recuperação de Áreas Degradadas. Além disso, visa também estabelecer um monitoramento remoto constante do nível e da turbidez dos córregos e do consumo de água da região do Hospital Universitário da UFSC. Para tal, um dos objetivos do projeto foi a instalação de estações hidrológicas (similares à representada pela figura 7) que realizem o monitoramento remoto de precipitações pluviométricas, vazão de cursos d'água e turbidez da água utilizando tecnologia de redes de sensores sem fios de baixa potência. Enquanto o Núcleo de Estudos

da Água é encarregado de realizar o projeto de recuperação, o LISHA é encarregado de fornecer a solução para o monitoramento remoto - sua plataforma *IoT*.

Figura 7 – Estação de monitoramento hidrológico



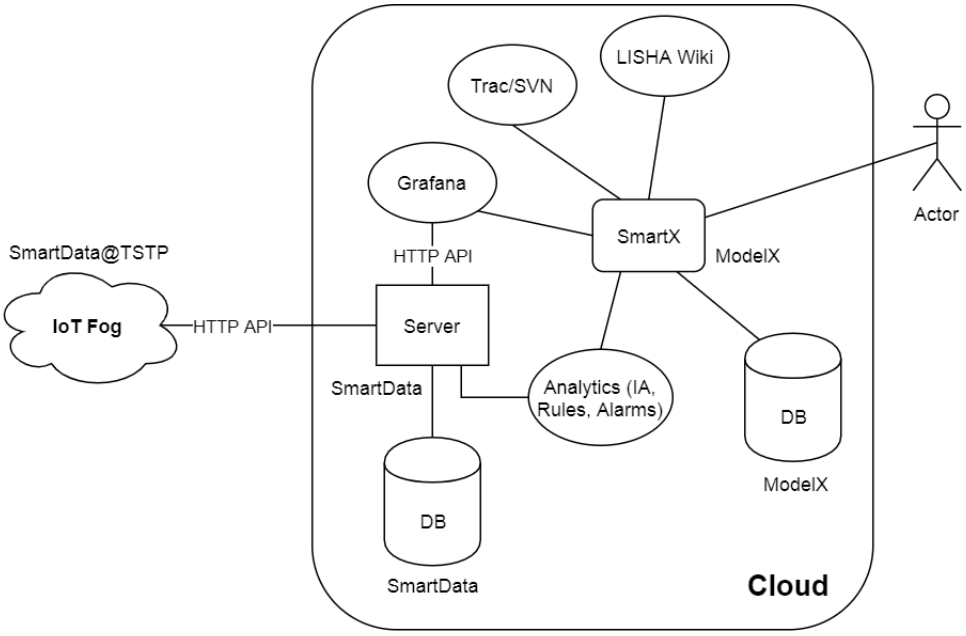
Fonte: LISHA

3 O Projeto

Este capítulo visa prover uma descrição conceitual das atividades desenvolvidas para se atingir os objetivos específicos propostos na seção 1.2 e descrever a metodologia usada.

Um dos objetivos específicos propostos foi a implementação de uma ferramenta de gerenciamento para estruturar a documentação do desenvolvimento de projetos *IoT* (descrito na seção 2.1), integrar as informações espalhadas por outras ferramentas (Grafana, Trac e Site do LISHA) e apresentar todas as informações relevantes num mesmo contexto. Essa ferramenta será implementada como um sistema web, denominado *SmartX*, cujo papel na arquitetura *IoT* do LISHA é representado pela figura 8.

Figura 8 – Arquitetura *IoT* do LISHA com o sistema *SmartX*



Fonte: Autor, 2019

O *SmartData* [4], como modelo de dados, é auto-contido em termos de semântica, localização, temporização e confiança. Isso porque utiliza padrões internacionais de representação de dados como o SI para a unidade, ECEF (*Earth-Centered Earth-Fixed*) para localização, e *POSIX* para tempo (microsegundos desde 1 janeiro de 1970). A abstração provida pelo *SmartData*, por si só, apesar de auto-contida, não representa o sistema *IoT* de forma intuitiva para o usuário (desenvolvedor, gestor, cliente etc).

A origem espacial do dado (ou a assinatura, no caso de um *SmartData* móvel), por exemplo, representa um local de instalação (ou estação) da perspectiva do processo

de desenvolvimento. Esse local possui materiais, dispositivos, pertence a um projeto e cumpre um propósito. Utilizar o *ECEF* do *SmartData* para se referenciar a esses locais, no dia-a-dia, não é nada intuitivo. O que acontece, naturalmente, é a utilização de apelidos para tais locais.

Essa abstração do *SmartData* tem sido feita intuitivamente e de forma não padronizada nas plataformas de documentação e apresentação de dados utilizadas no LISHA; por exemplo, no *Grafana*, através da criação de *Dashboards* de apresentação de dados separados por projetos e estações; na *Wiki* através da criação de páginas para a concentração de informações de projetos e dispositivos; no sistema de versionamento de códigos *SVN* através da estrutura de diretórios e da nomenclatura das aplicações EPOS; e nas ferramentas de planejamento e acompanhamento, como *Trello* e *Web2Projetc*, através de *tags* e da organização de tarefas em projetos, estações e dispositivos.

Portanto, visando estruturar e padronizar essa abstração, será instituído o *ModelX*; um modelo de dados que complementará o *SmartData* com informações relevantes para o processo de desenvolvimento de projetos *IoT*, deixará as informações mais intuitivas e que servirá de base para a implementação do *SmartX*.

Após a implementação do sistema, o mesmo será introduzido ao processo de desenvolvimento de projetos *IoT*, no LISHA, para a realização de um estudo de caso, como prevê o segundo objetivo específico na seção 1.2. Esse estudo terá o intuito de validar o sistema desenvolvido frente aos objetivos do trabalho e criar um banco de dados de sintomas, análises, causas e tratamentos de problemas analisados na etapa de gerenciamento de falhas do processo de desenvolvimento.

3.1 Metodologia

O desenvolvimento do trabalho foi realizado através da modelagem e implementação do sistema *SmartX* e da inserção do mesmo no processo de desenvolvimento de projetos *IoT* para a execução de um estudo de caso do sistema, frente ao desenvolvimento dos projetos *IoT* em andamento no laboratório. Com isso, buscou-se identificar as causas raiz dos problemas que levaram à situação de alta demanda por manutenção, documentando o processo no sistema de forma a gerar um banco de dados de sintomas, análises, causas e tratamentos de problemas analisados; o que facilitará os processos de gerenciamento de falhas futuros. Sendo assim, esse foi um trabalho prospectivo.

O sistema *SmartX* foi modelado através do levantamento de requisitos e da representação de dados e comportamento por diagramas Entidade-Relação (ER) e *UML*. Para implementar o sistema, estudou-se o gerenciador de conteúdo *TikiWiki*. Essa ferramenta foi escolhida por questões de praticidade e agilidade, visto que a mesma já é utilizada para o site do LISHA e, por tanto, já está instalada e configurada. A análise de viabilidade,

documentada na seção 4.6.1, mostrou que a ferramenta é capaz de implementar o sistema modelado.

Para a realização do estudo de caso, o sistema implementado foi introduzido ao processo de desenvolvimento de projetos *IoT* através da utilização do mesmo por alguns desenvolvedores do LISHA. Os dados de projetos, estações e dispositivos (vide figura 10), referentes aos projetos em desenvolvimento (seção 2.1.3), foram adicionados e passaram a ser apresentados pelas visões contextualizadas do sistema. Conforme os acontecimentos, *Tickets* foram adicionados para documentá-los. Ao fim do estudo de caso, formou-se um banco de dados de sintomas, análises, causas e tratamentos de problemas analisados no processo de *Root Cause Analysis* (seção 4.5) da etapa de gerenciamento de falhas. Desse banco de dados, abstraiu-se um esquema de classificação e utilizou-se diagramas para evidenciar o processo de *RCA*, validando o sistema frente à análise e documentação de falhas.

4 Conceitos e Ferramentas utilizados

Neste capítulo serão apresentados os diversos conceitos e ferramentas utilizados no desenvolvimento do trabalho e necessários para entender o contexto desse desenvolvimento.

4.1 Sistemas embarcados

Sistemas embarcados consistem da integração de componentes de *hardware* e *software* em meio a sistemas mecânicos e elétricos, desenvolvidos para cumprir um propósito específico, ao contrário de sistemas computacionais de uso geral, como computadores pessoais [7]. Por ser um sistema dedicado a tarefas específicas, é passível de um design com menos recurso computacional que um sistema de propósito geral, com tamanho reduzido e com menor custo. Essas características possibilitam sua implementação em sistemas que vão desde o domínio doméstico e não crítico, até sistemas com requisitos de tempo real, como sistemas de controle.

4.2 Rede de sensores sem fio

Uma rede de sensores é uma infraestrutura que consiste de elementos de sensoriamento, computação e comunicação, e que permite observar e reagir a eventos e fenômenos de um determinado ambiente [8]. Em diversas aplicações, o ambiente monitorado possuí restrições, como difícil acesso ou local hostil, que inviabilizam a utilização de redes cabeadas, dificultam o acesso humano e requerem tolerância à falhas. Nesse contexto, aplicam-se RSSFs energeticamente auto-suficientes (com baixo consumo energético e alimentação de baterias recarregadas por painéis solares) e tolerantes a falhas. Sistemas embarcados são essenciais na composição das RSSF por serem de baixo custo, terem dimensões reduzidas e permitirem um consumo reduzido de energia, possibilitando a escalonabilidade do sistema, a instalação em ambientes confinados e a redução da necessidade de manutenção, respectivamente.

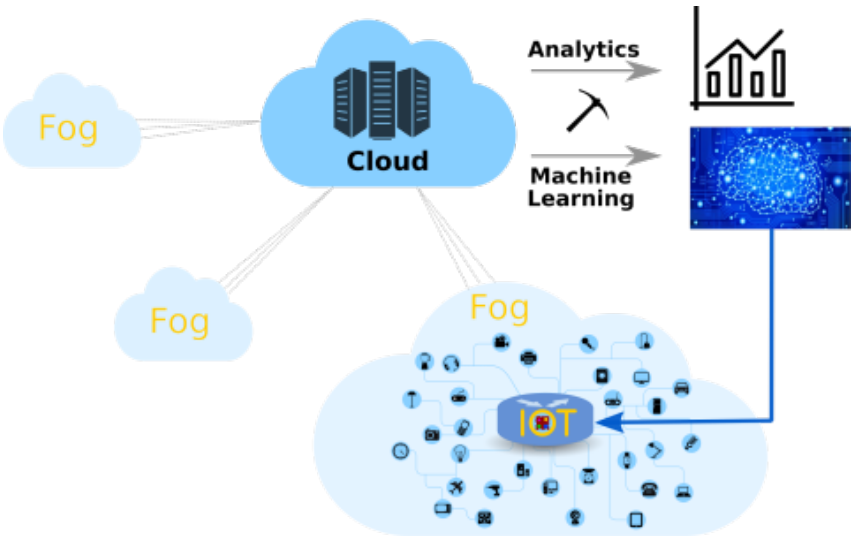
4.3 Internet das coisas

A internet das coisas é composta de uma rede auto-configurável, adaptativa e complexa de "coisas"conectadas entre si e à internet através do uso de protocolos padrão de comunicação. As "coisas"possuem sensores e/ou atuadores que possibilitam o monitoramento e atuação de forma local ou remota. Através de identificadores únicos e do

sensoriamento, é possível coletar informações sobre as "coisas"e configurá-las de qualquer lugar, em qualquer momento, através de qualquer coisa [9].

De um ponto de vista arquitetural, a rede de sensores e atuadores interconectados (*Fogs*) se conectam à "nuvem"(internet) através de *Fog gateways*, que agem como uma barreira do ponto de vista de segurança, *design*, confiança e eficiência, e implementam protocolos padrão de conexão com a internet e *APIs* de interação com servidores. Na nuvem, salvam-se os dados de sensores e de estado dos dispositivos através de servidores e bancos de dados preparados para um grande volume (*Big Data*), e são oferecidas aplicações de análise (IA e *Machine Learning*) e apresentação de dados. A figura 9 representa a arquitetura descrita para os sistemas *IoT*.

Figura 9 – Arquitetura *IoT*



Fonte: LISHA

4.4 Modelagem de software orientada a objetos

A modelagem de software orientada a objetos consiste na construção de modelos que expliquem diversos aspectos de um sistema através de diagramas baseados no paradigma da orientação a objetos. Os aspectos modelados compreendem, basicamente, estruturas ou dados e comportamento.

Para representar tais aspectos, utiliza-se diagramas baseados em UML bem como em modelos ER. O UML consiste de um conjunto de notações gráficas que ajudam a descrever e modelar conceitos tanto dos aspectos estruturais, quanto dos comportamentais, pela orientação a objeto [10]. Por outro lado, modelos ER apresentam uma notação gráfica mais simplificada e são voltados à representação de dados.

A modelagem de dados é representada pelos modelos conceitual, lógico e físico. O modelo conceitual provê uma abstração de alto nível para o domínio, através da representação em entidades, atributos e relações, utilizando-se diagramas ER ou diagramas UML de classes. O modelo lógico introduz conceitos mais específicos de banco de dados, como o conceito de chaves, tipo de dados e tabelas, e também pode ser representado pelo diagrama de classes (com algumas alterações para identificar as chaves primárias e secundárias). Por fim, o modelo físico inclui informações altamente ligadas ao banco de dados e é representada por código *Structured Query Language*.

A modelagem comportamental visa especificar aspectos dinâmicos do sistema, como o fluxo de mensagens em informações através do tempo, as mudanças de estado e a interação sistema-usuário. Para esses exemplos citados, em específico, utiliza-se diagramas de sequência, de estado e de casos de uso, respectivamente.

4.5 *Root Cause Analysis*

A análise de causas raiz, do inglês *Root cause analysis* (*RCA*), é um processo sistemático de investigação de problemas que busca encontrar e prevenir suas causas raiz [11]. Esses problemas são identificados nos sistemas investigados através de sintomas sutis ou enfáticos e utilizados como entrada do processo de *RCA*. Esse processo, por sua vez, consiste do uso de informações do sistema, como características específicas e seu estado antes, durante e após o sintoma, que junto ao uso de ferramentas (*5 Whys, FMEA, Fishbone*, etc) e do raciocínio, permitem criar uma análise de causa-efeito de acontecimentos. Ao fim desse processo, são identificadas as possíveis causas raiz e, baseado nessas, são feitas sugestões para evitar o acontecimento desses eventos e/ou controlar seus efeitos. Sendo assim, o processo de *RCA* pode ser resumido nas seguintes etapas:

a) Identificação do problema;

b) Identificação das causas;

c) Identificação dos relacionamentos de causa-efeito;

d) Identificação das causas raiz;

e) Sugestões de melhora para solucionar, mitigar, ou evitar as causas raiz do problema.

