Universidade de Aveiro
2013

Departamento de Electrónica, Telecomunicações e Informática.

Gonçalo Miguel Candeias Duarte

Simulador de eventos de Contexto em Domótica

Universidade de Aveiro
2013

Departamento de Electrónica, Telecomunicações e Informática.

Gonçalo Miguel Candeias Duarte

Simulador de eventos de Context em Domótica

Projecto apresentado pelo aluno Gonçalo Miguel Candeias Duarte, com o número mecanográfico 43518 à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Sistemas de Informação, realizada sob a orientação científica do Doutor. Diogo Gomes, professor auxiliar do Instituto de Telecomunicações da Universidade de Aveiro.

o júri

Presidente	Professor Doutor Luis Filipe de Seabra Lopes Professor Associado, Universidade de Aveiro
Vogal - Arguente Principal	Mestre Telma Susana Pinto Mota Especialista, Pt Inovacao
Vogal - Orientador	Doutor Diogo Nuno Pereira Gomes Professor Auxiliar Convidado, Universidade de Aveiro

agradecimentos

Em primeiro lugar agradeço ao professor Diogo Gomes ao ter depositado em mim a confiança para a realização deste projecto.

Aos meus pais João Duarte e Carolina Candeias, responsáveis por tudo aquilo que sou hoje.

À minha namorada Ana Delusinne, por todo o apoio e compreensão que me deu ao logo do meu percurso académico.

Ao Eng. José Mota da empresa Lightenjin Lda. ao dar-me a possibilidade de tornar o meu estágio profissional flexível para fosse possível frequentar o Mestrado em Sistemas de Informação.

A todos os colegas e amigos, que durante o seu desenvolvimento contribuíram com o seu conhecimento e opinião.

palavras-chave Tecnologia, Simulação, Domótica, Software, Comunicação, Sensores, Componente, Fenomenos, Casa.

resumo O presente projecto visa a criação de um simulador de eventos *contexto* em domótica, em que o principal ponto de destaque se prende com a possibilidade de criar uma simulação entre o sensor, componente e divisão da casa, o qual poderá auxiliar o desenvolvimento de serviços e aplicações baseadas em contexto recolhido a partir de uma plataforma de domótica.

keywords Technology, Simulation, Home Automation, Software, Communications, Sensors, Component, Phenomena, House.

abstract This project consists to create a home automation simulator, where the main point it is the possibility of creating a simulation between the sensor and the component division of the house.
You can support the development of services and applications, based on context collected from a home automation platform.

Índice

Sumário

Índice de Figuras

Índice de Tabelas

Capítulo 1 – Introdução Geral

1.1. Enquadramento e Motivação

Numa época em que a palavras de ordem são a racionalização de custos, e tendo a tecnologia um papel importante no dia-a-dia da sociedade actual, é importante prever como um investimento elevado pode contribuir para o bem-estar do seu utilizador. Assim, recorrer a um simulador torna-se a forma mais económica de prever e estudar os resultados de como um sistema se comporta no seu ambiente.

A motivação vai ao encontro do actual momento que se vive em casa das famílias, onde estas exigem uma melhoria na racionalização do consumo energético.

1.2. Estado da arte

Neste capítulo abordar-se-á um conjunto de aplicações que têm como objectivo simular eventos contexto na área da domótica.

Siafu - An opensource Contexto Simulator: Sem dúvida o simulador *Context* mais popular na internet é o que tem maior comunidade e maior suporte, devendo-se ao facto de ser *opensource* e ter o apoio da NEC, daí ser um projecto que tem tido continuidade no seu desenvolvimento.

Após os testes efectuados à aplicação, verificou-se que esta dá mais importância há forma como o “humano” se comporta num espaço e de como executa as suas tarefas. Também se pôde constatar que nos *plugins* disponibilizados no *site* oficial, os cenários a serem estudados são em cidades, entre cidades e em escritórios. Não foi encontrada qualquer referência a fenómenos como a claridade, temperatura ou humidade. O mais próximo que foi encontrado de como se comporta um fenómeno foi a propagação das ondas

WiFi. De referir que esta aplicação permite o envio de dados em rede via TCP/IP.

Free domotic: Este simulador gratuito permite a construção de um cenário a ser implementado numa casa. Os componentes existentes apresentam muitas opções de personalização. Os contras encontrados são os seguintes: 1) ausência de fenómenos, 2) só é apresentado o estado do componente e valor da leitura do mesmo e de sensores, 3) o tratamento dado a um sensor é igual ao componente, 4) não é possível adicionar, construir a planta da casa com os componentes e sensores é demasiado complicado, e por fim, 5) não é possível criar novos componentes e sensores.

SP Simulator: Após a análise deste simulador (pago) pôde-se concluir que executa as mesmas tarefas que o simulador estudado anteriormente (Free domotic), mas a grande diferença e ao mesmo tempo o ponto forte da seguinte aplicação é a apresentação 3D da simulação.

1.3. Objectivo

O objectivo genérico do projecto é o desenvolvimento de um software capaz de simular a variação dos fenómenos existentes numa residência, assim como os diferentes componentes (televisores, iluminação, electrodomésticos, etc.). O simulador deve ser capaz de se interligar com uma aplicação externa, de forma a receber e enviar dados com o objectivo de, durante a simulação, se conseguir uma optimização do consumo energético e gerar o conforto desejado pelos seus habitantes.

Para a simulação do sistema de domótica criado na simulação devemos ter em conta os seguintes pontos:

- Sensores: os sensores estão sempre activos e lêem os valores do local, como a temperatura, luminosidade, humidade, entre outros.
- Componentes: os componentes afectam os fenómenos no local quando ligados ou desligados. Podem eles ser ares condicionados, televisores, iluminação entre outros.

- Comunicação: as leituras realizadas pelos sensores devem ser enviadas para a rede informática, e os componentes devem receber ordens a partir da rede informática.

1.4. Metodologia

Segundo o PROMODEL® USER'S GUIDE (2002), a definição dos passos a serem seguidos para se obter bons resultados numa simulação com o mínimo de recursos utilizados varia muito de acordo com o projecto, porém os procedimentos básicos são essencialmente os mesmos. Esses passos básicos de "como realizar uma simulação" são divididos em seis:

· *Plano de estudo*. Consiste em estabelecer objectivos e definir ferramentas; restrições, principalmente quanto a tempo e custo; especificação da simulação, que define o seu planeamento (tamanho e complexidade da actividade a ser simulada), nível de detalhe (quanto maior, maior será o custo), tipo de experiência (natureza e número de soluções diferentes), forma do resultado final, orçamento.

