

FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO



FEUP

Framework para o desenvolvimento rápido de aplicações baseadas em redes de sensores sem fios

Rúben Tiago Ribeiro da Costa

Mestrado Integrado em Engenharia Informática e Computação

Orientador: Gil Manuel Magalhães de Andrade Gonçalves (Engenheiro)

28 de Junho de 2010

Framework para o desenvolvimento rápido de aplicações baseadas em redes de sensores sem fios

Rúben Tiago Ribeiro da Costa

Mestrado Integrado em Engenharia Informática e Computação

Aprovado em provas públicas pelo júri:

Presidente: José Manuel Magalhães Cruz (Professor Auxiliar)

Vogal Externo: Gustavo Ribeiro da Costa Alves (Professor Adjunto)

Orientador: Gil Manuel Magalhães de Andrade Gonçalves (Assistente Convidado)

28 de Junho de 2010

Resumo

No mundo actual, o ser humano encontra-se numa constante campanha em busca da total mobilidade trazendo consigo o máximo de informação possível. Por esta razão a área das tecnologias sem fios tem sido tão desenvolvida nos últimos tempos. Dentro desta área, e aliada a uma fonte de informação importante, encontra-se a tecnologia de redes de sensores sem fios que são constituídas por dispositivos autónomos (Motes) espacialmente distribuídos que, de forma cooperativa, monitorizam condições físicas ou ambientais, tais como temperatura, humidade, luminosidade, som, pressão, vibrações, etc., em diferentes localizações.

Graças a estas capacidades estes aparelhos têm sido o foco de desenvolvimento de muitas aplicações inovadoras nos tempos actuais. No entanto, este desenvolvimento tem encontrado muitos obstáculos.

Como é uma área inovadora, não existe nenhum método de desenvolvimento nem ferramentas dedicadas para completar tal tarefa. Poucas são as vezes em que o fabricante fornece algo de útil para tal.

Esta dissertação tem como objectivo colmatar esta necessidade e oferecer uma ferramenta que ajude no desenvolvimento destas aplicações

O trabalho foi realizado sobre uma primeira versão desta ferramenta, a WiSe Framework. Foram criadas ferramentas de suporte para a criação das dependências da Framework. Estas ferramentas ainda oferecem capacidades de depuração e optimização para as mesmas. A adaptação e melhorias feitas nos restantes componentes permitem retirar o utilizador da edição de código fonte, permitem a importação automática das dependências, a criação de plugins, etc. Não menos importante foi ainda desenvolvido um método de desenvolvimento que varia de acordo com a utilização pretendida.

As ferramentas certas aliadas a um método estruturado de desenvolvimento produzem assim um óptimo resultado e, indo além do objectivo proposto, permitem assim ultrapassar os obstáculos encontrados no desenvolvimento de aplicações que usem tecnologias de redes de sensores sem fios.

Abstract

In the actual world, the human being finds himself in a constant campaign for the search of total mobility with the maximum information possible. For this reason the area of wireless sensor technologies has been so developed over the last years. Within this area, and allied to an important source of information, lays the wireless sensor networks technology which is constituted by small autonomous devices(Motes), space distributed, that of cooperative form monitor physical or ambient conditions, such as temperature, moisture, luminosity, sound, pressure, vibrations, etc., in different locations.

Thanks to these abilities, these devices have been being the focus of the development of lots of innovative applications in the modern times. However, this development has found lots of obstacles.

Since it is an innovative area, there's not a dedicated development method or dedicated tools to achieve this task. Few are the options that the manufacture offers.

This dissertation has the objective to eliminate these needs and offer a tool that helps in the development of such applications.

The work was done over a initial version of this tool, called WiSe Framework. Supporting tools were created for the development of the dependencies of the framework. Other than that, these tools offer debugging capabilities and optimization extras. The adaptation made to the rest of the components allowed the elimination of the necessity for the user to alter source code, the automatic importation of dependencies, and the creation of plugins amongst other things. Not less important was the establishment of a development method that varies with the wanted utilization.

The right tools allied to the right method of development produce a great result that goes beyond the proposed objective allowing it to overcome the obstacles found in the development of wireless sensors network applications.

Conteúdo

1	Introdução	1
1.1	Contexto e Enquadramento	1
1.2	Motivação e Objectivos	2
1.3	Estrutura da Dissertação	3
2	Revisão Bibliográfica	4
2.1	Redes de Sensores sem Fios	4
2.1.1	Introdução teórica	4
2.1.2	Tecnologias de Rede	5
2.2	Áreas de Aplicação	6
2.3	Trabalhos Relacionados	7
2.4	Arquitectura geral de uma aplicação RSSF	10
2.5	Outros projectos	11
2.5.1	CrossBow MoteWorks	11
2.5.2	Sentilla	12
2.5.3	ZigBit	13
2.5.4	CodeBlue	13
2.5.5	Octavex	13
2.6	WiSe Framework, versão inicial	15
2.7	Comparações	17
2.8	Resumo e Conclusões	18
3	WiSe Framework, versão inicial	20
3.1	Arquitectura Geral	21
3.2	Conceitos comuns	23
3.2.1	Sensores	23
3.2.2	Parsing de mensagens	23
3.2.3	WiseML	23
3.3	Camada dos Sensores	24
3.3.1	Universal Gateway	24
3.3.2	Funcionamento Geral da Universal Gateway	29
3.4	Camada dos Dados	32
3.4.1	WiSeServer	32
3.4.2	Funcionamento Geral do WiSeServer	35
3.5	Base de Dados	36
3.5.1	O Generator	36
3.5.2	Funcionamento Geral do Generator	36

CONTEÚDO

3.6	Camada do Interface	37
3.6.1	WiSeApp	37
3.7	Framework Wizard	39
3.7.1	O que é?	39
3.7.2	Para que serve?	39
3.7.3	Funcionamento Geral do Wizard	40
3.8	Conclusão	41
4	Apresentação e descrição do problema	42
4.1	O acto de criação de aplicações para RSSF	42
4.2	Utilização da WiSe Framework, versão inicial	43
4.2.1	1ª dependência: O parsing das mensagens	44
4.2.2	2ª dependência: O processamento dos dados	44
4.2.3	3ª dependência: O interface	44
4.3	Resumo e Conclusões	45
5	Solução proposta - O método	47
5.1	O método de utilização	47
5.2	Componentes do método	48
5.2.1	Desenvolver o parser com a Parser_tool	48
5.2.2	Desenvolver o workflow com a Workflow_tool	48
5.2.3	Criar Projecto e compilar	49
5.2.4	Criar Base de Dados	49
5.2.5	Configurar a aplicação final	49
5.2.6	Alterar ou adicionar novo interface	49
5.3	O antes e o depois	49
5.3.1	Camada dos Sensores/Serviços	49
5.3.2	Camada de Dados	50
5.3.3	Framework Wizard	51
5.3.4	Base de Dados	52
5.3.5	Camada do Interface	52
6	Concepção e Implementação	54
6.1	Camada Sensores	54
6.1.1	Optimizações na Universal Gateway	54
6.1.2	O Parser_tool	56
6.2	Camada Dados	59
6.2.1	Optimizações ao WiSeServer	59
6.2.2	A Workflow_tool	61
6.3	Camada Interface	64
6.3.1	Melhoramentos	64
6.3.2	Novos gráficos	66
6.3.3	Motor de Plugins	67
6.4	Base de Dados	68
6.4.1	Optimizações ao Generator	68
6.5	Framework Wizard	71
6.5.1	Optimizações	71

CONTEÚDO

6.5.2	Projecto VS Configurações	72
6.5.3	Novas funcionalidades	72
6.5.4	Novas opções	73
6.5.5	Funcionamento actual do Framework Wizard	74
6.6	Testes	83
6.6.1	Introdução	83
6.6.2	Inclusão de um novo serviço - Amplitude de Som	83
6.6.3	Implementação de um novo serviço externo - Serviço de Locali- zação e Seguimento	84
7	Conclusões e Trabalho Futuro	86
7.1	Satisfação dos Objectivos	86
7.2	Limitações	87
7.3	Trabalho Futuro	88
	Referências	89
A	Testes	92
A.1	Inclusão de um novo serviço - Amplitude de Som	92
A.2	Implementação de um novo serviço externo - Serviço de Localização e Seguimento	97

Lista de Figuras

2.1	Topologia de RSSF	5
2.2	Alguns tipos de redes de sensores sem fios	6
2.3	Diagrama do myTag	7
2.4	Interface do MoteTrack	8
2.5	Localização dos sensores no estudo de Parkinson	8
2.6	Arquitectura do CrossBow MoteWorks	11
2.7	Arquitectura de uma rede Sentilla	12
2.8	WSN Monitor GUI do Kit ZigBit	13
2.9	Arquitectura da Octavex	14
2.10	Arquitectura física da WiSe Framework	15
3.1	Arquitectura Geral de uma aplicação resultante da Wise Framework	21
3.2	Aplicação resultante criada pela WiSe Framework	22
3.3	Arquitectura Geral da Universal Gateway	24
3.4	Exemplo de uma mensagem recebida	25
3.5	Diagrama de Actividade - Parsing	30
3.6	Diagrama de Actividades - Comandos	31
3.7	Arquitectura Geral do WiSeServer	32
3.8	Serviços disponibilizados pelo WiSeServer	33
3.9	Diagrama de um Workflow Sequencial	34
3.10	Exemplo da estrutura de Base de Dados de uma aplicação resultante	36
3.11	Arquitectura Geral da WiseApp	37
3.12	Interface do Framework Wizard	39
6.1	Ilustração das flags de início e de fim	55
6.2	O parser_tool	56
6.3	Web Services disponíveis actualmente	60
6.4	Ecra de arranque do Workflow_tool	62
6.5	Valores de entrada e saída	62
6.6	Variveis de tempo de execução	63
6.7	Gráficos sem Identificação do Hardware	65
6.8	Gráficos com Identificação do Hardware	65
6.9	Gráfico linear e Gráfico de pontos	66
6.10	Gráfico de Barras e Gráfico de de densidade	66
6.11	Feedback do Generator	69
6.12	Interface simples para novos parâmetros	71
6.13	Separação entre serviços Internos e Externos	72

LISTA DE FIGURAS

6.14	Opções de Compilação	73
6.15	Ecran inicial do Wizard da WiSe Framework	74
6.16	Serviços do projecto actual	75
6.17	Informação dos utilizadores	76
6.18	Interface da Aplicação final	77
6.19	Opções de compilação	78
6.20	Serviços internos	79
6.21	Pop-Up novo serviço interno	79
6.22	Serviços Externos	80
6.23	Pop-Up novo serviço externo	80
6.24	Workflows	81
6.25	Pop-Up novo Workflow	81
6.26	Botão cmpilar projecto	82
6.27	Botão criar base de dados	82
A.1	Interface inicial que obriga a criar ou abrir um projecto	92
A.2	Pedido de um ficheiro pelo interface	93
A.3	Desenvolvendo um Parser	94
A.4	Workflows no mesmo ambiente de desenvolvimento que a WiSeApp	95
A.5	Adição de tipos de sensor no código fonte do generator	96
A.6	Interligação entre o código fonte	96
A.7	Ambiente de desenvolvimento de workflows e a Workflow_tool	98
A.8	SLS - Interface final	99

Lista de Tabelas

2.1	As 3 camadas de uma aplicação RSSF	10
2.2	Comparações entre os Trabalhos realizados	17
5.1	Correspondência entre método e possíveis utilizações	48
5.2	Antes e Depois - Sensores/Serviços	49
5.3	Antes e Depois - Dados	50
5.4	Antes e Depois - Framework Wizard	51
5.5	Antes e Depois - Base de Dados	52
5.6	Antes e Depois - Interface	52

Lista de Listagens

3.1	Exemplo de um ficheiro WiSeML	23
3.2	Exemplo de um ficheiro sensores.xml	27
3.3	Exemplo de um ficheiro portas.xml	28
6.1	Exemplo de um ficheiro portas.xml actual	55
6.2	Ficheiro das configurações do Parser_tool	57
6.3	Exemplo de classes para o carregamento dinâmico	59
6.4	Ficheiro das configurações do Wotkflow_tool	63
6.5	Exemplo de ficheiro WiseConfig.xml	64
6.6	Exemplo de uma chamada de Render Partial	67
6.7	Motor de carregamento de ficheiros javascript	67
6.8	Motor de carregamento de ficheiros css	68

Abreviaturas e Símbolos

AJAX	Asynchronous Javascript And XML
API	Application Programming Interface
DLL	Dynamic-link library
ECG	Electro Cardiograma
GPS	Global Positioning System
GSM	Global System for Mobile Communications
GUI	Graphical User Interface
IDE	Integrated Development Environment
IEEE	Institute of Electrical and Electronics Engineers
MVC	Model-View-Controller
NMEA	National Marine Electronics Association
PAC	Presentation-Abstraction-Control
RF	Radio Frequência
RSSF	Redes de Sensores Sem Fios
SOAP	Simple Object Access Protocol
WSN	Wireless Sensor Network
WWF	Windows WorkFlow Foundation
XML	eXtensible Markup Language

Capítulo 1

Introdução

Este capítulo apresenta o enquadramento e a motivação do trabalho, identifica os problemas abordados, resume as metodologias utilizadas e apresenta uma breve descrição dos capítulos posteriores.

1.1 Contexto e Enquadramento

Esta dissertação é realizada na área das redes de sensores sem fios. Esta é uma área onde se têm verificado imensos avanços tecnológicos nos últimos anos. O desenvolvimento e a pesquisa realizada nesta área levaram à criação de padrões e protocolos que servem de base para futuros trabalhos a serem desenvolvidos.

Esta dissertação decorreu na empresa INOVAMAIS, Serviços de Consultadoria em Inovação Tecnológica, SA.

“A INOVAMAIS é, desde a sua criação em 1997, um líder em inovação, ocupando o espaço entre o mundo científico e o mundo empresarial e colocando os seus conhecimentos e experiência em processos de inovação ao serviço dos líderes de amanhã.” [[dCdAdIS](#)]

Aquando da realização desta dissertação, a empresa INOVAMAIS, conta com 2 projectos já desenvolvidos na área das RSSF e outros 2 em desenvolvimento. Não sendo uma empresa dedicada exclusivamente a esta área, a experiência adquirida pela mesma permitiu identificar um conjunto de necessidades no desenvolvimento destes projectos que serviu de inspiração para o tema desta dissertação.

O facto de esta dissertação decorrer em ambiente empresarial permitiu ainda o acesso a vários tipos de hardware e de conhecimento que contribuiu para uma maior valia do resultado final.

1.2 Motivação e Objectivos

Existem actualmente inúmeros projectos deste tipo com diferentes objectivos e de elevada importância, como por exemplo, na área da saúde. Este tipo de importância faz levantar o interesse, das pessoas envolvidas em tais projectos, de criar soluções informáticas cada vez mais inovadoras.

No entanto, são constatados vários problemas na criação destes projectos. Um dos maiores problemas é a dificuldade que existe em desenvolver software para RSSF mas a um nível superior, isto é, partindo do princípio que a RSSF e o seu funcionamento estão correctos, fazer a sua abstracção e providenciar soluções informáticas para o processamento e tratamento da informação resultante da rede não tem qualquer método definido e fica sempre a cargo do programador tal decisão. Este tipo de abordagem é lenta, difícil e os resultados, nem sempre são os melhores.

Assim, e como o título indica, esta dissertação tem por objectivo desenvolver uma ferramenta que permita superar estas barreiras e promover o rápido desenvolvimento destas aplicações.

Esta ferramenta deverá ser o mais transparente e abstracta quanto possível. Quer isto dizer que deverá libertar o utilizador de tarefas complicadas e diminuir ao máximo a programação. O ideal seria escolher o tipo de hardware a usar, como os dados desse hardware devem ser processados e compilar tudo. Infelizmente, existem inúmeras variáveis a ter em conta, o hardware muda com frequência, os objectivos finais mudam, as tecnologias de suporte mudam. Torna-se então muito difícil seguir um método de desenvolvimento concreto que resolva o problema mas não impossível.

Em questões de hardware, esta ferramenta deve permitir a implementação fácil de novos componentes sendo que as únicas variáveis neste aspecto deverão ser dependentes do hardware a adicionar.

Em questões de processamento de dados, a ferramenta deve incluir um método simples que permita ao utilizador "programar" de forma intuitiva o tratamento a dar aos mesmos.

É necessário também existir todo um conjunto de operações no interface final para dar a liberdade ao utilizador de escolher a forma de visualizar os dados.

Para que tudo isto aconteça, é necessário numa fase preliminar, fazer um estudo sobre o que já existe neste campo e identificar os pontos fortes e os pontos a desenvolver para que sirvam de orientação ao desenvolvimento desta dissertação. O próximo passo será colmatar esses pontos a desenvolver tentando assim obter uma ferramenta que satisfaça estes objectivos.

Em poucas palavras, o objectivo desta dissertação é desenvolver uma ferramenta ágil que permita o desenvolvimento rápido de aplicações baseadas em redes de sensores sem fios.

É de salientar que a INOVAMAIS impôs uma condição de desenvolvimento. Condição que diz que este desenvolvimento deve ter por base uma primeira versão já existente de uma ferramenta intitulada de WiSe Framework.

1.3 Estrutura da Dissertação

Esta dissertação encontra-se dividida em 7 capítulos. O primeiro e último capítulo dizem respeito à introdução e às conclusões respectivamente. Nestes é explicado os objectivos e a motivação desta dissertação e também os resultados concluídos da conclusão dos mesmos.

Nos capítulos intermédios é descrito todos os passos que decorreram na execução desta dissertação. Inicialmente é feita uma análise ao estado da arte actual nesta área onde é também justificado a condição imposta pela empresa em questão. No capítulo seguinte é descrito em mais pormenor a WiSe Framework para facilitar a compreensão das matérias discutidas nos restantes capítulos.

Os capítulos restantes consistem em apresentar o problema na sua totalidade, apresentar a solução e as suas vantagens e por fim, descrever com mais pormenor o que foi feito para atingir esta solução.

Capítulo 2

Revisão Bibliográfica

Neste capítulo é feita uma descrição do estado da arte na área das RSSF. É feita uma síntese sobre a base do projecto assim como as aplicações relacionadas. É feita referência a alguns trabalhos já realizados neste domínio e também aos problemas encontrados durante a sua elaboração.

No final, apresenta-se uma descrição geral de projectos compatíveis a esta dissertação e é feita uma comparação entre eles.

2.1 Redes de Sensores sem Fios

Esta é a área de desenvolvimento desta dissertação. Tudo aqui retratado está directa ou indirectamente relacionado com este tipo de redes.

2.1.1 Introdução teórica

Uma rede de sensores sem fios (wireless sensor network) é uma rede de aparelhos denominados de nós que podem sentir o ambiente que os envolve e comunicar tal informação através de ligações sem fios [BPC⁺07]. Esses aparelhos são então designados por sensores.

Os dados são transmitidos, se possível e necessário através de múltiplos nós, para um nó receptor muitas vezes denominado como a base (sink). A este nó base está geralmente associado uma aplicação que pode ser local ou distribuída e que se encarregará de outras funções. Os nós da rede podem estar parados ou em movimento, podem ou não estar conscientes da sua localização e podem ou não ser homogéneos assim como muitas outras características dependentes do objectivo da rede.

Na Figura 2.1 temos um exemplo do layout de um rede de sensores sem fios(a) e um exemplo de uma possível trajectória que a informação pode ter(b).

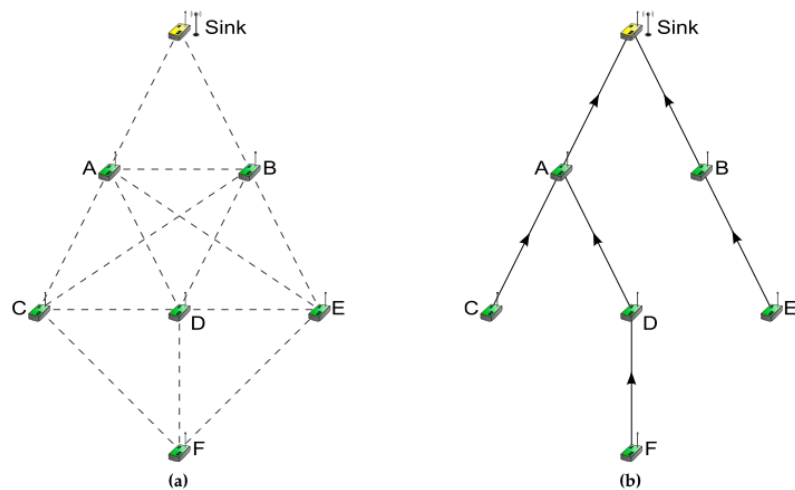


Figura 2.1: Topologia de RSSF

Tal trajectória é possível através de um algoritmo de routing incluído na programação dos nós.

A funcionalidade deste tipo de redes é influenciada pela distância entre cada nó e pelas suas condições técnicas (estado da bateria, sensores incluídos, etc.).

2.1.2 Tecnologias de Rede

Não existe nenhuma obrigatoriedade sobre qual a tecnologia a utilizar neste tipo de redes [BPC⁺07]. No entanto é possível designar quais as tecnologias que actualmente são usadas em produção e conferem a produtos estáveis. São estas a tecnologia ZigBee e Bluetooth.

ZigBee é a especificação para protocolos de comunicação que se destina a equipamentos de rádio frequência de pouca potência baseados na especificação IEEE 802.15.4-2006. Este protocolo contém duas variantes: uma para largura de banda normal e outra para largura de banda larga.

Bluetooth é um protocolo aberto que se destina à troca de dados entre aparelhos de rádio frequência de onda curta.

A grande diferença entre estas duas tecnologias encontra-se na topologia da rede. Com a tecnologia ZigBee é possível estabelecer um rede tipo malha (mesh network) isto é, os nós podem trocar mensagens entre si. Num rede com a tecnologia Bluetooth, cada nó troca mensagens directamente com o receptor.

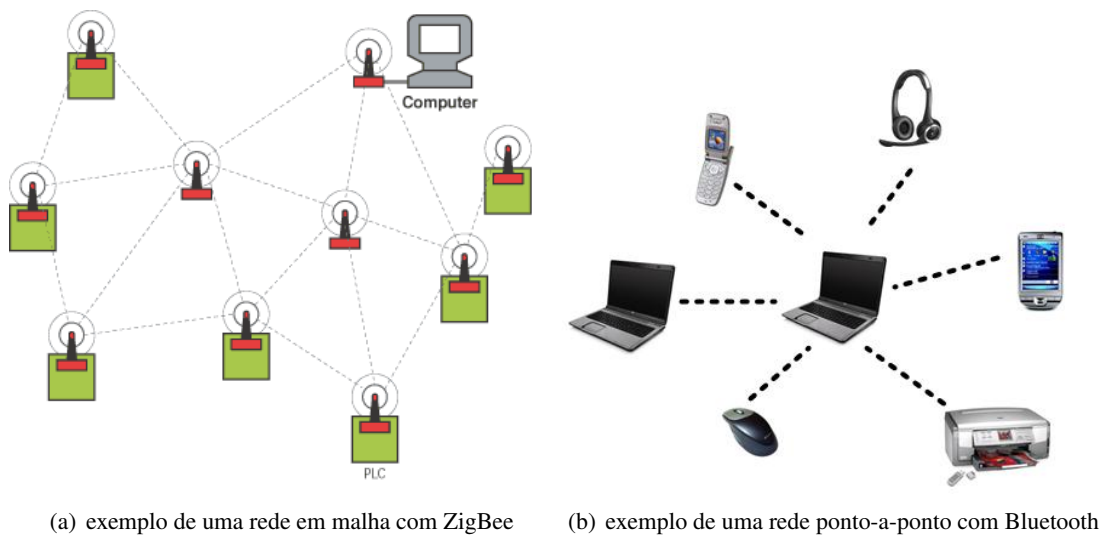


Figura 2.2: Alguns tipos de redes de sensores sem fios

A vantagem de uma rede em malha quando comparada com uma rede com a tecnologia bluetooth é que o nó mais distante, se não estiver dentro do raio de comunicação com o nó base, pode fazer o envio da sua mensagem, através dos outros nós, aumentando assim a área de cobertura da rede.

2.2 Áreas de Aplicação

As áreas nas quais se pode incluir este tipo de tecnologia são inúmeras. No entanto, é possível agrupá-las nos seguintes tópicos:

- Monitorização do Ambiente
- Cuidados na Saúde
- Serviços sobre o estado de espírito
- Posicionamento e localização de animais
- Entretenimento
- Logística
- Transportes
- Casa e escritório
- Aplicações industriais

Na monitorização do ambiente podemos encontrar aplicações que servem ambientes de interior ou de exterior. O tipo de sensores utilizado em redes de sensores sem fios permite obter detalhes localizados sobre espaços onde os métodos comuns não chegam. Na maior parte das vezes, estas aplicações servem para soluções de vigilância e segurança.

Na área da saúde encontram-se aplicações para a monitorização remota de dados vitais humanos, localização de pacientes e doutores, administração de drogas em hospitais, etc. O uso destas aplicações faz com que a vida tanto dos pacientes como a dos doutores seja mais fácil, ou seja, é possível fazer diagnósticos e outros processos independentemente da posição dos intervenientes assim como operações de triagem dentro dos hospitais, por exemplo. Tais aplicações são vitais pois lidam com dados humanos.

Na área dos serviços sobre o estado de espírito encontram-se as aplicações que tentam processar de forma inteligente o estado de espírito do utilizador. Isto é, a medição de dados corporais através de uma rede de sensores sem fios permite identificar a emoção do utilizador que pode depois ser incluída numa conversa on-line ou numa chamada telefónica.

Aplicações que trabalham na área de posicionamento e localização de animais servem simplesmente para monitorizar posições e localizações. Na área de entretenimento encontram-se as aplicações que usam varias fontes de dados para dar ênfase ao entretenimento do utilizador.

Na área de logística, transportes, industria e casa e escritório, encontram-se as aplicações com grandes semelhanças entre si. Todas elas usam os dados provenientes de várias fontes para melhorar a eficiência e eficácia dos seus objectivos.

2.3 Trabalhos Relacionados

Existem imensos trabalhos relacionados com a área das redes de sensores sem fios. Ao estudar estes trabalhos verifica-se que cada um deles tem o seu próprio software desenhado única e exclusivamente para esse efeito.

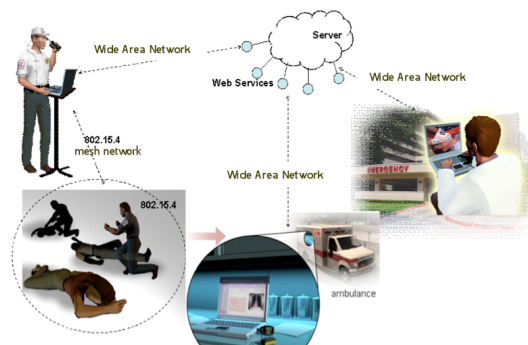


Figura 2.3: Diagrama do myTag

O myTag [TG] (medical information tag) é uma aplicação desenvolvida nas instalações da universidade de Harvard cujo objectivo é resolver problemas no congestionamento aquando de situações de emergência médica. Suporta diversos tipos de sensores (GPS, pulsação, oximetria, pressão sanguínea, temperatura, ECG) através de uma rede de sensores em malha.

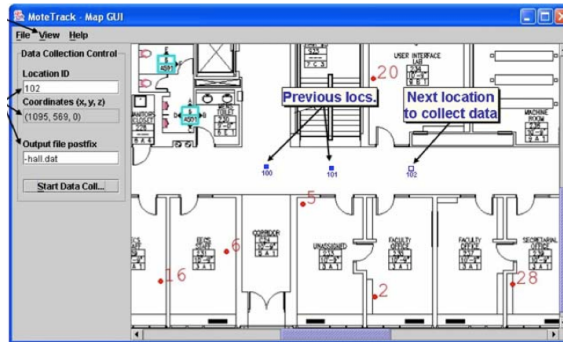


Figura 2.4: Interface do MoteTrack

O MoteTrack [LW07] é uma aplicação que não depende de qualquer servidor ou de nenhuma outra rede. A sua finalidade é estabelecer a localização dos vários nós através da sua assinatura RF.

Pela figura 2.4 podemos constatar que o MoteTrack foi desenvolvido apenas com a funcionalidade de detectar a localização dos nós.

Também é possível encontrar outro tipo de projectos na área da saúde. De facto, esta é a área em que se encontram as maiores inovações no campo das redes de sensores sem fios.

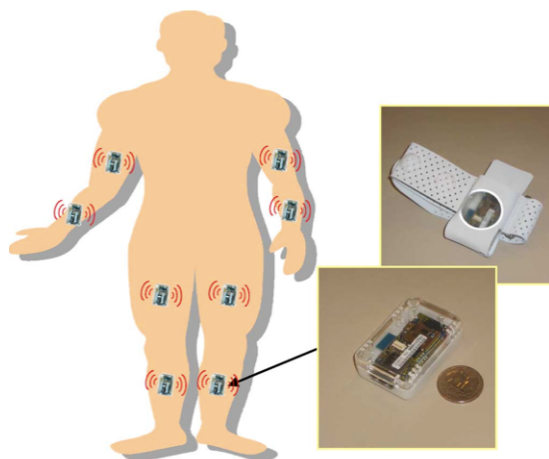


Figura 2.5: Localização dos sensores no estudo de Parkinson

Um desses projectos focou-se em monitorar flutuações motoras em doentes de Parkinson [PLH⁺09]. Usando um rede de sensores com acelerómetro, criaram uma aplicação que fazia a leitura e permitia a análise estatística para fazer o estudo dos sintomas desta

doença.