4.6 Ferramentas

4.6.1 *TikiWiki CMS Groupware*

A TikiWiki [12] é um sistema de gerenciamento de conteúdo (*CMS*, do inglês *Content Management System*) baseado em páginas *Wiki* e voltado a aplicações de desenvolvimento colaborativo (*Groupware*). Um *CMS* é uma aplicação de *software* que fornece soluções prontas para o gerenciamento de diversos aspectos da criação de conteúdo colaborativo (gerenciamento de páginas, buscas, gerenciamento de arquivos, controle de versionamento, etc) por usuários finais. Uma *Wiki* é um site colaborativo de compartilhamento e gerenciamento de conhecimento, onde usuários criam e editam páginas através de interfaces *Web*, utilizando uma linguagem simples (*Markup*), sem a necessidade de conhecimento em *Web Design*.

As páginas *Wiki* são tratadas, geralmente, como um quadro em branco, com usuários gerenciando seu conteúdo diretamente nas páginas, sem uma estrutura definida. Sistemas baseados em banco de dados, por sua vez, são altamente estruturados e permitem a padronização da apresentação de conteúdo, porém tornam o gerenciamento e o trabalho colaborativo mais rígido e especializado. O meio termo entre esses dois tipos de aplicações *Web* são as *Wiki* estruturadas. Esse conceito une a facilidade do gerenciamento de conteúdo e trabalho colaborativo das *Wikis* com a padronização e geração de visões, em trechos específicos, de sistemas baseados em banco de dados.

Para a implementação de *Wikis* estruturadas na *TikiWiki*, utiliza-se a ferramenta *Trackers* unida a um sistema de *plugins* e *templates*. A ferramenta *Trackers* consiste de um bando de dados baseado em *MySQL* acessível e gerenciável por interfaces gráficas prontas, APIs e *plugins*, que fornecem a implementação das operações *CRUD* (*Create, Read, Update* e *Delete*). O sistema de *templates* permite a criação de estruturas e visões reutilizáveis, bem como uma maior customização de páginas *wiki*, através da programação de arquivos com o *framework* PHP *Smarty*.

A *TikiWiki* ainda implementa uma série de *plugins* que fornecem soluções prontas para diversas funcionalidades. A lista a seguir apresenta alguns dos *plugins* que foram importantes no desenvolvimento do projeto.

- Plugin Tracker: cria um formulário para a inserção dos dados referentes aos atributos de um *Tracker*, com um botão "Salvar"para submeter a criação do *tracker item*. O *plugin* verifica o *Field Type* dos atributos para criar os campos adequados à inserção dos dados do item (áreas de texto, listas *dropdown*, caixas de marcação, inserção de arquivos, etc).
- Plugin TrackerList: implementa a realização de buscas nos *Trackers* pelos seus atributos e a apresentação do resultado, que, por padrão, é representado em formato

de tabela.

- Plugin List: possibilita a busca por diversas estruturas da *TikiWki* (inclusive *tracker items*), permitindo também a aplicação de filtros por essas estruturas. Possui uma opção que permite lidar com o resultado da busca através de *templates* que, por sua vez, podem ser codificados utilizando-se o *smarty*.

- Plugin Map: provê uma *API* com diversos parâmetros para controlar aparência, dimensões e interação com o usuário. Provê também a ferramenta *search layers*, que aceita parâmetros para realizar uma busca filtrada de *tracker items* e para controlar e customizar a apresentação dos mesmos, utilizando, por exemplo, ícones customizados salvos em atributos de *tracker items*. Internamente, o *plugin* traduz sua *API* para a *API* da biblioteca *OpenLayers*, que, por sua vez, implementa de fato a criação do mapa (em *HTML*, *CSS*, *JavaScript*, etc).

4.6.2 *Grafana*

O *Grafana* [13] é uma ferramenta *Web*, *open source*, de análise e monitoramento que, através de *plugins*, faz buscas em bancos de dados e exibe os resultados em gráficos, mapas, tabelas, listas, etc. Além disso, permite que usuários criem e editem quadros contendo essas apresentações, a partir de uma interface gráfica de fácil uso.

O LISHA utiliza o *Grafana* como solução de monitoramento de dados de sua plataforma *IoT*, descrita na seção 2.1.1.

4.6.3 *Subversion* e *Trac*

O *Subversion* (SVN) [14] é um sistema de controle de versões, com código aberto, desenvolvido pela comunidade *Apache Software Foundation*. Esse sistema gerencia a edição colaborativa de arquivos e diretórios por usuário através da internet, salvando o histórico de alteração dos mesmos através do tempo. Isso permite analisar esse histórico de mudanças e recuperar versões antigas. Na arquitetura do sistema, de um lado há o programa do repositório que contém todos os dados versionados fazendo o papel de servidor, de outro há o programa de clientes que gerencia reflexões locais de partes desses dados versionados.

O *Trac* é uma ferramenta *Web* de código aberto de *bug tracking* e gerenciamento de projetos de software que possui integração com a maioria dos sistemas de versionamento (incluindo *SVN*). Possui visões do conteúdo versionado, do histórico de alterações, da lista de *bugs* e do planejamento.

O LISHA utiliza repositórios *SVN* para versionar todo código produzido no laboratório e o *Trac* como ferramenta *Web* para a apresentação do histórico de alterações e *bug tracking*.

5 Atividades desenvolvidas

O desenvolvimento do trabalho foi dividido em duas etapas principais: Desenvolvimento do sistema *SmartX* e Introdução do sistema no processo de desenvolvimento de projetos *IoT* para a realização de um estudo de caso. As seções a seguir aprofundam a composição dessas etapas.

5.1 Desenvolvimento do sistema *SmartX*

O *SmartX* foi o sistema *Web* de gerenciamento de projetos *IoT* proposto para estruturar e sistematizar a documentação e a apresentação do processo de desenvolvimento de tais projetos. Para isso, integra as plataformas e ferramentas utilizadas no laboratório, guia uma documentação mais completa das etapas e proporcionando análises mais eficazes, através de visões com um contexto amplo do processo de desenvolvimento, estruturadas por um modelo de dados que facilita o entendimento (*ModelX*).

O desenvolvimento deste sistema foi realizado de forma cíclica entre as etapas de modelagem e implementação (re-modelando, adicionando funcionalidades e corrigindo *bugs* quando necessário), sempre buscando um Produto Mínimo ou protótipo funcional a cada ciclo. Na modelagem, foram levantados o modelos de dados (*ModelX*), requisitos funcionais e modelos de comportamento, utilizando-se *UML* e diagramas. Na implementação, utilizou-se a ferramenta de gerenciamento de conteúdos *TikiWiki* para implementar o sistema. Essas etapas são aprofundadas a seguir.

5.1.1 Modelagem

O capitulo 3 proveu um modelo conceitual inicial do sistema, através da contextualização do mesmo dentro da arquitetura dos sistemas *IoT* (figura 8) e de uma análise inicial para a criação do *ModelX*.

Partindo disso, analisou-se o linguajar utilizado na documentação e na comunicação entre os desenvolvedores para abstrair os conceitos do sistema *IoT* e do seu processo de desenvolvimento. Com isso, definiu-se as seguintes entidades para compor o *ModelX*: Projeto, Estação, Dispositivo, *Ticket* e Usuário.

5.1.1.1 Modelagem conceitual de dados

As entidades Estação e Dispositivo representam o local de instalação de um nodo da *IoT Fog*, na arquitetura de sistemas *IoT* (figura 9); e a entidade Projeto, o propósito desses nodos e dos dados produzidos por eles. A abstração desses nodos, como dito no

capítulo 3, não era bem representado pelo *SmartData* por si só, da perspectiva do processo de desenvolvimento, por não ser intuitivo e por não contemplar certas informações sobre o local de instalação e os dispositivos e materiais que o compõem. Sendo assim, as entidades Estação, Dispositivo e Projeto complementam o *SmartData*, em tal sentido.

A entidade *Ticket* representa um evento que, por sua vez, simboliza uma falha, uma constatação (análise de um acontecimento) ou uma tarefa (um evento programado para ocorrer). Essa entidade dá a característica de linha temporal na documentação do desenvolvimento dos projetos *IoT* e, portanto, é a peça chave do *SmartX*. Já a entidade Usuário, representa o ator na arquitetura do *SmartX* (figura 8), que, por conseguinte, retrata um desenvolvedor, gestor ou cliente do sistema *IoT*. A descrição das entidades do *ModelX* é resumida na tabela 1.

Tabela 1 – Descrição das entidades do modelo de dados *ModelX*

Entidade	Descrição
Projeto	Representa uma abstração comum para um conjunto de objetivos e requisitos. Essa entidade é importante como agregador das outras entidades em análises de planejamento.
Estação	Representa um conjunto de materiais e dispositivos ligados fisicamente ou conceitualmente, coexistindo no espaço-tempo (móvel ou estático) sobre o mesmo contexto. Do ponto de vista da RSSF do sistema *IoT*, representa um nodo.
Dispositivo	Representa um componente físico conceitualmente importante para o sistema e complexo o suficiente para necessitar de uma descrição mais elaborada, com *links* de *datasheets*, tutoriais de uso e *tickets* relacionados.
Ticket	Representa um evento no tempo, que aconteceu (falhas ou *logs*) ou que vai acontecer (tarefa). Essa é a entidade chave para a documentação das etapas do processo de desenvolvimento de um projeto *IoT* como um todo; mas especialmente do processo de gerenciamento de falhas, que permitirá a criação um banco de dados de sintomas, análises e causas de problemas.
Usuário	Representa um usuário do sistema *IoT* e, consequentemente, do sistema *SmartX* (desenvolvedor, gestor ou cliente).

Unindo-se a análise utilizada para definir as entidades ao levantamento de informações sobre o desenvolvimento de projetos *IoT* ausentes na documentação, mas que são relevantes para o trabalho, foram definidos os atributos do modelo conceitual dessas entidades. Por hora foram ignoradas chaves primárias, secundárias e atributos específicos da implementação. Esses conceitos fazem parte do modelo lógico e precisam de mais contexto para serem explicadas, e portanto serão comentadas mais à frente na descrição das atividades. O resultado pode ser verificado na tabela 2.

O relacionamento entre essas entidades ocorre da seguinte forma: Estações possuem Dispositivos e fazem parte de um Projeto; *Tickets* se referem a Projetos, Estações e/ou a Dispositivos; Usuários fazem parte de Projetos (como desenvolvedor, gestor ou cliente), são

Tabela 2 – Atributos do modelo conceitual do *ModelX*

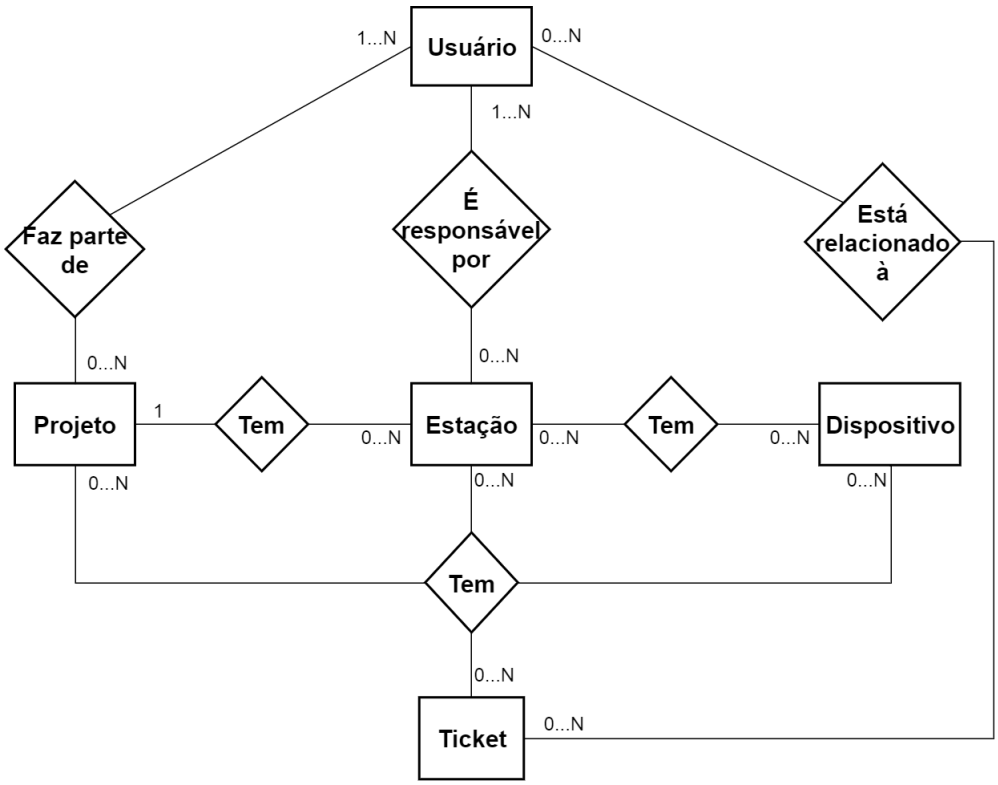
Entidade	**Atributo**	**Descrição**
Projeto	Nome	Nome oficial do projeto.
	Código	Código para referência interna.
	Status	Ativo ou finalizado.
Estação	Código	Código para referência interna.
	Apelido	Nome comumente utilizado entre os desenvolvedores para se referir à estação.
	Status	OK, Apresentando problemas ou desativado.
	Conteúdo	Lista de materiais que não são representados pela entidade Dispositivos. Ex: Caixa de proteção, bateria, etc.
	Descrição	Uma descrição sobre a estação em texto livre.
	AppLink	*Link* para a versão do *firmware* instalado no sistema embarcado da estação.
Dispositivo	Código	Código para referência interna.
	Modelo	Modelo do dispositivo.
	Marca	Marca do dispositivo.
	Versão	Versão do dispositivo.
	Categoria	Categoria do dispositivo dentro do sistema *IoT* (no LISHA, dividido em *Transducer*, *EPOSMote* e Outros).
	Descrição	Tipo do dispositivo, dentro de sua categoria (sensor de temperatura da categoria *Transducer*, por exemplo).
Ticket	Timestamp_Criação	*Timestamp* em que o *Ticket* foi criado.
	Timestamp_Referente	*Timestamp* ao qual o *Ticket* faz referência.
	Título	Título do *Ticket*.
	Tipo	Tarefa, *Log* ou *Issue*.
	Status	a fazer/a resolver, pendente, feito/resolvido
	Descrição	Descrição, em texto livre, sobre o evento.
Usuário	Nome	Nome do usuário
	E-mail	E-mail do usuário.
	Senha	Senha de login do usuário.
	Admin	Booleano que define se o usuário tem permissão de administrador.
	Tipo	Define a função do usuário para o sistema; se é desenvolvedor, gestor ou cliente.

responsáveis por Estações (desenvolvedores) e são relacionados a *Tickets*. Essas relações são expressas no diagrama ER da figura 10.