· *Definindo o sistema.* Identificação do modelo conceitual no qual a simulação será baseada. Nesse passo há a tomada de dados, e para ajudar a organizar o processo de junção e análise desses dados para a definição do sistema os seguintes passos são indicados: definição dos dados requeridos, fontes apropriadas de dados, fazer certas considerações quando necessário, converter os dados apropriadamente e documentar e aprovar os dados.

· *Construção do modelo.* O objectivo da construção do modelo é prover uma representação válida do sistema definido. Refinamento dos dados, possível expansão do sistema, verificação e validação do modelo, são alguns passos dessa fase.

· *Testes.* Baseado no resultado da simulação o modelador tem uma resposta sobre a validade de suas hipóteses, postas no sistema. Em um teste de simulação há a entrada de variáveis definidas no modelo, tais variáveis são independentes e podem ser manipuladas. O efeito dessa manipulação é então medido e correlacionado. Embora o software ajude nessa fase, cabe ao

modelador decisões do tipo: número de replicações do teste, o tamanho e complexidade da simulação, etc.

· *Análise dos resultados.* Quando se faz a condução de experiências de simulação, há a necessidade de cuidados ao analisar os resultados. O maior benefício da simulação é dar ideias de “o que acontece se”.

· *Reportar os resultados.* O último passo é dar recomendações para melhorias no sistema actual baseado nos resultados da simulação. Tais resultados devem ser claramente apresentados para a tomada de decisão final.

1.5. Estrutura do projecto

A estrutura deste relatório encontra-se agrupada em cinco capítulos distintos.

O segundo capítulo deste relatório faz uma apresentação da arquitectura do sistema.

No terceiro capítulo são apresentadas as tecnologias, ferramentas e bibliotecas utilizadas no desenvolvimento do projecto em questão.

O quarto capítulo visa explanar todo o processo de desenvolvimento e implementação da aplicação. Neste é detalhado o código produzido e explicadas as decisões tomadas ao longo do processo de desenvolvimento.

No quinto capítulo são apresentadas as conclusões retiradas do trabalho desenvolvido bem com apresentadas considerações gerais acerca deste.

2.1. Ambiente gráfico

A visualização da planta da casa seria uma mais-valia para a analisar o sistema proporcionaria ao utilizador um ambiente de trabalho mais amigável. Esta representação gráfica permitiria também ter uma visão centralizada, contendo toda a informação relevante, da simulação.

Na concepção da arquitectura deste simulador, entendeu-se que esta deveria prever mecanismos que facilitassem a criação da representação gráfica da planta da casa, bem como os sensores e componentes.

Como tal, foi apenas definida a necessidade de especificar os sensores e componentes, sendo que o ambiente gráfico, permite a interacção com o sistema, gerando eventos automaticamente.

Foi determinado que o aspecto gráfico iria ser idêntico ao apresentado na ferramenta de desenvolvimento Visual Studio, já que, após a análise do objectivo do sistema, esta estrutura gráfica ser a que permitiria a maior facilidade de utilização.