Através de uma rede com sensores GPS embutidos, é possível criar aplicações para, por exemplo, medir a radiação nuclear em zonas pré-determinadas [DSY⁺09]. Tais aplicações usam sensores especiais para medir a radiação que juntamente com o sensor de GPS podem identificar o valor da radiação e atribuir o local. Com os sensores de GPS é também possível e relativamente mais fácil, criar aplicações de posicionamento e localização [NSBO08].

Aplicações mais simples podem consistir em simples sensores de temperatura que comunicam a sua informação para a estação base. Exemplo disso é uma aplicação em cujos sensores medem a temperatura da apanha do camarão e a base transmite os dados por GSM para um servidor on-line [KC05]. Desta forma é possível fazer uma melhor gestão do produto.

Os exemplos aqui mostrados foram escolhidos por critérios de originalidade e complexidade. Através de trabalhos como estes é possível verificar que existe um grande interesse nesta área.

2.4 Arquitectura geral de uma aplicação RSSF

Apesar de existirem inúmeros projectos e inúmeras variantes de aplicações, é necessário estabelecer pontos comuns entre elas e definir então parâmetros semelhantes para que futuras análises sejam discutidas na mesma base.

O conhecimento mais importante a ter em conta, quando se retrata uma aplicação baseada em redes de sensores sem fios, é a sua arquitectura interna, ou seja, a forma em como ela está organizada.

Em qualquer aplicação baseada em redes de sensores sem fios existem pelo menos 3 camadas. São elas:

Tabela 2.1: As 3 camadas de uma aplicação RSSF

Camada dos Sensores	Camada dos Dados	Camada do Interface
Esta é a camada mais inferior do sistema encarregue de interpretar os sensores nos nós da rede e interagir com eles.	Esta é a camada do meio do sistema (middleware) responsável pelo tratamento dos dados provenientes dos sensores e pela lógica da aplicação.	Esta é a camada mais superior do sistema que trata da transformação dos dados em algo perceptível ao utilizador.

Em muitos projectos e ferramentas, a finalidade, encontra-se apenas na camada dos sensores. Tais projectos têm como objectivo apenas o desenvolvimento das redes em si e a sua manipulação. Para estes projectos, a camada de interface é o menos importante pois ao utilizador interessa apenas a visualização dos dados. A análise posterior será feita por outros métodos ou outras pessoas.

A camada dos sensores é, provavelmente, a mais importante pois é a camada que faz a ponte entre os sensores e o resto da aplicação. Dependendo do tipo de sensores a utilizar, esta camada pode ser a mais complexa ou a mais simples. Normalmente, os aparelhos encarregues de processar o software desta camada em tempo real são simples e não requerem muito poder de processamento.

Se é o desejo do utilizador ter um sistema que dê apoio à decisão então é necessário investir na camada dos dados. Esta será então uma camada que poderá requerer um elevado poder de processamento dependendo dos objectivos da aplicação.

Normalmente, estas camadas encontram-se numa disposição vertical em que a camada dos sensores não consegue comunicar com a camada de interface sem passar pela camada dos dados. No entanto, podem surgir casos em que tal seja necessário havendo então um atalho entre camadas.

Estas 3 camadas são as mais básicas pelo que pode acontecer uma subdivisão de qualquer uma destas camadas em vários níveis ou até a existência de outras camadas extra.

2.5 Outros projectos

Na execução deste documento foram analisados outros projectos de carácter semelhante ao desta dissertação. São projectos com o objectivo de ajudar ao desenvolvimento de aplicações que trabalhem com redes de sensores sem fios. Em seguida é feita uma análise descritiva desses projectos.

2.5.1 CrossBow MoteWorks

O kit de desenvolvimento da CrossBow MoteWorks [Cro07] é uma plataforma de desenvolvimento de redes de sensores sem fios que abrange as 3 camadas gerais de uma aplicação para RSSF.

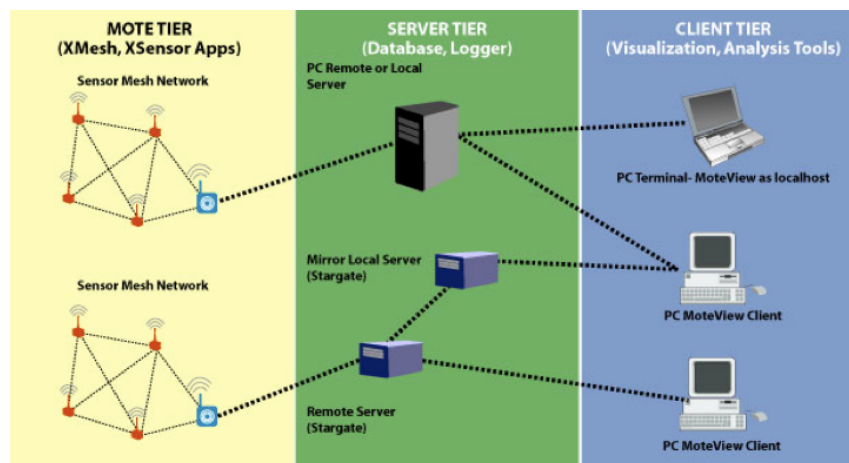


Figura 2.6: Arquitectura do CrossBow MoteWorks

Na primeira camada, a camada dos sensores, encontra-se o XMesh software que é o responsável pelos algoritmos que formam a rede de sensores e as trajetórias de reenca-minhamento das mensagens entre os vários nós.

Na segunda camada, a camada dos dados, encontra-se o software XServe e o XOtap que são os responsáveis pelo processamento dos dados desde a camada dos sensores até à camada de interface. Esta aplicações também se podem incluir na camada dos sensores pois são elas que recebem os dados dos mesmos.

Para a terceira camada, a camada de interface, é fornecido um software de visualização denominado MoteView mas o XMesh pode ser usado com outro tipo de software que terá de ser feito de propósito para o efeito.

2.5.2 Sentilla

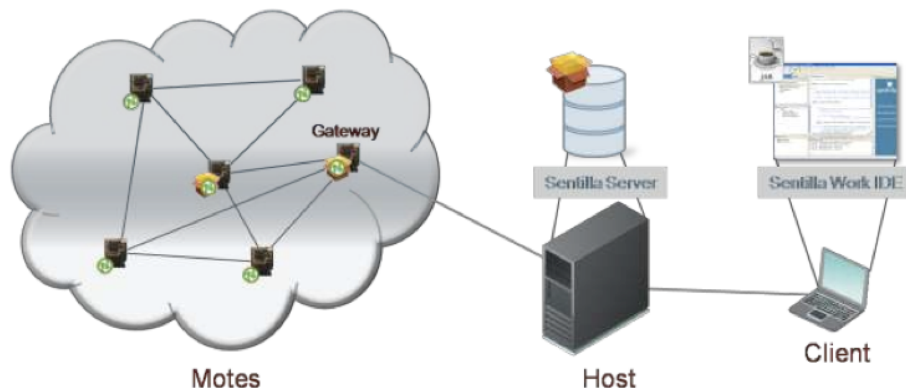


Figura 2.7: Arquitectura de uma rede Sentilla

O kit de desenvolvimento da Sentilla [Sen08] é mais focado para o desenvolvimento e manipulação de uma rede de sensores sem fios do tipo malha. Como tal, incide apenas nas duas primeiras camadas gerais, a camada dos sensores e a camada dos dados. De notar que é na camada dos sensores que este kit de desenvolvimento sobressai. Ele contém um conjunto de APIs que ajudam o utilizador a configurar e extrair os dados da rede. Na camada dos dados, apenas encontramos um servidor que permite a comunicação com esta rede. Para o processamento de dados nas duas camadas superiores é necessário que o utilizador implemente o software necessário, que pode ser feito através do Sentilla Work IDE. Esta ferramenta usa o Eclipse como base para o desenvolvimento geral das aplicações.

2.5.3 ZigBit

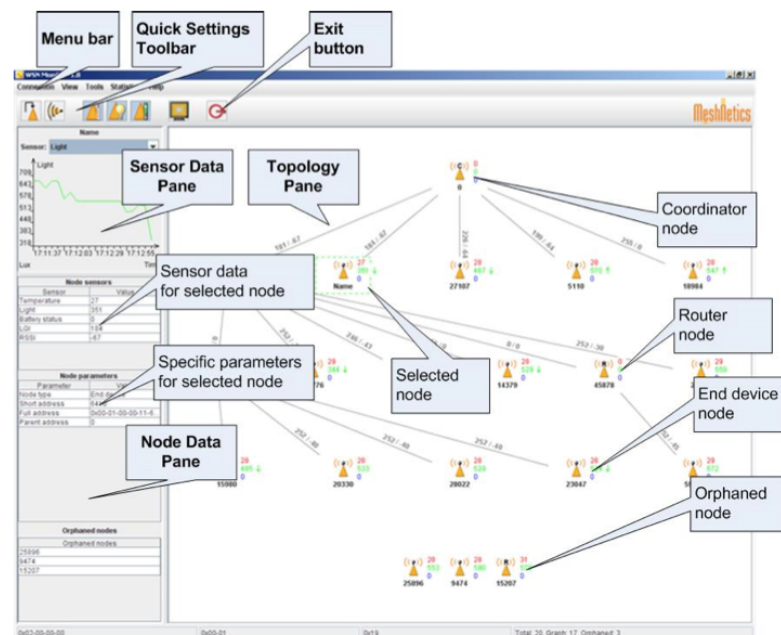


Figura 2.8: WSN Monitor GUI do Kit ZigBit

O kit de desenvolvimento ZigBit [Mes08] é mais uma plataforma de desenvolvimento destinada á camada mais baixa de uma aplicação geral. Contém as ferramentas necessárias para a criação e execução da rede de sensores mas todo o tratamento de dados e a visualização na camada de interface é deixada ao critério do utilizador. No entanto, existe uma ferramenta, a WSN Monitor, que pode ser usada para ver a topologia da rede e outras informações dos sensores como mostra a figura 2.8.

2.5.4 CodeBlue

Esta aplicação abrange as três camadas de uma aplicação geral, no entanto, o seu espaço de desenvolvimento é muito limitado em questões de manipulação de dados. Contém o software necessário para o desenvolvimento da rede e contém uma aplicação em Java pronta para a visualização dos dados. O CodeBlue [SrCL⁺05] está destinado a monitorização de dados vitais e é ainda um projecto em desenvolvimento na Universidade de Harvard.

2.5.5 Octavex

Esta ferramenta é a que mais se aproximada da totalidade dos objectivos assumidos para esta dissertação. A Octavex [OCT06] é uma suite desenvolvida para a implementação nas 3 camadas gerais de uma aplicação para RSSF. No entanto, existem várias aplicações nas duas camadas superiores.

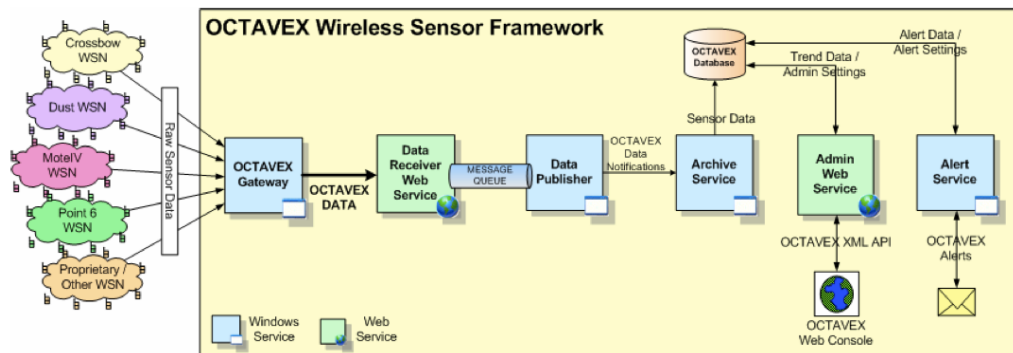


Figura 2.9: Arquitectura da Octavex

Ao contrário dos outros projectos, a Octavex não fornece ferramentas para a programação dos nós da rede de sensores nem para a sua manipulação. Apenas se destina a trabalhar com os dados que eles transmitem.

Na **camada dos sensores** encontra-se a gateway (Octavex Gateway) que é responsável pela interpretação dos dados provenientes dos sensores e transmiti-los para a camada superior.

Na **camada dos dados**, existem 4 aplicações distintas:

- Recepção de dados (Data Receiver Web Service) - é a aplicação responsável pela agregação dos dados provenientes das gateways e pelo seu reencaminhamento para a aplicação de publicação de dados.
- Publicação de dados (Data publisher) - é a aplicação que extrai a informação das mensagens recebidas pela aplicação de recepção de dados e decide as aplicações de destino para essa informação. Para que uma aplicação seja uma aplicação de destino tem de se subscrever junto da aplicação de publicação de dados. Isto é feito através de um sistema de troca de mensagens no formato SOAP XML próprio. É sempre necessário que haja pelo menos uma aplicação subscrita para que o fluxo de dados seja útil.
- Serviço de Arquivo (Archive service) - é a aplicação que recebe notificações e dados da aplicação de publicação de dados e armazena-os para análises futuras.
- Serviço de Alertas (Alert Service) - é uma aplicação que interroga a base de dados com frequência em busca de alterações de estado nas secções de alertas.

Caso haja alguma alteração, esta é reportada para os destinos configurados. Por defeito são alertas por email ou por log na base de dados.

Na **camada de interface** existem duas aplicações:

O serviço de administração permite configurar as aplicações externas que vão utilizar os dados do sistema.

A consola Web permite a interface do utilizador com a framework octavex. Com ela é possível interagir com as motes, ver gráficos de dados, administrar alertas e configurar outras formas de visualização.

2.6 WiSe Framework, versão inicial

Esta Framework encontra-se numa versão inicial e vai ser a base para o desenvolvimento do projecto desta dissertação.

No seu estado inicial, a WiSe Framework já atravessa as 3 camadas gerais de uma aplicação para RSSF. No entanto, ela foi desenhada com o objectivo de se abstrair da rede de sensores e trabalhar apenas com os seus dados. Como tal, não oferece nenhum tipo de aplicação para configuração da rede ou para a programação dos nós.

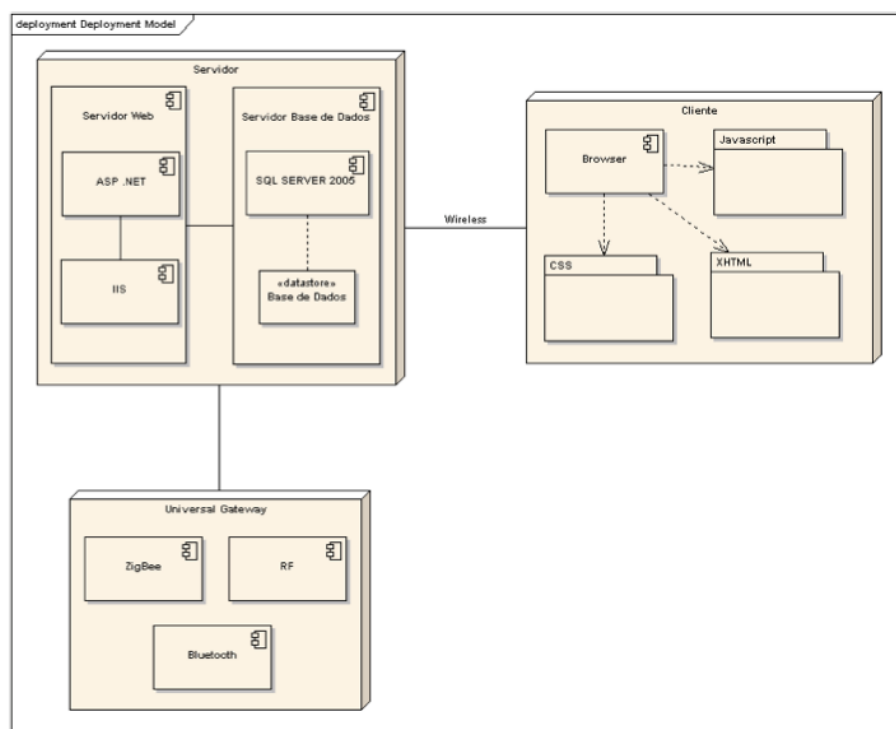


Figura 2.10: Arquitectura física da WiSe Framework

Na **camada dos sensores** existe uma aplicação denominada de Universal Gateway. Esta aplicação consiste em estabelecer a ponte entre os sensores e a camada dos dados.

O dados recebidos pelos sensores são interpretados e convertidos para um formato XML específico da WiSe Framework. Devido à abstracção da rede de sensores, a Universal Gateway consegue trabalhar com virtualmente qualquer rede desde que a recepção dos dados na máquina de destino seja feita através da porta de série ou da sua virtualização.

Para uma interpretação dos dados dos sensores é necessário existir uma pré programação de métodos que permitam interpretar as mensagens recebidas pela Universal Gateway. Esta programação pode ser feita em Java ou C#.

Na **camada dos dados** encontra-se o Main Server que é encarregue da recepção das mensagens provenientes da Universal Gateway, a sua interpretação e o seu reenvio para os workflows correspondentes.

Os workflows são como mini-aplicações programadas sobre a forma de um fluxograma. Estes workflows estão encarregues do processamento dos dados e das consequências dos mesmos. São eles também os responsáveis pelo lançamento de alertas. O uso de workflows dá uma grande expansibilidade a esta framework pois dá ao utilizador a capacidade de usar programar o processamento dos dados sem ter de usar aplicações externas ou aplicações pré-compiladas.

Na camada superior, a **camada de interface**, encontra-se uma aplicação Web que faz o interface entre o utilizador e o sistema. Esta aplicação permite a organização dos vários intervenientes do sistema, permite a visualização dos seus dados, a recepção de alertas, a disponibilização de feeds e a configuração de parâmetros do sistema (limites de valores, associação de sensores, grupos, etc.).

2.7 Comparações

De seguida é feita uma comparação nas categorias em que os trabalhos realizados mais se diferenciam.

Tabela 2.2: Comparações entre os Trabalhos realizados

	Camada de Sensores			Camada de Dados			Camada de Interface	
	Criação da rede	Tecnologias	Envio de comandos aos sensores	Expansibilidade	Armazenamento dos dados	Serviços	Administração	Funções
CrossBow MoteWorks	Sim	ZigBee Mesh Networks, tinyOS	Não	Possível integração com aplicações externas	Não	N.d.	Não	Só visualização do estado da rede
Sentilla	Sim	Sentilla, tinyOS	Sim	Possível integração com aplicações externas	Não	N.d.	Não	Só visualização do estado da rede
ZigBit	Sim	ZigBee Mesh Networks, tinyOS	Não	Possível integração com aplicações externas	Não	N.d.	Não	Só visualização do estado da rede
CodeBlue	Sim	ZigBee Mesh Networks, tinyOS	Não	Não	Sim	Alertas	Pequena administração dos sensores e respectivos pacientes	Visualização de dados, utilizadores, gráficos
OctaveX	Não	Limitada mas com capacidade para expansão	Não	Sim, através da subscrição de aplicações externas.	Sim	Alertas por email ou por evento na base de dados	Parcial	Associação de sensores, parâmetros e alertas
WiSe Framework	Não	Virtualmente qualquer tecnologia através da virtualização da porta de série e configuração de drivers	Sim, sem precisar de os pré programar	Sim, através do uso de workflows	Sim	Alertas, feeds, GPS, possível expansão para serviços externos	Apenas de parâmetros de configuração do sistema e nunca da rede de sensores.	Visualização de gráficos, histórico, criação de comandos, configuração de alertas, etc.

2.8 Resumo e Conclusões

Após o levantamento do estado da arte e de uma análise feita a outros projectos na área, é possível concluir que todos os trabalhos relacionados sofrem do mesmo problema: a capacidade de desenvolver as aplicações finais de uma forma rápida e coerente.

De facto, não existe no mercado nem fora deste, nenhuma ferramenta que consiga suportar completamente o desenvolvimento de aplicação de redes de sensores sem fios, desde a criação da rede até à criação do seu interface final.

Não é, no entanto, objectivo desta dissertação criar uma ferramenta que sirva para o desenvolvimento da rede e dos seus nós. Mas sim, criar uma ferramenta que, aparte disso, sirva para criar tudo o resto de uma forma rápida e dinâmica.

Este será um óptimo método de desenvolvimento, usar um dos kits de desenvolvimento para criar e configurar a rede de sensores e depois usar uma ferramenta para produzir uma aplicação final de forma rápida e eficaz.

De todo este estudo realizado apenas a Octavex e a WiSe Framework chegam aos níveis desejados para esta dissertação. Ambas as frameworks conseguem de certo modo, cumprir aquilo que foi descrito na motivação.

Em questões de hardware ambas estão preparadas para a inclusão de qualquer tipo de sensor. A Octavex já suporta alguns tipos de sensores mas tem um API que permite implementar virtualmente qualquer sensor. A WiSe Framework faz a virtualização das portas de série e como tal, qualquer tipo de mensagem que seja recebida tem a possibilidade de passar para esta Framework através da pré configuração de uma espécie de “driver” (interpretador).

Em questões de processamento de dados, a Octavex contém uma API para a criação de aplicações que depois terão de fazer uma subscrição junto da Framework e assim os dados podem ser processados como o utilizador quiser. A WiSe Framework suporta a tecnologia Windows WorkFlow Foundation que permite ao utilizador estruturar o processamento da informação como um fluxograma, abstraindo assim complicações de programação.

Destas duas, só a WiSe Framework é que oferece uma aplicação final com inúmeras opções de configuração que vai desde os limites que disparam os alertas até à criação de grupos de utilizadores. A Octavex contém um serviço de administração que permite criar tais funcionalidades.

A escolha cai na WiSe Framework por várias razões:

- **Suporta comunicação nos dois sentidos** - É possível configurar mensagens de comando, sem conhecimentos de programação, que serão transmitidas para os sensores. É necessário que os drivers dos sensores suportem esta funcionalidade. Esta funcionalidade é de extrema importância quando queremos, por exemplo, pedir um ECG ou reiniciar o modulo de GPS.
- **Fácil programação dos drivers dos sensores** - Estes drivers consistem na interpretação das mensagens recebidas na porta série (virtual ou não)
- **Implementação de workflows** - Ao contrário da Octavex, a WiSe Framework usa a tecnologia Windows Workflow Foundation para o processamento dos dados na segunda camada. É uma forma mais simples e directa do que a Octavex que requer a necessidade de criar aplicações para esse efeito.
- **Implementação de Serviços externos** - A abstracção da Wise Framework permite implementar serviços externos directamente na aplicação final.
- **Interface local da framework** - a Octavex contém um interface Web para a criação da aplicação, isto requer que toda a rede de sensores esteja a funcionar para ser possível fazer alterações. A WiSe Framework permite criar a aplicação final sem ser necessário que a rede de sensores esteja a funcionar.
- **Alertas e Feeds** - A WiSe Framework suporta feeds para tecnologias móveis e alertas para a aplicação final. Oferece ainda mais opções de administração de alertas que a Octavex.

A WiSe Framework será então a base para o desenvolvimento desta dissertação.

Capítulo 3

WiSe Framework, versão inicial

Como previamente descrito na secção 2.8, a WiSe Framework é uma ferramenta para ajudar ao desenvolvimento de aplicações baseadas em RSSF e foi a ferramenta seleccionada para servir de base para o desenvolvimento desta dissertação.

Ao longo deste capítulo vão ser descritos os conceitos e funcionalidades principais da WiSe Framework cuja importância está directamente relacionada para a compreensão deste documento. Todos os outros conceitos que não forem aqui mencionados podem ser aprofundados através de uma consulta às referências.

A WiSe Framework já suporta as 3 camadas gerais de uma aplicação para RSSF e como tal este capítulo está organizado nessa estrutura. Existem também outras duas secções intituladas de Base de Dados e Framework Wizard que juntamente com as 3 camadas gerais de uma aplicação para RSSF fazem os 5 componentes que a WiSe Framework contém. Porém, inicialmente é feita uma descrição geral da sua arquitectura e uma explicação dos conceitos comuns a todos os 5 componentes.

3.1 Arquitectura Geral

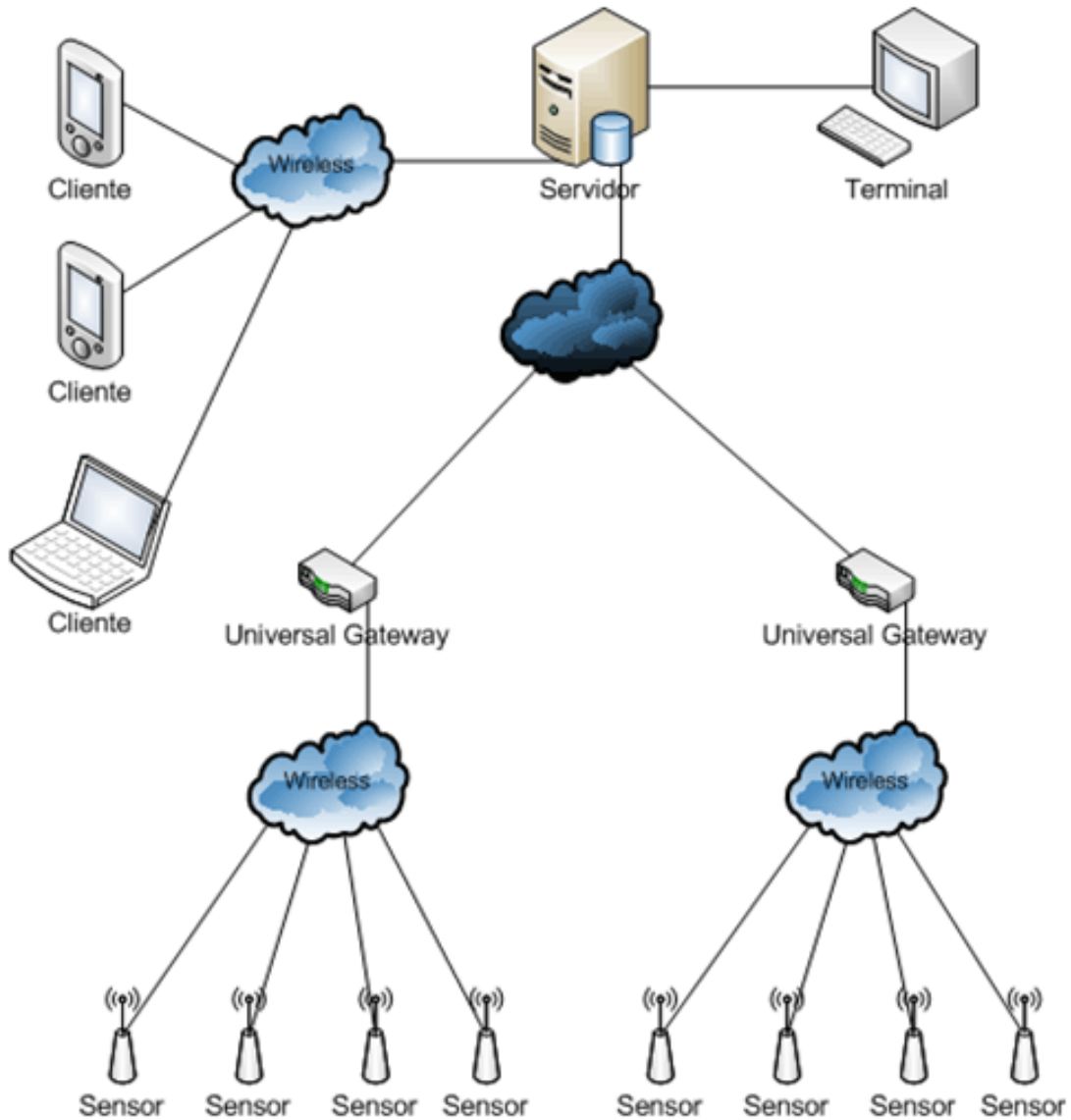


Figura 3.1: Arquitectura Geral de uma aplicação resultante da Wise Framework

A figura 3.1 demonstra a arquitectura de uma aplicação produzida com a WiSe Framework. Estas aplicações vão obedecer as 3 camadas gerais de uma aplicação para RSSF. No geral, os sensores vão comunicar com a Universal Gateway presente na camada dos sensores. Esta vai ficar encarregue de interpretar a informação e comunica-la ao servidor, o WiSeServer, que se encontra na camada dos dados. O WiSeServer por sua vez disponibiliza a informação que poderá ser consultada pela camada de interface. Para tal, o utilizador deverá aceder à aplicação de interface, a WiSeApp.

Na figura 3.1 não é mostrada a componente da Base de Dados nem o Framework Wizard. Mas na figura 3.2 temos um exemplo típico de uma aplicação resultante que foi criado através do Framework Wizard.