5.1.1.2 Modelagem comportamental

Para modelar o comportamento do sistema, realizou-se um levantamento de requisitos funcionais e, em seguida, utilizou-se as funcionalidades definidas nesses requisitos para a criação de um diagrama *UML* de casos de uso. Os requisitos funcionais são exigências que

Figura 10 – Diagrama Entidade-Relacionamento do *ModelX*



Fonte: Autor, 2019

o sistema precisa implementar para funcionar da maneira esperada, ou seja, para atacar os problemas que motivaram esse trabalho e ser considerado suficiente. Para defini-los, tomou-se como base a motivação e os objetivos do trabalho, unido ao recém definido modelo conceitual. O resultado é expresso na tabela 3.

Casos de uso descrevem como ocorre o uso de funcionalidades do sistema. Um diagrama *UML* de casos de uso, por sua vez, descreve como os atores do sistema (Usuários sistêmicos ou humanos) interagem com essas funcionalidades e como as mesmas se relacionam entre si. Sendo assim, o diagrama de casos de uso que representa o sistema modelado é retratado na figura 11.

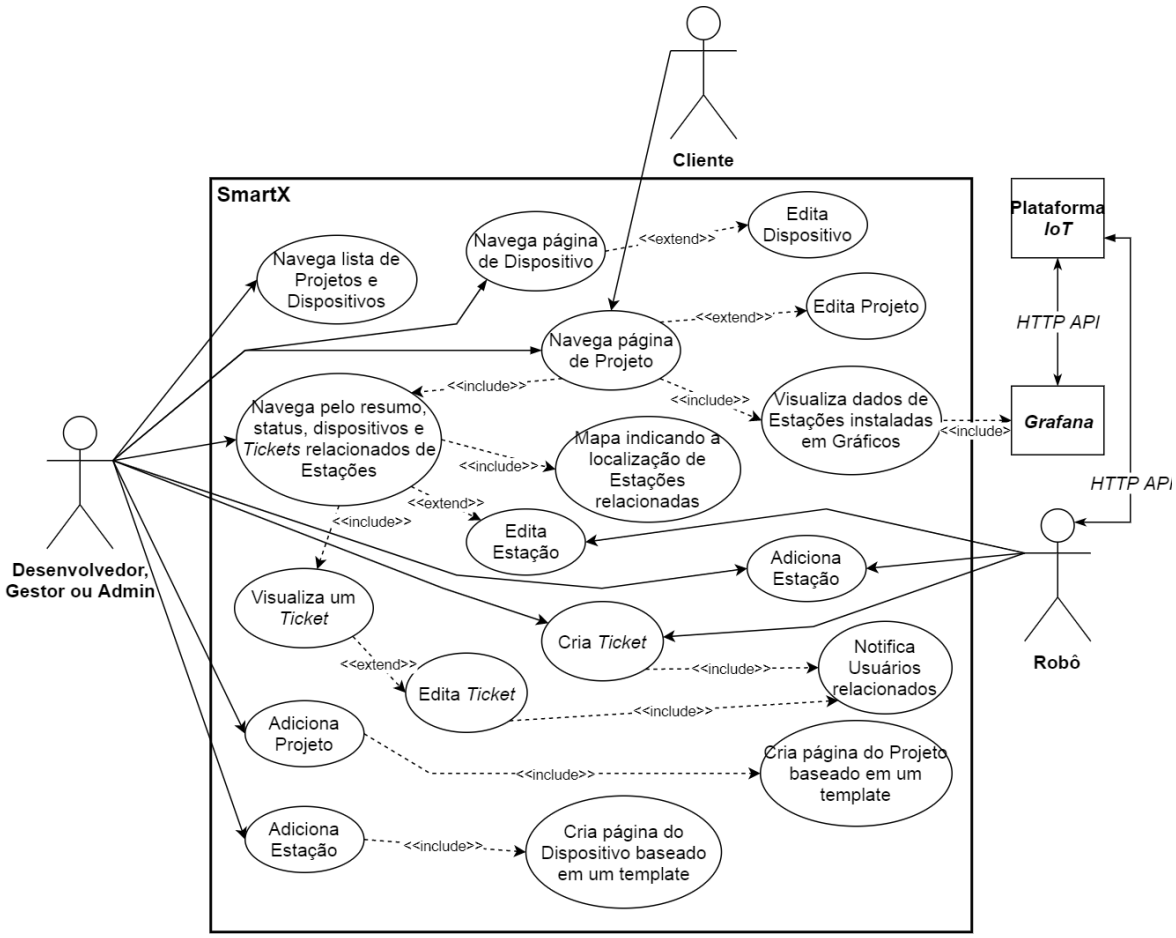
A funcionalidade de RF2 foi retratada no diagrama de casos de uso já apresentando o vínculo com o *Grafana*, sendo que a forma como esse vínculo foi implementado será explicado no capítulo seguinte. Além disso, adicionou-se um ator sistêmico ao modelo do *SmartX*, que visa apresentar a possibilidade de automatização de algumas funcionalidades, como a adição e edição de Estações e a criação de *Tickets*, através da análise dos dados sensoreados por um *script*.

Tabela 3 – Requisitos funcionais

Código	Título	Descrição
RF1.1	Listar a localização das Estações em um mapa	O sistema deve indicar a localização de estações em um mapa através de ícones.
RF1.2	Apresentar resumo da Estação ao clicar no seu ícone no mapa	O sistema deve permitir que o usuário clique num ícone do mapa de estações para apresentar um resumo de suas informações.
RF2	Apresentar dados de Estações instaladas em gráficos e afins	Gráficos para apresentação temporal dos dados sensoreados.
RF3	Listar *Tickets*	O sistema deve apresentar listas de Tarefas, *Logs* e *Issues*, filtradas por Projetos, Estações, Dispositivos e Usuários.
RF4	Apresentar as informações de um Projeto em uma página	O sistema deve apresentar todas as informações de um projeto em uma página.
RF5	Apresentar as informações de um Dispositivo em uma página	O sistema deve apresentar todas as informações de um Dispositivo em uma página.
RF6	Apresentar o *status* de desenvolvimento de todas as estações do LISHA em uma página	O sistema deve apresentar, em uma página, o resumo e o status de todas as estações.
RF7	Gerenciamento de *Tickets* por usuários	O sistema deve permitir que usuários adicionem e editem *Tickets*.
RF8	Gerenciamento de Projetos por usuários desenvolvedores, gestores ou admins	O sistema deve permitir que usuários dos tipos desenvolvedor ou gestor (ou ainda um admin) adicionem e editem Projetos.
RF9	Gerenciamento de Estações por usuários desenvolvedores, gestores ou admins	O sistema deve permitir que usuários dos tipos desenvolvedor ou gestor (ou ainda um admin) adicionem e editem Estações.
RF10	Gerenciamento de Dispositivos por usuários desenvolvedores, gestores ou admins	O sistema deve permitir que usuários dos tipos desenvolvedor ou gestor (ou ainda um admin) adicionem e editem *Dispositivos*.
RF11	Gerenciamento de Usuários por admins	O sistema deve permitir que admins adicionem e editem Usuários.
RF12	Cadastrar novos usuários	O sistema deve permitir que novos usuários se cadastrem.
RF13	*Login* de Usuários	O sistema deve permitir Usuários façam *login*, para acessar conteúdo personalizado e restrito.

Fonte: Autor

Figura 11 – Diagrama de casos de uso do *SmartX*



Fonte: Autor

5.1.2 Implementação na ferramenta *TikiWiki*

Como já exposto na metodologia do trabalho (seção 3.1), a ferramenta *TikiWiki CMS* foi escolhida para implementar o sistema *SmartX* por conta da praticidade, tanto pelo fato de que essa ferramenta já estava instalada, configurada e sendo usada nos sites do LISHA, quanto por implementar *templates* e *plugins* que facilitam a criação e gerenciamento de conteúdo. Na seção 4.6.1, descreveu-se o funcionamento e os mecanismos da *TikiWiki*; mostrando que era capaz de implementar o sistema planejado na seção 3.

Sendo assim, o *SmartX* foi implementado no mesmo ambiente que o site do LISHA, onde os requisitos referentes a Usuários já estavam implementados (itens RF11, RF12 e RF13, da tabela 3) e já haviam diversas páginas de documentação (não padronizada) de Dispositivos e Projetos.

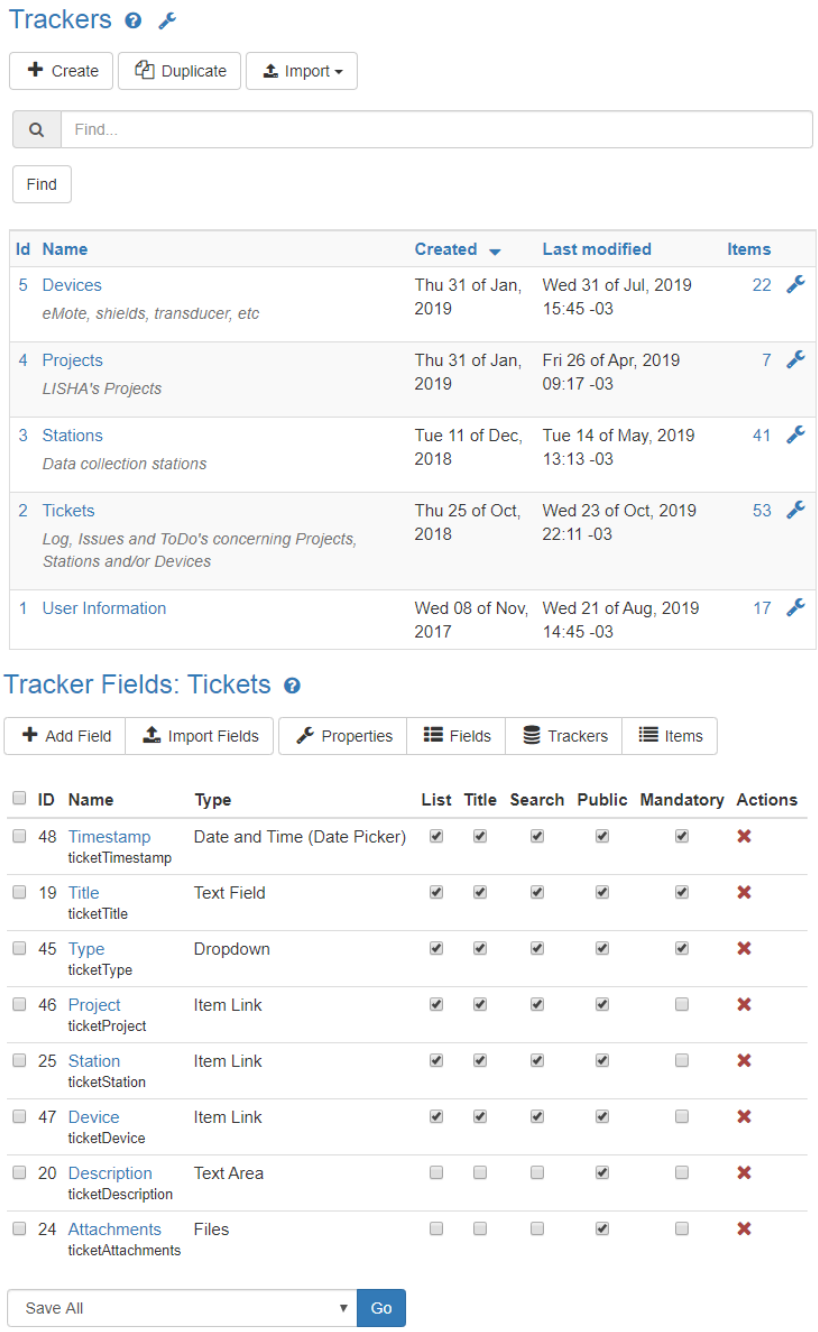
5.1.2.1 Implementação do Banco de Dados com Trackers

A ferramenta *Tracker*, como dito na seção 4.6.1, implementa um banco de dados relacional acessível e gerenciável por interfaces gráficas prontas, APIs e *plugins*, que

fornecem a implementação das operações *CRUD*. Para implementar o banco de dados baseado no *ModelX*, utilizou-se essa ferramenta.

Considerando que a tabela Usuário já estava implementada e que os atributos identificador, *timestamp* de criação, usuário criador e *status* são, por padrão, criados e atualizados automaticamente, as tabelas restantes do *ModelX* (Projeto, Estação, Dispositivo e Ticket) e seus atributos (tabela 2) foram criados utilizando-se a interface gráfica de *Trackers*, como mostra as figura 12.

Figura 12 – Interface gráfica para o gerenciamento de *Trackers*



Fonte: *SmartX*

Na criação dos atributos, lhes são concedidos *Field types*, que são definições da *TikiWiki* que determinam como os atributos serão tratados em buscas e formulários de inserção e edição. Para a definição das chaves estrangeiras, por exemplo, tais atributos são classificados como o tipo *Item Link*. Esse tipo de atributo suporta a adição de múltiplos valores, como um *array*. Com isso, foi possível implementar as relações N para N sem a criação de tabelas auxiliares, como normalmente teria de ser feito em bancos de dados tradicionais.

Somado aos atributos definidos na tabela 2, foram adicionados atributos específicos para esta implementação. À entidade Projeto, incluíram-se os atributos *wiki* e *dashboard*. O primeiro forma base para a realização do requisito RF4, fazendo o link entre a página *wiki* de um Projeto e seu *Tracker item*, e o segundo dá base à realização do requisito RF2, fazendo o link entre um dashboard do *Grafana* referente a um Projeto e seu *Tracker item*.