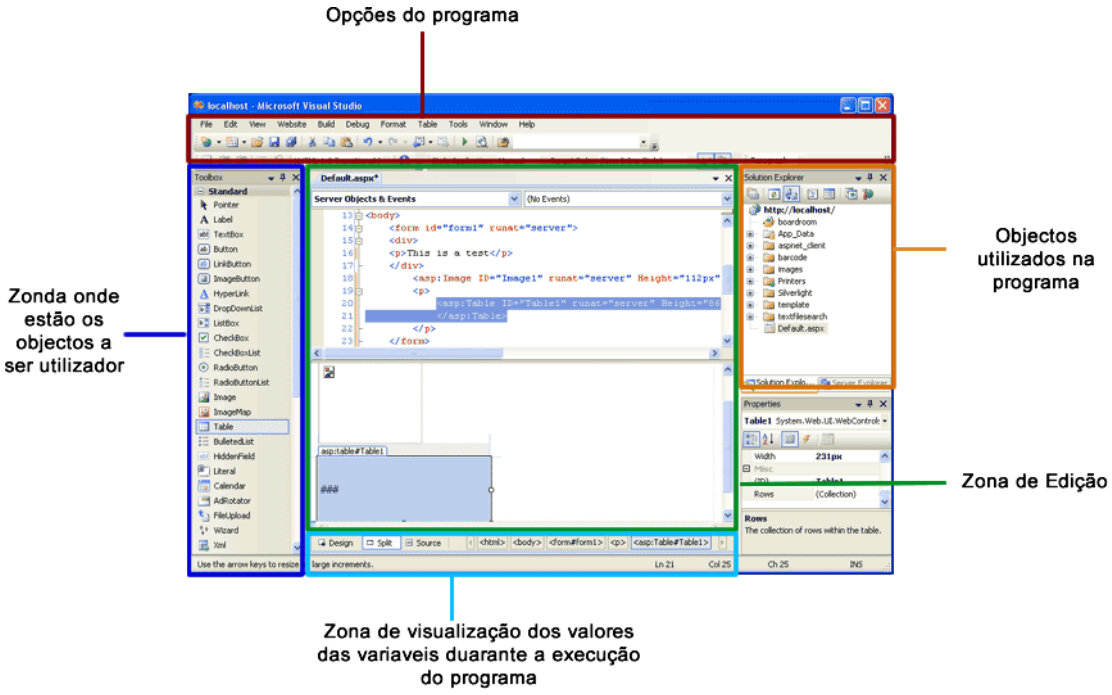

Figura 1 - Aspecto gráfico do Visual Studio

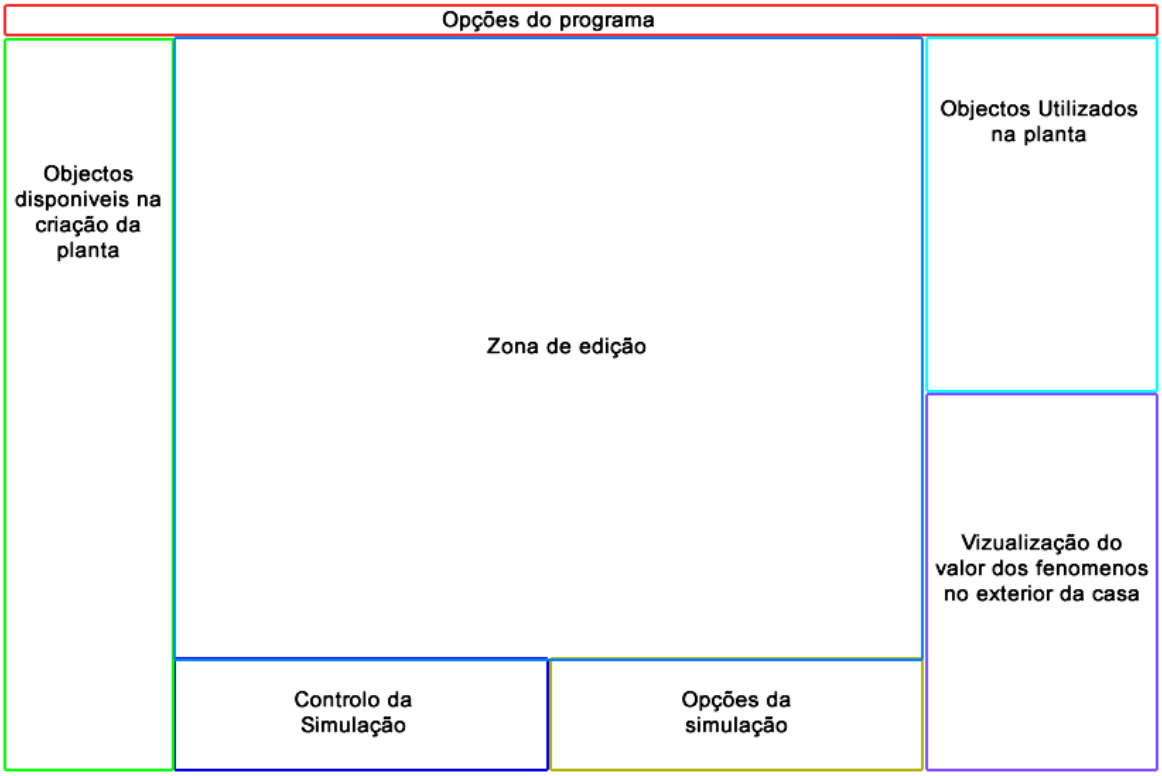

Figura 2 - Aspecto gráfico proposto

O seguinte texto apresenta uma breve descrição da figura 2 para ajudar a perceber melhor o conteúdo do aspecto gráfico que o sistema irá apresentar:

Opções do programa (vermelho):

Zona destinada às opções do programa, onde vão ser apresentadas opções, como abrir ficheiro, gravar planta, criar sensores, criar componentes, entre outras opções.

Objectos disponíveis na criação da planta (verde):

Zona onde devem ser apresentadas as geometrias disponíveis para a construção da planta da casa, assim como os componentes e sensores possíveis de colocar no interior da planta da casa.

Zona de edição (Azul escuro):

Área disponível para desenhar a planta da casa. Apresentar como se fosse uma folha de desenho de um arquitecto.

Objectos utilizados na planta (Azul marinho):

Zona onde vão ser apresentadas as divisões da casa que foram inseridas na zona de edição, assim como os sensores e componentes. É importante referir que é possível ver a relação entre salas e o seu conteúdo.

Controlo da simulação (Roxo):

Área onde estão colocados os botões Play e Stop, assim como o sistema de controlo de velocidade da simulação.

Opções da simulação (Amarelo):

Área onde é possível editar os fenómenos que ocorrem no exterior da casa. Mas também disponibiliza as opções para escolher a data de início da simulação, mostrar legendas e se é pretendido guardar os dados resultantes da simulação.

Visualização dos fenómenos no exterior da casa (violeta):

Zona que permite visualizar em tempo real a oscilação dos fenómenos que ocorrem no exterior da casa.

2.2. Arquitectura do simulador

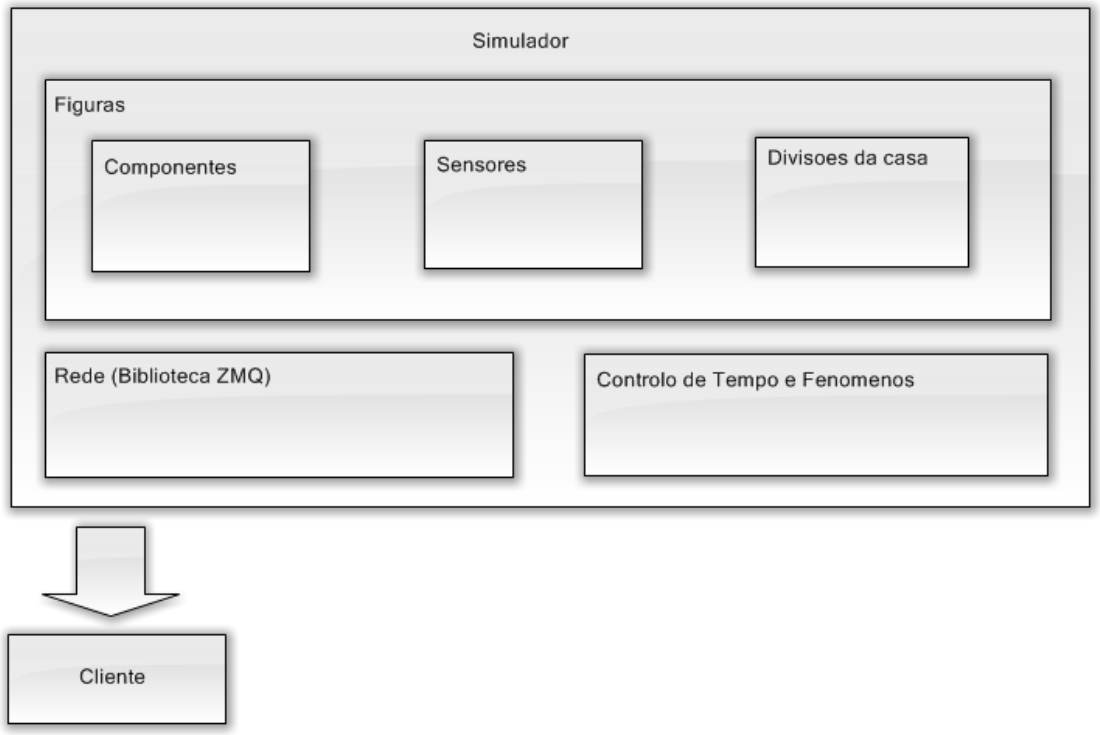

Figura 3 - Arquitectura

O sistema a ser desenvolvido deve apresentar a arquitectura representada na Figura 3. Desta forma, o mesmo será capaz de simular as diferentes divisões da casa de forma independente, assim como o conteúdo do interior de cada uma.

Termos um módulo responsável pelo controlo dos eventos relacionados com o tempo e fenómenos, por forma a aproximar as suas variações de acordo com as diferentes estações do ano.

Por fim é necessário um módulo capaz de enviar e receber pacotes para a rede informática, estando já definida a utilização da biblioteca *ZeroMQ*.