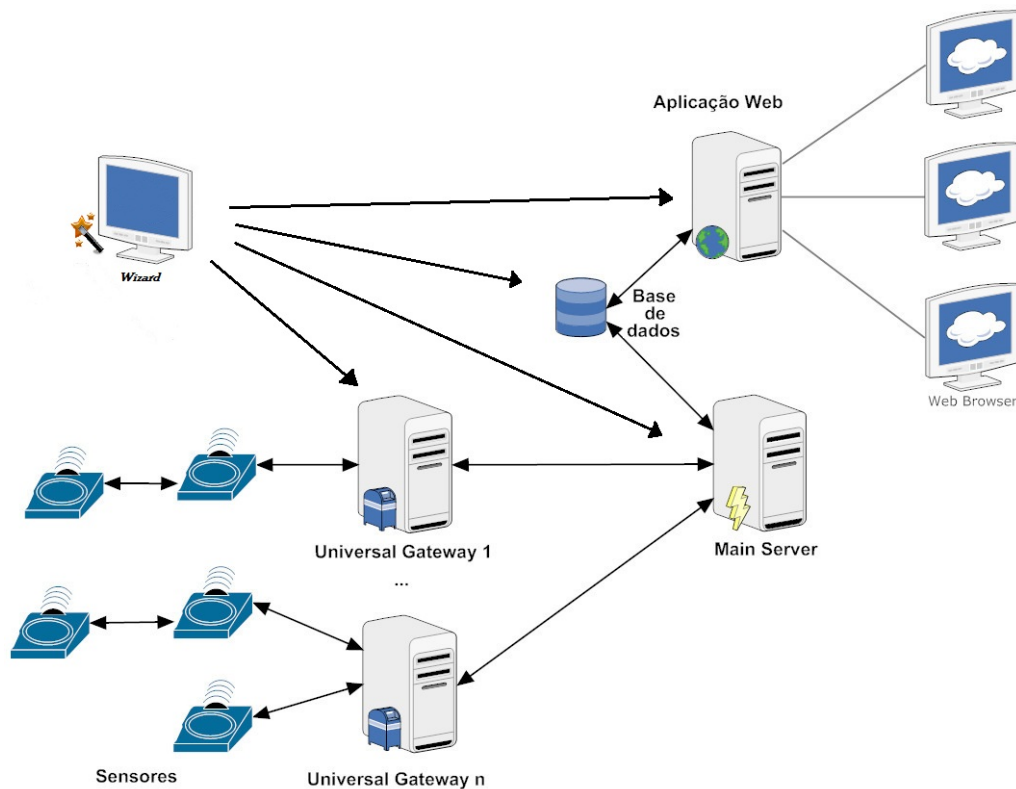


Figura 3.2: Aplicação resultante criada pela WiSe Framework

O Framework Wizard serve para configurar e compilar uma aplicação resultante. A componente da Base de Dados serve para guardar toda a informação que vai circular entre o WiSeServer e a WiSeApp.

3.2 Conceitos comuns

Nesta secção são explicados os conceitos comuns entre os vários componentes da WiSe Framework que vão estar directamente relacionados com esta dissertação.

3.2.1 Sensores

Ao longo deste capítulo é muitas vezes referido a palavra “sensores”. Estes sensores são os pequenos aparelhos referidos na Revisão bibliográfica na secção 2.1.1 que medem valores comuns como a temperatura, humidade, pulsação, etc. Podem também referenciar valores mais complexos como um Electrocardiograma ou até coordenadas GPS.

Mais tarde, estes sensores serão denominados de “serviços” por questões de logística e para uma melhor compreensão da WiSe Framework. A explicação para esta alteração encontra-se posteriormente na secção 4.3.

3.2.2 Parsing de mensagens

Parsing, é acto pelo qual se analisa porções de texto para determinar elementos na sua estrutura. O parsing referido ao longo deste capítulo e também ao longo deste documento é o acto de interpretar a informação recebida por parte dos sensores e transformá-la em algo legível para o ser humano ou para a aplicação em questão.

3.2.3 WiseML

O WiseML é um formato de mensagem em XML, criado exclusivamente para a WiSe Framework que vai circular entre os seus componentes e consiste nos seguintes campos:

- **Id:** valor que identifica o hardware;
- **Tipo:** tipo de sensor (temperatura, GPS, etc);
- **Dados:** valores medidos, separados por “|” (pipe) ou outra condicional, desde que tratada posteriormente;
- **Stamp:** data e hora da leitura.

```

1 <?xml version="1.0" encoding="utf-8"?>
2 <Messages xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
3   <sensor id="3" tipo="SLS" dados="0.00011.00014.000" stamp="18-05-2010 12:33:21" />
4   <sensor id="3" tipo="SLS" dados="1.00010.00013.000" stamp="18-05-2010 12:34:21" />
5   <sensor id="3" tipo="SLS" dados="0.00010.00012.000" stamp="18-05-2010 12:35:21" />
6   <sensor id="3" tipo="SLS" dados="1.00011.00011.000" stamp="18-05-2010 12:36:21" />
7   <sensor id="3" tipo="SLS" dados="0.00011.00015.000" stamp="18-05-2010 12:37:21" />
8 </Messages>

```

Listagem 3.1: Exemplo de um ficheiro WiSeML

3.3 Camada dos Sensores

Nesta secção vai ser descrito os componentes da WiSe Framework relativamente à camada dos sensores.

A camada dos sensores é a primeira meta de interacção entre os sensores e a aplicação resultante. A WiSe Framework contém uma aplicação denominada de Universal Gateway que vai operar apenas nesta camada e cujo objectivo é fazer a ponte entre esta e as outras camadas. Existem também alguns ficheiros de suporte a esta aplicação que serão descritos respectivamente.

3.3.1 Universal Gateway

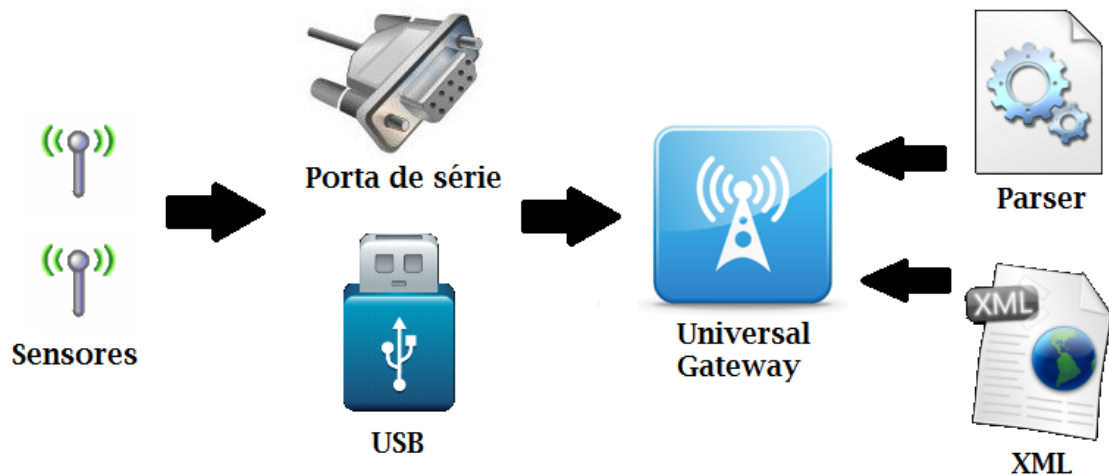


Figura 3.3: Arquitectura Geral da Universal Gateway

A Universal Gateway é uma pequena aplicação que está encarregue da leitura e interpretação dos dados enviados pelos sensores e o seu reencaminhamento para a camada dos dados.

Todos os sensores utilizados são emulados como porta de série. A Universal Gateway apenas se interessa pelas mensagens que são enviadas para estas portas. No seu núcleo, a Universal Gateway é apenas um motor de parsing das mensagens recebidas. Este motor utiliza classes externas denominadas de Parsers que contêm os métodos necessários para a interpretação das mensagens recebidas.

A Universal Gateway também está encarregue da configuração das portas de série e da formatação dos dados em WiseML para posteriormente serem enviados para a camada de dados.

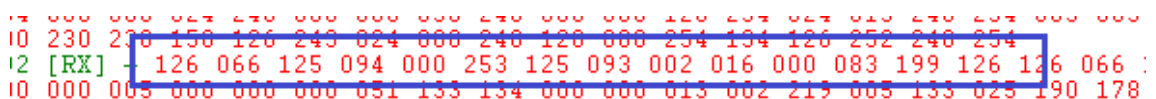
3.3.1.1 Mensagens recebidas

As mensagens recebidas nas portas de série são a fonte de dados principal para a Universal Gateway.

A documentação de suporte da Universal Gateway identifica vários tipos de mensagem a processar. Algumas das características comuns mencionadas são as seguintes:

- Tamanho da mensagem variado;
- Cabeçalho da mensagem diferente do fim.

Os valores dos dados contidos nas mensagens são também afectados pela tecnologia de transmissão das mesmas (Bluetooth, Zigbee, RF), pela velocidade de emissão e pela configuração das portas. Pelo que, qualquer processamento feito às mensagens recebidas tem de ter presente estes conceitos.



```

i0 230 230 150 126 243 024 000 240 120 000 254 134 126 252 240 254
i2 [RX] 126 066 125 094 000 253 125 093 002 016 000 083 199 126 126 066
i0 000 005 000 000 000 051 133 134 000 000 013 002 213 003 133 025 190 178

```

Figura 3.4: Exemplo de uma mensagem recebida

A imagem em cima demonstra o valor decimal de uma cadeia de bytes. O primeiro e o ultimo byte tem o mesmo valor decimal.

3.3.1.2 Métodos de Leitura

Como a principal preocupação da Universal Gateway são as mensagens recebidas e como é impossível de prever o tipo de mensagem a receber, existe então dois métodos de leitura para as mesmas.

- **BIN** - Método que lê uma cadeia de bytes tal como é enviado para a porta de série;
- **ASCII** - Método que lê caracteres á medida que estes vão chegando à porta de série.
A leitura termina quando é lido o carácter de final de linha.

O método ASCII é mais eficaz para a leitura de mensagens do tipo NMEA 0183 (GPS) [Bad] e mensagens recebidas através de bluetooth. Este método é muito eficaz para mensagens recebidas por linha. Por outro lado, o método BIN é muito eficiente para todas as outras leituras. Este método efectua a leitura de todos os bytes até ao comprimento máximo da cadeia de bytes que foi recebida. Apesar de muito eficiente este método provoca alguns erros na leitura que advêm principalmente da linguagem de programação na qual a Universal Gateway foi desenvolvida, C#.

3.3.1.3 Pré-Processamento de Mensagens

Segundo a documentação de suporte da Universal Gateway, a linguagem de programação C# é uma linguagem de alto nível e no decorrer da implementação dos métodos de leitura foram descobertos alguns problemas.

Por vezes, a mensagem recebida era nula ou muito pequena e provocava uma desordem no funcionamento da gateway. Para contornar este problema, a Universal Gateway foi implementada com algumas restrições:

- Nenhuma mensagem nula é processada;
- Nenhuma mensagem muito pequena é processada;
- Todas as outras mensagens terão que ser validadas no pós-processamento.

Qualquer mensagem que não seja apanhada por estas restrições será processada.

3.3.1.4 Parser

Após a recepção de uma mensagem válida no pré-processamento, a Universal Gateway precisa de processar os seus dados. Como podem existir uma imenso conjunto de sensores, a primeira tarefa é identificar o tipo de sensor emissor. Conhecendo o emissor, a Universal Gateway processa a mensagem e os seus dados, criando uma nova mensagem no formato WiseML e envia o resultado para a camada de dados.

Todas estas tarefas estão dependentes de métodos específicos a cada tipo de sensor. Estes métodos devem ser implementados numa classe própria para o efeito. O motor de parsing da gateway irá usar esta classe e os seus métodos para interpretar as mensagens recebidas. Os métodos incluídos nesta classe deverão ser implementados por um programador que conheça a estrutura das mensagens a receber para que estas sejam bem interpretadas. A esta classe dá-se o nome de Parser. Cada tipo de Hardware deve ter associado pelo menos um Parser.

Cada sensor precisa de ter 3 métodos para fazer o parsing da sua mensagem:

- **Validação:** método encarregue de validar a mensagem caso ela seja do seu sensor;
- **Identificação:** método encarregue de ler a mensagem e identificar a mote/nó/hardware emissor;
- **Recolha de dados:** método encarregue de ler a mensagem, recolher os seus dados e se necessário processá-los.

Estas deverão ser as únicas funções que um programador deverá desenvolver. No entanto, segundo a documentação de suporte, o programador pode criar funções extra para ajudar ao processamento destas.

Existe ainda a possibilidade de implementar um método extra para a criação de mensagens de comando a ser enviadas ao hardware dos sensores. Não é possível prever todos os comandos ou a sua funcionalidade, como tal, este método fica a cargo do programador. A informação que é fornecida para este método através da Universal Gateway é:

- **ID de destino** - Vai identificar qual o hardware que deve receber o comando;
- **Um parâmetro** - Este parâmetro pode ser qualquer informação. É necessário que o programador tenha particular atenção nesta função para que haja coerência com os comandos suportados pelo hardware.

Julga-se que estes dois campos são mais que suficientes para processar toda e qualquer mensagem de comando.

3.3.1.5 Sensores.xml

Ficheiro XML gerado automaticamente pelo Framework Wizard cuja funcionalidade é conter a informação necessária para que a Universal Gateway conheça os sensores que lhe estão associados.

```

1 <?xml version="1.0" encoding="utf-8"?>
2 <ListaSensores xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
3   <sensor nome="Temperature"
4     protocolo="RF"
5     dll="XBOW"
6     descricao="Temperature"
7     foto="foto"
8     isSelected="false"
9     validacao="is_Temperature"
10    id="get_sensorID"
11    dados="getTemp_XBow"
12    request="" />
13 </ListaSensores>

```

Listagem 3.2: Exemplo de um ficheiro sensores.xml

Este ficheiro contém as seguintes *tags*:

- **Nome:** corresponde ao tipo do sensor;
- **Protocolo:** é a tecnologia de comunicação. Não afecta em nada o funcionamento e serve apenas para informação;
- **DLL:** é o nome do ficheiro que contém o Parser do sensor;
- **Descrição:** é apenas uma descrição associada ao sensor;

- **Foto:** contém o caminho para a foto associado ao sensor e apenas serve para fins ilustrativos no Framework Wizard e não na aplicação resultante;
- **isSelected:** informação para o Framework Wizard;
- **Validação:** nome do método do Parser que vai testar a validação da mensagem;
- **id:** nome do método do Parser que vai retornar o id da mensagem;
- **Dados:** nome do método do Parser que vai retornar os dados contidos na mensagem;
- **Request:** nome do método do Parser que vai criar mensagens de comando a enviar aos sensores.

3.3.1.6 Portas.xml

Ficheiro XML gerado automaticamente pela Universal Gateway cuja funcionalidade é conter a informação necessária para configurar as portas e o métodos de leitura respectivo.

```
1 <?xml version="1.0" encoding="utf-8"?>
2 <ListaPortas xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
3   <porta porta="COM5"
4     baudrate="57600"
5     paridade="None"
6     dataBits="8"
7     stopBits="Two"
8     tipoLeitura="BIN"
9     reserved="false"
10  </ListaPortas>
```

Listagem 3.3: Exemplo de um ficheiro portas.xml

Este ficheiro contém os seguintes campos:

- **Porta:** nome da porta;
- **Baudrate:** velocidade de transferência dos dados. Este campo vai influenciar a interpretação das mensagens recebidas;
- **Paridade:** este campo corresponde à paridade associada à porta;
- **dataBits:** este campo corresponde aos databits associados à porta;
- **stopBits:** este campo corresponde aos stopbits associados à porta;
- **tipoLeitura:** este campo define o método de leitura que a Universal Gateway vai utilizar. BIN ou ASCII;
- **Reserved:** se este campo estiver a true, a gateway ignora esta porta.

3.3.2 Funcionamento Geral da Universal Gateway

Nesta secção é descrito o funcionamento geral da Universal Gateway.

3.3.2.1 Inicialização

No arranque, a Universal Gateway inicializa as portas e os sensores conforme os ficheiros portas.xml e sensores.xml respectivamente. É feito também a instalação de um handler em cada porta activa para a recepção das mensagens. É criada uma sessão junto do servidor na camada de dados para o processamento das mensagens de comando. A partir desta etapa, o processamento fica a cargo do handler que foi instalado nas portas e dos eventuais comandos recebidos por parte do servidor.

3.3.2.2 Data_port Handler

Este handler é o responsável pelo processamento das mensagens. As suas funções consistem no seguinte:

1. Identificar a sua porta e o tipo de leitura;
2. Ler a mensagem;
3. Executar o motor de parsing com o Parser respectivo;
4. Enviar resultados para o método WiSeMICreator.

3.3.2.3 Parsing das mensagens

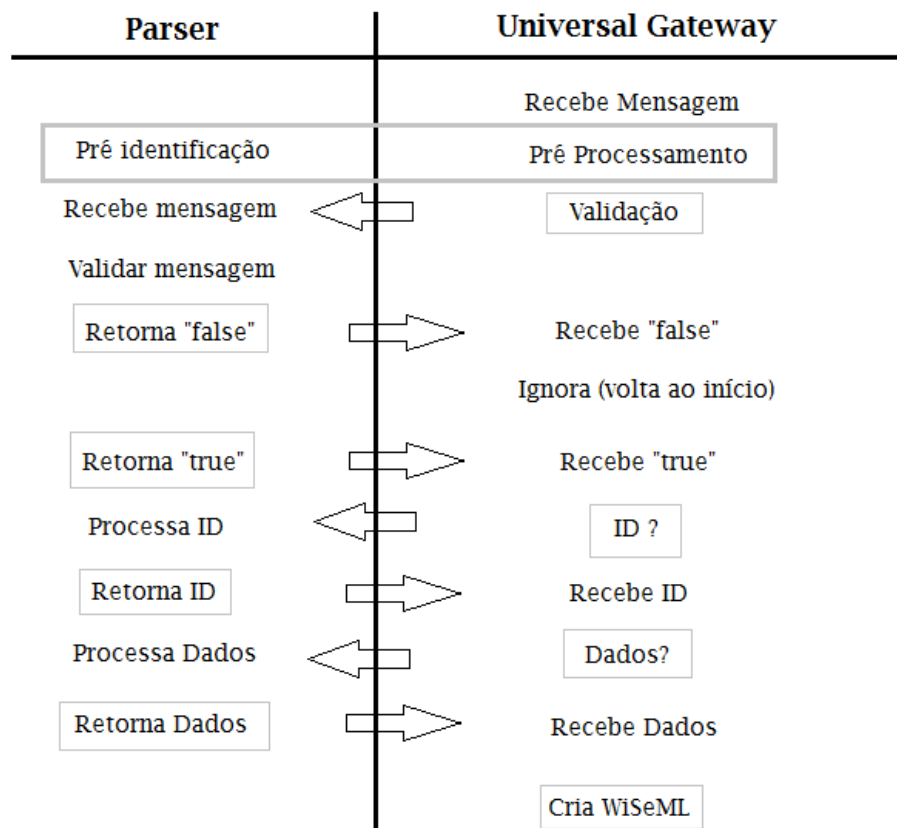


Figura 3.5: Diagrama de Actividade - Parsing

A figura 3.5 demonstra as actividades que ocorrem entre a Universal Gateway e o parser aquando da altura de executar o parsing a uma mensagem recebida.

3.3.2.4 WiSeMLCreator

O WiSeMLCreator é a função que a Universal Gateway usa para formatar os resultados obtidos da interpretação das mensagens em WiSeML e posteriormente envia-los para o servidor na camada de dados.

3.3.2.5 Requests

A Universal Gateway contém um handler para os comandos que chegam da camada de dados. Este handler está encarregue de processar estes comandos com a ajuda do Parser respectivo ao sensor e escrever o comando na porta associada também.

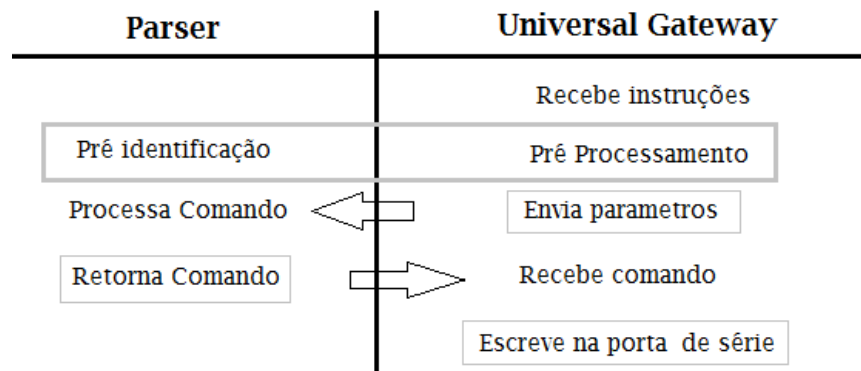


Figura 3.6: Diagrama de Actividades - Comandos

A figura 3.6 demonstra as actividades que ocorrem entre a Universal Gateway e o parser aquando da altura de processar um comando para o hardware.

3.4 Camada dos Dados

Nesta secção vão ser descritos os componentes da WiSe Framework relativamente à camada dos dados.

A camada dos dados é a segunda meta de interacção entre os sensores e a aplicação resultante. A WiSe Framework contém uma aplicação denominada de WiSeServer que vai operar apenas nesta camada e cujo objectivo é fazer a ponte entre a Universal Gateway presente na camada dos sensores e a Base de Dados. Existem também alguns ficheiros de suporte a esta aplicação que também serão descritos.

3.4.1 WiSeServer

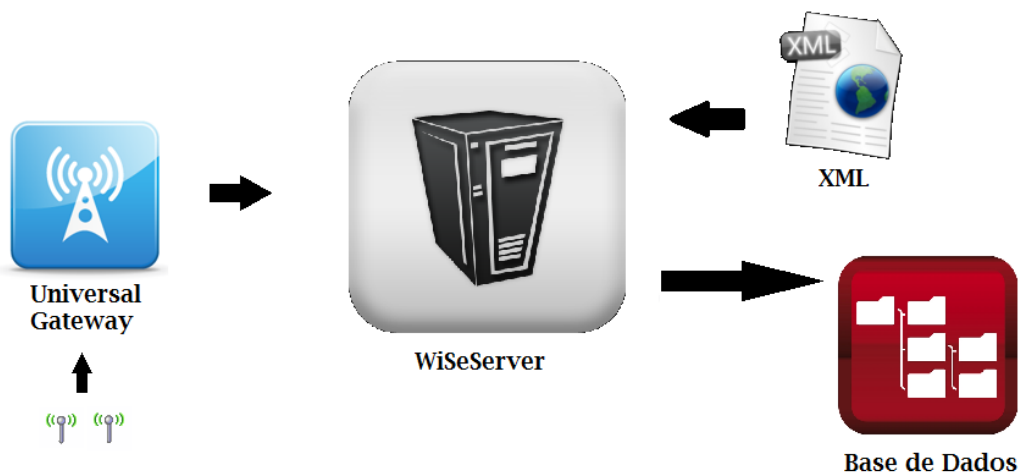


Figura 3.7: Arquitectura Geral do WiSeServer

Existente na camada de dados encontra-se o WiSeServer. Este é um servidor que oferece os serviços necessários ao funcionamento da aplicação resultante e é provavelmente a peça fundamental de todas as aplicações resultantes do uso da WiSe Framework.

Este WiSeServer é a camada intermédia entre os sensores e a visualização dos dados dos mesmos. O WiSeServer também está encarregue do processamento de workflows. Estes workflows fazem parte da componente Windows Workflow Foundation [Micd] presente na .NET Framework [Micc] e são descritos com maior detalhe na secção 3.4.1.3.

A escolha deste tipo de tecnologia deveu-se á facilidade de desenvolvimento dos workflows que a .net Framework oferece. E assim, também será possível expandir o uso da WiSe Framework no sentido em que qualquer programador pode programar a sequência de processamento dos dados adquiridos na camada inferior.

Após toda a informação resultante do processamento estar válida, não é impressa no ecrã mas sim, guardada na Base de Dados. Esta técnica permite uma enorme compatibilidade com qualquer tipo de visualização de dados. Na camada de Interface, a aplicação encarregue dessa função, tem apenas de lidar com a Base de Dados, até mesmo, para interações com o hardware.

3.4.1.1 Mensagens Recebidas

As mensagens recebidas por parte do WiSeServer vêm no formato WiseML. Este formato permite encapsular a informação necessária para o bom funcionamento de toda a aplicação resultante.

3.4.1.2 Web Services

O WiSeServer disponibiliza alguns serviços para o funcionamento da aplicação.

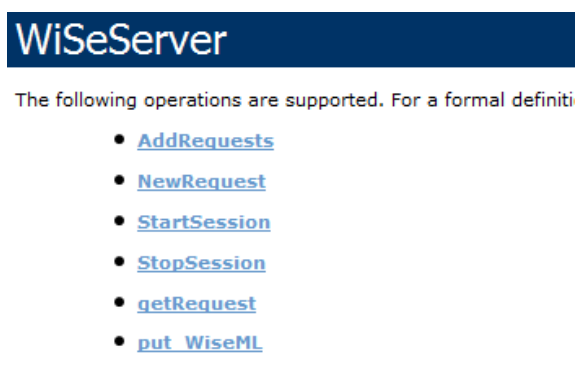


Figura 3.8: Serviços disponibilizados pelo WiSeServer

- **put_WiSeML** - O serviço encarregue de receber as mensagens chama-se: put_WiSeML. A sua função principal é a decodificação das mensagens recebidas em parâmetros que possam ser usados pelo resto da aplicação, mais propriamente, pelo workflow associado. Este método também se encarrega de fazer uma triagem inicial dos dados e chamar o workflow associado com os mesmos;
- **getRequest** - Este é o serviço que a Universal Gateway vai utilizar que retornar os pedidos que tem de fazer aos sensores;
- **StopSession** - Este serviço serve para a Universal Gateway terminar a sua sessão junto com o WiSeServer;
- **StartSession** - Este serviço serve para a Universal Gateway iniciar a sua sessão junto com o WiSeServer;

- **NewRequest** - Este serviço é usado pela WiSeApp para criar um novo comando para os sensores;
- **AddRequests** - Este serviço é usado pela WiSeApp para adicionar um novo comando para os sensores.

3.4.1.3 Workflows

Os Workflows são o “core” do processamento de todos os dados no WiSeServer. Estes consistem numa forma simples e intuitiva de programar baseada em fluxogramas. A sequência do fluxograma é bastante fácil de criar e o código a incluir é bastante rápido. Este é o conceito do Windows Workflow Foundation [Micd].

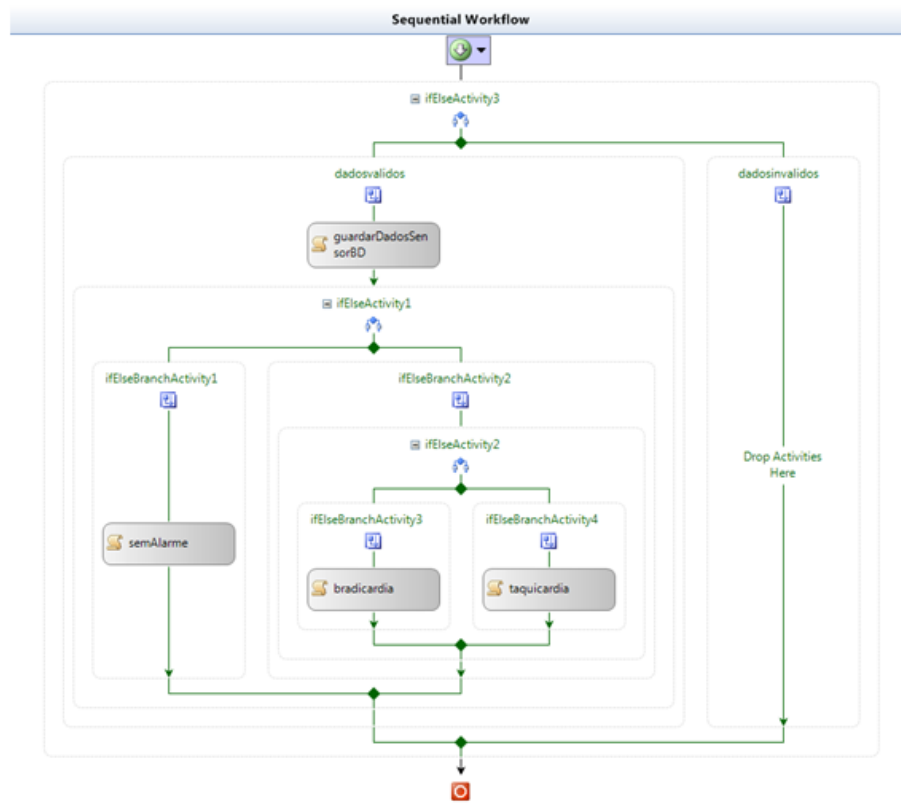


Figura 3.9: Diagrama de um Workflow Sequencial

Cada tipo de hardware ou tipo de sensor, deve ter um workflow associado. A associação é feita antes da compilação da aplicação resultante. Este pode e deve ser programado por quem sabe quais os objetivos finais dos dados em questão.