À entidade Dispositivo, adicionou-se o atributo *wiki*, visando atacar o requisito RF5 da mesma forma feita para a entidade Projeto. À entidade *Ticket*, acrescentou-se um atributo de anexos: esse salva o *id* de arquivos do sistema que, portanto, dá base para a adição de anexos junto a *Tickets*. Por fim, também acrescentou-se o atributo ícone à entidade Estação, possibilitando a customização do ícone da estação que será apresentada no mapa. A figura 13 representa o modelo lógico de dados, em formato de diagrama UML de classes, resultante da implementação do banco de dados com *Trackers*.

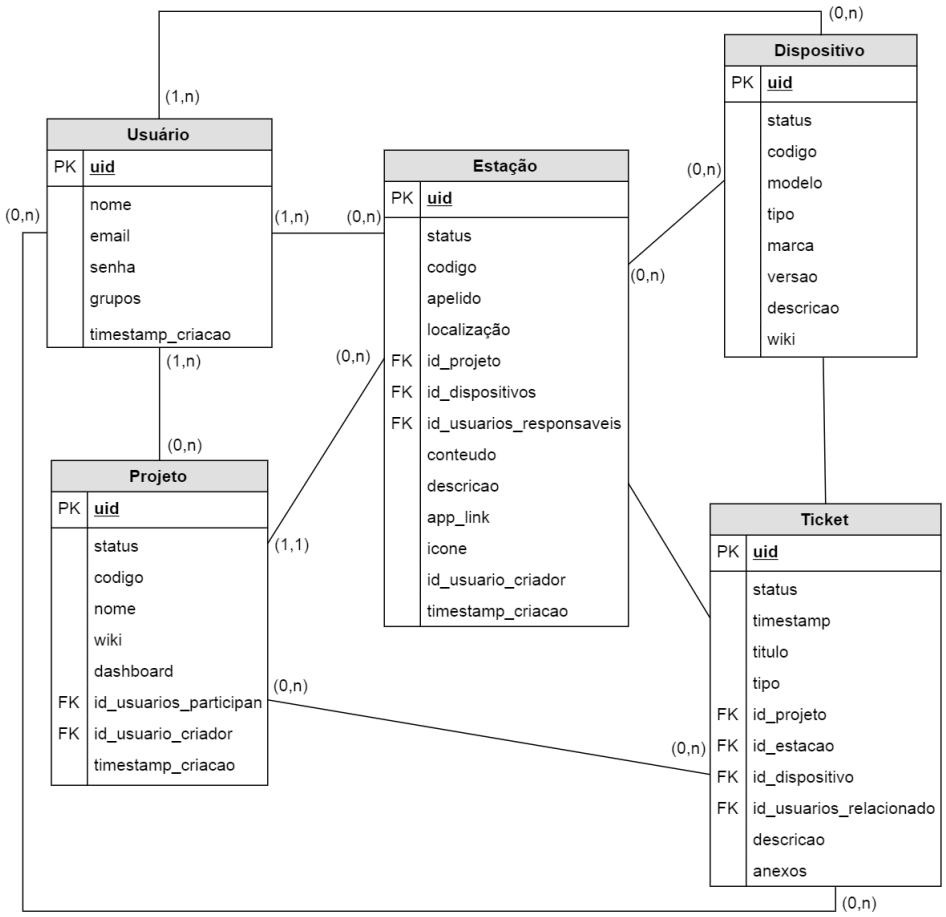
5.1.2.2 Gerenciamento de Projetos, Estação, Dispositivos e Tickets

A interface gráfica de gerenciamento de *Trackers* (figura 12), utilizada para criar o banco de dados na seção anterior, implementa as ferramentas de *CRUD* necessárias para gerenciar os dados. Entretanto, para realizar os requisitos de RF7 a RF10, notou-se a necessidade de permitir a criação e edição de Projetos, Estações, Dispositivos e *Tickets* (*tracker items*) por usuários através de páginas *wiki* customizáveis e com um contexto mais amplo.

Para implementar a criação de *tracker items* em páginas *wiki*, utilizou-se o *Plugin Tracker*. Esse cria um formulário para a inserção dos dados referentes aos atributos de um *Tracker*, com um botão "Salvar"para submeter a criação do *tracker item*. O *plugin* verifica o *Field Type* dos atributos para criar os campos adequados à inserção dos dados do item (áreas de texto, listas *dropdown*, caixas de marcação, inserção de arquivos, etc).

Para implementar a edição de *tracker items*, partindo de páginas *wiki*, utilizou-se a mesma página *template* de edição de *tracker items* utilizada pela interface gráfica de gerenciamento de *Trackers*. Essa interface tem o item de edição definido por uma variável de *URL* (*Uniform Resource Locator*). Com isso, a página *template* utiliza o *Plugin Tracker* para criar o formulário com os itens preenchidos com os atributos atuais do *tracker item* e que pode ser editado, e adiciona também um botão "Salvar"para submeter a edição.

Figura 13 – Modelo lógico de dados do *SmartX*



Fonte: Autor

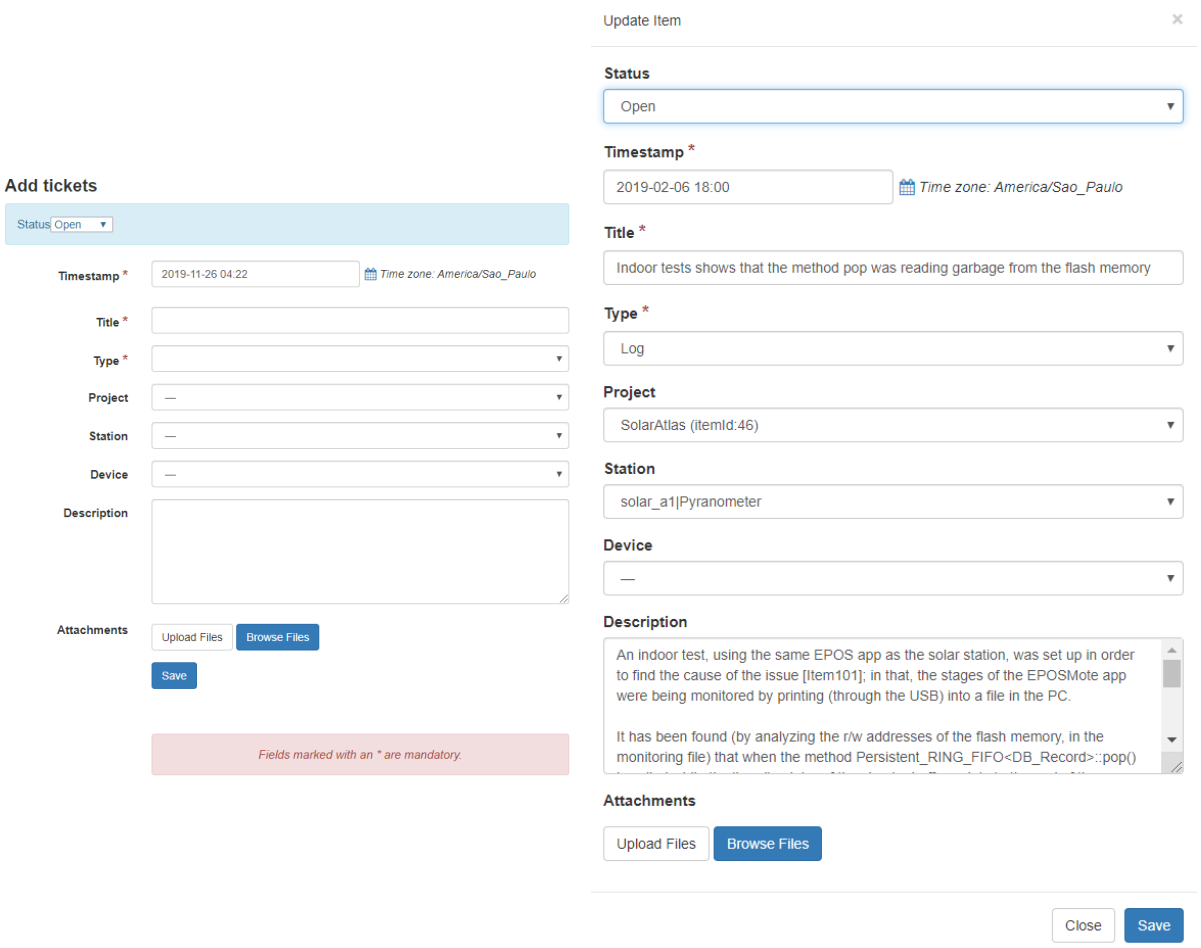
Os formulários resultantes para a criação e edição de um *tracker item* (no exemplo, um *Ticket*) são representandos, respectivamente da esquerda para direita, na figura 14.

5.1.2.3 Listagem de *Tickets*

Para implementar o requisito RF3 foi necessário possibilitar a listagem de *Tickets* em páginas *wiki*. Considerando uma quantidade de itens possivelmente extensa e a dispensabilidade de um alto nível de customização da apresentação dos itens, a lista de *Tickets* foi implementada no formato de tabela. Para tal, utilizou-se o *Plugin Tracker List*, que implementa a realização de buscas nos *Trackers* pelos seus atributos e a apresentação do resultado, que, por padrão, é representado em formato de tabela. Isso possibilita a filtragem de *Tickets* por Projetos, Estações, Dispositivos e Usuários, para que o resultado possa ser posto em um contexto específico e adequado; cumprindo o requisito RF3.

O *Plugin Tracker List* também possibilita certa customização na apresentação dos itens. Para a listagem dos *Tickets*, foram realizadas a seguintes customizações: apresentação dos atributos do item em formato de *link*, apontando para a página *template* de edição de *tracker items* (como na seção anterior); seleção dos atributos que resumem o *Ticket*

Figura 14 – Criação e edição de *tracker items* via *Plugin Tracker*



Fonte: *SmartX*

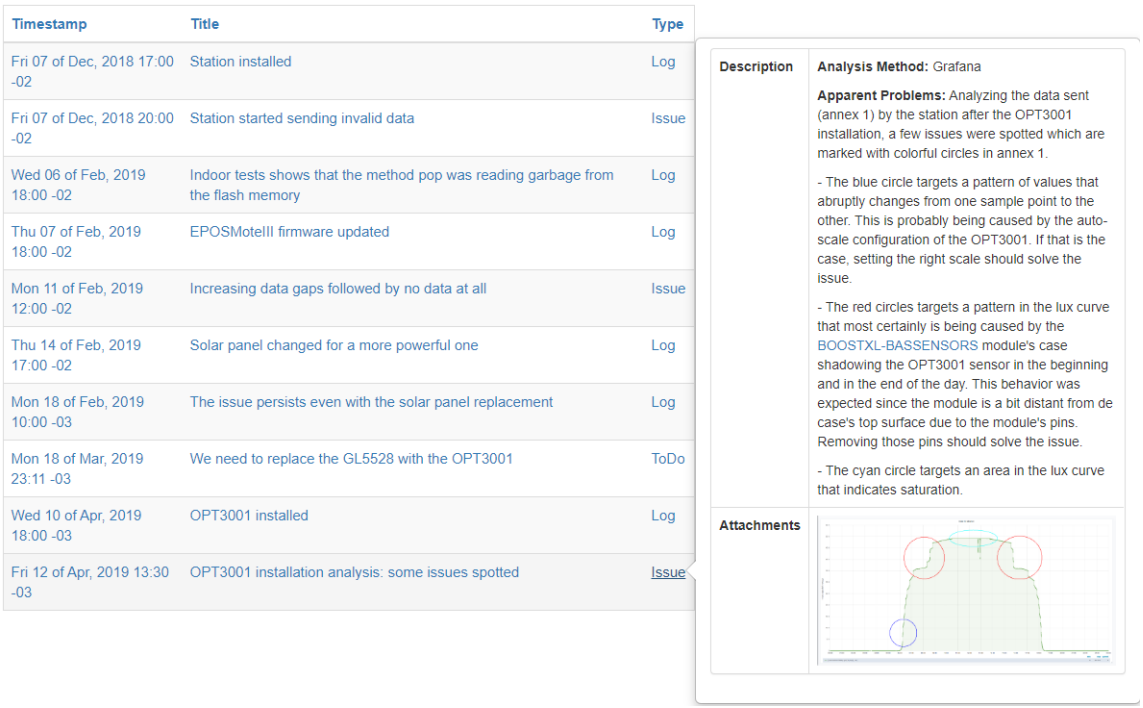
(*timestamp*, título, e outros dependendo do contexto) para aparecerem na tabela de apresentação dos mesmos; apresentação de outros atributos importantes (descrição e anexos) em uma janela flutuante, acionada pelo *hover* do cursor (passagem por cima) no item da tabela de *Tickets*. O resultado da implementação desse recurso está representado na figura 15 por um exemplo de listagem de *Tickets* filtrado por uma Estação.