2.3. Conceitos importantes

De seguida, são apresentados os conceitos que foram considerados durante o desenvolvimento da plataforma.

2.3.1. Abstracção

É um dos princípios para se gerir a complexidade no desenvolvimento de software. Permite analisar o problema com um nível de generalização, independentemente dos detalhes de implementação, ou seja: a abstracção é a representação concisa duma ideia ou objecto mais complexa, sobre as características essenciais do objecto.

2.3.2. Modularidade

Princípios para se gerir a complexidade no desenvolvimento de software. Modularidade é a decomposição lógica e física de conceitos em unidades mais elementares, de forma a facilitar a aplicação dos princípios da engenharia de software. A modularidade contribui para uma melhor compreensão dos problemas secundários que ao software é suposto resolver, para uma melhor integração, e para facilitar a introdução das alterações em módulos específicos, o que reduz o impacto de falha nos restantes módulos, e consequentemente em todo o sistema.

2.3.3. Usabilidade

É um dos factores de qualidade mais importantes para uma aplicação. Aplicações difíceis de utilizar levam os utilizadores a rejeitá-las. O objectivo é, portanto, projectar o programa com o qual os utilizadores possam atingir os seus objectivos de forma eficaz e eficiente.

Capítulo 3 – Tecnologias utilizadas no desenvolvimento

As tecnologias de suporte à aplicação são um ponto importante a definir. É a escolha destas tecnologias e as opções tomadas neste âmbito que vão delimitar as capacidades futuras da solução alcançada e a sua usabilidade e interoperabilidade.

3.1. C#

Em Janeiro de 1999, uma equipa de desenvolvimento foi formada por Anders Hejlsberg, que fora escolhido pela Microsoft para desenvolver a linguagem. Dá-se início à criação da linguagem chamada Cool. Em 2000, passa a chamar-se C#, cujo nome viria duma sobreposição de quatro símbolos +.

C# (CSharp) é uma linguagem de programação orientada a objectos criada pela Microsoft, faz parte da sua plataforma.Net 2.0 (2000). A Microsoft baseou-se em linguagens como o C++ e o JAVA.

Esta nova linguagem é considerada a linguagem símbolo do .NET, por ter sido criada praticamente do zero para funcionar na nova plataforma, sem preocupações de compatibilidade com código existente.

A Microsoft submeteu o C# à ECMA para uma padronização formal. Em Dezembro de 2001, a associação publicou a *Standard ECMA-334 C# Language Specification*. Em 2003, tornou-se um padrão ISO (ISO/ IEC 23270). Há algumas implementações em desenvolvimento, destacando-se a Mono, implementação *open source* da Novell, o dotGNU e o Portable.NET, implementações da Free Software Foundation.

Devido à ligação do C# com a Microsoft, a discussão política continua em relação à legalidade da sua normalização, as semelhanças com Java, o futuro como uma linguagem de utilização geral. Alguns peritos em segurança encontram-se desconfiados em relação à eficácia do mecanismo de segurança do CLR e criticam a sua complexidade.

3.2. Visual Studio 2010/12

O *Microsoft Visual Studio* é um conjunto de programas da *Microsoft* para desenvolvimento de software especialmente dedicado ao .NET *Framework.*

Também é um grande produto de desenvolvimento na área web, usando o ASP.NET. As linguagens com maior destaque plataforma são: VB.NET (Visual Basic .Net) e o C# (C Sharp).

Na versão mais recente do Visual Studio (2012) trouxe várias novidades, tais como o suporte ao desenvolvimento de aplicações para o Windows 8 (através da nova interface conhecida como Metro), *sites* baseados em HTML 5, recursos visando uma maior produtividade no dia-a-dia dos programadores, de entre outras funcionalidades.

3.3. ZeroMQ ("The Intelligent Transport Layer")

ZeroMQ é uma biblioteca de mensagens que permite a criação de um sistema de comunicação complexo, mas executado sem muito esforço.

No início do projecto era apresentado como *Middleware,* mas mais tarde evoluiu para TCP, e actualmente é uma camada sobre a pilha de rede.

ZeroMQ é de fácil implementação na programação de uma aplicação. Basicamente, dá acesso á *socket interface,* possibilitando uma construção rápida de um sistema de troca de mensagens.

A vantagem na utilização do ZeroMQ está no seu equilíbrio. É possível ter a flexibilidade e o desempenho do baixo nível e ainda ter a facilidade de implementação de alto nível. No entanto, a gestão de *sockets* é difícil e complicada quando se quer implementar um sistema escalável. Um sistema de alto nível funciona bem se for utilizado para o propósito que foi projectado, mas pode ser difícil de alterar os elementos principais do sistema e sua facilidade de uso, que muitas vezes vem com um custo de desempenho.

3.4. XML

O XML é recomendação W3C para gerar linguagens de marcação de dados. Foi desenvolvida para o transporte de informação, focando-se no que esta representa e separando o conteúdo da formatação do documento.

Assenta numa estrutura baseada em elementos, hierarquizados sob a forma de árvore, e não possui *tags* definidas, sendo estas criadas pelo autor do documento. É uma linguagem auto descritiva e facilmente entendível, tanto por humanos como por computadores.

3.5. Json

O JSON é um formato leve para troca de dados informáticos. É baseado num subconjunto da linguagem de programação *JavaScript*, *Standard ECMA-262 3rd Edition - December* 1999. JSON é um formato de texto que é completamente independente da linguagem, mas usa convenções que são familiares para os programadores de linguagens da família C, incluindo C, C + +, C #, Java, *JavaScript*, e muitos outros.

O JSON é uma alternativa ao XML, mas das vantagens reivindicadas do JSON sobre o XML como um formato para troca de dados, é o facto de ser muito mais fácil escrever um analisador JSON. Porém, o facto de o JSON ser uma alternativa ao XML não invalida que ambos possam ser usados na mesma aplicação.

3.6. Newtonsoft.Json

É uma *framework* JSON para .Net que permite a serialização de um objecto para o formato JSON, sendo também possível fazer o inverso. Caso não exista uma classe que corresponda ao JSON, terá de se fazer o processamento de forma manual.