A WiSe Framework já suporta o lançamento de alertas para valores editáveis. Estas funções já estão implementadas nos workflows existentes.

3.4.2 Funcionamento Geral do WiSeServer

Nesta secção é descrito o funcionamento geral do WiSeServer.

3.4.2.1 Inicialização

Ao iniciar o WiSeServer faz o carregamento de todos os parâmetros que foram programados pelo Framework Wizard. Nomeadamente, inicializa o tipo de sensores a processar, os seus workflows, os seus limites, os alertas e as funções de comunicação com as camadas adjacentes.

3.4.2.2 Recepção de mensagens

O servidor recebe mensagens através da função `put_WiSeML`. Aquando da recepção da mesma, esta função vai-se encarregar de decodificar o seu conteúdo, identificando o workflow de destino para os dados. De seguida, é feito alguns testes para verificar a veracidade da mensagem e se corresponde ao workflow de destino. Caso a mensagem que passe nestes testes é enviada para o workflow, caso contrário, é descartada.

Para prevenir sobre carregamento da Base de Dados, o reencaminhamento dos dados só é iniciado quando um determinado sensor for associado a um agente. Esta associação é feita na camada do interface, a aplicação WiSeApp.

3.4.2.3 Processamento de Workflows

O processamento presente num workflow é deixado a cargo do programador, no entanto, existem funcionalidades genéricas que têm de acontecer.

1. **Tratamento dos dados** - os dados recebidos devem ser tratados de alguma maneira. É aqui que se deve fazer as principais computações;
2. **Armazenamento na Base de Dados** - os dados devem ser armazenados a uma dada altura;
3. **Lançamento de Alertas** - No caso de existirem, os alertas também têm que ser lançados pelo workflow.

Qualquer outro tipo de computação extra pode existir e fica ao critério de programador.

3.4.2.4 Armazenamento na Base de Dados

Todo o tipo de resultados provenientes de um workflow serão posteriormente armazenados. Actualmente, estamos a falar de valores dos sensores e de variáveis de alertas mas no futuro, poderão existir outro tipo.

3.5 Base de Dados

Todos os dados necessários à execução da aplicação, especialmente por parte da WiSeApp, encontram-se na Base de Dados.

3.5.1 O Generator

O generator é um pequeno programa criado automaticamente pela WiSe Framework com a finalidade de construir a estrutura da Base de Dados que vai suportar as configurações da aplicação resultante.

3.5.1.1 Exemplo de Base de Dados

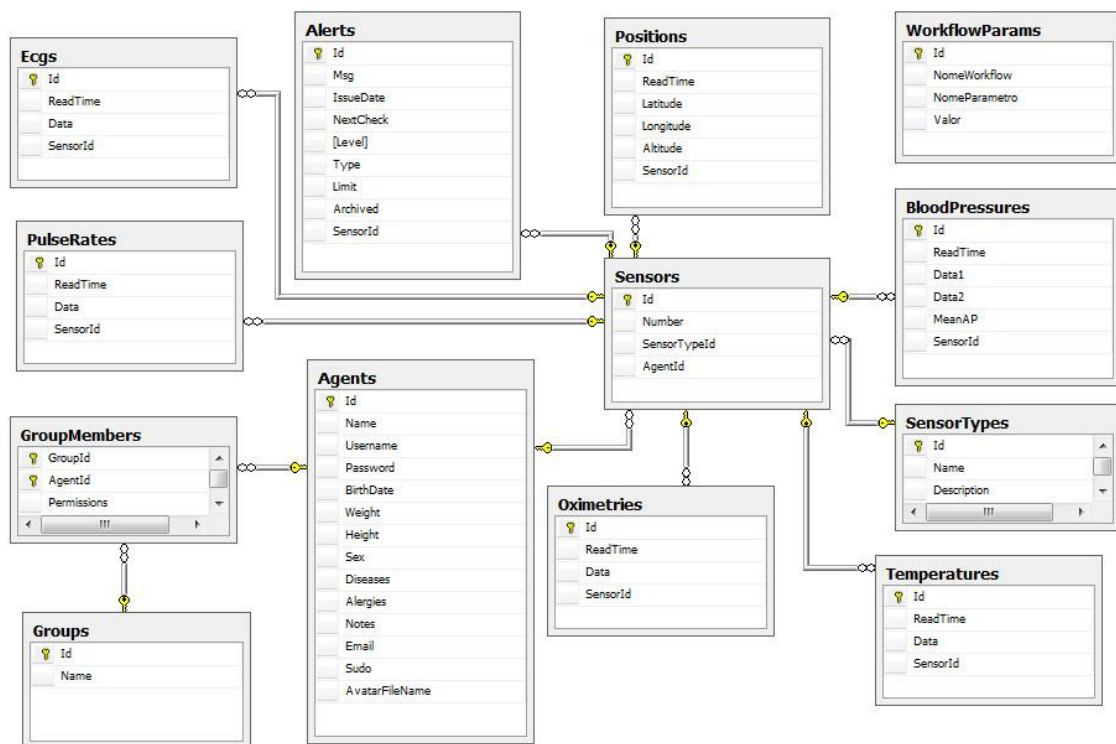


Figura 3.10: Exemplo da estrutura de Base de Dados de uma aplicação resultante

Na figura 3.10 podemos ver ali alguma personalização no tipo de sensores escolhidos como a Pulsação e Pressão sanguínea. Este é o tipo de estrutura que uma aplicação resultante tem quando criada através da WiSe Framework.

3.5.2 Funcionamento Geral do Generator

O funcionamento do generator é simples. Inicialmente faz o carregamento dos parâmetros todos que necessita. E de seguida cria a base de dados baseado no script que a WiSe Framework criou automaticamente.

3.6 Camada do Interface

Nesta secção vão ser descritos os componentes da WiSe Framework relativamente à camada do interface.

A camada do Interface é a ultima meta de interacção entre os sensores e a aplicação resultante. A WiSe Framework contém uma aplicação denominada de WiSeApp que vai operar apenas nesta camada e cujo objectivo é fazer a ponte entre a Base de Dados e o utilizador final. Existem também alguns ficheiros de suporte a esta aplicação que serão descritos respectivamente.

3.6.1 WiSeApp

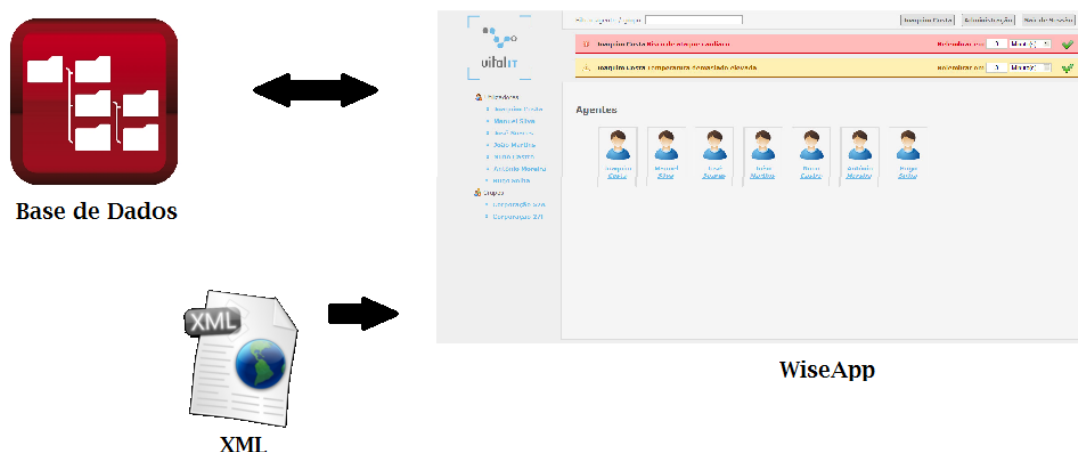


Figura 3.11: Arquitectura Geral da WiseApp

A WiSeApp desenvolvida foi inicialmente desenvolvida com base num caso prático de monitorização de equipas de emergência em missões de busca e salvamento. Contém um conjunto de funcionalidades que permitem a monitorização remota de utilizadores no terreno, sendo possível a visualização da sua posição e sinais vitais.

A WiSeApp foi desenhada com este propósito mas as suas capacidades vão um pouco além. É possível agrupar utilizadores e também é possível editar os parâmetros que vão influenciar o processamento do workflow de cada sensor. É também possível a programação de comandos que irão ser enviados de volta aos sensores.

No geral, a WiSeApp é uma aplicação bastante robusta mas o seu propósito é apenas a monitorização de sinais vitais em gráficos lineares.

3.6.1.1 Ficheiros de Tradução

O carregamento dos ficheiros de tradução é necessário na medida em que a WiSeApp foi desenvolvida de maneira a conter o menor texto possível, para fomentar a reutilização de código.

O ficheiro “Translations.po” contém informações relativas a certas “strings” que aparecem na WiSeApp como por exemplo o que escrever caso encontre o campo “nome”. Isto faz com que funcione com todo o tipo de ficheiros sem ser preciso modificar os ficheiros de cada vez que se mudam os nomes dos campos correspondentes.

3.6.1.2 WiseConfig.xml

O ficheiro “WiseConfig.xml” contém as informações que a WiSeApp supostamente deve pedir ao utilizador, tais como a altura ou o peso e ainda a informação sobre os sensores a disponibilizar. Esta informação é editada no Framework Wizard e criada automaticamente pela WiSe Framework. Se esta informação não existir, a aplicação dá erro de página não encontrada.

3.6.1.3 Arquitectura MVC

É um padrão de arquitetura de software que visa separar a lógica de negócio da lógica de apresentação, permitindo o desenvolvimento, teste e manutenção isolado de ambos [Micb].

A aplicação é feita em ambiente MVC, de forma a otimizar ao máximo a reutilização de código. Toda a estrutura e o tipo de informação para cada entidade que a WiSeApp e o WiSeServer vão utilizar fazem parte dos “models”. Os “views” contêm o layout da página actual e os “controllers” são a componente encarregue de fazer as operações e retornar valores da Base de Dados de acordo com a estrutura dos “models”.

3.6.1.4 AJAX

É o uso metodológico de tecnologias como Javascript e XML, providas por navegadores, para tornar páginas Web mais interativas com o usuário, utilizando-se de solicitações assíncronas de informações [Mica].

A aplicação efectua pedidos AJAX de modo a evitar fazer um refrescamento da página inteira. Isto é, é apenas feito o refrescamento das secções de página que contêm dados que necessitam de ser mostrados em tempo real.

3.7 Framework Wizard

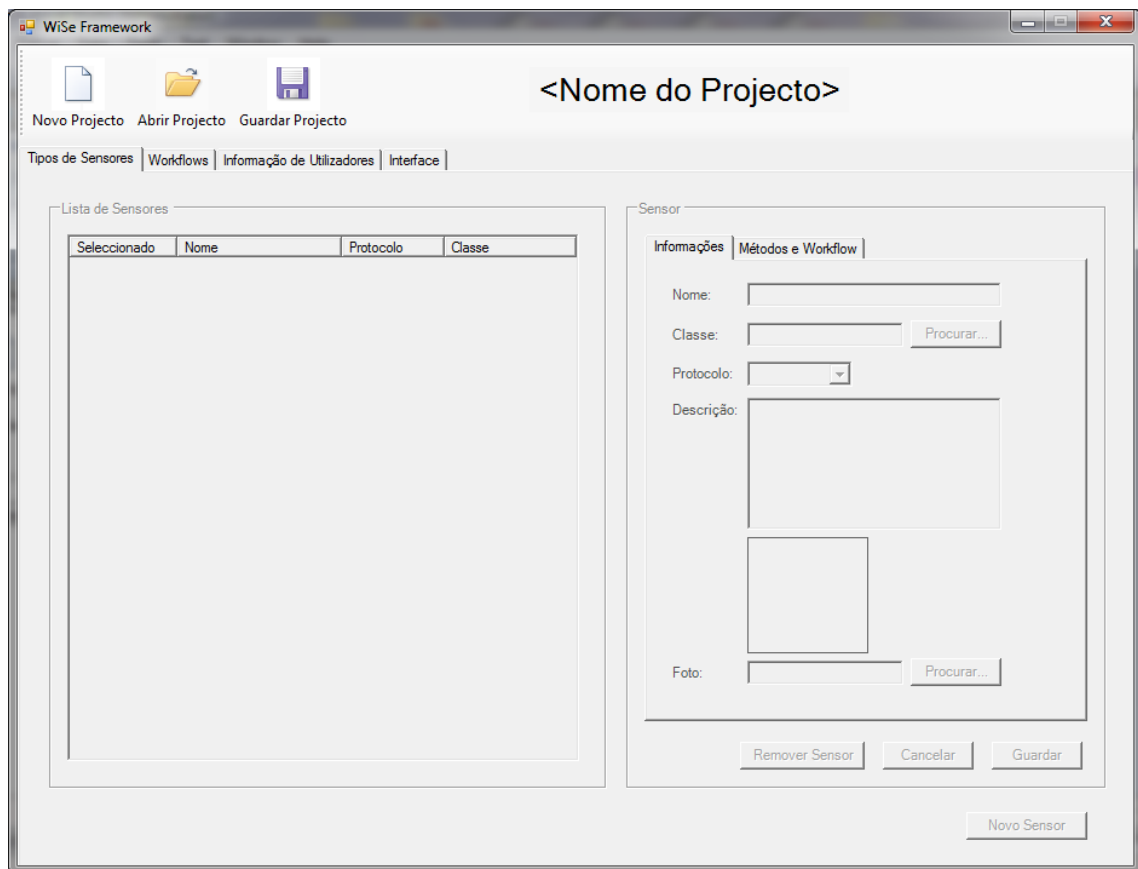


Figura 3.12: Interface do Framework Wizard

3.7.1 O que é?

O Framework Wizard é apenas um programa com a aparência de um configurador. Este programa serve para fazer algumas configurações que podem ir desde a importação de novos ficheiros até à simples mudanças de nome de um sensor.

3.7.2 Para que serve?

O Framework Wizard serve para configurar todos os parâmetros referentes a um projecto associado a RSSF dentro da WiSe Framework. É possível configurar, alterar, compilar, mudar parâmetros, adicionar parsers, workflows, etc.

3.7.3 Funcionamento Geral do Wizard

O seguinte texto foi extraído da documento de suporte da WiSe Framework:

“Aquando da criação de um novo projecto, o Framework Wizard dá ao utilizador a escolha do directório onde guardar o projecto; caso seja criado um novo directório, o projecto fica automaticamente com esse nome. O Framework Wizard cria então um ficheiro de extensão .wise, com o mesmo nome que o projecto. Este ficheiro .wise não é nada mais que um ficheiro XML vazio.

Quando se efectua uma alteração no projecto, o Framework Wizard actualiza automaticamente o ficheiro XML, adicionando as tags conforme os detalhes alterados.

Quando se cria um workflow novo, o Framework Wizard cria um ficheiro de extensão .cs e o nome do ficheiro contém as letras WF (workflow) e o nome do projecto, juntamente com um ficheiro do Designer e adiciona-os ao projecto de workflows.

Para visualizar o projecto, é necessário gravar o projecto em primeiro lugar.

Para guardar o projecto, a Framework compila o ficheiro wiseapp.csproj e workflows.csproj juntamente com a parte relativa à Universal Gateway (compilada à parte com o compilador de C#, uma vez que é mais rápido desta maneira que adicionar a um projecto e compilar).” [Vit09]

A citação em cima demonstra que o Framework Wizard funciona como um configurador de parâmetros para que a aplicação resultante suporte as opções pretendidas.

3.8 Conclusão

O conjunto total dos 5 componentes da WiSe Framework forma uma forte ferramenta de desenvolvimento de aplicações para RSSF. De facto, o suporte para as 3 camadas gerais de uma aplicação para RSSF é fulcral neste aspecto e as funcionalidades extra do Framework Wizard e do generator na componente da Base de Dados aumenta o valor da WiSe Framework.

No entanto, é fácil de identificar algumas melhorias que podem ser realizadas, principalmente ao nível da WiSeApp.

A WiSe Framework foi a base de desenvolvimento escolhida pela empresa autora desta dissertação e foi uma boa escolha por todos os pontos referidos até agora. Nos próximos capítulos vai ser apresentado o porque da WiSe Framework ainda não corresponder ao objectivo desta dissertação e os passos necessários para lá chegar.

Capítulo 4

Apresentação e descrição do problema

Neste capítulo vão ser descritos todos os problemas encontrados ao tentar fazer uma aplicação baseada em RSSF utilizando a versão inicial da WiSe Framework.

4.1 O acto de criação de aplicações para RSSF

Ao iniciar o desenvolvimento de uma aplicação baseada em RSSF, o programador vê-se a par com imensas dificuldades.

Numa aplicação tradicional deste género, a ideia principal de desenho deverá algo como a seguinte:

- Ler a informação dos sensores;
- Tratar os dados;
- Apresentar resultados.

No entanto, o programador terá que fazer compromissos entre vários assuntos como a eficiência da aplicação, o tempo de desenvolvimento, o custo associado, etc. As suas decisões vão ser afectadas pelo tipo de hardware a ser suportado, pelas funcionalidades desejadas, pelas tecnologias a utilizar, entre outros. Aliado a esta composição, cada aplicação tem de ser criada de raiz pois na maior parte das vezes o custo de uma adaptação é superior.

Não existe um qualquer método de desenvolvimento para a criação destas aplicações. O programador vê-se então com mais uma dificuldade a ultrapassar.

É também do senso comum que grande parte dos projectos sofrem alterações a meio do seu desenvolvimento. Ainda que estas alterações seja de pouca relevância, podem reflectir enormes modificações em toda a estrutura da aplicação resultante aumentando assim o tempo de desenvolvimento.

Este tipo de abordagem é lenta, difícil e as aplicações que resultam deste processo, apesar de potencialmente muito eficientes e com uma taxa de erros muito baixa, são obviamente muito limitadas em todas as suas funcionalidades e expansibilidade. Nomeadamente, foram desenhadas para um pequeno conjunto de funções e para um hardware específico.

4.2 Utilização da WiSe Framework, versão inicial

A WiSe Framework foi desenhada com outro objectivo diferente do tópico anterior mas a ideia principal das aplicações resultantes está presente e estruturada nas 3 camadas gerais de uma aplicação para RSSF na tabela 2.1:

O objectivo principal da WiSe Framework é o desenvolvimento de aplicações que vão incidir nestas 3 camadas. Estas aplicações não são previsíveis nem dirigidas a um hardware específico. Também não é sabido qual ou quais as funcionalidades que estas aplicações têm de corresponder. Como tal, a WiSe Framework é mais complexa em toda a sua arquitectura para que a sua utilização final seja muita mais fácil e rápida, tentando corresponder aos objectivos principais do seu desenvolvimento:

Uma ferramenta ágil que permita o desenvolvimento rápido de aplicações que assentam em tecnologias de redes de sensores sem fios.

Apesar da WiSe Framework conter características que levaram à sua selecção de entre os outros projectos disponíveis e apesar de já contribuir em parte para o objectivo desta dissertação, possui ainda grandes falhas que foram detectadas nos primeiros instantes da utilização da mesma.

Para a averigação de todos os problemas foi necessário ter em conta todos os tipos de utilização e todos os tipos de operações possíveis de realizar. Para tal foi realizado um teste de utilização que consistiu em criar uma aplicação genérica de raiz para medições de um tipo de sensor novo utilizando a WiSe Framework.

O hardware de sensores escolhido também foi um hardware nunca antes utilizado, as motes da Crossbow. O sensor escolhido foi o sensor de Amplitude de Som. A aplicação final deverá ter as características genéricas de uma aplicação deste género.

Os passos realizados durante esta tarefa não são importantes para o resto do capítulo mas podem ser encontrados no anexo A.1 deste documento.

Após a conclusão desta tarefa foram apontadas diversas falhas no método de utilização. Estas falhas devem-se maioritariamente às dependências impostas. Foram então identificadas as 3 principais dependências. Uma em cada camada:

4.2.1 1ª dependência: O parsing das mensagens

Um dos pilares principais da WiSe Framework encontra-se na camada dos sensores e é a capacidade de interpretar as mensagens que são enviadas pelos sensores. Para que isso seja possível terá de existir uma classe respectiva que fará o parsing dessas mensagens em algo útil para a Aplicação.

Essa classe está à responsabilidade do programador. Ele é que fica encarregue de programar os métodos correspondentes de acordo com o hardware utilizado. Para resolver este problema, é mesmo necessário que exista essa classe com os métodos correctos.

Às classes desse tipo deram-se o nome de Parsers [3.3.1.4](#). Pelo que, deverá existir pelo menos um Parser para cada tipo de sensor.

4.2.2 2ª dependência: O processamento dos dados

Na camada dos dados da WiSe Framework existe um outro pilar principal, desta vez, é a sua capacidade de processamento de dados. Este processamento fica encarregue a um workflow [3.4.1.3](#) da tecnologia Windows Workflow Foundation.

Este workflow também está à responsabilidade do programador. Ele é que está encarregue de desenvolver o workflow de acordo com os resultados esperados. O workflow também é o responsável pela inserção dos dados na Base de Dados.

Terá sempre que existir um workflow para cada tipo de sensor independentemente da carga de processamento dos dados.

4.2.3 3ª dependência: O interface

A possibilidade de alterar o interface é o terceiro e ultimo pilar da WiSe Framework e encontra-se na camada de Interface.

Após a compilação, a aplicação produzida arranca com um interface genérico. Para que seja possível ao utilizador final ou para programadores, configurar o interface ou criar novos tipos de interface é necessário recorrer ao código fonte da aplicação resultante.

Para além destas 3 dependências, surgiram outros problemas que merecem destaque:

- Como proceder para utilizar outro tipo de informação que não seja proveniente de sensores.
- Como proceder para utilizar sensores cuja informação seja proveniente de fontes que não a Universal Gateway?

- Como proceder para criar interfaces diferentes?

Não há uma solução para estes problemas na versão inicial da WiSe Framework.

4.3 Resumo e Conclusões

A WiSe Framework foi sem dúvida a escolha correcta para o desenvolvimento desta dissertação. Para além dos factores referidos na secção 2.8 a WiSe Framework suporta as 3 camadas gerais de uma aplicação para RSSF, no entanto, existem ainda inúmeros obstáculos que a fazem distanciar do objectivo desta dissertação.

A dificuldade em criar um parser ou um workflow respectivo, a necessidade de alterar código fonte para obter um novo interface, a falta de suporte para outro tipo de informação que não a proveniente de sensores e a impossibilidade de expansão para fontes externas são apenas algumas das barreiras que impedem que a WiSe Framework corresponda ao objectivo proposto para esta dissertação.

Mediante estas barreiras tomaram-se as seguintes decisões de implementação:

Todas as fontes de informação, quer sejam elas sensores ou de outro tipo passarão a denominar-se de “serviços”.

Novos serviços cuja fonte de informação seja conhecida (porta de série, bluetooth, RF) poderão e deverão ser inseridos na camada dos Sensores e serão denominados de “serviços internos”. Para tal, deverá existir uma ferramenta que ajude na configuração e no desenvolvimento do Parser para estes novos sensores ou serviços. Denominou-se esta ferramenta de Parser_tool.

Fontes de informação desconhecida (aplicações externas, cálculos externos) serão denominados de “serviços externos” deverão ser inseridos directamente na camada dos dados. Para tal irão ser disponibilizadas novas funcionalidades no WiSeServer.

Independentemente, da fonte da informação, deverá sempre existir um workflow respectivo. Para ajudar ao desenvolvimento destes workflows deverá existir uma ferramenta própria. Denominou-se esta ferramenta de Workflow_tool.

O interface final deverá ser configurável ao nível da forma como se apresenta a informação. Deverá ser possível escolher o tipo de gráfico a representar os dados assim como o tipo de informação lá contida.

Também deverá ser possível a criação de novos gráficos ou novas formas de visualização dos dados, por exemplo: a utilização de dados GPS (recebidos da camada de sensores) em mapas do Google Maps embutidos directamente na interface final. Então, deverá existir um conjunto de regras que ajude na criação de novas formas de visualização e também na implementação destas.

Por fim, após uma consideração séria aos problemas apresentados e mediante as soluções á disposição, pode-se concluir que para atingir o objectivo que esta dissertação propõe é necessário que a ferramenta a produzir obdeça a um método de utilização conciso e coerente.

Foi então necessário estabelecer esse método e depois, aplicar as alterações ou adições necessárias à WiSe Framework para que obdeça ao mesmo.

Capítulo 5

Solução proposta - O método

Neste capítulo vai ser apresentada a solução proposta para resolver o problema, um método de utilização estruturado, e também irão ser descritos os seus componentes. No final, será feita uma comparação entre o antes e o depois de empregar este método.

5.1 O método de utilização

Estabelecer um método de utilização e criação de aplicações para a WiSe Framework é solução para atingir o objectivo desta dissertação. A partir daí resta apenas implementar as alterações necessárias à WiSe Framework para que este método seja possível de empregar.

Este método deve garantir que os problemas encontrados anteriormente não se verifiquem assim como também deve garantir que todo processo de criação de aplicações para RSSF com a WiSe Framework esteja optimizado ao máximo.

O método estabelecido foi então o seguinte:

- Desenvolver o Parser com a Parser_tool;
- Desenvolver o Workflow com a Workflow_tool;
- Criar Projecto e compilar;
- Criar Base de Dados;
- Configurar a aplicação final;
- Alterar ou adicionar novo interface.

Também foi criada uma correspondência entre cada componente do método e as suas possíveis utilizações:

Tabela 5.1: Correspondência entre método e possíveis utilizações

Método de Utilização da WiSe Framework	Novos Sensores/Serviços	Aplicações, Serviços ou Processamento Externo	Desenvolvimento do Interface	Utilização Normal
Desenvolver o Parser com a Parser_tool	Necessário	---	---	---
Desenvolver o Workflow com a WF_tool	Necessário	Necessário	---	---
Criar Projecto e compilar	Necessário	Necessário	Necessário	Necessário
Criar Base de Dados	Necessário	Necessário	Necessário	Necessário
Configurar a aplicação final	---	---	Necessário	Necessário
Alterar ou adicionar novo interface	---	---	Necessário	---

A tabela 5.1 ilustra quais os passos do método que são necessários mediante o objectivo para o qual se vai utilizar a WiSe Framework. Um utilizador que tencione usar a WiSe Framework para uma tarefa normal, necessita apenas de criar um projecto, compilá-lo, criar a base de dados e configurar a aplicação resultante. Por outro lado, um programador que esteja interessado em desenvolver apenas novos serviços a serem utilizados posteriormente necessita apenas de desenvolver um Parser respectivo com o Parser_tool.

5.2 Componentes do método

Nesta secção são descritos os 6 componentes do método.

5.2.1 Desenvolver o parser com a Parser_tool

Esta componente do método consiste em desenvolver um Parser com uma ferramenta própria para o efeito, o Parser_tool. De seguida deve-se executar os testes respectivos para validar o respectivo Parser.

5.2.2 Desenvolver o workflow com a Workflow_tool

Um workflow também é necessário para cada serviço na WiSe Framework. Como tal, esta componente consiste em criar o respectivo workflow com a Workflow_tool. Posteriormente, deverão ser executados os testes necessários para validar o workflow produzido.

5.2.3 Criar Projecto e compilar

Esta componente consiste em utilizar o Framework Wizard da WiSe Framework para criar todo o projecto, associar os respectivos serviços e parametros assim como como importar parsers ou workflows novos. No final, deverá compilar-se todo o projecto.

5.2.4 Criar Base de Dados

A estrutura da Base de Dados terá de corresponder aos serviços incluídos na WiSe Framework. Este método consiste em utilizar a ferramenta para o efeito, produzida automaticamente no acto de compilação de projecto, o generator. Esta ferramenta criará a estrutura da Base de Dados respectiva.

5.2.5 Configurar a aplicação final

Esta passo do método consiste em simplesmente configurar a aplicação resultante de acordo com os requisitos necessários.

5.2.6 Alterar ou adicionar novo interface

Se o interface genérico não for suficiente, o programador pode implementar outro tipo de interfaces. Sendo que essa função deverá ser executada apenas nesta ultima fase.

5.3 O antes e o depois

Nesta secção será descrito o panorama antes e depois da elaboração deste método, relativo a cada um dos componentes da WiSe Framework.

5.3.1 Camada dos Sensores/Serviços

Tabela 5.2: Antes e Depois - Sensores/Serviços

Antes	Depois
Perda de Mensagens	Adição de buffer, Sem perda de mensagens
Problemas de comunicação com as outras camadas	Sem problemas de comunicação
Processamento das mensagens recebidas no Parser	Pré-processamento fora do Parser
Impossível de fazer o Parser sem instalar uma aplicação genérica	Desenvolvimento do Parser com a Parser_tool

Uma das maiores dificuldades encontradas na camada dos sensores era o desenvolvimento do Parser respectivo a cada serviço. Para agilizar este processo foi criada uma

ferramenta denominada de Parser_tool. Com esta ferramenta, um utilizador pode, entre outras tarefas, fazer uma depuração ao Parser que está a ser criado.