5.1.2.4 Listagem de Estações

Como parte da realização do requisito RF6, foi necessário implementar uma lista estruturada de resumo das estações. A complexidade dessa listagem, entretanto, se difere da realizada anteriormente, utilizando apenas o *Plugin Tracker List*, pois, para desenvolver uma apresentação completa, intuitiva e bem estruturada, teriam de ser realizadas sub-buscas dentro de buscas. A estratégia de implementação dessa listagem seria similar à seguinte, em pseudo-código:

```
def busca(entidade, filtro):
    % define a busca por itens de uma entidade,
```

Figura 15 – Exemplo de apresentação de *Tickets* filtrado por uma Estação



Timestamp	Title	Type
Fri 07 of Dec, 2018 17:00 -02	Station installed	Log
Fri 07 of Dec, 2018 20:00 -02	Station started sending invalid data	Issue
Wed 06 of Feb, 2019 18:00 -02	Indoor tests shows that the method pop was reading garbage from the flash memory	Log
Thu 07 of Feb, 2019 18:00 -02	EPOSMoteIII firmware updated	Log
Mon 11 of Feb, 2019 12:00 -02	Increasing data gaps followed by no data at all	Issue
Thu 14 of Feb, 2019 17:00 -02	Solar panel changed for a more powerful one	Log
Mon 18 of Feb, 2019 10:00 -03	The issue persists even with the solar panel replacement	Log
Mon 18 of Mar, 2019 23:11 -03	We need to replace the GL5528 with the OPT3001	ToDo
Wed 10 of Apr, 2019 18:00 -03	OPT3001 installed	Log
Fri 12 of Apr, 2019 13:30 -03	OPT3001 installation analysis: some issues spotted	Issue

Fonte:*SmartX*

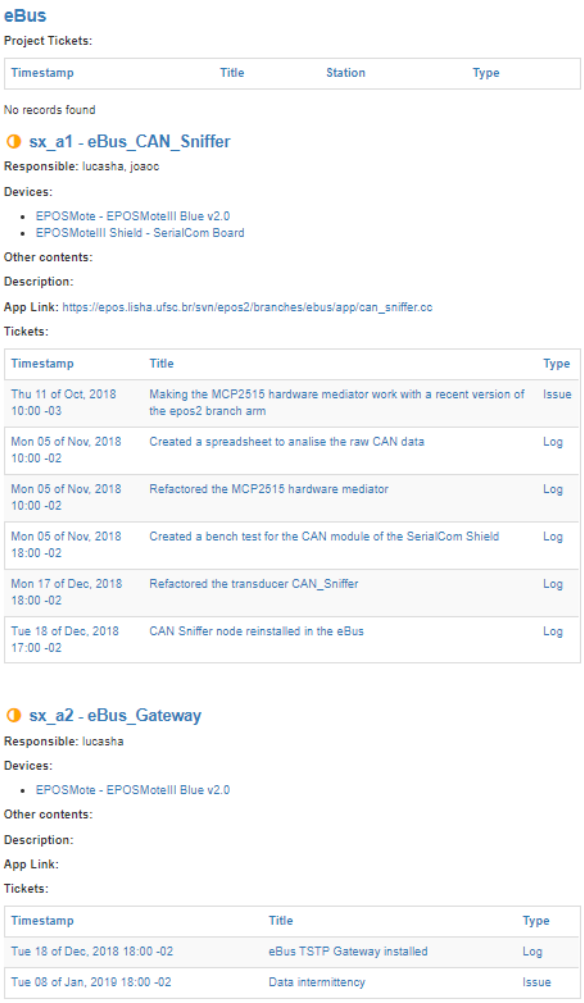
```
    % aplicando filtros
def h(nivel,texto):
    % cria um cabecalho HTML
def trackerlist(entidade,filtro):
    % cria uma lista de itens atraves do plugin trackerlist
projetos = busca("Projeto",...)
for projeto in projetos:
    h(...,projeto.nome)
    trackerlist("Ticket",id_projeto=projeto.id)
    estacoes = busca("Estacao",id_projeto=projeto.id)
    for estacao in estacoes:
        % cria as estruturas HTML para apresentar os
        % atributos da estacao
        trackerlist("Ticket",id_estacao=estacao.id)
```

Para implementar, de fato, a estratégia representada pelo pseudo-código acima, foi utilizado o *Plugin List*. Esse possibilita a busca por diversas estruturas da *TikiWki* (inclusive *tracker itens*), permitindo também a aplicação de filtros por essas estruturas. Traçando-se um paralelo com o pseudo-código, o *Plugin List* representa a função "busca".

Entretanto, parece não ser possível codificar a parte do *loop*, nem a das sub-buscas, direto em páginas *wiki*. Para contornar isso, utilizou-se a função de *templates* do *Plugin*

List. Por fim, foi possível implementar a ideia do pseudo-código com o *Plugin List* tratado por um *template smarty*. O resultado é retratado na figura 16 por um trecho da listagem de estações, referente ao Projeto *eBus*.

Figura 16 – Exemplo de apresentação de resumo das estações



Fonte: *SmartX*

5.1.2.5 Mapa de Estações

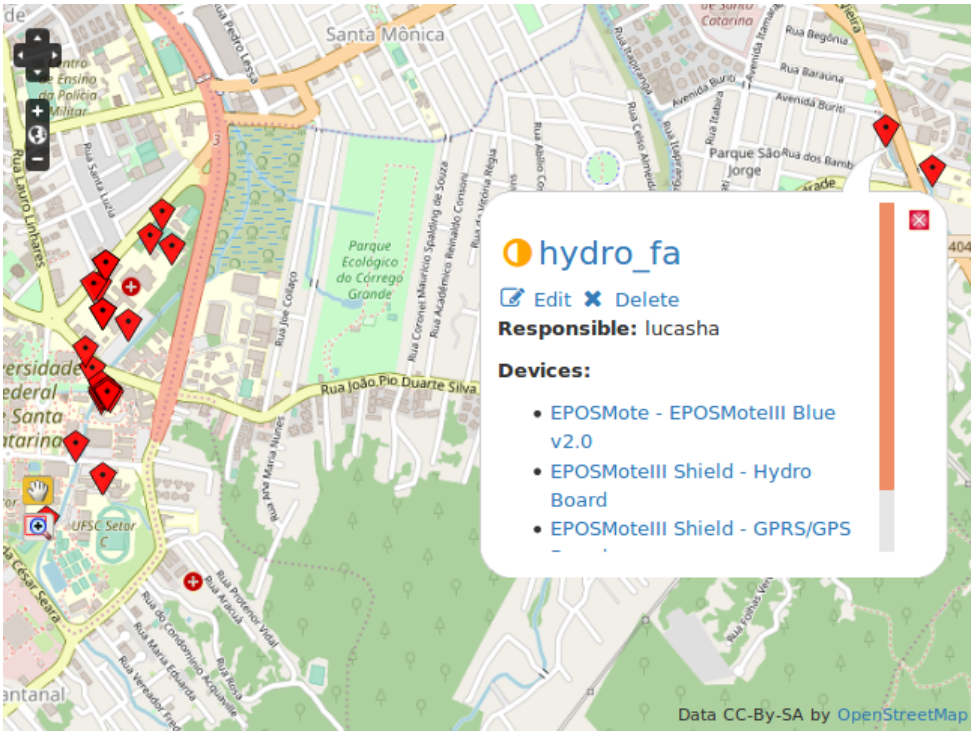
Para a implementação dos requisitos RF1.1 e RF1.2, foi utilizado o *Plugin Map* para embutir mapas, em páginas *wiki*, que buscam *tracker items* e os representam como ícones. O *plugin* provê uma *API* com diversos parâmetros para controlar aparência, dimensões e interação com o usuário; provê, também, a ferramenta *search layers* que aceita parâmetros para realizar uma busca filtrada de *tracker items* e para controlar e customizar a apresentação dos mesmos, utilizando, por exemplo, ícones customizados salvos em atributos de *tracker items*, como no caso do *Tracker* Estação (cujo atributo ícone é representado na figura 13). Internamente, o *plugin* traduz sua *API* para a *API*

da biblioteca *OpenLayers*, que, por sua vez, implementa de fato a criação do mapa (em *HTML*, *CSS*, *JavaScript*, etc).

Uma das características que o *Plugin Map* implementa é a ação de criar uma janela flutuante ao clicar com o cursor (ou outra ação, como o *hover*, visto que esta é configurável) em um ícone apresentado no mapa. Essa janela, por padrão, apresenta o *tracker item* com um de seus atributos como título (passado como parâmetro na *API* do *plugin*), um ícone que representa o *status* do item e links para gerenciar o item (editar ou deletar). Entretanto, é possível customizar essa janela alterando-se o arquivo *template infobox.tpl* no diretório da *TikiWiki*.

Sendo assim, para implementar RF1.2, especificamente, ativou-se a característica de criação da janela flutuante com o clique do cursor num ícone do mapa, representando a localização de uma estação, e alterou-se o arquivo *infobox.tpl* para customizar a apresentação das estação. O resultado da implementação dos requisitos RF1.1 e RF1.2 é apresentado no mapa de Estações da figura 17.

Figura 17 – Exemplo de apresentação das estações em mapa



Fonte: *SmartX*

5.1.2.6 Apresentação pelo *Grafana* de dados sensoreados

A apresentação de dados sensoreados de estações instaladas, referentes a projetos *IoT* do LISHA, é realizada, atualmente, através da interface *Web* de análise de dados da plataforma de *IoT* do laboratório (seção 2.1.1), implementada pela ferramenta *Grafana*.

Essa ferramenta, como dito na seção 4.6.2, permite apresentar os dados em formato de gráficos e afins e a separá-los em *dashboards*, para organizar e criar contexto. Por padrão, no processo de desenvolvimento de projetos *IoT* do LISHA (seção 2.1.2), cria-se um *dashboard* por Projeto para representá-lo. Podem haver outros, mas utiliza-se um como o principal.

Deste modo, para implementar a apresentação de tais dados no *SmartX* e, portanto, cumprir o requisito RF2, utilizou-se o *Plugin HTML* para embutir *dashboards* do *Grafana* em páginas *wiki*, através da criação uma estrutura *HTML iframe*. Para estruturar a apresentação desses *dashboards*, visando relacioná-los com a entidade Projetos, o atributo *dashboard* foi adicionado ao *Tracker* Projeto. Nesse, salva-se a *URL* da página do *Grafana* (da plataforma *IoT* do LISHA) que se refere ao *dashboard* do Projeto em questão. Através dessa *URL*, são passados parâmetros como o período de tempo a ser mostrado nos gráficos, o período de atualização dos gráficos, o modo de visualização da página (modo TV, kioski, painéis auto-dimensionáveis), dentro outros. Com isso, o *Grafana* configura a página em questão.

A figura 18 mostra o resultado da implementação do requisito RF2, representando um *dashboard* de um projeto embutido em uma página *wiki* através da criação de um *iframe* pelo *Plugin HTML*. A *URL* do *dashboard* utilizada no *iframe*, foi adquirida selecionando-se o atributo na busca por um Projeto com o *Plugin List*.

5.1.2.7 Visão de Projeto

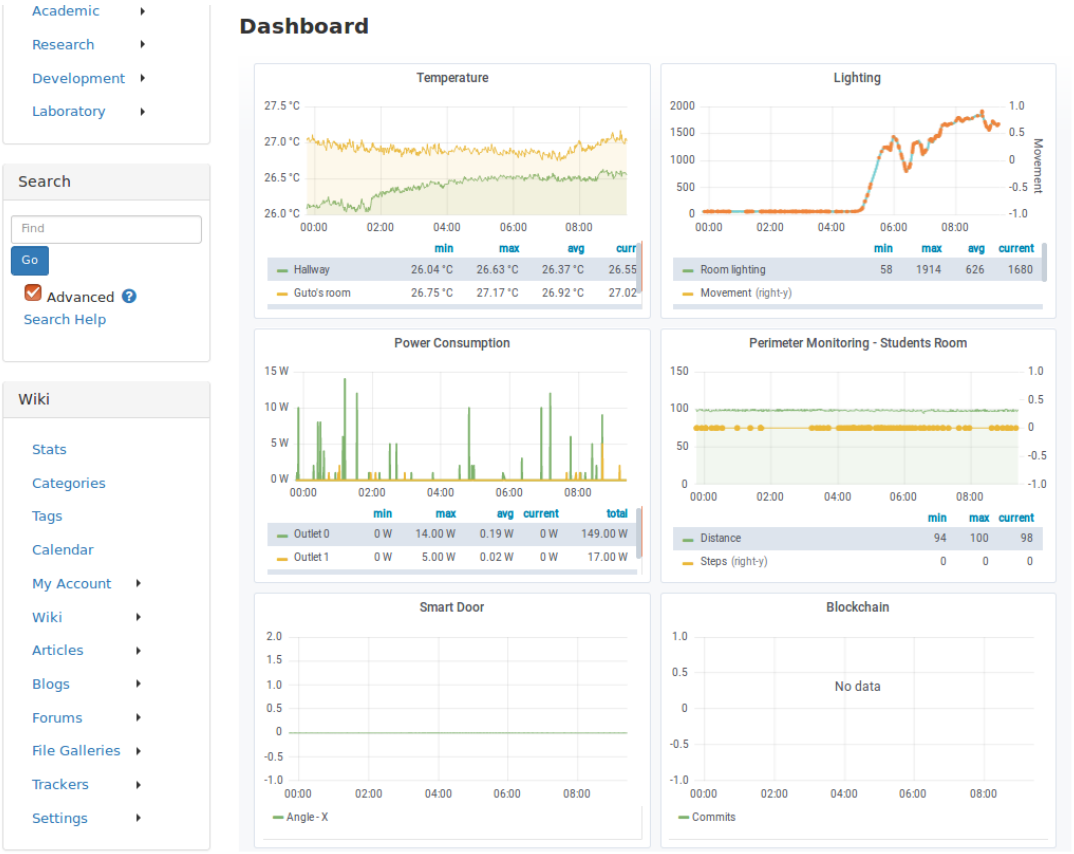
Até o momento, citou-se apenas a implementação de características pontuais do sistema. A criação de visões que unam essas características é o que cria o contexto para analisar, planejar e documentar o processo de desenvolvimento de projetos *IoT* do LISHA. Pensando nisso que os requisitos RF4, RF5 e RF6 foram criados.

Nesse sentido, para implementar o requisito RF4 e, portanto, criar visões com o contexto referente à entidade Projetos, instituiu-se que junto à criação de cada *tracker item* de Projeto, cria-se uma página *wiki* estruturada, referente a esse, e salva-se o link dessa página no atributo *wiki* de tal *tracker item*. Essas páginas têm o intuito de fornecer conteúdo estruturado (buscado do banco de dados) junto à possibilidade de se documentar conteúdo não estruturado. A criação de conteúdo não estruturado é mais intuitiva da perspectiva de trabalho colaborativo (a própria essência do sistema *wiki*) e possibilita a documentação de aspectos não modelados.

Sendo assim, definiu-se que o conteúdo estruturado da visão de projeto seria composto das seguintes *features*:

- *Dashboard* do *Grafana*, referente ao projeto; *feature* representada pelo requisito funcionais RF2 da tabela 3, definida na seção 5.1.2.6 e exemplificado pela figura 18);

Figura 18 – Exemplo do *dashboard* do *Grafana* de um Projeto, embutido em uma página *wiki*



Fonte: *SmartX*

- Mapa de estações, filtrado pelo projeto em questão; *feature* representada pelos requisitos funcionais RF1.1 e RF1.2 da tabela 3, definida na seção 5.1.2.5 e exemplificada pela figura 17);

- Listagem das estações, filtrado pelo projeto em questão; *feature* definida na seção 5.1.2.4 e exemplificada pela figura 16);

- Formulários para inserção de Estações e *Tickets*; *feature* representada nos requisitos funcional RF7 e RF9 da tabela 3, definida na seção 5.1.2.2 e exemplificada pela figura 14);

Para automatizar a criação dessas páginas, utilizou-se uma configuração do *Field Type* do atributo *wiki* (*Page Selector*), que faz com que junto à criação de um *tracker item*, seja criada uma página *wiki* baseada em um *template*. Nesse *template*, por sua vez, utilizou-se o *Plugin List* e um *template smarty* para criar toda a estrutura, citada acima, de forma dinâmica, baseando-se no *id* da página wiki referente ao Projeto.