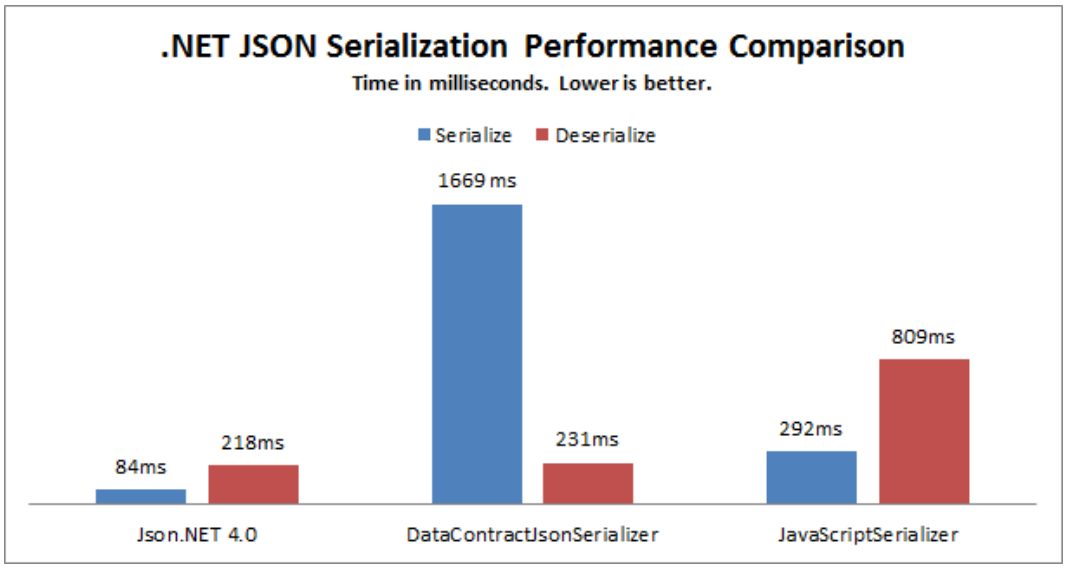

Figura 4- .NET JSON Serialization Teste de comparação

3.7. Framework .net 4

É um conjunto de bibliotecas com várias funções e recursos prontos para o programador usar no desenvolvimento dos seus programas. Quando usado o *.Net* e a utilização dessas bibliotecas para programar, o programa que vai ser criado vai ter como requisito a instalação da *framework* no *pc*. Sendo que inicialmente os programas só seriam executados em Windows, mas actualmente já é possível correr esses mesmos programas em *Linux* e *Mac OS X*, para tal é necessário recorrer ao mono. Devendo ter em conta que esta conversão tem algumas limitações, pois existem bibliotecas que não podem ser convertidas.

Capítulo 4 – Implementação

Este capítulo visa explicar detalhadamente o processo de implementação da aplicação desenvolvida.

O simulador foi desenvolvido em C#, deixando um canal de comunicação entre o simulador e o exterior, onde a troca de dados é realizada através de JSON e XML.

4.1. Esquematização do geral sistema

4.1.1. Casos de utilização

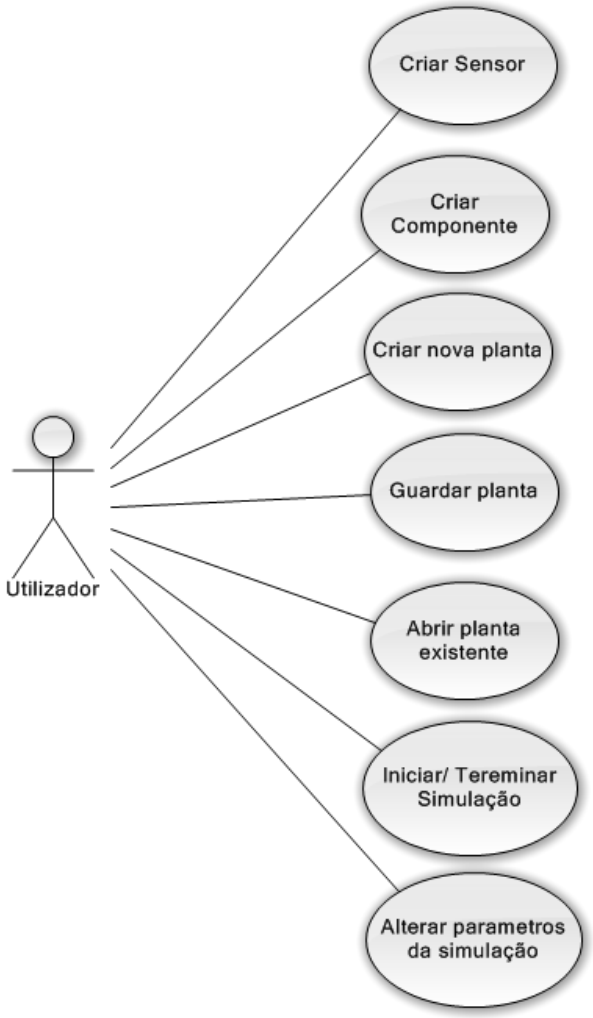

Figura 5 - Casos de utilização gerais

4.1.1.1. Descrição textual dos casos de utilização

Com base no processo de identificação dos casos de uso executado no ponto anterior, apresentar-se-á de seguida a descrição textual de cada um dos casos referidos:

4.1.1.1.1. Criar Sensor

Actor: utilizador.
Objectivo: aquando da falta de um sensor no simulador, ser possível a criação do mesmo, evitando programar o sistema.
Descrição: o caso de utilização inicia-se quanto o utilizador inicia a opção Sensores. O sistema vai apresentar um formulário onde todos os campos correspondentes aos parâmetros a devem ser preenchidos.
Após o preenchimento dos mesmos, o sistema valida os campos, e apresenta uma mensagem de sucesso. Caso algum campo não esteja preenchido ou incorrectamente, será apresentada uma mensagem de erro.

4.1.1.1.2. Criar componente

Actor: utilizador.
Objectivo: aquando da falta de um componente no simulador, ser possível a criação do mesmo, de forma a evitar programar o sistema.
Descrição: o caso de utilização inicia-se quanto o utilizador inicia a opção Componentes. O sistema vai apresentar um formulário onde todos os campos correspondentes aos parâmetros devem ser preenchidos.
Após o preenchimento dos mesmos, o sistema valida os campos, e apresenta uma mensagem de sucesso. Caso algum campo não esteja preenchido ou incorrectamente, será apresentada uma mensagem de erro.

4.1.1.1.3. Criar nova planta

Actor: utilizador.

Objectivo: criar a planta da casa onde se pretende efectuar a simulação.

Descrição: o caso de utilização inicia-se a partir do momento em que este tem “Folha de desenho” limpa e escolhe uma figura geométrica disponível e começa por desenhá-la. O sistema permite ao utilizador desenhar três tipos de figuras geométricas para desenhar as divisões da casa, e ainda disponibiliza os componentes e sensores. Estes só podem ser colocados dentro das divisões da casa.

4.1.1.1.4. Guardar planta

Actor: utilizador.

Objectivo: guardar a planta da casa para ser utilizada mais tarde numa nova simulação.