Graças a esta vantagem foi possível identificar um grande problema na leitura das mensagens. De facto, ao desenvolver o Parser para as motes da CrossBow no teste de utilização no anexo A.1 constatou-se que inúmeras mensagens não estavam a ser apanhadas pela Universal Gateway. Através do Parser_tool foi possível verificar que apesar do Parser desenvolvido corresponder ao protocolo da CrossBow, cerca de 7 em cada 10 mensagens recebidas eram descartadas. O Parser_tool obedece à mesma arquitectura que a Universal Gateway e graças às suas funções de depuração foi possível chegar à conclusão que as mensagens descartadas eram na verdade mensagens partidas em segmentos e que a junção dos vários segmentos correspondia a uma mensagem completa.

Para contornar este problema a arquitectura da Universal Gateway e do Parser_tool foi alterada de forma a suportar um buffer. O número de mensagens descartadas diminuiu para cerca de 1 em cada 20 mensagens recebidas. Entende-se que as mensagens descartadas agora já não são mensagens partidas em vários segmentos.

Anteriormente, cada mensagem recebida era enviada para todos os Parsers para ser pré-validada. Apenas um destes retornava com a validação no entanto, todos os Parsers eram executados. Isto levava a um desempenho medíocre. Actualmente, as mensagens são pré-validadas antes de ser chamado o Parser respectivo. Gerou-se assim um aumento de desempenho na Universal Gateway.

Inicialmente, também existiam alguns problemas de comunicação com a camada de dados. Estes problemas deixaram de existir depois da reestruturação da arquitectura da Universal Gateway.

5.3.2 Camada de Dados

Tabela 5.3: Antes e Depois - Dados

Antes	Depois
Adição manual de Workflows	Adição automática de Workflows
Alta dificuldade na Criação de Workflows	Criação de Workflows facilitada com a Workflow_tool
Sem suporte para serviços externos	Suporte para serviços externos
Parametros dos serviços inseridos no código-fonte	Parametros dos serviços inseridos automaticamente
Tipos de serviços inseridos no Código	Tipos de serviços inseridos automaticamente

Assim como na camada dos serviços, a camada de dados também depende de classes externas denominadas de workflows. O desenvolvimento de workflows implicava a

grandes alterações no código fonte da aplicação resultante o que resultava num desenvolvimento muito lento. Para contornar este problema foi criada a Workflow_tool. Esta ferramenta permitiu o desenvolvimento rápido de workflows.

Anteriormente apenas serviços internos eram suportados 3.2.1. Actualmente esta camada já suporta serviços provenientes de fontes externas, estes serviços são designados de serviços externos 4.3.

Associado a cada serviço está o seu workflow, o seu tipo e os seus parâmetros. O seu tipo permite ao WiSeServer identificar o workflow respectivo e os seus parâmetros servem para algum tipo de processamento específico a realizar no workflow. Anteriormente, estas 3 entidades teriam de ser inseridas manualmente através do código fonte. Actualmente este processo é automático e dinâmico.

5.3.3 Framework Wizard

Tabela 5.4: Antes e Depois - Framework Wizard

Antes	Depois
Alterações manuais de código-fonte necessárias	Alteração automática de Código- fonte
Adição manual de workflows	Adição automática de Workflows e Parsers
Apenas um tipo de serviços: Sensores internos	Suporte para serviços internos e externos
Necessária configuração manual do Generator	Generator auto-configurável
Necessário abrir projecto para configurar o wizard	Configuração do wizard sem precisar de abrir projecto
Sem funções de Suporte	Funções de suporte e manutenção

Seja qual fosse o objectivo, era sempre necessário abrir ou criar um projecto para se poder trabalhar no Framework Wizard. Para contornar esta situação foi necessário fazer algumas alterações, sendo que o resultado final permite assim a alteração de todos os parâmetros do Framework Wizard sem necessitar de ter um projecto aberto.

Foi também adicionado o suporte para serviços externos e também funções de importação para Parsers e Workflows eliminando assim a necessidade do utilizador editar o código-fonte para adicionar estas entidades.

Havia também uma necessidade em editar o código fonte do generator 3.5.1 para suportar os novos serviços adicionados. Esta necessidade foi suprimida com alterações feitas no Framework Wizard mas também no generator.

Foram também adicionadas funções de suporte ao Framework Wizard para permitir uma maior flexibilidade na compilação dos projectos e na criação da Base de Dados.

5.3.4 Base de Dados

Tabela 5.5: Antes e Depois - Base de Dados

Antes	Depois
Necessário alterar manualmente código-fonte antes de compilar um projecto	Sem necessidade de alterar código-fonte
Sem feedback para o utilizador do estado actual	Com feedback para o utilizador

O maior problema neste componente era o facto de que a ferramenta que criava a base de dados, o generator, necessitar de alterações ao seu código fonte para poder corresponder a alterações feitas no Framework Wizard. Este problema foi resolvido, sendo que agora esta ferramenta é completamente automática e dinâmica.

Um outro problema era o facto de não haver feedback nenhum para o utilizador do que se passava na execução desta ferramenta. O utilizador não tinha noção se o generator foi bem executado ou não. Este problema foi resolvido sendo que agora o utilizador tem completa noção do que a ferramenta está a executar.

5.3.5 Camada do Interface

Tabela 5.6: Antes e Depois - Interface

Antes	Depois
Apenas utilizadores e grupos	Suporte para utilizadores, grupos e agentes
Edição de parametros de workflow iniciais através de ficheiro externo	Edição dos parametros de workflow dinamicamente
Apenas um tipo de serviço para cada utilizador	Suporte para n tipo de serviços para cada entidade, diferenciados pela identificação do hardware
Tipo de informações sempre iguais	Tipo de informação consoante a entidade
Apenas um só tipo de gráfico: linear	Vários tipos de gráficos suportados
Novos interfaces apenas por edição do código-fonte	Novos interfaces por adição de plugins

O interface é claramente uma das partes com maior importância para o utilizador final, visto que este não terá acesso a mais nada.

Inicialmente, apenas era suportado duas entidades: utilizadores ou grupos de utilizadores. Actualmente existe uma nova entidade denominada de agentes. Ao arrancar, o interface fazia loading dos parâmetros através de um ficheiro auxiliar. Este ficheiro poderia estar corrompido ou ser alterado por fontes externa e implicava que todo o interface deixasse de funcionar correctamente. Actualmente, estes parâmetros são inicializados pelos valores reais contidos na Base de Dados.

Solução proposta - O método

Inicialmente também era apenas suportado um tipo de serviço para cada entidade, isto é, um utilizador só poderia ter um sensor de cada tipo. Actualmente, cada entidade pode ter vários tipos de serviços quantas vezes necessárias e estes são diferenciados entre si pela identificação do hardware.

Para permitir a expansibilidade do Interface e para tirar as limitações impostas pelo interface genérico, foi criado um motor de plugins que permite a adição dos mesmos. Estes plugins dão ao utilizador a vantagem de poder ter o seu interface de acordo com as suas necessidades.

Capítulo 6

Concepção e Implementação

Neste capítulo vão ser abordados todos os componentes da WiSe Framework que foram otimizados ou melhorados assim como todas as novas funcionalidades e ferramentas que foram criadas. Irá também ser feita uma síntese das vantagens de cada alteração ou ferramenta. Para ajudar a uma melhor compreensão do trabalho realizado, serão mostrados alguns excertos de código ou conteúdo de ficheiros.

6.1 Camada Sensores

6.1.1 Optimizações na Universal Gateway

Nesta secção encontram-se as optimizações efectuadas á Universal Gateway assim como as vantagens que daí resultaram.

6.1.1.1 Acumulador

No pré processamento das mensagens existiam algumas falhas devido a alguns problemas das bibliotecas de comunicação pela porta série da .NET Framework [Micc]. Acontecia que algumas das mensagens recebidas eram descartadas pelas seguintes razões:

- Eram mensagens nulas;
- Eram mensagens demasiado pequenas;
- Eram mensagens com lixo.

Após um estudo a estas mensagens, concluiu-se que elas continham informação necessária e que eram partes de outras mensagens que quando juntas eram validadas.

Para contornar esta situação desenvolveu-se um acumulador, cuja funcionalidade é acumular todas as partes de uma mensagem até que esta esteja completa. O acumulador

verifica se o primeiro valor de uma mensagem corresponde á flag de início e vai acumulando até que o valor da última posição acumulada seja igual á flag de fim.

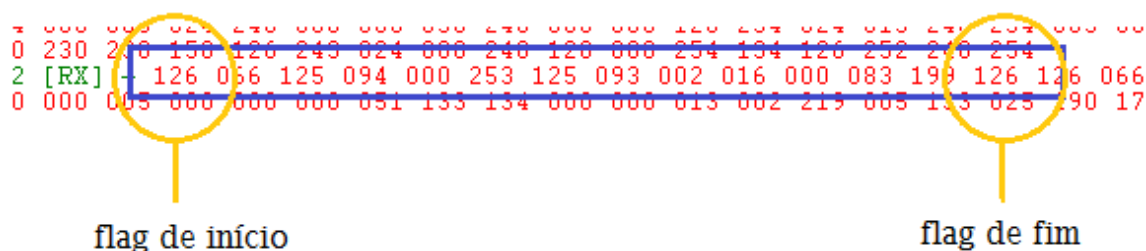


Figura 6.1: Ilustração das flags de início e de fim

Esta operação só é feita caso a propriedade de acumulação esteja a “true” nas configurações da porta. Neste caso é também necessário definir qual a flag de início e de fim nas mesmas configurações. Finalmente, os valores a comparar podem ser em valor inteiro ou em string.

6.1.1.2 Portas.xml

A estrutura deste ficheiro foi modificada para suportar a nova funcionalidade do acumulador.

```

1 <?xml version="1.0" encoding="utf-8"?>
2 <ListaPortas xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
3   <porta porta="COM5">
4     baudrate="57600"
5     paridade="None"
6     dataBits="8"
7     stopBits="Two"
8     tipoLeitura="BIN"
9     reserved="false"
10    acumular="true"
11    flag_inicio="126"
12    flag_fim="126"
13    flag_Type="int" />
14 </ListaPortas>

```

Listagem 6.1: Exemplo de um ficheiro portas.xml actual

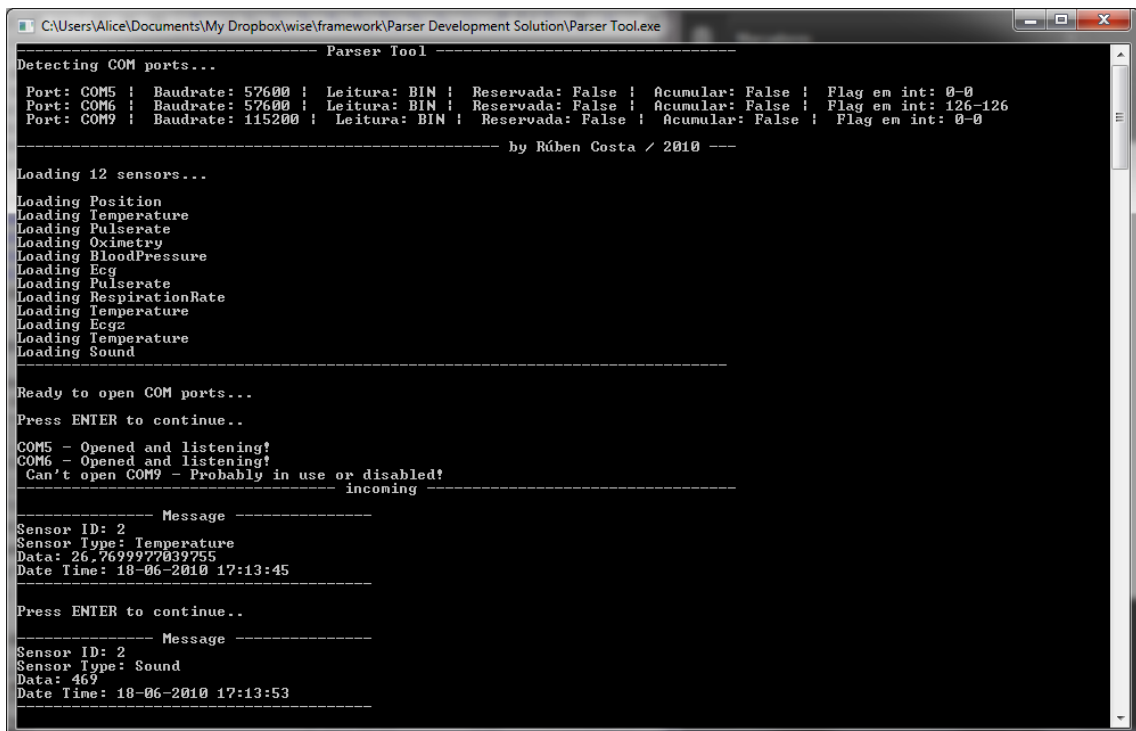
6.1.1.3 Vantagens

O uso da propriedade de acumulação resulta num aumento do número de mensagens a serem processadas pela Universal Gateway. Isto quer dizer que o método anteriormente utilizado não processava todas as mensagens recebidas o que resultava em perda de informação importante.

Com os 2 métodos de leitura disponíveis 3.3.1.2 mais a nova funcionalidade de acumulação é possível expandir a variedade de mensagens a processar pela gateway sendo estas agora, virtualmente qualquer tipo de mensagem.

No pós processamento, para validar uma mensagem, a Universal Gateway tinha de inicializar uma estância em todos os parsers existentes para que estes pudessem ou não validar a mensagem como sua. Uma vez que o pré processamento pode agora usufruir da nova funcionalidade de acumulação, ficou mais robusto libertando assim algum trabalho por parte do motor de parsing, o que resulta numa diminuição da carga de processamento. Temos apenas uma função a executar ao invés de 1*n Parsers funções a executar.

6.1.2 O Parser_tool



```

C:\Users\Alice\Documents\My Dropbox\wise\framework\Parser Development Solution\Parser Tool.exe
Parser Tool
Detecting COM ports...
Port: COM5 : Baudrate: 57600 : Leitura: BIN : Reservada: False : Acumular: False : Flag em int: 0-0
Port: COM6 : Baudrate: 57600 : Leitura: BIN : Reservada: False : Acumular: False : Flag em int: 126-126
Port: COM9 : Baudrate: 115200 : Leitura: BIN : Reservada: False : Acumular: False : Flag em int: 0-0
----- by Rúben Costa / 2010 -----
Loading 12 sensors...
Loading Position
Loading Temperature
Loading Pulserate
Loading Oximetry
Loading BloodPressure
Loading Ecg
Loading Pulserate
Loading RespirationRate
Loading Temperature
Loading Ecg
Loading Temperature
Loading Sound
-----
Ready to open COM ports...
Press ENTER to continue..
COM5 - Opened and listening!
COM6 - Opened and listening!
Can't open COM9 - Probably in use or disabled!
----- incoming -----
----- Message -----
Sensor ID: 2
Sensor Type: Temperature
Data: 26.7699977039755
Date Time: 18-06-2010 17:13:45
-----
Press ENTER to continue..
----- Message -----
Sensor ID: 2
Sensor Type: Sound
Data: 469
Date Time: 18-06-2010 17:13:53
-----

```

Figura 6.2: O parser_tool

O Parser_tool é uma ferramenta que foi desenvolvida para ajudar ao desenvolvimento dos Parsers.

Consiste numa versão simplificada da Universal Gateway que simula o ambiente normal de funcionamento e realiza vários testes unitários para que o programador possa avaliar o comportamento do Parser.

6.1.2.1 Funcionamento geral

O seu funcionamento é semelhante ao da Universal Gateway com a excepção de que os comandos recebidos por parte da camada de dados são simulados através de um conjunto de threads.

6.1.2.2 Funcionalidades Suportadas

- **Verbose:** o Parser_tool pode ou não imprimir no ecrã todas as mensagens produzidas. Esta opção é mais indicada para os inícios do desenvolvimento do Parser cuja preocupação principal nesta altura é integrar o Parser no motor de interpretação;
- **pressEnterToContinue:** esta é uma funcionalidade básica que quando activada, pausa a execução do programa após cada interpretação de mensagem. É de grande utilidade quando queremos verificar a validade dos resultados obtidos;
- **useExpectedMessages:** Esta funcionalidade é de extrema importância quando já se sabe á partida qual o resultado final que a interpretação de uma mensagem deve ter. Ao usar esta função, o Parser_tool compara automaticamente os resultados obtidos com os esperados e imprime no ecrã o resultado dessa comparação. Esta funcionalidade depende do ficheiro ExpectedMessages.xml;
- **processRequests:** o Parser_tool pode ou não processar comandos simulados. Esta funcionalidade depende do ficheiro Requests.xml;
- **onlyShowAnswers:** Esta funcionalidade permite apenas mostrar as respostas aos pedidos efectuados. De extrema utilidade quando se está a implementar o método que processa os comandos no Parser.

As opções para estas funcionalidades podem ser configuradas no ficheiro Configurations.xml:

```

1 <?xml version="1.0" encoding="utf-8"?>
2 <Configuration xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
3   <options verbose="false" pressEnterToContinue="true" useExpectedMessages="false" processRequests="false"
4     onlyShowAnswers="false" />
  </Configuration>

```

Listagem 6.2: Ficheiro das configurações do Parser_tool

Para além destas funcionalidades, esta ferramenta ainda faz o seguinte:

- Escreve para o ficheiro WiSeML.xml todas as mensagens interpretadas;
- Escreve para o ficheiro ExceptionsLog.xml todas as excepções encontradas.

6.1.2.3 Vantagens

Com esta ferramenta é possível obter um ambiente simulado de programação sem que o programador se preocupe em ter a camada de dados a funcionar.

A programação dos parsers é agora isolada do resto da WiSe Framework o que permite delegar tarefas a outras pessoas, isto é, podemos ter um programador com conhecimentos exclusivos sobre o hardware dos sensores a programar o parser sem precisar de ter conhecimentos específicos sobre a aplicação resultante em que vão ser utilizados.

A empresa detentora da WiSe Framework, pode agora pedir aos fabricantes do hardware dos sensores o desenvolvimento de um Parser respectivo, sem ter que revelar total ou parcialmente, o código fonte da WiSe Framework.

6.1.2.4 Requisitos

Os requisitos são os mesmos que a Universal Gateway mas ao contrário desta, o Parser_tool necessita que os seus ficheiros de configuração sejam configurados manualmente.

6.2 Camada Dados

6.2.1 Optimizações ao WiSeServer

Nesta secção encontram-se as optimizações efectuadas ao WiSeServer assim como as vantagens que daí resultaram.

6.2.1.1 Carregamento dinâmico

Na sua versão inicial o WiSeServer necessitava de alterações a certas classes do seu código fonte para poder suportar novos serviços que fossem adicionados. Actualmente tudo o que o WiSeServer necessita é carregado dinamicamente.

Isto é possível graças às capacidades de introspecção que o C# suporta. Por exemplo, a inicialização das entidades da Base de Dados é feita através da análise interna das classes incluídas na dll na WiSeApp.

O código seguinte ilustra esta solução:

```

1  using System;
2  using System.Reflection;
3  using System.Collections;
4
5  namespace WiseApp.Models
6  {
7      public class ModelTypes
8      {
9
10         public static ArrayList GetTypesInNamespace( string nameSpace)
11         {
12
13             Assembly asm = Assembly.GetExecutingAssembly();
14
15             ArrayList namespaceList = new ArrayList();
16
17             foreach (Type type in asm.GetTypes())
18             {
19                 if (type.Namespace == nameSpace)
20                     namespaceList.Add(type);
21             }
22
23             return namespaceList;
24         }
25
26         public static Type GetTypeInNamespace(string typeName)
27         {
28
29             Assembly asm = Assembly.GetExecutingAssembly();
30
31             foreach (Type type in asm.GetTypes())
32             {
33                 if (type.Name == typeName)
34                     return type;
35                 break;
36             }
37
38             return null;
39         }
40     }
41 }
42 
```

Listagem 6.3: Exemplo de classes para o carregamento dinâmico

6.2.1.2 Novas funcionalidades para serviços externos

A ideia dos serviços externos é uma forma de fazer um bypassing legítimo ao uso da Universal Gateway. Isto é, quando desconhecemos a fonte de origem dos dados ou não temos acesso ao seu hardware ou qualquer outra razão, poderemos efectuar a chamada por Web Service ao método `put_WiSeML`. Este método foi alterado para poder suportar estas funcionalidades. Também foi criado um novo Web Service denominado de `put_String_WiSeML` com o objectivo de suportar outro tipo de comunicações externas ou internas.



Figura 6.3: Web Services disponiveis actualmente

O Web Service `put_WiSeML` suporta a comunicação em XML formatado de acordo com os padrões WiSeML 3.2.3. Por outro lado o Web Service `put_String_WiSeML` suporta a comunicação em String formatado de acordo com os mesmos padrões do WiSeML. É aconselhável a comunicação em XML mas caso a tecnologia do serviço externo não o suporte, é possível a comunicação em Strings simples.

6.2.1.3 Vantagens

Para além do suporte oferecido a virtualmente qualquer tipo de comunicação, esta alteração permite que os dados sejam inseridos na base de dados mesmo que não estejam associados a um agente. Isto permite inserir dados de fontes externas que não dependam de agentes e também permite que não se percam dados importantes, já calculados e que não eram armazenados simplesmente porque não estavam associados a nenhum agente.

O carregamento dinâmico retira a necessidade que o programador tinha de editar código fonte e permite que a empresa detentora do mesmo não tenha a necessidade de o divulgar a nenhuma entidade externa.

6.2.2 A Workflow_tool

Esta ferramenta foi criada para colmatar uma necessidade que existia no processo de desenvolvimento de workflows.

6.2.2.1 Funcionamento geral

A ideia principal desta ferramenta é ajudar ao desenvolvimento dos workflows, simplificando esta tarefa através da simulação do ambiente circundante dos mesmos. Este processo é feito, utilizando mensagens em WiSeML tal e qual como seriam enviadas pela Universal Gateway. Estas mensagens são então decodificadas e o seu workflow é chamado respectivamente. Tanto o processo antes do workflow como após este é completamente simulado pela Workflow_tool, isto é:

1. Recepção das mensagens;
2. Validação dos dados;
3. Invocação do workflow respectivo;
4. Armazenamento na base de dados de qualquer resultado do workflow.

Assim, o funcionamento é simples. Para a recepção das mensagens utiliza-se um dos ficheiros que o Parser_tool produz, o WiseML.xml. A validação de dados ficou a cargo de um outro ficheiro xml criado com o objectivo de tirar trabalho a editar ficheiros de código. Chama-se SensorParam.xml e contém todos os tipos de sensores, os seus limites e os seus valores. A invocação dos workflows é feita pela mesma classe que é usada na aplicação final.

O armazenamento na base de dados é simulado por funções de escrita em ficheiros de log. Para esta funcionalidade existir, os modelos respectivos tiveram de ser alterados.

6.2.2.2 Funcionalidades Suportadas

O Workflow_tool contém as funcionalidades necessárias para o desenvolvimento dos workflows:

- Permite a visualização e interpretação de erros que possam surgir na configuração dos workflows e os seus parâmetros;

```

Workflow Tool

Loading Sensor Parameters:
- Opening SensorParam.xml file... Done!
- Loading Types and Parameters:
+ Temperature :: lower_limit :: upper_limit ...Done!
+ Ecg :: lower_limit ...Done!
+ Eegz :: lower_limit ...Done!
+ PulseRate :: lower_limit :: upper_limit ...Done!
+ RespiratoryFrequency :: lower_limit :: upper_limit ...Done!
+ Oximetry :: lower_limit ...Done!
+ BloodPressure :: upper_limit :: lower_limit ...Done!
+ Sound :: lower_limit :: upper_limit :: calibrate_value ...Done!
- Complete!

Loading Alerts... Done!

Loading Workflows:
- WFSound... Done!

Loading workflows configurations file... Done!
Configuring 0 Workflows from workflows.xml file...
- WFPosition... NOT FOUND!
- WFecg... NOT FOUND!
- WFEegz... NOT FOUND!
- WFTemperature... NOT FOUND!
- WFOximetry... NOT FOUND!
- WFPulse... NOT FOUND!
- WFBloodPressure... NOT FOUND!
- WFRespiratoryFreq... NOT FOUND!
- WFSound... OK!

Loading WiSeML.xml file... Done!
Configuring 2673 Messages from WiSeML.xml file... Done!

Ready to Begin processing...                               Press ENTER to continue!
    
```

Figura 6.4: Ecrã de arranque do Workflow_tool

- Apresenta os valores de entrada e de saída para cada mensagem recebida;

```

PROCESSING MESSAGES

----- Sensor Model
Message Number: 1
Operation: Create and Flush
Type: Temperature
id: 2
Press ENTER to continue...

----- Sensor Model
Message Number: 2
Operation: Create and Flush
Type: Sound
id: 2
Press ENTER to continue...

----- Sound Model
Message Number: 2
Operation: Create and Flush
Type: Sound
id: 2
Stamp: 10-03-2010 12:33:21
Output data: 481
Process runtime: 14,0000 ms
Press ENTER to continue...

----- Sound Model
Message Number: 4
Operation: Create and Flush
Type: Sound
id: 2
Stamp: 10-03-2010 12:33:22
Output data: 483
Process runtime: 1,0001 ms
Press ENTER to continue...

----- Sound Model
Message Number: 6
Operation: Create and Flush
Type: Sound
id: 2
Stamp: 10-03-2010 12:33:23
Output data: 493
Process runtime: 15,0000 ms
Press ENTER to continue...
    
```

Figura 6.5: Valores de entrada e saída

- Visualização de varias variáveis, incluindo a do tempo de execução do workflow.

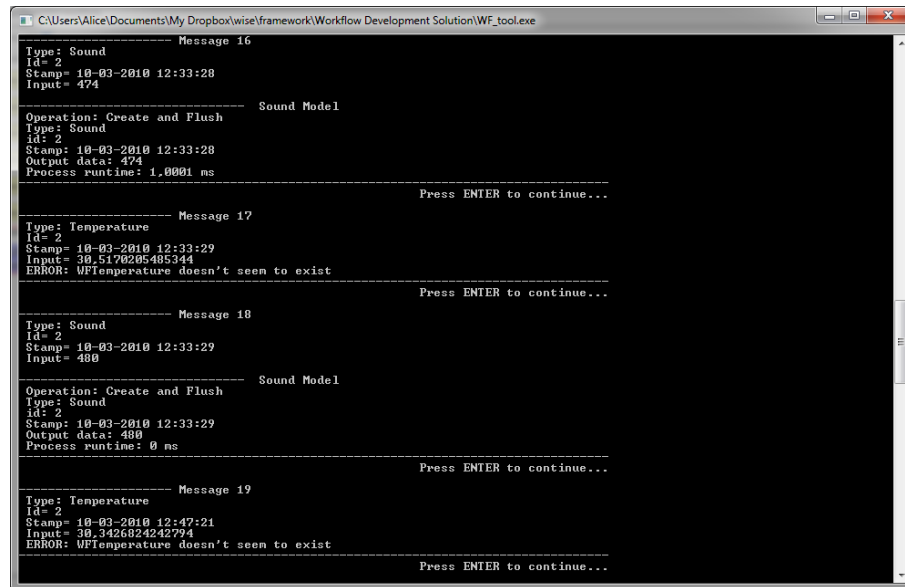


Figura 6.6: Variveis de tempo de execução

Para além destas funcionalidades, existem outras que podem ser alteradas num ficheiro de configuração com o nome: "configurations.xml", são elas:

- **onlyShow:** permite escolher o que queremos ver no ecrã. Sendo possível escolher entre visualização só de alertas, só de modelos ou de tudo;
- **pressEnterToContinue:** permitir escolher se queremos que a tool faça uma pausa entre cada mensagem processada.

```

1 <?xml version="1.0" encoding="utf-8"?>
2 <Configuration xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3   xmlns:xsd="http://www.w3.org/2001/XMLSchema">
4   <options onlyShow="everything" pressEnterToContinue="true" />
5 </Configuration>

```

Listagem 6.4: Ficheiro das configurações do Wotkflow_tool

6.2.2.3 Vantagens

Com esta ferramenta, é possível desenvolver e testar um workflow sem necessitar da WiSe Framework.

A simulação feita ao ambiente circundante do workflow permite que este possa ser posteriormente inserido no Framework Wizard sem necessitar de qualquer alteração prévia. A utilização de um ficheiro XML de configuração de parâmetros para os sensores permite que o programador modifique estes parâmetros sem alterar o código fonte. É

possível concentrar os esforços no desenvolvimento dos workflows sem nos preocuparmos com a WiSe Framework.

A criação de um workflow está directamente relacionada com o seu model, pelo que, para que o workflow funcione correctamente, o seu model terá de estar correcto. O model que resulta do desenvolvimento do workflow através desta ferramenta, serve de template para o model a criar no Framework Wizard. Isto diminui a carga de programação posterior no que respeita aos models.

6.2.2.4 Requisitos

O único requisito é a configuração manual dos ficheiros XML que esta ferramenta utiliza.

6.3 Camada Interface

6.3.1 Melhoramentos

A WiSeApp apenas continha dois actores para o sistema: Utilizadores e Grupos de utilizadores. No entanto, nem sempre era necessário que uma entidade que tivesse um serviço associado fosse um utilizador da aplicação. Esta entidade podia simplesmente não ter qualquer necessidade de actuar sobre o sistema, por exemplo:

- Um objecto - Uma casa, um automóvel, etc;
- Uma localização - Uma área de uma floresta, de uma zona urbana, etc;
- Um pessoa não actuante - Um paciente, um peão, etc.