5.1.2.8 Visão de Dispositivo

Para implementar a visão de Dispositivos proposta pelo requisito RF5, aplicou-se a mesma tática utilizada na implementação da visão de Projetos, descrita na seção anterior: a criação automática de uma página *wiki* mista (parte com conteúdo dinâmico, parte com conteúdo para documentação "normal"de *wikis*), baseada num *template* (parte dinâmica definida por buscas no banco de dados feitas pelo *Plugin List* e tratadas por um *template smarty*), junto à criação de um *tracker item* de Dispositivo. No caso da visão de Dispositivos, entretanto, possibilitou-se a representação de mais de um Dispositivo por página *wiki*. Em contraste com a visão de Projetos, tornou-se opcional a criação automática da página *wiki* no momento da criação do Dispositivo, permitindo que o preenchimento do atributo *wiki* com uma página existente.

Sendo assim, definiu-se que a parte estruturada (dinâmica) da visão de Dispositivos seria composta pelas seguintes *features*:

- Listagem de *Tickets*, filtrados pelos dispositivos em questão; *feature* representada pelo requisito funcional RF3 da tabela 3, definida na seção 5.1.2.3 e exemplificada pela figura 15);

- Formulários para inserção de *Tickets*; *feature* representada no requisito funcional RF7 da tabela 3, definida na seção 5.1.2.2 e exemplificada pela figura 14);

O resultado da implementação da visão de Dispositivos está representado, na figura 19, pela página *wiki* resultante da criação dos Dispositivos *Hydro Board* e *Hydro Board ESP* v2.0 (e com a posterior criação de *Tickets* referentes a esses Dispositivos).

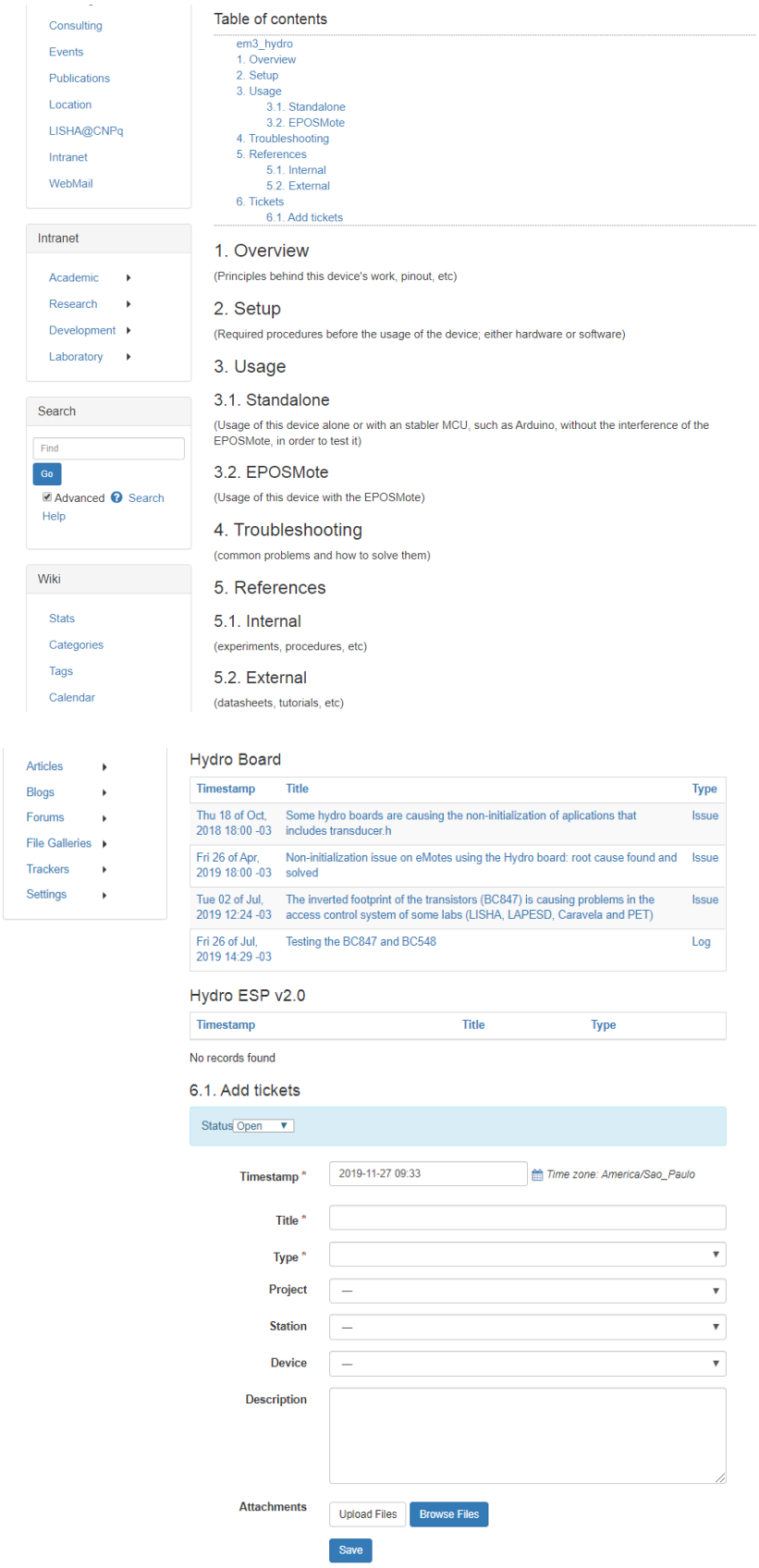
5.1.2.9 Visão de contexto geral

No requisito RF6, da tabela 3, solicitou-se uma visão geral do processo de desenvolvimento de projetos *IoT* (visão de contexto geral), contendo a localização, o estado e o resumo (materiais, dispositivos instalados, etc) de todas as estações cadastradas, assim como a lista de seus *Tickets* relacionados. Para implementar essas características, criou-se uma página *wiki* dinâmica utilizando a maioria das *features* definidas nas seções anteriores:

- Mapa com todas as estações cadastradas; *feature* representada pelos requisitos funcionais RF1.1 e RF1.2 da tabela 3, definida na seção 5.1.2.5 e exemplificada pela figura 17);

- Listagem de todas as estações; *feature* definida na seção 5.1.2.4 e exemplificada pela figura 16);

Figura 19 – Exemplo de página *wiki* dinâmica para a apresentação de dispositivos



Fonte: *SmartX*

- Formulários para inserção de Estações e *Tickets*; *feature* representada nos requisitos funcional RF7 e RF9 da tabela 3, definida na seção 5.1.2.2 e exemplificada pela figura 14);

Adicionalmente às estruturas já definidas, também criou-se uma tabela de conteúdos da página personalizada para indicar o resumo da lista de estações e seus estados de forma concisa e gráfica. Para tal, não foi possível utilizar o *Plugin Maketoc*, comumente utilizado para criar uma tabela de conteúdos baseada nas estruturas de cabeçalho, pois esse se baseia na sintaxe *wiki* de *headers* para criar a tabela de conteúdos, enquanto o *script smarty* da *feature* Listagem de estações cria diretamente as estruturas HTML de cabeçalhos. Portanto, utilizou-se *JQuery* no *script smarty* da *feature* Listagem de estações para inserir os títulos e os ícones de *status* à estrutura *HTML* da tabela de conteúdos.

A figura 20 retrata uma parte da visão de contexto geral resultante, não retratada por inteiro por ser extensa. Do restante, uma parte segue o mesmo padrão da listagem de Estações, mostrada nessa figura, para os outros projetos e estações, e a outra são os formulários de inserção de Estações e *Tickets*. Esse último pode ser visualizado pela figura 5.1.2.2, à esquerda.

5.1.2.10 Visão inicial

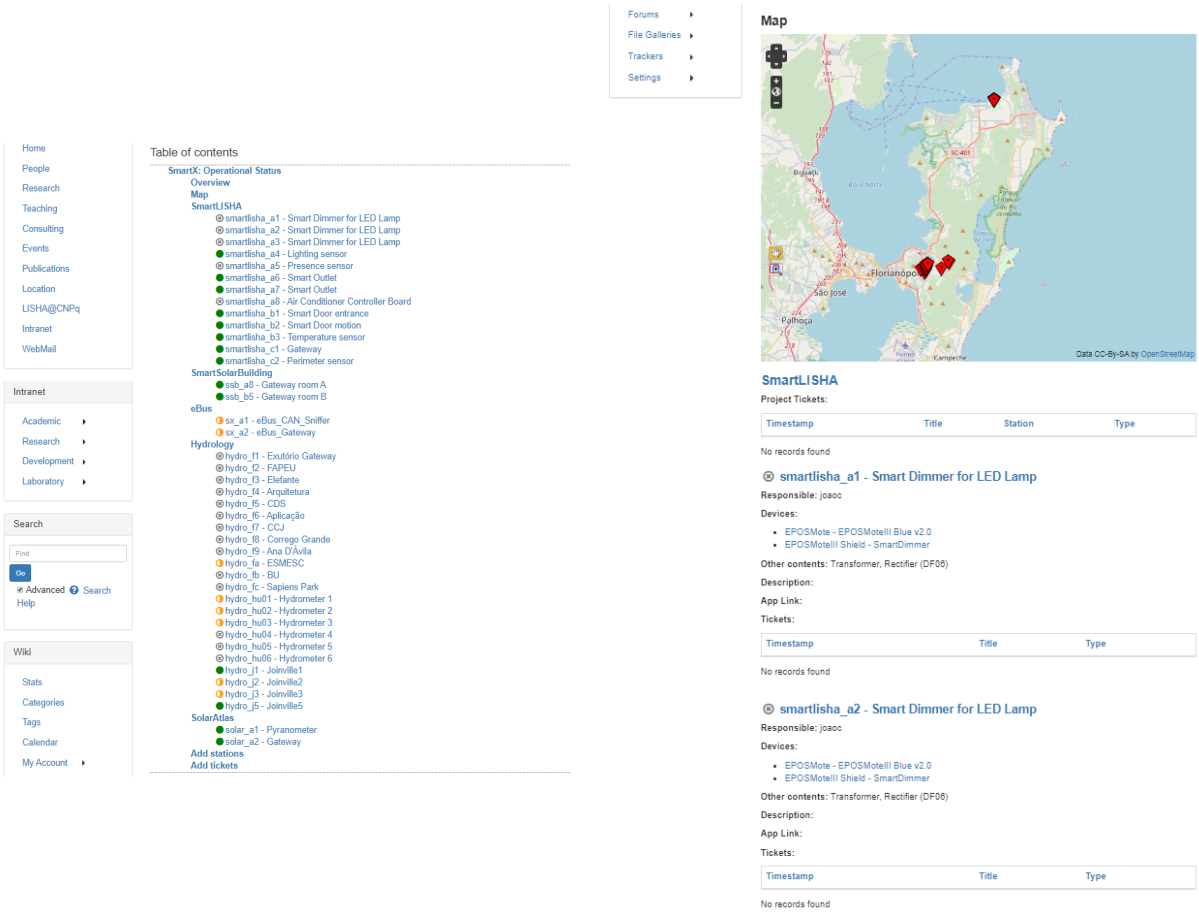
Para finalizar a implementação do sistema (na *TikiWiki*), criou-se uma página inicial para o *SmartX*, contendo o acesso para o restante das visões. Realizou-se isso através da listagem dos Projetos e Dispositivos, com links para suas respectivas páginas e provendo, também, o link para a visão de contexto geral.

5.2 Estudo de caso: Introdução do sistema no processo de desenvolvimento de projetos *IoT*

No plano de projeto deste trabalho (seção 3), determinou-se a realização de um estudo de caso para analisar o impacto do sistema *SmartX* no processo de desenvolvimento de projetos *IoT* no LISHA. Esse estudo teria o intuito de validar o sistema quanto aos objetivos de aprimorar certas etapas do processo de desenvolvimento (seção 1.2); e seria realizado através da introdução de uma verão funcional do sistema desenvolvido no processo de desenvolvimento dos projetos *IoT*.

Baseado nisso, alguns desenvolvedores, incluindo o autor, e gestores de projetos *IoT*, no LISHA, foram introduzidos ao sistema *SmartX* para a realização do estudo em questão. Por conta do sistema ter sido desenvolvido no mesmo ambiente e com a mesma ferramenta que o site do LISHA, os usuários já estavam configurados.