Descrição: após a criação de uma planta, o utilizador pode guardá-la, tendo apenas para tal de pressionar a opção “gravar”. De seguida, o sistema irá perguntar onde deseja guardar o ficheiro correspondente à planta. Depois de seleccionada a pasta e a confirmação, o sistema vai informar que o ficheiro foi guardado com sucesso.

4.1.1.1.5. Abrir planta existente

Actor: utilizador.

Objectivo: abrir a planta da casa para ser utilizada mais tarde numa nova simulação.

Descrição: após pressionar a opção “Abrir”, o sistema pergunta a localização e qual o ficheiro que pretende abrir. Logo que seleccionado, a planta deverá aparecer na “Folha de desenho”.

4.1.1.1.6. Iniciar / Terminar Simulação

Actor: utilizador.

Objectivo: iniciar simulação para obtenção de dados a serem estudados, ou finalizar a simulação quando necessário.

Descrição: após a existência de uma planta, com sensores e componentes, o utilizador poderá dar início à simulação, podendo terminá-la quando assim o entender.

4.1.1.1.7. Alterar parâmetros da simulação

Actor: utilizador.

Objectivo: guardar a planta da casa para ser utilizada mais tarde numa nova simulação.

Descrição: durante a criação da planta, e quando criada uma divisão da casa, o sistema apresenta um conjunto de parâmetros correspondentes à divisória, os quais podem ser alterados pelo utilizador. No fim do procedimento, o utilizador apenas terá de pressionar a opção “Gravar fenómeno”. Caso não sejam detectados erros, o sistema fechará a janela de edição; detectados erros, o sistema apresentá-los-á.

No decorrer da simulação, é possível alterar o estado dos componentes, bem como definir um novo valor para os mesmos (esta alteração pode ser feita pela rede informática).

4.1.2. Funcionamento do simulador

Este ponto tem como objectivo mostrar e descrever, passo-a-passo, como se iniciam as várias tarefas que são executadas durante a simulação.

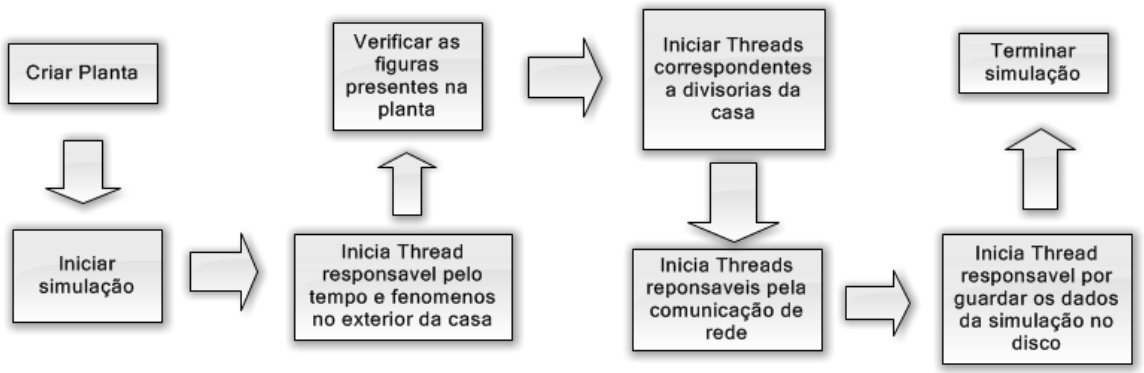

Figura 6 - Representação de início de Threads na simulação

Criar Planta:

Para que seja possível criar uma simulação, é obrigatório que na "Folha de desenho" esteja representada a planta de uma casa.

Pressionar a opção Iniciar simulação:

Só após o utilizador dar a ordem de início de simulação, é que o simulador inicia os serviços necessários para a criação de dados.

Inicia *Thread* responsável pelo tempo e fenómenos no exterior da casa:

Esta é a *thread* referência para toda a simulação. Isto porque é esta thread que controla a data e hora, velocidade da simulação, tamanho dos dias, nascer e pôr-do-sol, assim como vão variar os fenómenos ao longo da simulação. Por exemplo, no inverno os dias são mais pequenos, temperaturas baixas, e níveis de humidade mais elevados. Por outro lado, no verão os dias são maiores, temperaturas mais elevadas e menos humidade.

É importante lembrar que esta *thread* tem como objectivo simular as variações exteriores da casa.

Verificar as figuras presentes na planta:

Acção responsável pela verificação dos valores iniciais de todas as figuras representadas na planta.

Iniciar *Threads* correspondentes a divisórias da casa:

Para evitar problemas de concorrência durante a simulação, decidiu-se que cada divisória da casa vai ter uma *thread* associada. Esta vai ter a responsabilidade de verificar os valores dos diferentes fenómenos dentro da divisão, depois verifica quais os objectos que estão dentro da divisão que são componentes. Posteriormente verifica de forma sequencial qual o impacto que estes vão ter nos diferentes fenómenos, e actualiza-os se necessário.

Para finalizar, a *thread* procura os objectos que são do tipo sensores e verifica qual (ais) o(s) fenómeno(s) que cada sensor consegue ler e actualiza o valor da leitura.

Inicia *Threads* responsáveis pela comunicação de rede:

Threads responsáveis pela comunicação do simulador com outros sistemas.

Nesta etapa são iniciadas duas *threads*, uma de envio e outra de recepção.

A *thread* de envio recebe a leitura dos diferentes sensores, e caso não exista nenhum cliente ligado, os dados são guardados numa pilha, e quando um cliente se ligar ao sistema vai receber todos os dados que estão nessa pilha.

A *thread* de recepção tem como finalidade receber dados para a alteração de estado dos diferentes componentes existentes na planta.

No sistema de comunicação optou-se por uma *thread* de envio e recepção, as quais actuam em portas diferentes, de modo a evitar congestionamentos.

Inicia Thread responsável por guardar os dados da simulação no disco:

Durante a simulação pode ser necessário guardar todos os dados gerados pela mesma. Esta *thread* tem essa responsabilidade, podendo a sua activação ser opcional.

4.2. Descrição do modelo aplicado

A informação gerida e visualizada no âmbito do sistema desenvolvido consiste numa ontologia, cujos principais conceitos e relações são ilustrados nas figuras abaixo através de um diagrama de classes UML, onde estão representadas as classes principais do sistema.