Foi então desenvolvido o suporte para este tipo de entidades às quais se deu o nome de “agentes”.

O tipo de carregamento dinâmico que foi desenvolvido para o WiSeServer foi também desenvolvido e adaptado para a WiSeApp permitindo assim que a aplicação não necessitasse de ficheiros externos como o WiseConfig.xml para a inicialização de informação.

```

1 <?xml version="1.0" encoding="UTF-8" ?>
2 <application>
3   <user_info>
4     <info>birthdate</info>
5     <info>sex</info>
6     <info>allergies</info>
7   </user_info>
8   <sensors>
9     <sensor>Temperature | Temperature</sensor>
10    <sensor>Sound | Sound</sensor>
11  </sensors>
12 </application>

```

Listagem 6.5: Exemplo de ficheiro WiseConfig.xml

Concepção e Implementação

Uma das grandes vantagens é que esta alteração permite fechar um pouco a possibilidade de alteração do funcionamento da WiSeApp por mãos alheias. Isto é, anteriormente a WiSeApp dependia de ficheiros externos presentes no disco local da máquina em questão assim como de informação contida no seu código fonte. Actualmente, apesar do seu código fonte ser disponibilizado na máquina local, a WiSeApp está apenas dependente da Base de Dados. E o acesso a esta pode ser vedado.

Outro melhoramento foi o da implementação da capacidade de visualizar vários tipos de serviços na mesma entidade (utilizador, agente ou grupo). É agora possível, para além desta visualização, distinguir entre o Hardware como demonstra a figura 6.8.

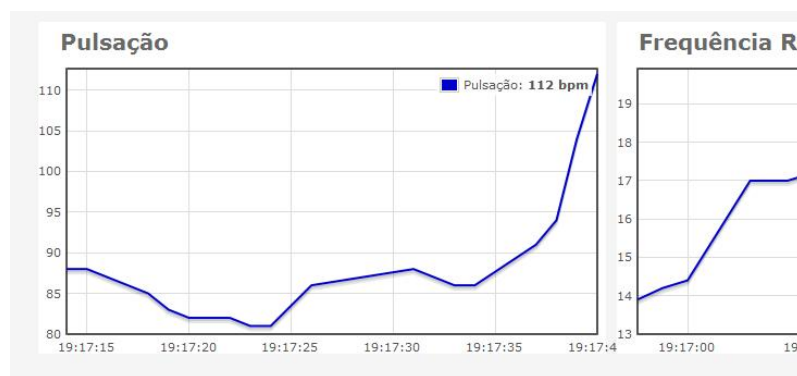


Figura 6.7: Gráficos sem Identificação do Hardware

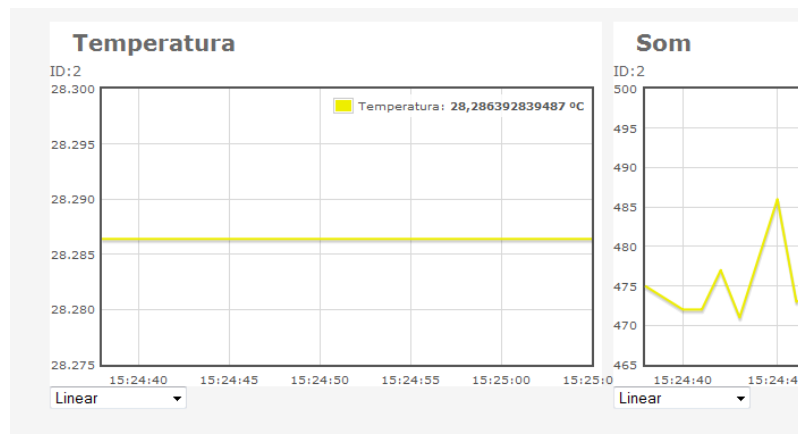
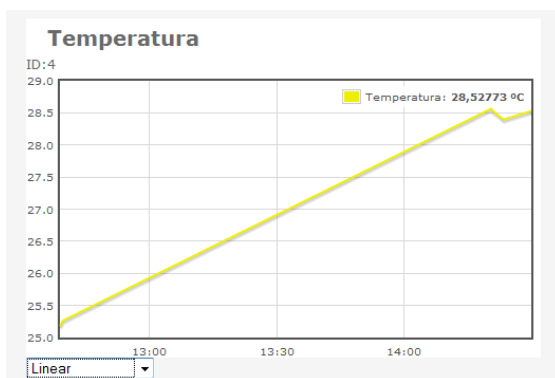


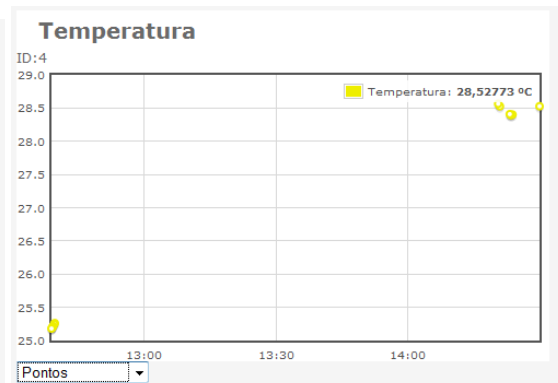
Figura 6.8: Gráficos com Identificação do Hardware

6.3.2 Novos gráficos

Existem agora vários tipos de gráficos disponíveis para que o utilizador possa escolher entre eles.

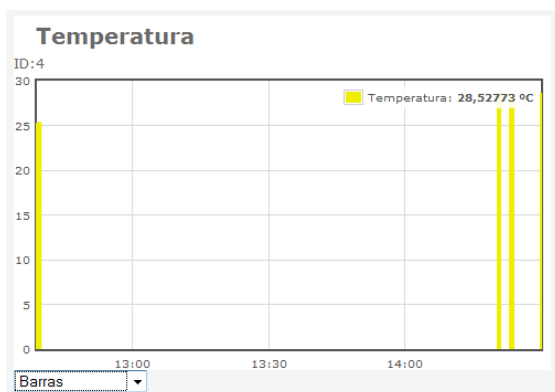


(a) Gráfico linear

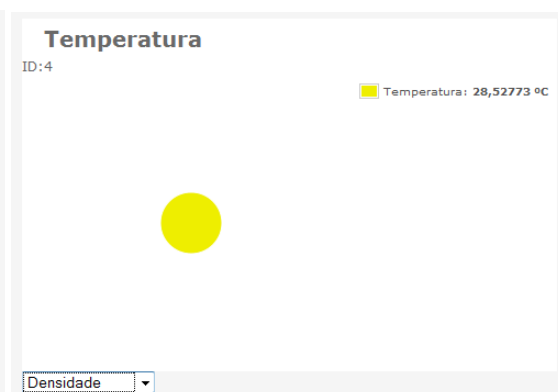


(b) Gráfico de pontos

Figura 6.9: Gráfico linear e Gráfico de pontos



(a) Gráfico de Barras



(b) Gráfico densidade

Figura 6.10: Gráfico de Barras e Gráfico de de densidade

6.3.3 Motor de Plugins

O motor de plugins é a grande novidade de toda a WiSeApp. De facto, não seria má ideia se toda a aplicação funcionasse apenas com este motor de plugins. No entanto, existiam algumas funcionalidades que não poderiam existir.

Este motor de plugins é dividido em 3 partes:

- O layout do interface;
- As funções necessárias;
- O seu estilo.

O layout é a peça mais importante e aquela que é indispensável. O layout é composto por um ficheiro “view” que depois ficará associado à arquitectura MVC [Micb] da WiSeApp. Este ficheiro “view” serve apenas para desenhar e identificar o que vai ser visto no plugin.

A cada plugin corresponde uma nova TAB no interface da WiSeApp. Esta tab só aparece disponível se o serviço disponibilizado pelo plugin estiver associado à entidade na qual estamos a navegar actualmente.

Esta Tab é adicionada ao interface através das capacidades dos métodos de Render-Partial [sta]

```

1 <div id="tab<%=tab %>">
2   <% Html.RenderPartial("~/Views/Shared/Plugin_Tabs/" + ficheiros_views[i].Name.Substring(0, ficheiros_views[i].Name
3     .Length), ViewData); %>
4 </div>

```

Listagem 6.6: Exemplo de uma chamada de Render Partial

Para ajudar ao processamento de funções como gráficos de tipos ainda não suportados é necessário um ficheiro javascript [w3sb]. Este ficheiro é carregado dinamicamente pelo motor de plugins.

```

1 <!--JavaScript engine loading-->
2   <%
3     string dir_framework = HttpContext.Current.Request.PhysicalApplicationPath.ToString();
4     dir_framework = dir_framework.Substring(0, dir_framework.LastIndexOf('\\'));
5     FileInfo[] ficheiros_java = new DirectoryInfo(dir_framework + @"\Content\Plugin_java").GetFiles("*.js");
6
7     for (int i = 0; i < ficheiros_java.Length; i++)
8     {
9       String java_name = ficheiros_java[i].Name.ToString();
10      %>
11      <script type="text/javascript" src="<%= Url.Content("~/Content/Plugin_java/" + java_name) %>"></
12        script>
13      <%
14      }
15    %>

```

Listagem 6.7: Motor de carregamento de ficheiros javascript

O estilo da Tab associada ao plugin deve ser definido no ficheiro .css [w3sa] e este também será carregado com o motor de plugin.

```

1  <!--CSS engine loading-->
2  <%
3      string dir_framework = HttpContext.Current.Request.PhysicalApplicationPath.ToString();
4      dir_framework = dir_framework.Substring(0, dir_framework.LastIndexOf('\\'));
5      FileInfo[] ficheiros_css = new DirectoryInfo(dir_framework + @"\Content\Plugin_css").GetFiles("*.css");
6
7      String CSS_contents = "";
8      File.WriteAllText(dir_framework + @"\Content\Plugin_css\plugin.css", CSS_contents); //empty the file
9
10     String plugin_css_name = "plugin.css";
11
12     for (int i = 0; i < ficheiros_css.Length; i++)
13     {
14         String contents = File.ReadAllText(ficheiros_css[i].FullName.ToString());
15         CSS_contents += contents;
16     }
17
18     File.WriteAllText(dir_framework + @"\Content\Plugin_css\plugin.css", CSS_contents);
19     plugin_css.Attributes.Add("href", " ../Content/Plugin_css/" + plugin_css_name);
20
21 %>

```

Listagem 6.8: Motor de carregamento de ficheiros css

Com esta nova funcionalidade, é possível criar novas tabs para serviços específicos sem afectar o resto da aplicação e é completamente integrado com os parâmetros definidos previamente no Framework Wizard. Os programadores podem então desenvolver o seu interface para um serviço específico. Isto aumenta exponencialmente a expansibilidade da WiSeApp.

6.4 Base de Dados

Todos os dados necessários à execução da aplicação, especialmente a parte da WiSeApp, encontram-se na Base de Dados.

Na sua primeira versão, o Framework Wizard era responsável por alterar código fonte do programa responsável pela criação da Base de Dados (o generator) e compilá-lo.

Isto levantava vários problemas. Qualquer utilizador da WiSe Framework, tinha acesso à estrutura da Base de Dados o que significava assim, vários riscos de segurança. Actualmente, o código fonte do generator não está disponível para edição e toda a base de dados é criada dinamicamente em relação à estrutura dos seus models.

6.4.1 Optimizações ao Generator

O generator é um programa de apoio criado pela WiSe Framework cuja funcionalidade principal é o de criar a Base de Dados de forma correcta para o projecto seleccionado. Na sua versão inicial, o generator apenas corria os comandos necessários para criar a base de dados. Na sua versão actual, o Generator imprime no ecrã todos os passos da sua

execução assim como falhas que possam ter ocorrido. Estas também são escritas num ficheiro de log.

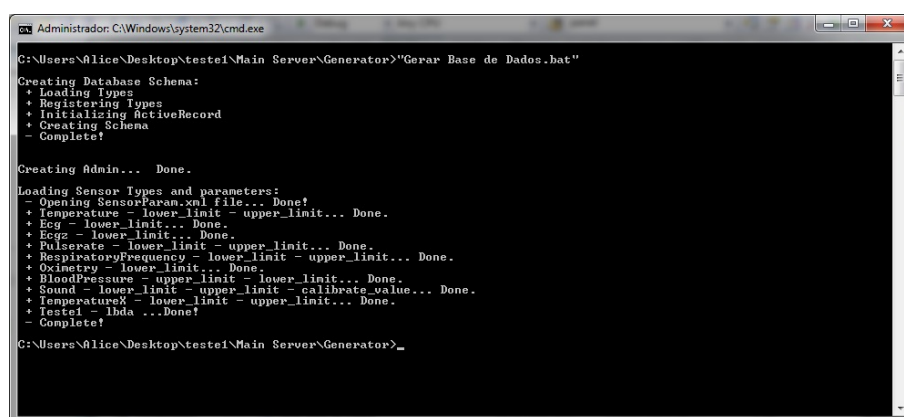
6.4.1.1 Optimizações

Devido a reestruturação da arquitectura para que não houvesse escrita de código por parte da WiSe Framework, o generator é agora completamente dinâmico. Isto é, faz o loading dos modelos dos serviços respectivos, dos seus parâmetros e dos tipos de serviços e cria a base de dados de acordo com estes.

A prevenção de erros também foi melhorada e é agora visível no ecrã a ocorrência de algum erro que possa acontecer.

6.4.1.2 Funcionamento actual

O funcionamento actual do generator é simples. Ao iniciar, faz o loading dos parâmetros todos que necessita através da classe WiseApp.dll e do ficheiro SensorParam.xml. E de seguida cria a base de dados baseado no script que a WiSe Framework criou.



```

C:\Users\Alice\Desktop\teste1\Main Server\Generator>"Gerar Base de Dados.bat"

Creating Database Schema:
+ Loading Types
+ Registering Types
+ Initializing ActiveRecord
+ Creating Schema
- Complete!

Creating Admin... Done.

Loading Sensor Types and parameters:
+ Opening SensorParam.xml file... Done!
+ Temperature - lower_limit - upper_limit... Done.
+ Ecg - lower_limit... Done.
+ Ecg2 - lower_limit... Done.
+ PulseRate - lower_limit - upper_limit... Done.
+ RespiratoryFrequency - lower_limit - upper_limit... Done.
+ Oximetry - lower_limit... Done.
+ BloodPressure - upper_limit - lower_limit... Done.
+ Sound - lower_limit - upper_limit - calibrate_value... Done.
+ Temperature2 - lower_limit - upper_limit... Done.
+ Teste1 - lbda ...Done!
- Complete!

C:\Users\Alice\Desktop\teste1\Main Server\Generator>_
    
```

Figura 6.11: Feedback do Generator

6.4.1.3 Funcionalidades Suportadas

- Criação da Base de Dados dinamicamente;
- Escrita em log file;
- Detecção de erros e sua escrita no ecrã.

6.4.1.4 Vantagens

O utilizador consegue com esta nova versão do generator, criar a sua base de dados sem ter de previamente editar código assim como, tem uma noção do trabalho que está a ser feita pela mesma, de forma a corrigir eventuais erros que aconteçam.

6.4.1.5 Requisitos

O generator necessita de vários ficheiros para que a sua execução seja viável. Os ficheiros são os seguintes:

- EnableMixedMode.reg;
- Castle.ActiveRecord.dll;
- Castle.Componentes.Validator.dll;
- Castle.Core.dll;
- Iesi.Collections.dll;
- Log4net.dll;
- NHibernate.dll;
- SensorParams.dll;
- WiseApp.dll;
- Gerar Base de Dados.bat;
- Script.sql;
- Appconfig.xml;
- SensorParam.xml;
- Generator.exe.

6.5 Framework Wizard

6.5.1 Optimizações

6.5.1.1 Criação de código manualmente VS dinamicamente:

Na utilização do Framework Wizard era muitas vezes necessário recorrer a edição de código para as seguintes tarefas:

- Adaptar os modelos de novos sensores;
- Adaptar novos Workflows;
- Inserir novos parâmetros para os limites dos sensores;
- Inserir novos tipos de sensores na base de dados;
- Carregar os workflows.

O que acontecia nestas situações era uma dependência entre o resultado final e a edição de código que levava a que o Framework Wizard tivesse uma utilização desagradável.

Actualmente, este problema já não acontece. Deixa de ser necessário ao utilizador editar o código das aplicações finais Universal Gateway e WiSeServer com excepção para a WiSeApp (cuja situação será descrita mais á frente). A criação dos Parsers e dos Workflows estão a cargo da Parser_tool e da Workflow_tool, respectivamente. Estando em posse destes é relativamente fácil importa-los para o Framework Wizard. Esta dispõe de um interface adequado para o efeito.

Ao inserir novos serviços, também era necessário editar os seus parâmetros de limites manualmente no código fonte. Este problema foi resolvido através da implementação de um interface simples.

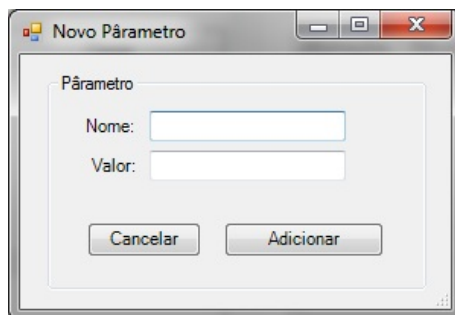


Figura 6.12: Interface simples para novos parâmetros

Este tipo de solução levou a uma reestruturação da arquitectura do WiSeServer, da WiSeApp e do generator como descrito previamente. Em relação a todos os parâmetros que estão associados a serviços, o seu loading é agora dinâmico, não dependendo de qualquer mudança no código fonte.

6.5.1.2 WiSeApp

A WiSeApp é a única excepção destas condições. Como é uma aplicação Web, o seu código fonte tem de estar presente para que possa ser “instalado” no Servidor respectivo e como é também a aplicação destinada a ser um interface, qualquer alteração que seja necessária, terá de ser feita no seu código fonte. Por esta razão, a WiSe Framework estará sempre acompanhada pelo código fonte da WiSeApp e sobre ele fará as alterações necessárias.

6.5.1.3 Serviços Internos e Externos

Para que a aplicação final suportasse serviços internos e externos, era necessário que o Framework Wizard também os suportasse.

Como tal, foi feita uma separação entre Serviços Internos e Serviços Externos.



Figura 6.13: Separação entre serviços Internos e Externos

Esta separação permite uma maior adaptação da Framework a eventuais utilizações futuras não previstas durante a fase do seu desenvolvimento.

6.5.2 Projecto VS Configurações

Uma das grandes funcionalidades novas criadas nesta fase foi também a separação entre o que é a WiSe Framework e o que é o Projecto a desenvolver.

Na primeira versão, o Framework Wizard exigia que se tivesse um projecto aberto para se poder modificar parâmetros que só faziam parte da WiSe Framework, isto é, adicionar ou remover sensores, workflows, entre outros.

Foi feita então uma separação destes contextos, sendo agora possível editar todos os parâmetros referentes à WiSe Framework sem precisar de ter um projecto aberto.

6.5.3 Novas funcionalidades

Em relação a todos os parâmetros que estão associados a sensores, o seu loading é agora dinâmico, não dependendo de qualquer mudança no código fonte. Esta mudança trouxe consigo algumas funcionalidades novas:

- É agora possível criar e editar um número indefinido de parâmetros para cada sensor e o seu uso será apenas ao nível de processamento do workflow;

- As aplicações finais não precisam de ser editadas ao nível do código;
- A base de dados é criada dinamicamente.

6.5.4 Novas opções



Figura 6.14: Opções de Compilação

- **Compilar WiSeApp** - Activar ou desactivar para que o Framework Wizard compile a WiSeApp;
- **Compilar Base de Dados** - Activar ou desactivar para que o Framework Wizard compile a Base de Dados;
- **Compilar WiSeServer** - Activar ou desactivar para que o Framework Wizard compile o WiSeServer;
- **Compilar Universal Gateway** - Activar ou desactivar para que o Framework Wizard compile a Universal Gateway;
- **Criar Base de Dados** - Activar ou desactivar para que o Framework Wizard crie a estrutura da Base de Dados na máquina em que está a correr.

6.5.5 Funcionamento actual do Framework Wizard

6.5.5.1 Inicialização

Ao arrancar, o Framework Wizard faz o loading dos seus parâmetros que vão servir para as configurações do projecto e da WiSe Framework. É feita sempre uma verificação a todos os ficheiros necessários e se for detectado algum erro, este será mostrado no ecrã para que possa ser corrigido pelo utilizador.

Em seguida é mostrado o Interface Inicial.

6.5.5.2 Interface Inicial

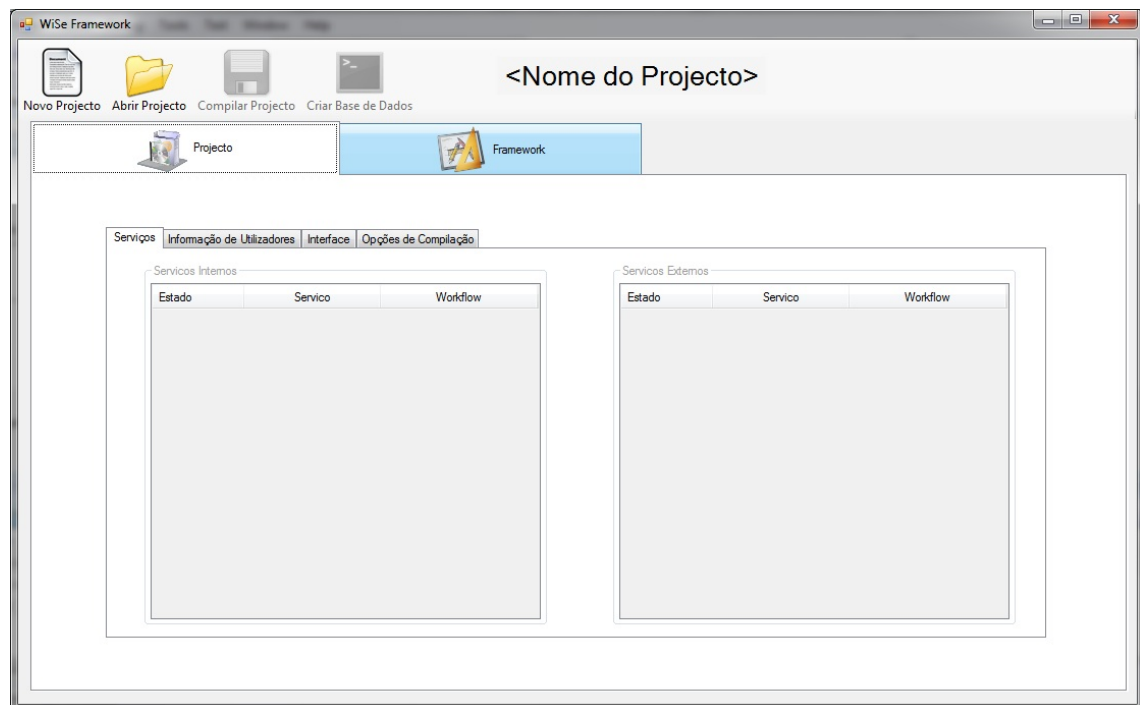


Figura 6.15: Ecran inicial do Wizard da WiSe Framework

Perante o Interface inicial, o utilizador tem três opções. Iniciar um projecto novo, Abrir um projecto já existente ou avançar para as configurações da WiSe Framework.

Se iniciar um projecto novo ou abrir um já existente, são apresentadas as configurações para o mesmo e é dado a possibilidade de editar essas configurações.

Ao navegar entre as várias tabs disponíveis, o utilizador tem a possibilidade de escolher outras opções.

6.5.5.3 Tab Projecto

Dentro da tab Projecto o utilizador tem a possibilidade de navegar entre 4 tabs:

1. **Serviços:** nesta tab encontra-se a lista dos serviços internos e externos disponíveis para associar ao projecto.

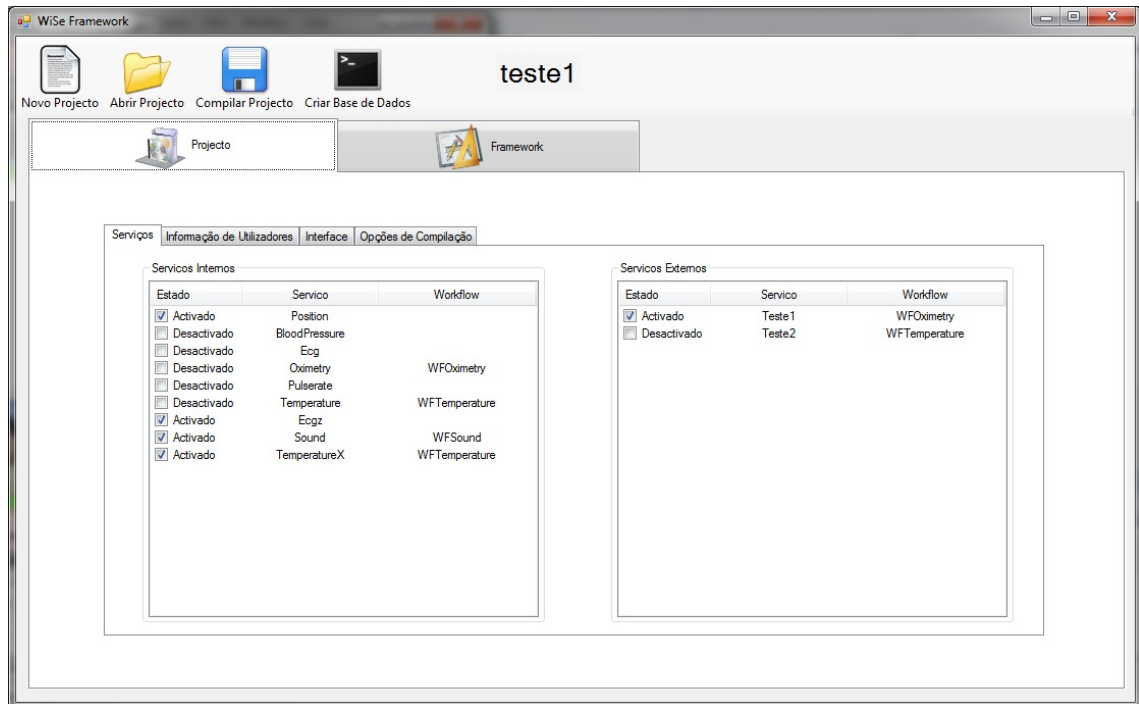


Figura 6.16: Serviços do projecto actual

Pela imagem podemos verificar duas listas distintas. Uma de serviços internos e outra de serviços externos. A activação ou desactivação destes é feita clicando na checkbox respectiva.

2. **Informação dos utilizadores:** nesta tab o utilizador pode seleccionar qual o tipo de informação que deve estar associado aos utilizadores da aplicação resultante.

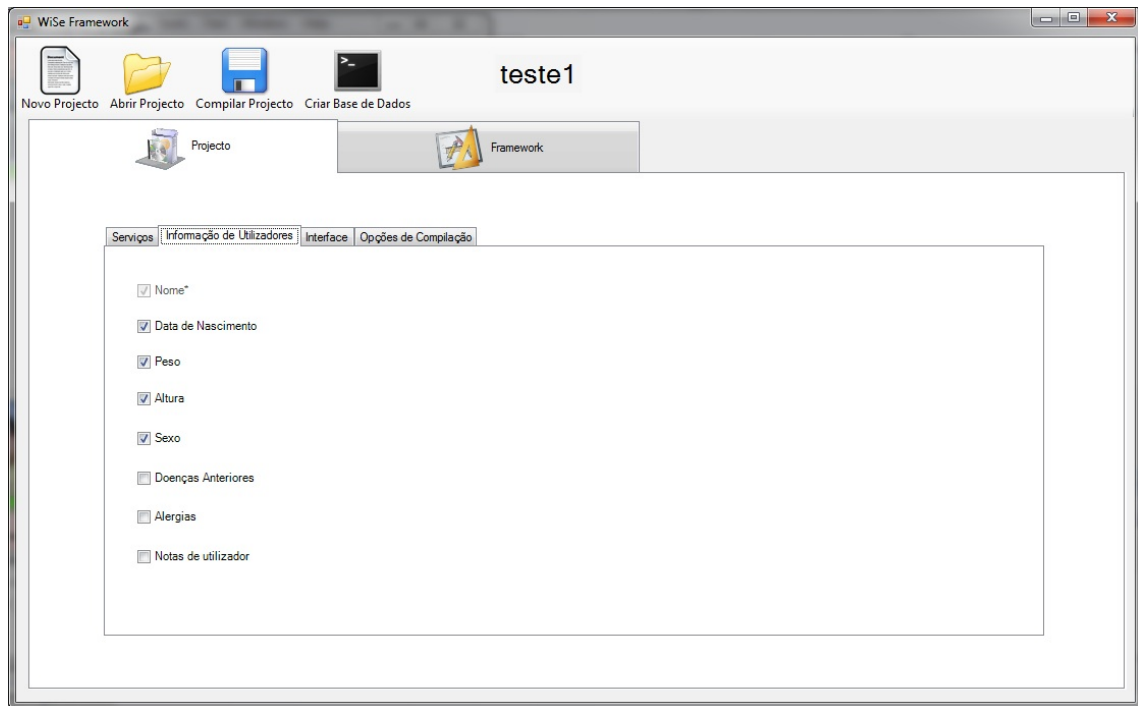


Figura 6.17: Informação dos utilizadores

Basta seleccionar a informação que se quer visualizar na aplicação final.