Figura 20 – Retrato da página *wiki* dinâmica de contexto geral



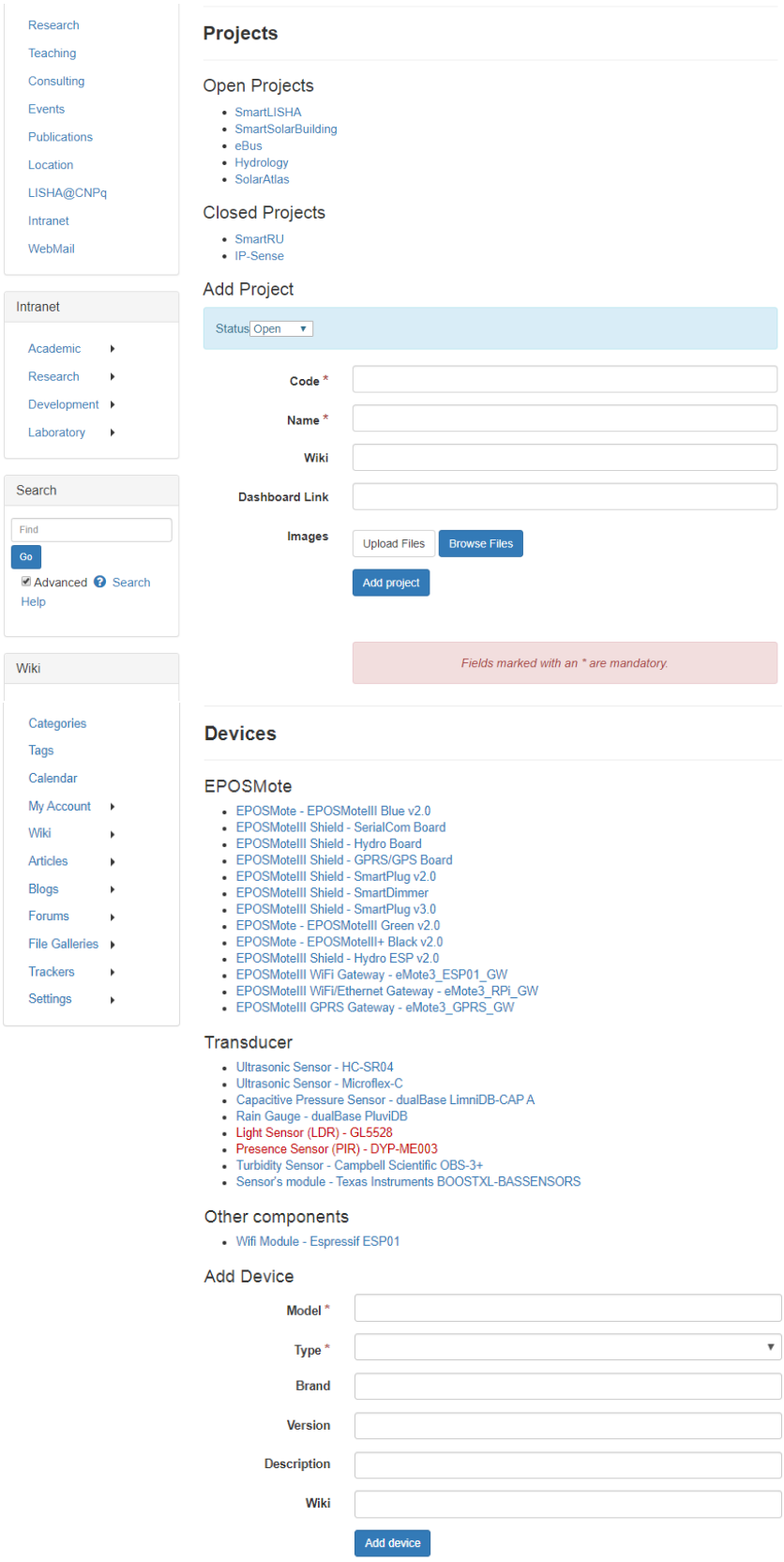
Fonte: *SmartX*

Sendo assim, o estudo de caso, resumidamente, foi realizado através da documentação, acompanhamento (análise) e planejamento do processo de desenvolvimento dos projetos *IoT* em atividade na época (*eBus*, PRAD e *SSB*, descritos na seção 2.1.3), por meio das visões de Projeto, Dispositivo e de contexto geral (descritas nas seções 5.1.2.7, 5.1.2.8 e 5.1.2.9), providas pelo *SmartX*. Traçando um paralelo com a representação do processo de desenvolvimento de projetos *IoT* na figura 4, o sistema passou a fazer parte da etapa de Documentação e Planejamento, que antes era realizada de forma não estruturada e não padronizada.

5.2.1 Utilização do *SmartX*

Com a inserção do *SmartX* no processo de desenvolvimento de projetos *IoT* do LISHA, o passo inicial para a condução do estudo e utilização do sistema em si, foi adicionar todos os Projetos, Estações e Dispositivos no sistema. A ferramenta *Trackers* permite a inserção de vários itens de uma só vez, através da importação de um arquivo *CSV* (estruturado conforme os atributos do *Tracker*), pela interface gráfica de gerenciamento de *Trackers*. Sendo assim, os desenvolvedores fizeram um levantamento das informações

Figura 21 – Visão inicial do sistema



Fonte: *SmartX*

dessas entidades e as organizaram em arquivos CSV. Esses, então, foram importados para o sistema, criando-se os *tracker items* de Projetos, Estações e Dispositivos.

Grande parte das informações, principalmente de Projetos e Dispositivos, se encontrava documentada em páginas da *Wiki* do LISHA, *dashboards* do *Trello* e tópicos do *Web2Project*. Algumas informações foram, ainda, extraídas de e-mails e conversas com os gestores. No caso das estações, entretanto, boa parte das informações ou não estava documentada, ou estava desatualizada. Tinha-se, porém, os dados de sensores e atuadores referentes a essas Estações, que foram enviados para a plataforma *IoT* e lá salvos. Esses dados, por serem *SmartData*, continham a localização das Estações, e portanto foi possível fazer um levante da localização de todas as Estações e visitá-las, coletando-se informações atualizadas sobre as mesmas.

Em seguida, realizou-se um levantamento retrospectivo das instalações e manutenções realizadas; dos problemas observados (sintomas, análises e diagnósticos) e suas soluções (se houveram); e das atividades planejadas. Essas informações também estavam documentados de forma não padronizada na *Wiki* do LISHA, no *Trello* e no *Web2Project*. Com isso, importou-se essas informações para o sistema, através de um arquivo CSV (assim como os Projetos, Estações e Dispositivos), no formato de *Tickets*.

A partir de então, as visões do sistema ganharam forma e os desenvolvedores passaram a utilizá-las para acompanhar e documentar o processo de desenvolvimento dos projetos *IoT* em atividade. Os gestores, por sua vez, tiveram um papel passivo nesse estudo, utilizando o sistema apenas para acompanhar o desenvolvimento.

5.2.2 PRAD

O projeto PRAD, descrito na seção 2.1.3, foi um dos projetos *IoT* do LISHA que tiveram bastante atividade durante o estudo de caso. Assim, gerou-se bastante documentação sobre o processo de desenvolvimento desse projeto para se analisar o impacto da utilização do sistema.

O PRAD é um projeto antigo, que já havia sido posto em funcionamento, contendo diversas estações hidrológicas instaladas, enviando dados para o servidor da plataforma *IoT* através de *gateways GPRS*. Porém, a limitação da escalonabilidade desses *gateways*, frente à projeção do custo do plano de dados celular dos mesmos, ao longo do tempo, fez com que gestores do projeto decidissem trocar a tecnologia de acesso à internet dos *gateways* para *WiFi* (caso houvesse rede, ou a possibilidade instalar uma).

Já havia uma solução de *gateway WiFi* com o desenvolvimento em curso no laboratório. Essa utilizava o dispositivo *ESP01* como modem *WiFi* do *EPOSMoteIII* (sistema embarcado parte da plataforma *IoT* do LISHA), com o auxilio da *HydroBoard ESP* (placa de expansão do *EPOSMote*). Essa solução foi apelidada de *eMote3-ESP01*.

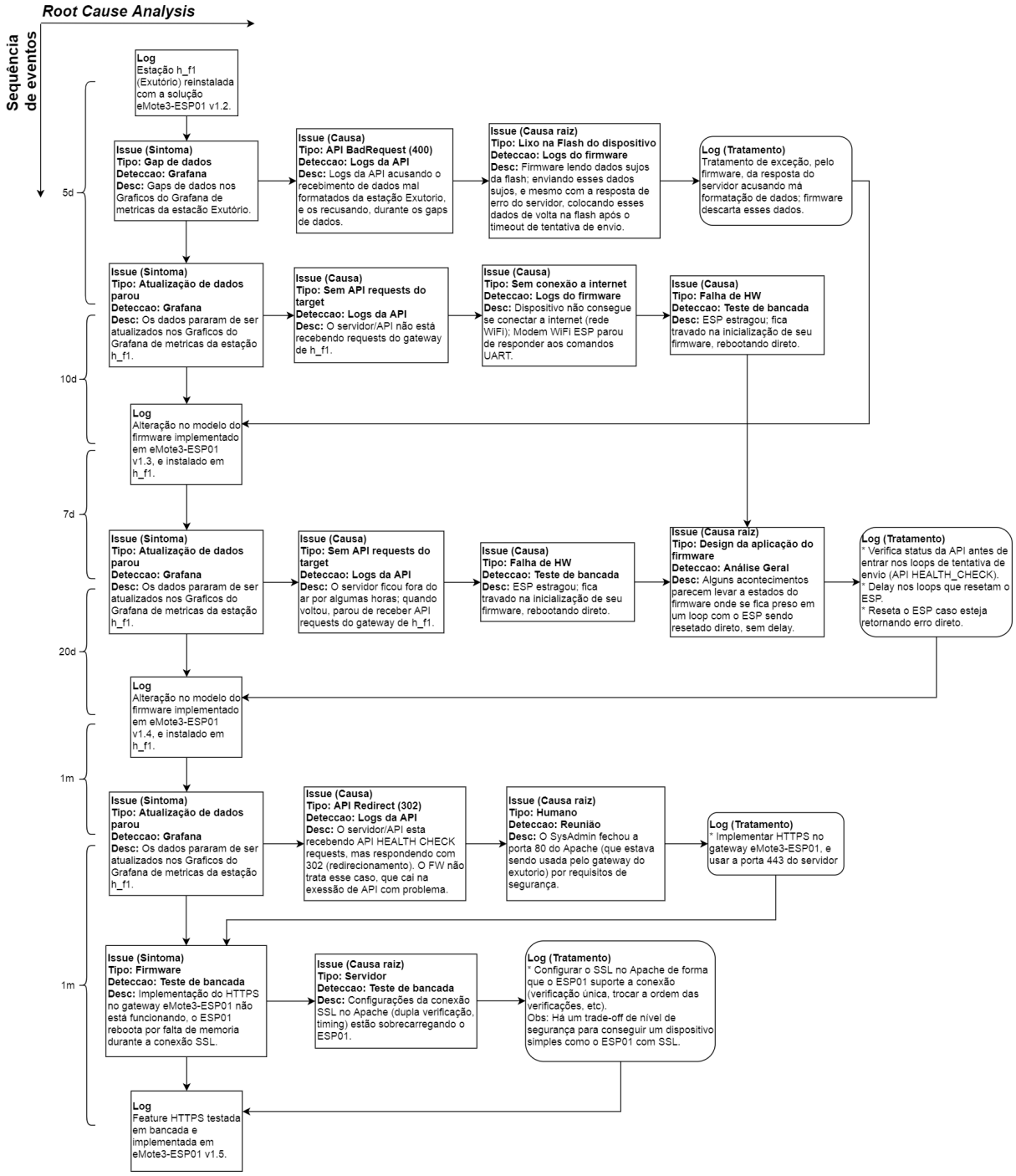
Sendo assim, o estudo de caso, nesse projeto, acompanhou o processo de desenvolvimento do *gateway WiFi eMote3-ESP01* na estação de código h_f1 (apelidada de Exutório). Esse processo passou por diversos ciclos de design e implementação, testes, instalação, monitoramento e detecção de falhas e gerenciamento de falhas (etapas do processo descrito na seção 2.1.2). Essas etapas foram sendo documentadas em *Tickets* no *SmartX*. A figura 22 exemplifica um trecho desse processo de desenvolvimento, através da mistura de um diagrama de causa-efeito com um diagrama de sequência (onde as flechas indicar uma relação entre os blocos), deixando evidente os processos de *Root Cause Analysis* das falhas detectadas.

5.2.3 eBus

O projeto eBus, descrito na seção 2.1.3, também foi um dos projetos *IoT* do LISHA que tiveram bastante atividade durante o estudo de caso. Essas atividades foram documentas em *Tickets* no sistema *SmartX*, o que levou à possibilidade de realizar uma análise desse processo.

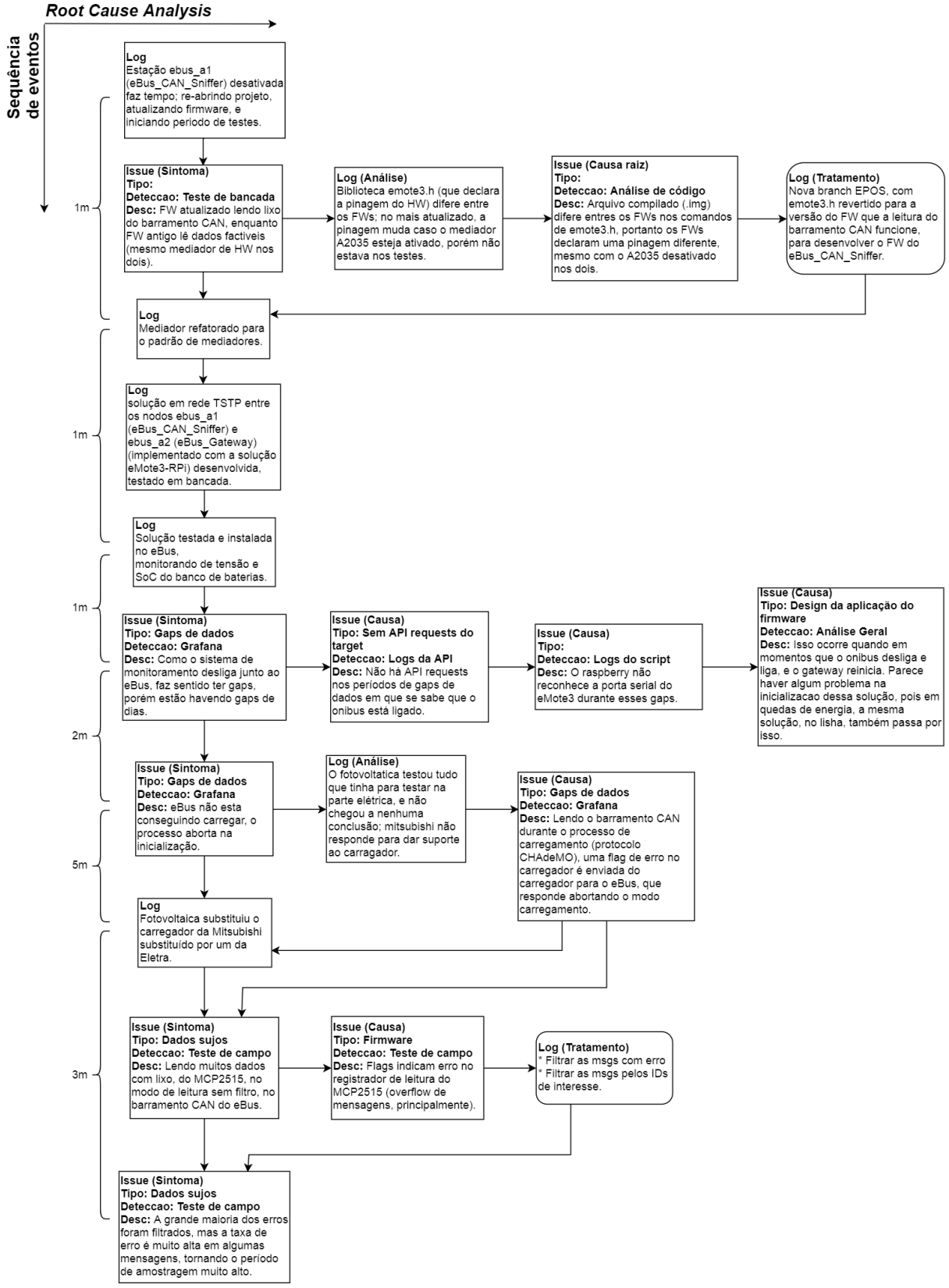
No início do estudo de caso, as Estações do *eBus* estavam desativadas e a versão atualizada do *firmware* da estação *eBus_CAN_Sniffer* estava apresentando erros nos testes de bancada. O estudo de caso, nesse projeto, acompanhou e documentou em *Tickets* o processo de reativação do monitoramento remoto de métricas do *eBus*, através da leitura de seu barramento *CAN*. A figura 23 exemplifica um trecho desse processo de desenvolvimento, através da mistura de um diagrama de causa-efeito com um diagrama de sequência (onde as flechas indicam uma relação entre os blocos), deixando evidente os processos de *Root Cause Analysis* das falhas detectadas.