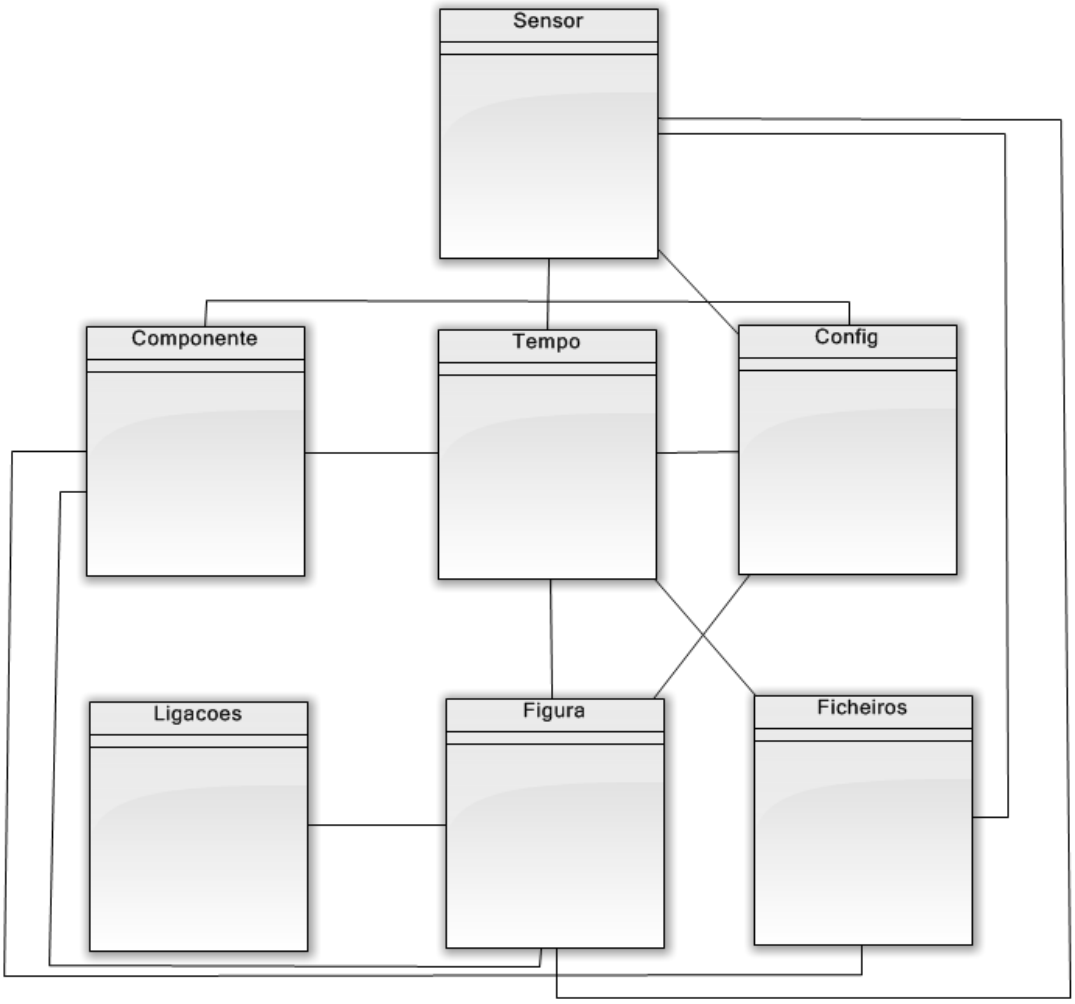

Figura 7 - Diagrama de classes de todo o sistema

4.2.1. Tempo e Fenómenos

O objectivo da classe Tempo é processar a data e hora no decorre da simulação. Com base nestas duas variáveis, será possível gerar o tempo solar diário, nascer do sol e pôr-do-sol. Para tal, foram utilizadas as seguintes fórmulas:

$$T_d = \frac{2}{15}\arccos(-\tan\phi.\tan\delta)$$

Td é o tempo de duração do dia

ϕ é a latitude da cidade (para cidades do hemisfério sul, o sinal é negativo)

δ é a declinação da Terra, que é calculada pela fórmula:

$$\delta = 23,45 sen\left(\frac{360}{365}(284+n)\right)$$

n é o dia sequencial do ano (1º de Janeiro é 1, 1º de Fevereiro é 32, ... 31 de Dezembro é 365 ou 366 se bissexto)

nascer do sol: 12-(td/2)

pôr-do-sol: 12+(td/2)

Esta classe também gera os valores referência da simulação, os quais são chamados de "valor da rua". Isto porque é a partir da classe em questão que vai informar ao sistema que no Inverno a temperatura é mais baixa e a humidade superior e que no Verão é o inverso. Deste modo, é possível fazer uma aproximação aos valores obtidos na realidade. A figura 8 é uma representação das variações de fenómenos que ocorrem no simulador.