3. **Interface:** Esta tab permite fazer um preview do aspecto do interface da aplicação Web final, escolher um logótipo para a mesma e também alterar o seu ficheiro css.

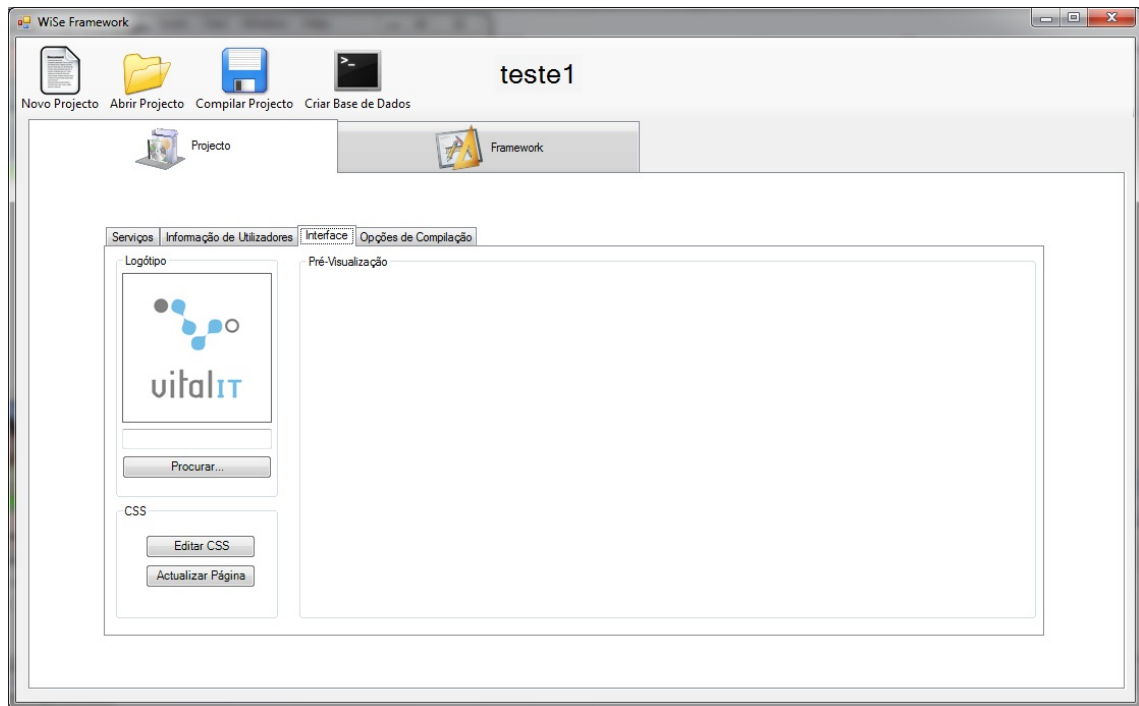


Figura 6.18: Interface da Aplicação final

Uma restrição destas funcionalidades é que para se poder pré-visualizar, é necessário compilar o projecto em primeiro lugar. Esta é uma óptima funcionalidade pois é possível ter uma noção de como ficará o interface sem ter de instalar a WiSeApp primeiro.

4. **Opções de Compilação:** esta tab oferece algumas opções de compilação. Esta é uma nova funcionalidade que permite ao utilizador distinguir quais as componentes a compilar e também, se pretende que a WiSe Framework cria a Base de Dados automaticamente após a compilação do Projecto.

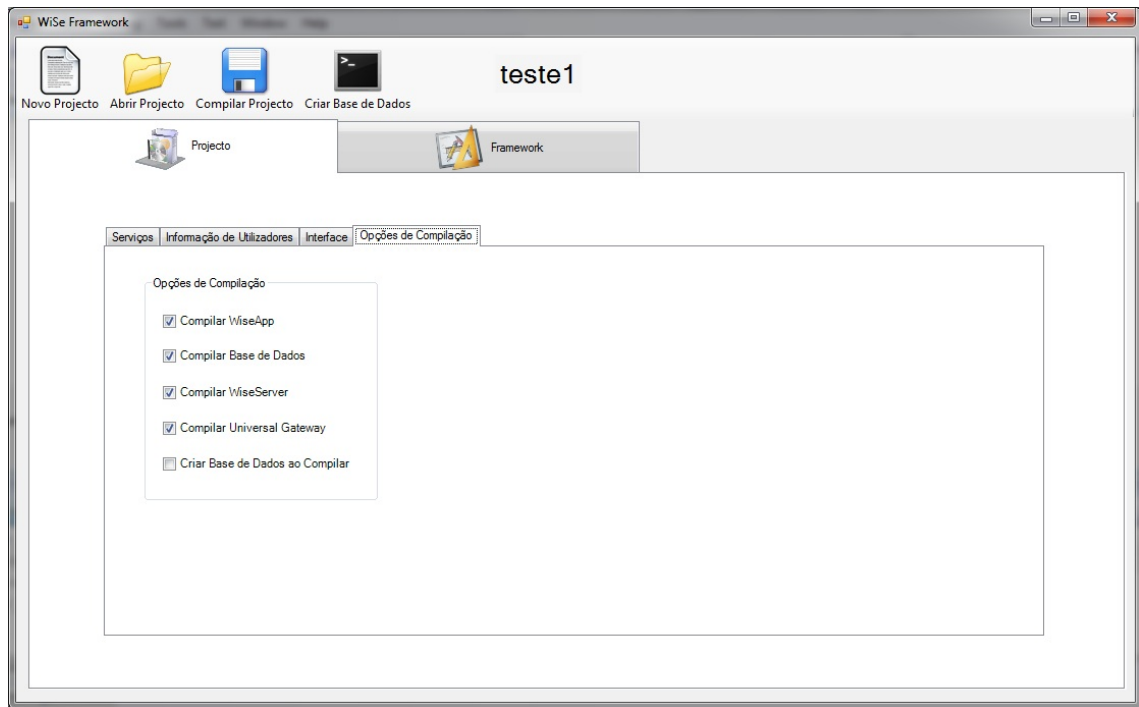


Figura 6.19: Opções de compilação

Estas opções são bastante intuitivas e ajudam ao desenvolvimento da aplicação final.

6.5.5.4 Tab Framework

Dentro da tab Framework o utilizador tem a possibilidade de navegar entre 3 tabs:

1. **Serviços Internos:** esta é a tab na qual é feito todo o trabalho de configurar serviços internos. O utilizador tem a possibilidade de adicionar, editar ou remover um serviço deste tipo como também, a possibilidade de editar todos os parâmetros correspondentes ao mesmo.

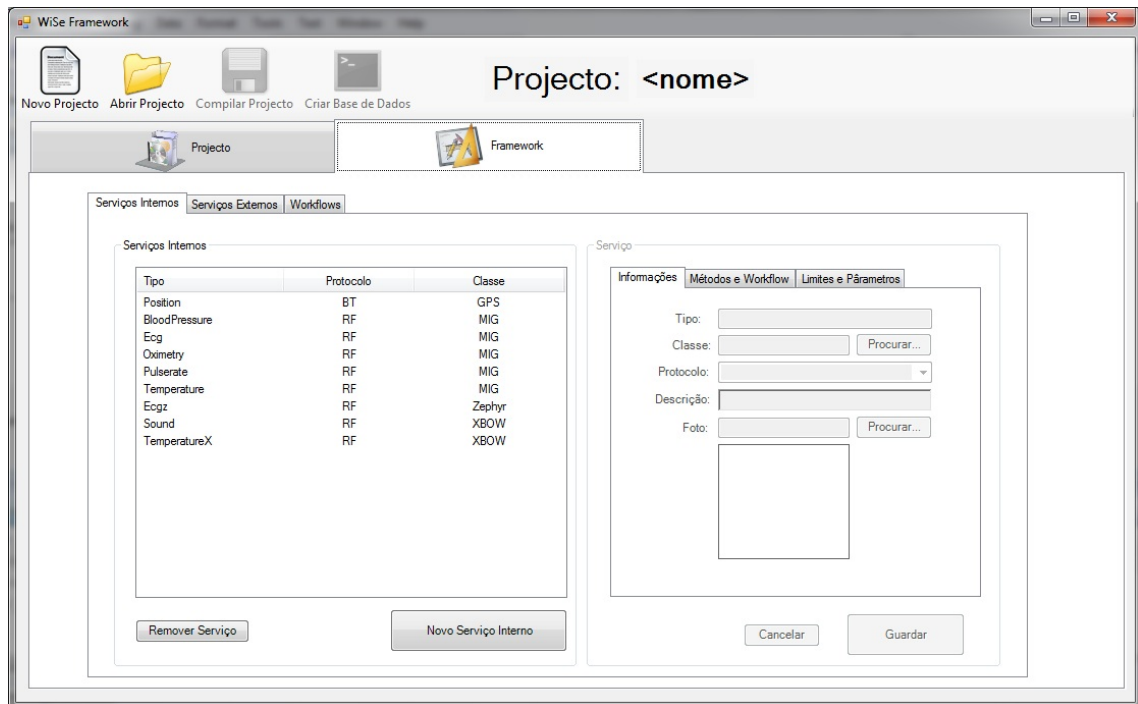


Figura 6.20: Serviços internos

Este é o pop-up que aparece para inserir um novo serviço interno:

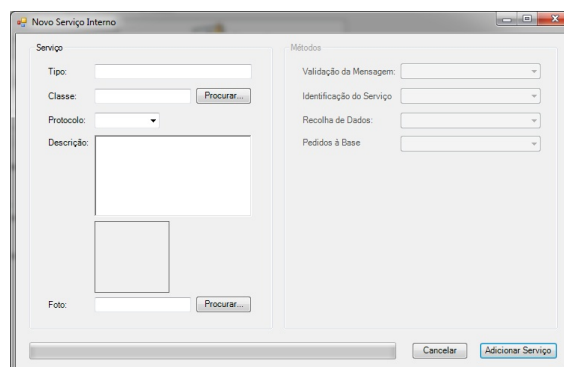


Figura 6.21: Pop-Up novo serviço interno

2. **Serviços Externos:** esta tab tem as mesmas funcionalidades que a tab anterior mas para serviços externos. Como os serviços externos não necessitam de um parser, as opções relativas a estes não existem nesta tab.

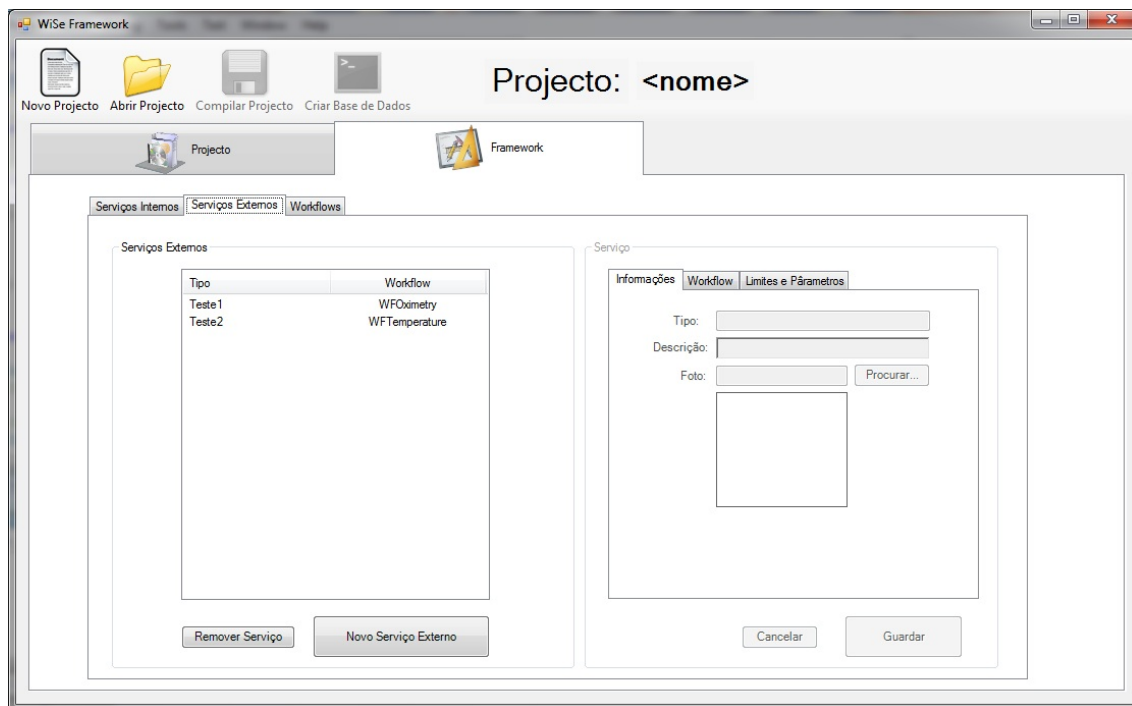


Figura 6.22: Serviços Externos

Este é o pop-up que aparece para inserir um novo serviço externo:

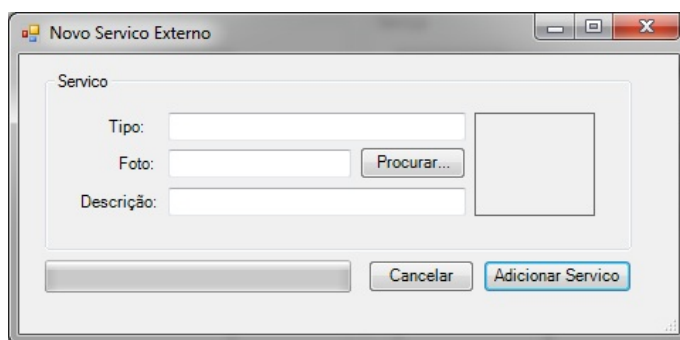


Figura 6.23: Pop-Up novo serviço externo

3. **Workflows:** Esta tab permite importar ou remover workflows. É possível remover vários workflows ao mesmo tempo. A remoção de um workflow implica a desassociação do mesmo ao respectivo serviço.

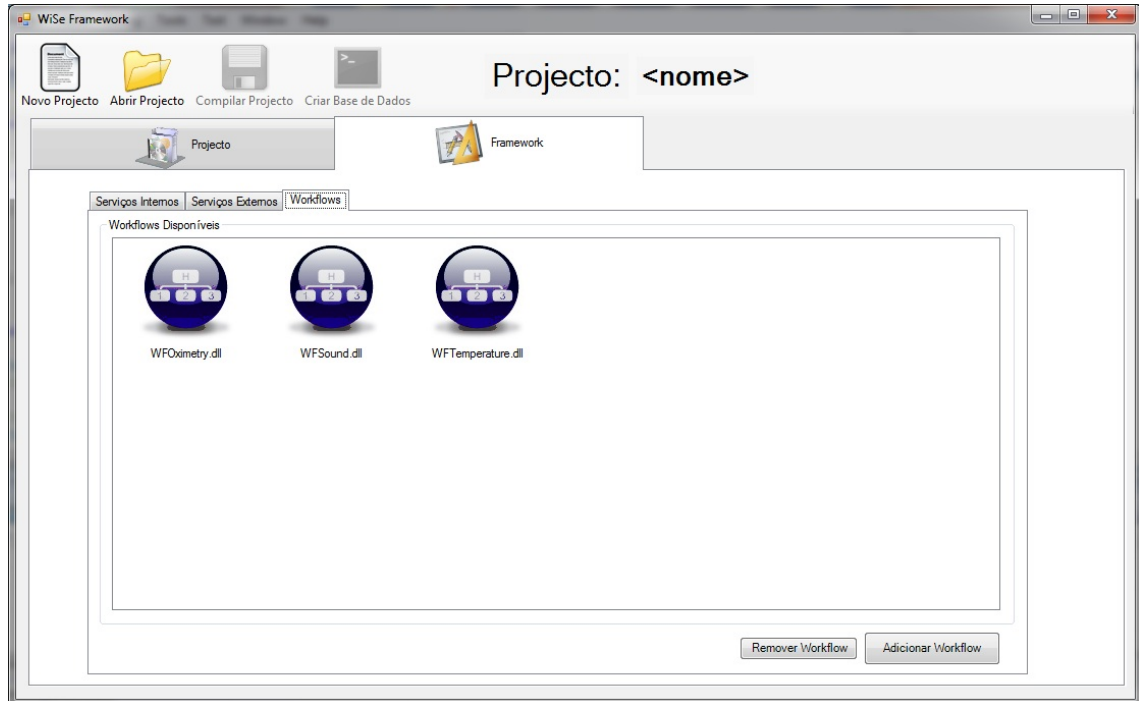


Figura 6.24: Workflows

Adicionar um workflow é bastante simples, basta escolher a classe e o workflow dessa classe:

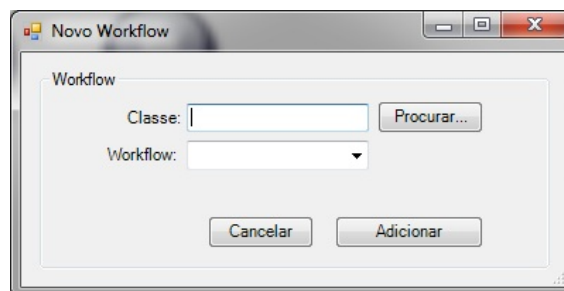


Figura 6.25: Pop-Up novo Workflow

6.5.5.5 Compilar Projecto

Após todos os dados e configurações do projecto estarem prontas, o utilizador pode então compilar o seu projecto para obter a aplicação final.

Deverá então fazê-lo pressionando botão "Compilar Projecto".

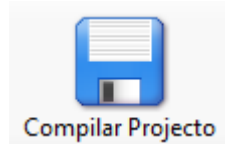


Figura 6.26: Botão compilar projecto

O Framework Wizard encarregar-se-á de compilar o Projecto e criar os ficheiros necessários á execução da aplicação final. Não esquecer que este processo é afectado pelas opções de compilação escolhidas.

6.5.5.6 Criar Base de Dados

Esta é uma nova funcionalidade que permite ao utilizador recriar a base de dados através do interface do Framework Wizard.

Esta opção só está disponível caso o projecto já tenha sido compilado previamente.

Para tal, basta que o utilizador pressione o seguinte botão:

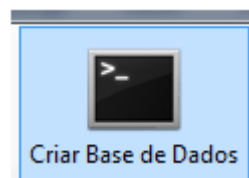


Figura 6.27: Botão criar base de dados

Esta opção faz lançar o generator remotamente. A sua janela aparecerá no ecrã.

6.6 Testes

6.6.1 Introdução

Neste capítulo são tratados os testes que foram executados para comprovar o sucesso ou insucesso desta dissertação.

Ao contrário da maior parte das aplicações informáticas que trabalham com valores específicos ou computam cálculos matemáticos em que é possível extrair taxas de sucesso ou insucesso, a WiSe Framework não tem qualquer tipo de métrica útil na qual seja possível estimar valores de sucesso ou insucesso. Em termos muito simples, se no final, a aplicação resultante funcionar como o esperado, é considerado sucesso. E caso não funcione como esperado, é considerado insucesso.

Podemos ver este tipo abordagem como um pequeno teste unitário:

- A aplicação funciona de acordo com o esperado? Se sim então sucesso, se não então insucesso.

Tendo em conta esta abordagem é então possível fazer uma série de testes unitários a cada uma das componentes da WiSe Framework. Na qualidade de autor deste documento e autor da implementação desta dissertação, é fácil de constatar que o ideal é comparar o antes e o depois da implementação. Como tal, foram realizados 2 tipos de testes, um com a versão inicial da WiSe Framework e outro com a versão final da mesma orientada pelo método de utilização proposto no capítulo 5. Em cada um destes dois tipos foram executadas as tarefas necessárias para determinar o sucesso da WiSe Framework.

Mais informação sobre estes testes pode ser encontrada no Anexo A.

6.6.2 Inclusão de um novo serviço - Amplitude de Som

Este teste foi um teste aplicado à versão inicial da WiSe Framework. Foi um teste muito completo pois abrangeu todas as áreas que a ferramenta tinha à disposição.

Depois de identificado o hardware a utilizar, a CrossBow, foi necessário criar o parser respectivo para a interpretação das suas mensagens. O primeiro obstáculo surgiu logo nesta etapa pois era necessária usar a própria Universal Gateway juntamente com as motes a funcionar para poder desenvolver este parser. Concluída esta tarefa faltava um workflow para trabalhar os dados obtidos.

Para poder fazer um workflow e verificar a sua fiabilidade foi necessário desenvolver em primeiro lugar, a estrutura da Base de dados. Ora, esta abordagem foi baseada em “tentativa erro” e o seu desenvolvimento foi muito lento.

Para a próxima etapa, era necessário adaptar as funcionalidades do interface e para que tal fosse possível, foi necessário alterar código fonte do mesmo. Esta tarefa foi muito delicada pois era necessário, alterar o código fonte e voltar ao Framework Wizard para poder gerar a aplicação resultante.

6.6.2.1 Conclusões:

Em termos gerais, o Framework Wizard da fase inicial da WiSe Framework não tem praticamente importância alguma pois quase todo o trabalho é feito nos ambientes de desenvolvimento de cada uma das aplicações criadas.

No entanto, a criação de código esqueleto de certos componentes é uma vantagem excepcional.

Não foi possível criar o Parser nem o Workflow sem ser nas soluções de cada uma das aplicações respectivas, a Universal Gateway e o WiSeServer. Houve imensos problemas na construção da Base de Dados e a solução para os mesmos implica alterar código fonte do generator.

6.6.3 Implementação de um novo serviço externo - Serviço de Localização e Seguimento

Este teste foi muito diferente do anterior pois foi executado sobre a nova versão da WiSe Framework. Seguindo os passos do método de utilização foi muito rápido iniciar o desenvolvimento. Como o serviço a adicionar é um serviço externo, não foi necessário criar um parser então avançou-se para a criação do workflow respectivo. Esta tarefa foi muito rápida devido ao suporte oferecido pela Workflow_tool.

Compilar o projecto com este workflow e o novo serviço foi a próxima etapa. A nova versão do Framework Wizard foi uma grande ajuda pois com o seu interface mais intuitivo permitiu uma rápida configuração de todos os parâmetros. A base de dados foi criada no acto de compilação o que diminuiu o trabalho para o utilizador.

Como tarefa final, faltava adaptar o interface ao serviço adicionado. Para tal foi criado um plugin e as dependências necessárias e este foi adicionado ao interface. Com alguns ajustes em pequenos parâmetros foi possível concluir todo este processo de forma rápida e eficaz.

6.6.3.1 Conclusões:

Sem nenhuma duvida que comparando os dois testes é possível identificar todas as melhorias e vantagens da versão actual da WiSe Framework. A ferramenta está muito mais ágil e não necessidade de alterar o código fonte aumenta a fluidez de desenvolvimento.

As ferramentas para criar o Parser e o Workflow são uma mais-valia que vão criar um ambiente isolado e permitem obter dois elementos importantes para o Framework Wizard e para a aplicação daí resultante.

A utilização de plugins para a criação de novas interfaces permite também uma enorme expansibilidade e possibilita ao utilizador a separação entre o que é a aplicação resultante e os plugins.

Capítulo 7

Conclusões e Trabalho Futuro

Neste último capítulo são apresentadas as conclusões sobre o resultado do trabalho que foi desenvolvido. É também mostrado em que medida este foi inovador, que limitações existem e, também, que melhoramentos podem advir em continuar o seu desenvolvimento.

7.1 Satisfação dos Objectivos

Para cumprir os objectivos inicialmente propostos foi dada como solução a criação de um método organizado e estruturado para a criação de aplicações baseadas em RSSF assim como a adaptação de uma versão inicial de uma ferramenta, a WiSe Framework.

De facto, ao que no início parecia apenas o desenvolvimento de uma ferramenta de trabalho levou ao desenvolvimento de uma metodologia de trabalho à qual foi aliada um conjunto de ferramentas. Entre estas ferramentas e a metodologia proposta, existe uma relação de dualidade. Ambas são dependentes entre si. Isto é, o método foi alcançado com base nas capacidades destas ferramentas e estas foram adaptadas ao método proposto.

Ao conjunto destas ferramentas intitulou-se WiSe Framework e á metodologia proposta de "método de utilização da WiSe Framework".

O objectivo desta dissertação era o de criar uma ferramenta ágil que permitisse o desenvolvimento rápido de aplicações baseadas em RSSF. Os resultados obtidos foram além do objectivo inicial. De facto, temos actualmente não só uma ferramenta mas um conjunto de ferramentas que para além de agilizarem o processo de desenvolvimento destas aplicações, permitem também agilizar o processo de depuração, análise de dados e distribuição de tarefas de desenvolvimento.

Aliado a estas ferramentas ainda temos o método que será como um guia para o uso das mesmas e talvez posso vir a tornar-se um padrão para o desenvolvimento deste tipo de aplicações.

No conjunto da WiSe Framework encontram-se 3 aplicações principais que vão incidir sobre as 3 camadas gerais de uma aplicação para RSSF, respectivamente, dando assim suporte para qualquer tipo de hardware, de dados a processar e também no tipo de interface desejado.

Para o desenvolvimento das dependências que cada uma dessas aplicações possui, existem outras ferramentas de suporte e funcionalidades extra como o Parser_tool, o Workflow_tool e o motor de plugins. Estas ferramentas permitem agilizar e acelerar o desenvolvimento das dependências e o motor de plugins permite a expansibilidade a novos tipos de interface.

Para além do suporte às 3 camadas gerais de uma aplicação para RSSF que a WiSe Framework oferece, existe também outros dois componentes. O componente da Base de dados, camada cuja funcionalidade é a de guardar todos os dados que circulam nas outras camadas, é suportada por uma aplicação, o generator, que é o responsável por criar toda a sua estrutura e adicionar a informação estática de forma completamente transparente para o utilizador. Por último, existe um outro componente, o Framework Wizard, que é o responsável por, literalmente, juntar "todas as peças" e criar a aplicação baseada em RSSF. O Framework Wizard contém um interface simples e permite a transparência, abstracção e boa usabilidade para o utilizador final.

O método proposto consiste em 6 passos dos quais nem todos são necessários executar. Isto é, a utilização desejada a dar à WiSe Framework é que vai definir o conjunto de passos a realizar. No entanto, nunca serão mais que 6 passos.

Mediante esta capacidade é possível dizer que existe uma grande satisfação com os resultados obtidos mas que como sempre, existem algumas melhorias possíveis.

7.2 Limitações

A maior limitação é, sem dúvida, ao nível do interface da aplicação final. O que acontece é que a sua estrutura é baseada em MVC e devido a esta aplicação ter como objectivo ser o mais dinâmica e expansível possível numa aplicação web, não é possível fechar o seu código fonte principalmente porque a criação de novas categorias (serviços internos ou externos) implica a adição de novo código fonte a esta aplicação. Felizmente, esta adição de código é feita de forma automática pelo Framework Wizard e transparente para o utilizador mas esta possibilidade é a maior limitação da WiSe Framework.

A segunda maior limitação é as dependências impostas pela WiSe Framework. Isto é, um parser, um workflow e um plugin (caso deseje outro tipo de interface). Era agradável se a aplicação fosse completamente auto-suficiente. Estas dependências são retratadas

como limitações porque caso não existem, o processo de criação de aplicações baseadas em RSSF era quase instantâneo. No entanto, esta limitação pode ser vista como uma mais-valia pois permite uma enorme diversidade de aplicações finais.

7.3 Trabalho Futuro

Todo o trabalho executado foi testado devidamente e tendo passado todos os testes não se acha nenhuma melhoria a fazer. No entanto, não caindo na categoria de melhorias mas sim de "novas funcionalidades", podemos considerar a mudança de arquitectura do interface final de MVC para outro tipo de arquitectura como por exemplo a PAC [[Arc](#)] ou uma arquitectura criada de propósito para o efeito, facilitando assim a não adição de código, o que levaria a uma aplicação final mais sólida e menos vulnerável que impeça o acesso a quem não tenha autorização.