Figura 22 – Diagrama de Sequência de Eventos x *RCA* do desenvolvimento na estação Exutório do projeto PRAD



Fonte: *SmartX*

Figura 23 – Diagrama de Sequência de Eventos x *RCA* do desenvolvimento das estações do projeto *eBus*



Fonte: *SmartX*

6 Análise dos resultados

Durante o desenvolvimento deste trabalho, buscou-se atender os objetivos descritos na seção 1.2, de aprimorar as etapas de gerenciamento de falhas e de planejamento e acompanhamento do processo de desenvolvimento de projeto *IoT* do LISHA. Para isso, desenvolveu-se um sistema de gerenciamento *Web* compilante com as características de projetos *IoT* (*SmartX*), que estruturou, sistematizou e integrou a documentação do processo de desenvolvimento desses projetos.

Uma versão funcional resultante do desenvolvimento do *SmartX* foi inserida no processo de desenvolvimento dos projetos *IoT* em atividade no LISHA, através da utilização do mesmo por desenvolvedores e gestores para a realização de um estudo de caso. Isso resultou em *feedbacks* da utilização do sistema por usuários e na documentação estruturada das etapas do desenvolvimento. Baseado nisso, foi possível validar o sistema e a sua inserção no processo de desenvolvimento, considerando os objetivos citados.

O *feedback* dos usuários foi, de maneira geral, positivo e construtivo. Os gestores elogiaram, principalmente, as visões contexto geral e de Projetos. Na primeira, endossou-se a possibilidade de acompanhar o andamento de diversos projetos em um mesmo contexto, com o auxílio de um mapa de Estações e contendo um leque abrangente de informações relevantes para a análise e planejamento do desenvolvimento desses projetos. Na segunda, exaltou-se a integração com o *Grafana*. Os desenvolvedores complementaram com críticas construtivas quanto a aspectos estruturais e de apresentação dos dados. Esses últimos foram s na seção de trabalhos futuros, abaixo.

Através da documentação estruturada do desenvolvimento dos projetos *eBus* e PRAD no estudo de caso, foi possível gerar diagramas (figuras 22 e 23) que evidenciam os processos de *Root Cause Analysis* realizados conforme a detecção de problemas durante o desenvolvimento desses projetos. Os processos de *RCA*, apresentados nos diagramas, mostram um aprofundamento das análises e da detecção de causas raiz desses problemas, comparado aos processos realizados anteriormente, sem o auxílio do sistema. Isso reflete no desenvolvimento de um tratamento mais adequado para o problema e reduz a realização de manutenções paliativas, um dos sintomas que motivaram o desenvolvimento deste trabalho.

6.1 Trabalhos futuros

Partindo de *feedbacks* da utilização do sistema por usuários (e pelo próprio autor) e de funcionalidades modeladas que acabaram não sendo implementadas, levantaram-se alguns pontos passiveis de serem abordados em trabalhos futuros.

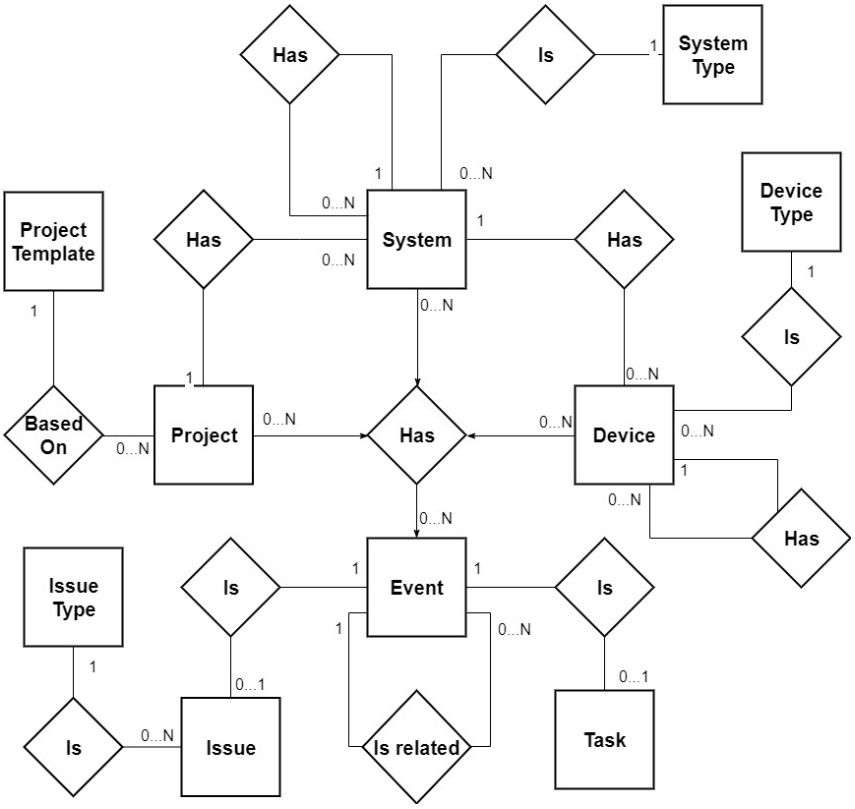
Alguns desses pontos, citados a seguir, levaram a uma nova versão do *ModelX* (figura 24), cuja implementação ficará para um trabalho futuro:

- Individualizar a representação de dispositivos: atualmente a entidade Dispositivo representa, na verdade, um grupo de Dispositivos. A ideia para a uma nova estrutura, seria individualizar a representação dos Dispositivos, possibilitando-se documentar *Tickets* individuais para esses dispositivos e integrar o sistema com o inventário do laboratório.

- Abranger as possibilidades de abstração dos sistemas IoT: atualmente, a representação da rede de sensores e atuadores pela entidade Estação é limitante quanto a semântica de grupos de sensores e suas relações. Não é possível representar, por exemplo, um grupo de nodos de um comodo de uma casa inteligente, como um sub-sistema semântico do sistema IoT completo.

- Estabelecer uma estrutura de classificação de falhas: atualmente, a entidade *Ticket* não fornece uma estrutura para classificar as falhas. No estudo de caso, por exemplo, foi levantado uma estrutura básica de classificação, que foi documentada em texto livre no atributo descrição.

Figura 24 – Diagrama ER da nova versão do *ModelX* para trabalhos futuros



Fonte: *SmartX*

Adequando-se o sistema ao novo modelo de dados e continuando a utilizá-lo, será possível, no futuro, realizar um estudo sobre os tipos de falhas mais comuns no desenvolvimento de projeto *IoT* no LISHA e agir em cima disso.

Além disso, citam-se também:

- Melhorar a interface das visão do sistema: segundo o *feedback* de usuários, as visões se alongam muito na vertical e exploram pouco a dimensão horizontal da tela
- Implementar a customização de visões por usuário (dependendo do grupo ao qual pertence): aspecto modelado porém não implementado, que permitirá a utilização do sistema por usuários do tipo cliente.
- Implementar uma visão de *RCA* para os *Tickets*: para deixar evidente as relações de causa-efeito, como nos diagramas das figuras 22 e 23.
- Implementar calendário de tarefas e prazos: segundo *feedbacks* dos usuários, a falta de visões focadas em planejamento os levaram a utilizar as ferramentas tradicionais de planejamento (*Trello* e *Web2Project*) para auxiliar o uso do *SmartX* durante o estudo de caso.
- Implementar alarmes e um esquema de regras para o disparo dos mesmos.

7 Considerações finais

Ao fim deste trabalho, concluiu-se, através da análise dos resultados do estudo de caso, que o sistema de gerenciamento de projetos *IoT* desenvolvido cumpriu os objetivos gerais traçados. Sendo assim, o trabalho foi capaz de aprimorar os processos de gerenciamento de falhas e de planejamento e acompanhamento das etapas do desenvolvimento de projetos *IoT* no LISHA. Apesar disso, o projeto ainda está em uma versão inicial e, portanto, é passível de melhorias.

Os conhecimentos adquiridos no curso de Engenharia de Controle e Automação foram essenciais na construção do projeto. A estrutura do curso proporciona um vasto conhecimento em diversas áreas, o que garantiu ao autor uma maior facilidade no estudo e compreensão de diversos aspectos do projeto.

Durante a realização deste trabalho, foram encontrados diversos desafios, em função do caráter multidisciplinar das atividades desenvolvidas. A criação do sistema *Web* mostrou-se particularmente difícil, devido à falta de experiência do autor neste âmbito. Além disso, a identificação do processo de desenvolvimento dos projetos *IoT* tomou bastante tempo, em razão da ausência de documentação deste quesito no laboratório. As dificuldades, entretanto, foram uma oportunidade para aprender novos conceitos e colocá-los em prática.

O projeto apresentou um apelo especial para o autor pelo fato do mesmo trabalhar há dois anos no LISHA com o desenvolvimento das soluções *IoT*, que foram de suma importância na formação profissional do autor, e que foram contempladas neste trabalho. Além disso, o sistema *SmartX* resultante deste projeto permitirá a melhoria da gestão interna dos projeto do *LISHA*, facilitando o trabalho dos gestores e desenvolvedores e possibilitando a entrega de melhores produtos aos clientes e parceiros.

Referências

1 LISHA, U. *EPOS: Embedded Parallel Operating System)*. 2019. Disponível em: <https://epos.lisha.ufsc.br/>. Citado 2 vezes nas páginas 23 e 25.

2 FRöHLICH, A. A. *Application-Oriented Operating Systems*. 200 p. Tese (Doutorado) — Technical University, Berlin, 2001. Ph.D. Thesis. Citado na página 23.

3 FRöHLICH, A. A.; STEINER, R.; RUFINO, L. M. A Trustful Infrastructure for the Internet of Things based on EPOSMote. In: *9th IEEE International Conference on Dependable, Autonomic and Secure Computing*. Sydney, Australia: [s.n.], 2011. p. 63–68. ISBN 978-1-4673-0006-3. Disponível em: <http://www.lisha.ufsc.br/pub/Frohlich_DASC_2011.pdf>. Citado 2 vezes nas páginas 24 e 25.

4 FROHLICH, A. A. et al. A Cross-layer Approach to Trustfulness in the Internet of Things. In: *9th Workshop on Software Technologies for Embedded and Ubiquitous Systems (SEUS)*. Paderborn, Germany: [s.n.], 2013. p. 1–8. Disponível em: <http://www.lisha.ufsc.br/pub/Frohlich_SEUS_2013.pdf>. Citado 2 vezes nas páginas 24 e 31.

5 RESNER, D.; FRöHLICH, A. A. Design Rationale of a Cross-layer, Trustful Space-Time Protocol for Wireless Sensor Networks. In: *20th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA 2015)*. Luxembourg, Luxembourg: [s.n.], 2015. p. 1–8. ISBN 978-1-4673-7929-8. Citado na página 24.

6 GIL, G. S. R. et al. Sistema de Monitoramento Hidrológico com Telemetria em uma Pequena Bacia Urbana. In: *XXII Simpósio Brasileiro de Recursos Hídricos*. Florianópolis, Brazil: [s.n.], 2017. p. 1–8. Disponível em: <http://evolvedoc.com.br/xxiisbrh/detalhes-172_sistema-de-monitoramento-hidrologico-com-telemetria-em-uma-pequena-bacia-urbana>. Citado na página 29.

7 ALUR, R. *Principles of Cyber-Physical Systems*. [S.l.]: The MIT Press, 2015. ISBN 0262029111, 9780262029117. Citado na página 35.

8 SOHRABY, K.; MINOLI, D.; ZNATI, T. *Wireless Sensor Networks: Technology, Protocols, and Applications*. New York, NY, USA: Wiley-Interscience, 2007. ISBN 0471743003. Citado na página 35.

9 MINERVA, R.; BIRU, A.; ROTONDI, D. Towards a definition of the internet of things. In: . IEEE Internet Initiative, 2015. Disponível em: <https://iot.ieee.org/images/files/pdf/IEEE_IoT_Towards_Definition_Internet_of_Things_Revision1_27MAY15.pdf>. Citado na página 36.

10 FOWLER, M. *UML Distilled: A Brief Guide to the Standard Object Modeling Language*. 3. ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2003. ISBN 0321193687. Citado na página 36.

11 LATINO, M.; LATINO, R.; LATINO, K. *Root Cause Analysis: Improving Performance for Bottom-Line Results, Fourth Edition*. CRC Press, 2016. ISBN 9781439851272.

Disponível em: <https://books.google.com.br/books?id=geig30dBoxgC>. Citado na página 37.

12 Tiki Software Community Association. *Tiki Wiki CMS Groupware Introduction*. 2019. Disponível em: <https://tiki.org/Introduction>. Citado na página 38.

13 Grafana Labs. *Grafana Documentation*. 2019. Disponível em: <https://grafana.com/docs/>. Citado na página 39.

14 COLLINS-SUSSMAN, B.; FITZPATRICK, B. W.; PILATO, C. M. *Version Control with Subversion*. O'Reilly, 2002. Disponível em: <http://svnbook.red-bean.com/>. Citado na página 39.