Referências

- [Arc] Garfixia Software Architectures. Presentation-abstraction-control. http://www.dossier-andreas.net/software_architecture/pac.html.
- [Bad] Glenn Baddeley. Glenn baddeley - gps - nmea sentence information. <http://home.pacific.net.au/~gnb/gps/nmea.html>.
- [BPC⁺07] Paolo Baronti, Prashant Pillai, Vince W. C. Chook, Stefano Chessa, Alberto Gotta e Yim-Fun Hu. Wireless sensor networks: A survey on the state of the art and the 802.15.4 and zigbee standards. *Computer Communications*, 30(7):1655–1695, 2007.
- [CCS09] Xianghui Cao, Jiming Chen e Youxian Sun. An interface designed for networked monitoring and control in wireless sensor networks. *Computer Standards & Interfaces*, 31(3):579–585, 2009.
- [CLT08] Wan-Young Chung, Seung-Chul Lee e Sing-Hui Toh. Wsn based mobile u-healthcare system with ecg, blood pressure measurement function. *Conf Proc IEEE Eng Med Biol Soc*, 2008:1533–6, 2008.
- [Cro07] CrossBow. *MoteWorks Getting Started Guide*, March 2007.
- [DBL06] *Proceedings of the International Conference on Systems and Networks Communications (ICSNC 2006), October 29 - November 3, 2006, Papeete, Tahiti, French Polynesia*. IEEE Computer Society, 2006.
- [dCdAdIS] Eurico Neves Presidente do Conselho de Administração da INOVA+ S.A. Mensagem. <http://www.inovamais.pt>.
- [DSY⁺09] Fei Ding, Guangming Song, Kaijian Yin, Jianqing Li e Aiguo Song. A gps-enabled wireless sensor network for monitoring radioactive materials. *Sensors and Actuators A: Physical*, 155(1):210 – 215, 2009.
- [GGW⁺05] Tia Gao, Dan Greenspan, Matt Welsh, Radford Juang e Alex Alm. Vital signs monitoring and patient tracking over a wireless network. *Conf Proc IEEE Eng Med Biol Soc*, 1:102–5, 2005.
- [HA06] Stephan Hengstler e Hamid K. Aghajan. Application development in vision-enabled wireless sensor networks. In *ICSNC* [DBL06], page 30.
- [JHS08] Won-Suk Jang, William M. Healy e Mirosław J. Skibniewski. Wireless sensor networks as part of a web-based building environmental monitoring system. *Automation in Construction*, 17(6):729 – 736, 2008.

REFERÊNCIAS

- [KC05] Séamus Murphy Mark Roantree Dermot Diamond Karl Crowley, June Frisby. Web-based real-time temperature monitoring of shellfish catches using a wireless sensor network. *Sensors and Actuators*, A(122):222–230, May 2005.
- [LVCÁ⁺07] Fernando Losilla, Cristina Vicente-Chicote, Bárbara Álvarez, Andrés Iborra e Pedro Sánchez. Wireless sensor network application development: An architecture-centric mde approach. In Oquendo [Oqu07], pages 179–194.
- [LW05] Konrad Lorincz e Matt Welsh. Motetrack: A robust, decentralized approach to rf-based location tracking. In Strang e Linnhoff-Popien [SLP05], pages 63–82.
- [LW07] Konrad Lorincz e Matt Welsh. Motetrack: a robust, decentralized approach to rf-based location tracking. *Personal and Ubiquitous Computing*, 11(6):489–503, 2007.
- [Mes08] MeshNetics. *ZigBit Development Kit 2.3 user's guide*, October 2008.
- [Mica] Microsoft. Ajax: The official microsoft asp.net site. <http://www.asp.net/ajax?tabid>.
- [Micb] Microsoft. Asp.net mvc: The official microsoft asp.net site. [http://www.asp.net/\(S\(d35rmemuonolwvmlgsp2n45\)\)/mvc](http://www.asp.net/(S(d35rmemuonolwvmlgsp2n45))/mvc).
- [Micc] Microsoft. Microsoft .net framework. <http://www.microsoft.com/net/>.
- [Micd] Microsoft. Windows workflow foundation. <http://msdn.microsoft.com/en-us/library/ms735967.aspx>.
- [MOJ06] Aleksandar Milenkovic, Chris Otto e Emil Jovanov. Wireless sensor networks for personal health monitoring: Issues and an implementation. *Computer Communications*, 29(13-14):2521–2533, 2006.
- [NSBO08] E.S. Nadimi, H.T. Søgaaard, T. Bak e F.W. Oudshoorn. Zigbee-based wireless sensor networks for monitoring animal presence and pasture time in a strip of new grass. *Computers and Electronics in Agriculture*, 61(2):79 – 87, 2008.
- [OCT06] OCTAVEX. *User's guide*, November 2006.
- [Oqu07] Flávio Oquendo, editor. *Software Architecture, First European Conference, ECSA 2007, Aranjuez, Spain, September 24-26, 2007, Proceedings*, volume 4758 of *Lecture Notes in Computer Science*. Springer, 2007.
- [PLH⁺07] Shyamal Patel, Konrad Lorincz, Richard Hughes, Nancy Huggins, John H Growdon, Matt Welsh e Paolo Bonato. Analysis of feature space for monitoring persons with parkinson's disease with application to a wireless wearable sensor system. *Conf Proc IEEE Eng Med Biol Soc*, 2007:6291–4, 2007.

REFERÊNCIAS

- [PLH⁺09] Shyamal Patel, Konrad Lorincz, Richard Hughes, Nancy Huggins, John Growdon, David Standaert, Metin Akay, Jennifer Dy, Matt Welsh e Paolo Bonato. Monitoring motor fluctuations in patients with parkinson's disease using wearable sensors. *IEEE Trans Inf Technol Biomed*, 13(6):864–73, Nov 2009.
- [RBZ05] Jason Redi, Hari Balakrishnan e Feng Zhao, editors. *Proceedings of the 3rd International Conference on Embedded Networked Sensor Systems, SenSys 2005, San Diego, California, USA, November 2-4, 2005*. ACM, 2005.
- [Sen08] Sentilla. *User's guide*, Abril 2008.
- [SLP05] Thomas Strang e Claudia Linnhoff-Popien, editors. *Location- and Context-Awareness, First International Workshop, LoCA 2005, Oberpfaffenhofen, Germany, May 12-13, 2005, Proceedings*, volume 3479 of *Lecture Notes in Computer Science*. Springer, 2005.
- [SrCL⁺05] Victor Shnayder, Bor rong Chen, Konrad Lorincz, Thaddeus R. F. Fulford Jones e Matt Welsh. Sensor networks for medical care. In Redi et al. [RBZ05], page 314.
- [sta] stackOverflow. Render partial from different folder (not shared). <http://stackoverflow.com/questions/208421/render-partial-from-different-folder-not-shared>.
- [TG] Leo Selavo Tia Gao, Christopher Pesto. Wireless medical sensor networks in emergency response: Implementation and pilot results.
- [TKCM08] A Triantafyllidis, V Koutkias, I Chouvarda e N Maglaveras. An open and reconfigurable wireless sensor network for pervasive health monitoring. *Methods Inf Med*, 47(3):229–34, 2008.
- [Vit09] VitalIT. *WiSe Framework, Manual do Utilizador*, June 2009.
- [w3sa] w3schools.com. Css tutorial. <http://www.w3schools.com/css/>.
- [w3sb] w3schools.com. Javascript tutorial. <http://www.w3schools.com/js/default.asp>.

Anexo A

Testes

A.1 Inclusão de um novo serviço - Amplitude de Som

Na fase inicial da WiSe Framework não há nenhum procedimento ou método a seguir portanto, a primeira tarefa que um utilizador tem é a de abrir o Framework Wizard e tentar a partir daí iniciar alguma acção.

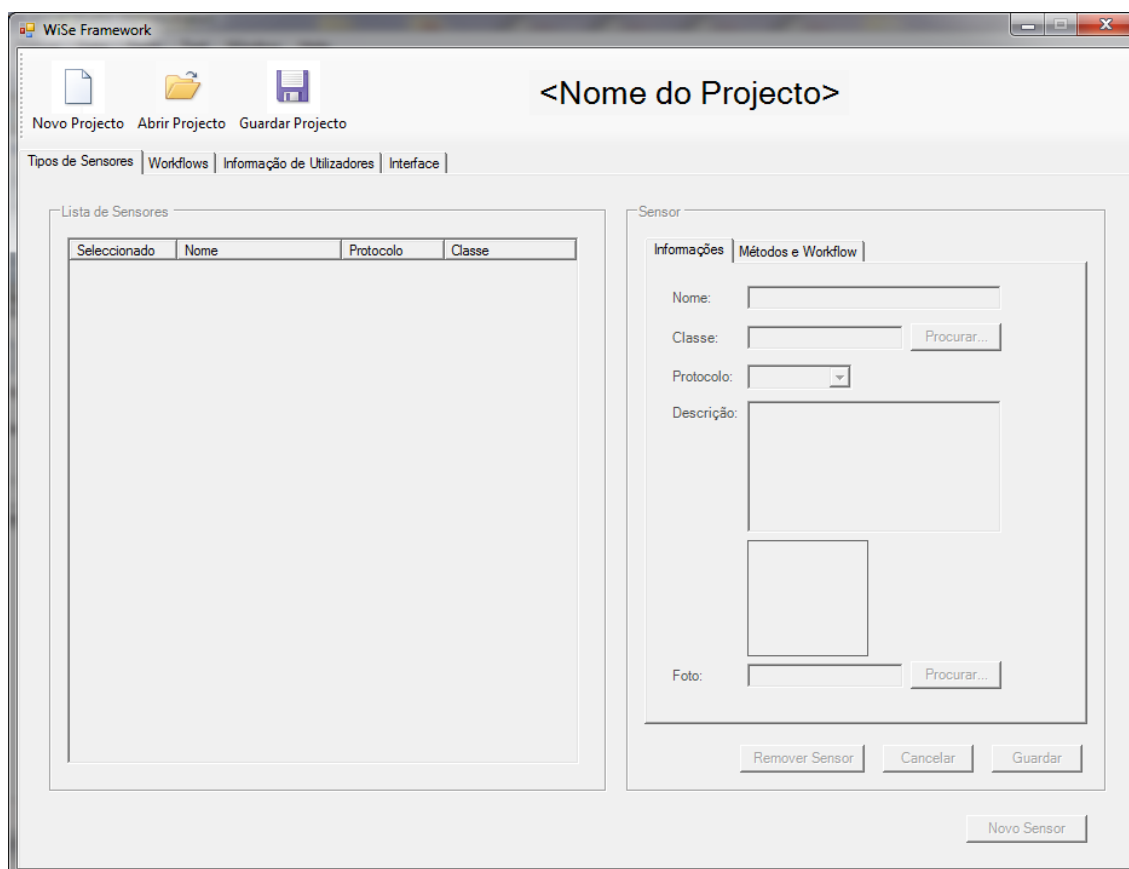


Figura A.1: Interface inicial que obriga a criar ou abrir um projecto

Para iniciar alguma acção no Framework Wizard, o utilizador tem de ter um projecto aberto (ou criado), então, a primeira tarefa consiste em criar um projecto novo ou abrir

um existente.

O objectivo deste teste é a adição de um novo sensor, então vamos criar esse novo sensor no Framework Wizard. Por sinal, o interface de adição de um novo sensor pede o ficheiro de parser correspondente e como esse ficheiro ainda não foi criado não podemos adicionar o novo sensor.

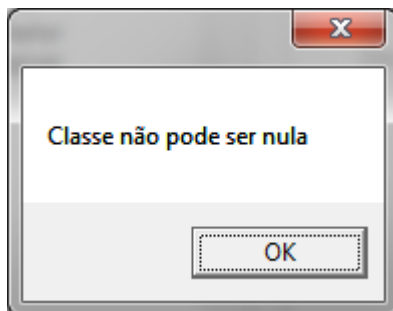


Figura A.2: Pedido de um ficheiro pelo interface

Para criar esse o ficheiro o utilizador deve programa-lo mas para fins de teste do mesmo não tem outra hipótese senão utilizar a Universal Gateway em tempo real com o sensor de som a funcionar em tempo real. Perante esta restrição, é necessário então ter uma aplicação resultante da WiSe Framework a funcionar para permitir o desenvolvimento do Parser do novo sensor.

Foi então compilada assim uma aplicação com o Framework Wizard mas com todas as configurações em branco, isto é, sem nenhum outro sensor ou informação adicionada para facilitar os testes ao novo sensor. Esta aplicação resultante foi instalada e executada verificando-se que tudo estava a funcionar correctamente.

Iniciou-se o desenvolvimento do Parser através da solução de desenvolvimento da Universal Gateway como mostra a figura A.3. Com o sensor de som a funcionar e com a Universal Gateway a imprimir para o ecrã as leituras efectuadas, procedeu-se à eliminação de erros do Parser e à sua optimização. Foi também necessário editar manualmente o ficheiro sensores.xml para que a Universal Gateway conseguisse reconhecer o novo sensor em questão.

Testes

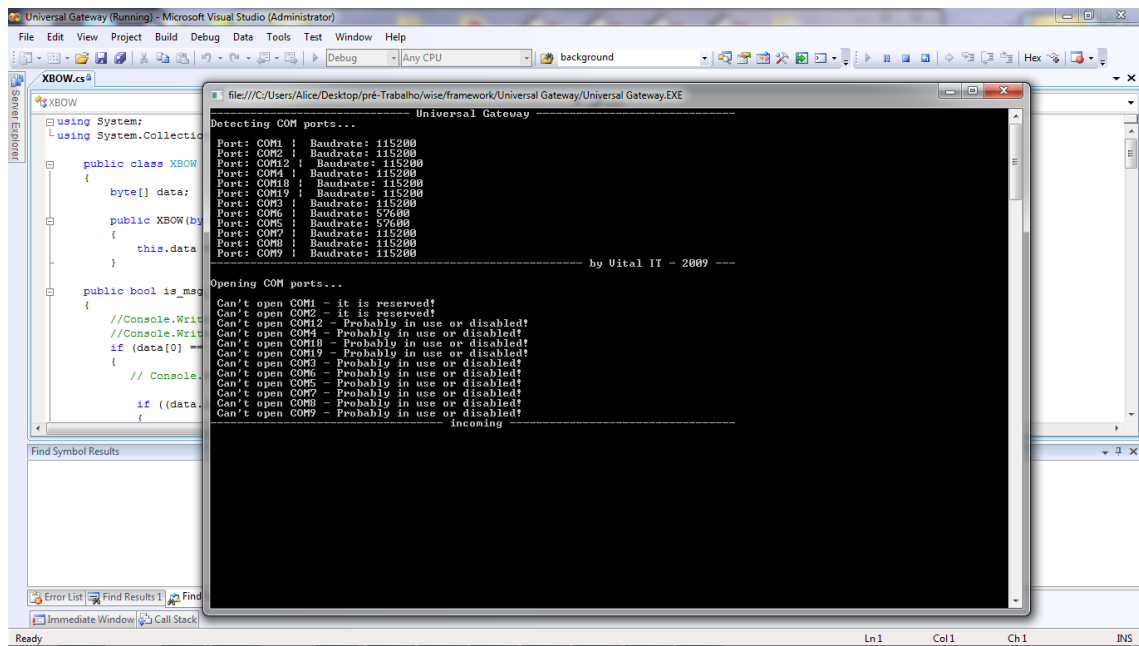


Figura A.3: Desenvolvendo um Parser

Nesta fase, é importante que o WiseML esteja a ser criado correctamente pois é essa a mensagem que vai passar para a camada seguinte. Tendo-se verificado essa condição, deu-se por encerrado o desenvolvimento do Parser e prosseguiu-se com a sua inserção no Framework Wizard.

Voltando à tarefa de inserir um novo sensor no Framework Wizard conseguiu-se ultrapassar a barreira na qual o Framework Wizard pedia por um Parser para o novo sensor de som. No entanto, outra barreira se levantaram: O workflow do sensor. Para tal o Framework Wizard disponibiliza um botão que gera um novo workflow mas com um esqueleto de código de um workflow genérico. É necessário agora editar esse esqueleto dentro da solução de desenvolvimento no WiSeServer de forma a suportar o novo sensor. Para fins de teste, mais uma vez, é necessário usar uma aplicação resultante mas desta vez em conjunto com a Universal Gateway e o Parser desenvolvido anteriormente.

Testes

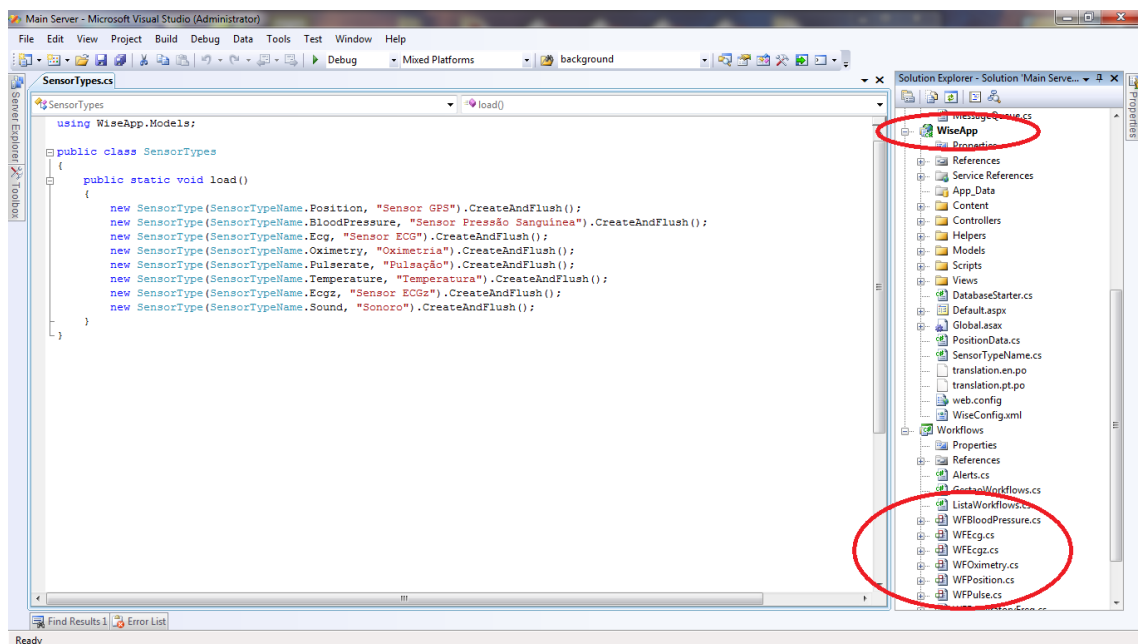


Figura A.4: Workflows no mesmo ambiente de desenvolvimento que a WiSeApp

Ao testar o workflow, verificou-se que é impossível saber se está a funcionar correctamente pois a base de dados não está preparada para os dados do novo sensor. Foi necessário resolver este problema. Como o Framework Wizard cria uma ferramenta automática para a criação da base de dados, o generator, optou-se por configurar o Framework Wizard para o novo sensor juntamente com o Parser e workflow associados. Isto faz com que o generator já cria a estrutura da base de dados com o suporte desejado para o novo sensor de som.

Testes

Ao correr o generator houve um erro que impediu a sua concretização. Verificou-se que se devia à falta de algumas configurações no mesmo que teriam de ser feitas com inserção manual de código, nomeadamente no ficheiro `SensorTypes.cs` e `WorkflowsParameters.cs`.

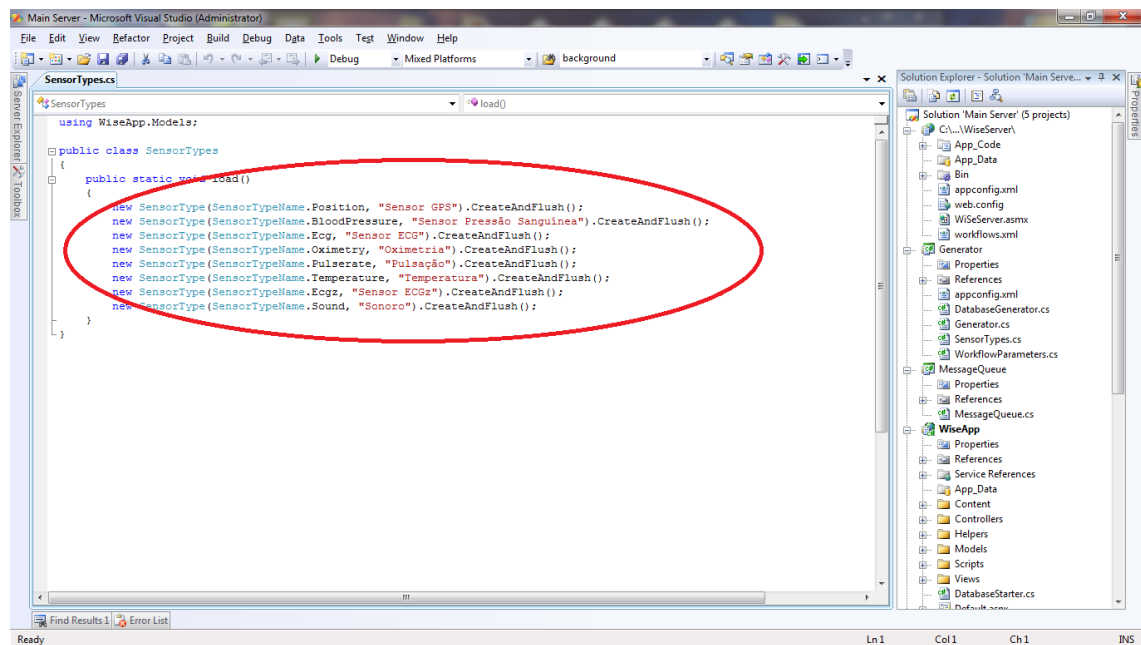


Figura A.5: Adição de tipos de sensor no código fonte do generator

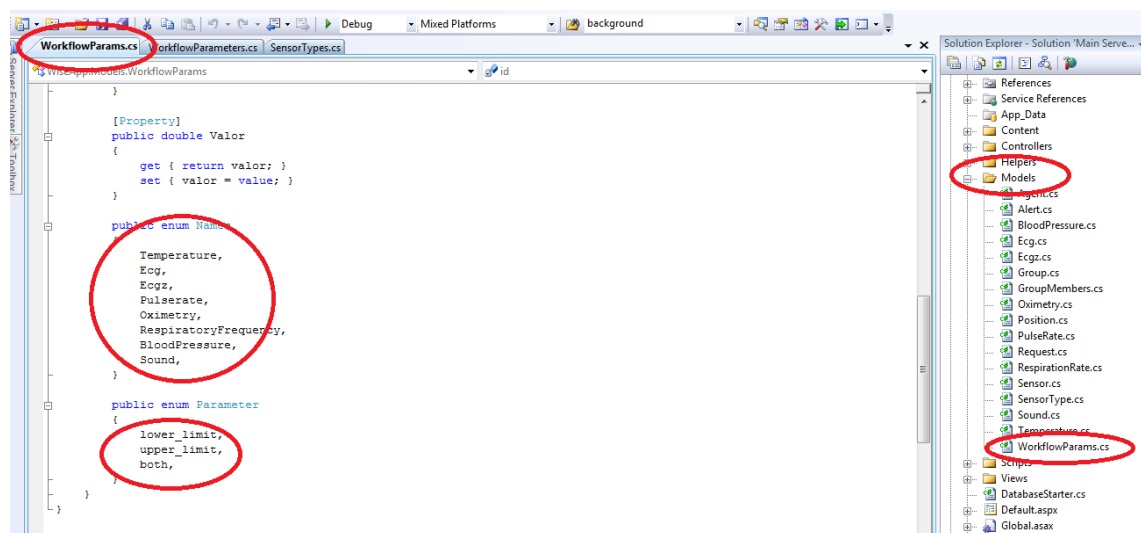


Figura A.6: Interligação entre o código fonte

Após estas alterações o generator correu sem problemas e criou a base de dados respectiva. Finalmente foi possível testar o workflow criado anteriormente e estava tudo de acordo com o esperado.

A próxima fase consiste no interface. É necessário que a WiSeApp consiga mostrar os valores que vão estar na base de dados de forma correcta. Felizmente a estrutura de dados do sensor de som, não difere muito da estrutura genérica de um sensor, como tal, o seu “model” não necessita de alterações. Por outro lado, o controller, componente que vai definir os dados a visualizar ainda não está completo.

Foi necessário editar o controller para corresponder ao novo sensor. Após esta tarefa já foi possível visualizar os dados do sensor som. Para finalizar, foi então feita a configuração manual dos ficheiros de tradução terminando assim o processo de inserção de um sensor novo, o sensor de som.

A.2 Implementação de um novo serviço externo - Serviço de Localização e Seguimento

Este teste obteve uma abordagem muito diferente do anterior. O objectivo é o de adicionar um novo serviço externo. Um serviço denominado de Serviço de Localização e Seguimento.

“Trata-se de um sistema que permite obter a localização (absoluta - sistema de coordenadas e/ou simbólica - sala nº2) de um objecto móvel. Este sistema recorre a uma Rede de Sensores sem fios, composta por motes (pequenos dispositivos de hardware...) que comunicam entre si, e que medem a intensidade de sinal RF nas mensagens recebidas”

Este teste teve imensa importância pois é um teste que valida quase todas as etapas do método de solução proposto anteriormente.

Como se tratava de um serviço externo apenas necessitamos de um workflow. Então o método inicia-se na fase de criação de um workflow.

Testes

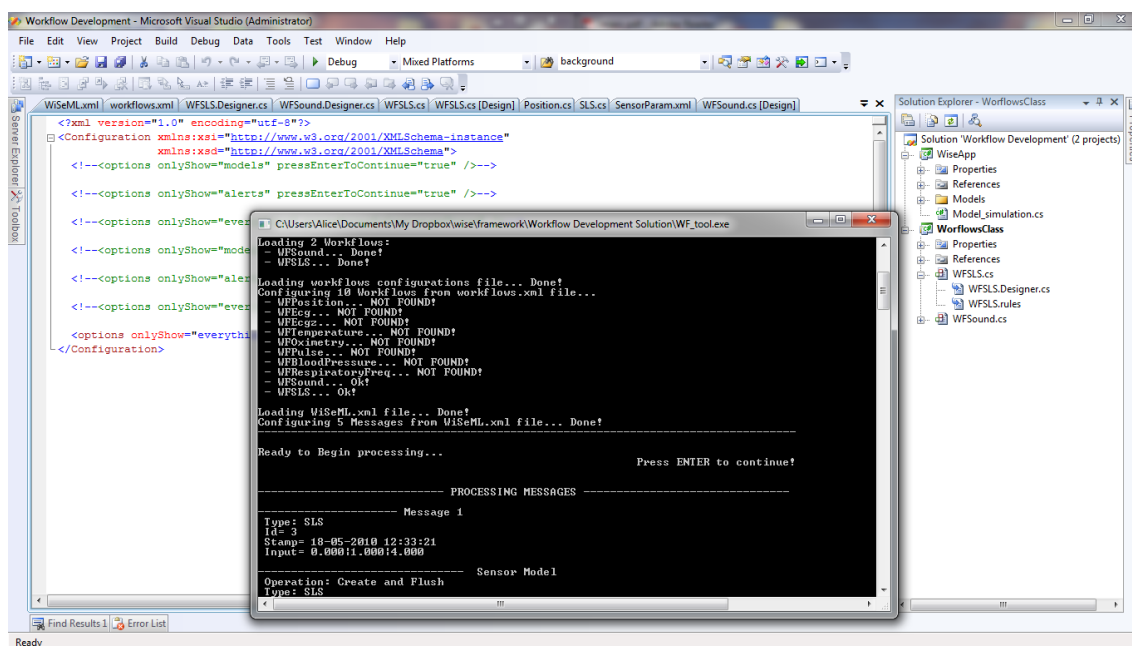


Figura A.7: Ambiente de desenvolvimento de workflows e a Workflow_tool

Com a ajuda da ferramenta Workflow_tool foi fácil desenvolver o workflow. Verificou-se que as funcionalidades de teste da Workflow_tool permitiram otimizar o processamento do workflow e estando todos os testes desta ferramenta superados, seguiu-se para a fase seguinte.

A fase seguinte consiste em criar o Projecto com o Framework Wizard e compilar. Esta etapa foi relativamente simples bastando alguns cliques para alcançar a configuração pretendida. O Framework Wizard oferece a opção de criar a base de dados no final da compilação. Esta opção foi aceite e com apenas um clique foi possível compilar o projecto e criar a base de dados.

Devido à reestruturação dos vários componentes da WiSe Framework já não se verificou a necessidade de editar código fonte de nenhum dos componentes. No entanto, o interface ainda não se encontrava pronto para suportar este tipo de serviço.

A última fase do método, visto que é um serviço externo e é um serviço específico que não funciona com um simples gráfico, é a de criação de um plugin para o Interface.

A utilização de plugins permite que não seja necessário editar nenhum código fonte da WiSeApp. No entanto, pelas razões discutidas na secção 6.5.1.2 é sempre necessário editar os controllers. Sendo que esta tarefa é agora muito mais simples.

O serviço de localização e seguimento é um serviço que fornece como dados um par de coordenadas (X,Y) e para uma correcta visualização é necessário imprimir essas coordenadas sobre um mapa da área em questão. Para tal, usou-se uma adaptação da biblioteca de gráficos existente na WiSeApp para fazer a sobreposição de pontos em cima de uma imagem estática pré-definida. O SLS suporta também um factor de erro que define

Testes

um raio de incerteza da posição real. Este valor foi usado como raio da circunferência a desenhar no mapa. Isto é, quanto maior o erro, maior a circunferência.



Figura A.8: SLS - Interface final

A criação de plugins pode ser feita em qualquer ambiente desde que o resultado final obedeça as regras impostas pelos mesmos mas para esta fase, optou-se por se criar o plugin com a ajuda da aplicação resultante da compilação prévia por parte do Framework Wizard.

Ficou então concluído o segundo teste.