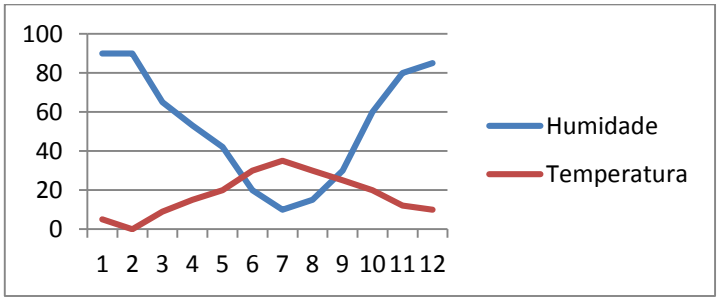

Figura 8 - Variação de Fenómenos ao longo do ano

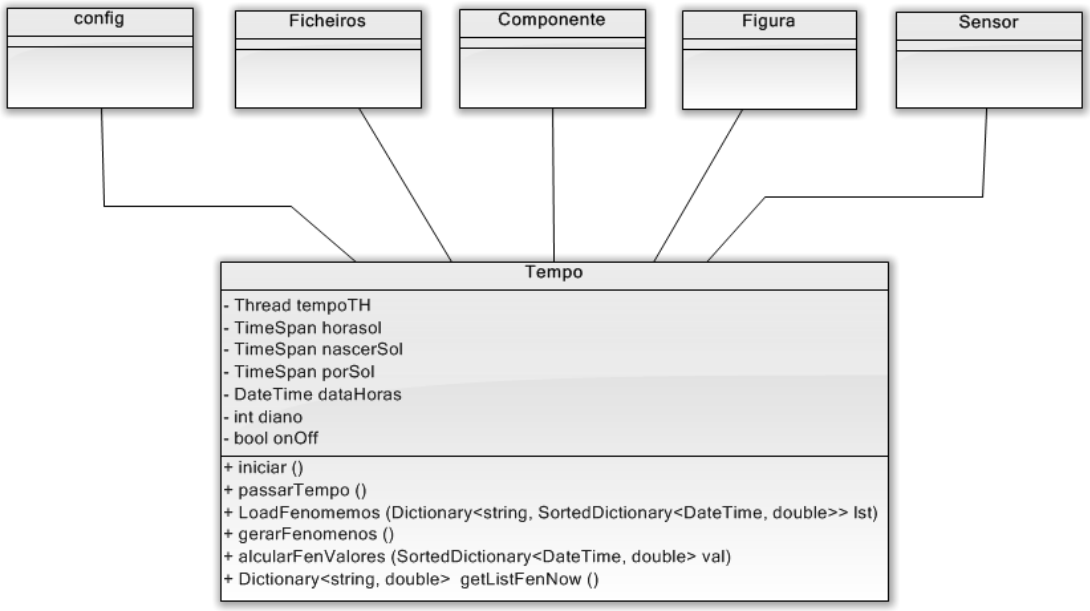

Figura 9 - Diagrama de classes correspondente à classe Tempo

Variáveis	
Thread tempoTH	Responsável pelo controlo do tempo.
TimeSpan horasol	Tamanho do dia solar.
TimeSpan nascerSol	Hora do nascer do sol
TimeSpan porSol	Hora do pôr-do-sol
DateTime dataHoras	Responsável por guardar a data e hora da simulação.
int diano	Informa o número do dia em relação ao ano.
bool onOff	Saber se a simulação esta a decorrer ou não.
Operações	
iniciar()	Iniciar variáveis para início de simulação
passarTempo()	Operação que contem o ciclo onde é feita a incrementação na alteração das horas.
LoadFenomemos(Dictionary<string, SortedDictionary<DateTime, double>> lst)	
	Carregar os fenómenos disponíveis no simulador.
gerarFenomenos()	Actualiza o valor dos fenómenos referencia da simulação (Fenómenos na rua)
calcularFenValores(SortedDictionary<DateTime, double> val)	
	Actualiza o valor do fenómeno em função do dia e hora na pilha de valores.
Dictionary<string, double> getListFenNow()	
	Consultar todos os valores correrspondentes aos vários fenómenos na pilha.

Tabela 1- Descrição do da classe tempo

4.2.2. Ligações (Rede)

Classe responsável pelas comunicações do programa. Foi concebida a pensar nas necessidades actuais e futuras, daí neste momento estas comunicarem por ZeroMQ, mas no futuro poder-se-á aplicar um novo protocolo, por exemplo o TCP/IP ou até mesmo comunicar a utilização de *web-services.* Abaixo, na figura 10, será apresentado o diagrama de classes referente à comunicação do sistema.

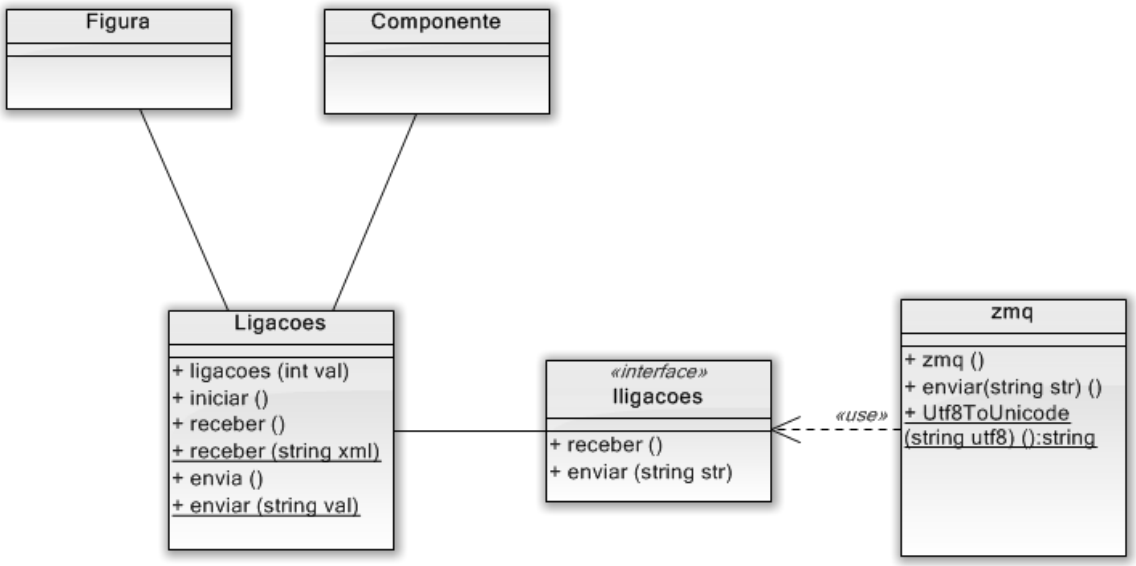

Figura 10 - Diagrama de classes correspondente as ligações

Variáveis	
Operações	
zmq()	Inicializador da Classe. Nesta operação é definida a porta de comunicação de envio (8051)
enviar(string str)	Operação que converte a string para bytes e envia os dados para um receptor.
Utf8ToUnicode(string utf8)	Operação estática que converte o charset da string de utf8 para Unicode.

Tabela 2 - Descrição da classe zmq

Variáveis	
Operações	
ligacoes(int val)	Inicializador da classe, o parâmetro de entrada refere-se ao tipo de ligação. O número 1 corresponde ao ZMQ, num futuro haverá a possibilidade de serem introduzidos novos protocolos de comunicação.
iniciar()	Operação responsável pelo início da *thread* responsável pelo envio e pela recepção de dados.
receber(string xml)	Operação estática que recebe os dados, valida-os e reencaminha-os para o componente de destino.
receber()	Operação que se liga com o Receber da interface.
Envia()	Operação que se liga com o Envia da interface. E recebe o parâmetro que se destina ao envio.
Enviar(string val)	Recebe o parâmetro que se pretende enviar, e coloca-o numa lista, onde vai permanecer até ser enviado.

Tabela 3 - Descrição da classe Ligações.

4.3.2.2. Zmq

Como apresentado na Tabela 2 – descrição da classe zmq, tal como o nome indica, esta classe refere-se à utilização da biblioteca ZeroMQ. A aproximação implementada neste caso foi *Divide and Conquer.*

Esta aproximação permite um processamento de tarefas em paralelo. Assim sendo, o "consumidor/ cliente" pode ultrapassar o "servidor/ produtor" na ordem de execução de tarefas. Isto irá fazer com que seja utilizada memória, e o processo de limpeza pode ser

difícil para o sistema, e em caso de erro pode dar-se a perda de informação. Isto pode ser evitado com a utilização do ZMQ_REP e *sockets* ZMQ_REQ, que neste caso aumenta a segurança, mas, em contrapartida, ocorrerá perda de velocidade, que é um dos pontos fortes do ZeroMQ.

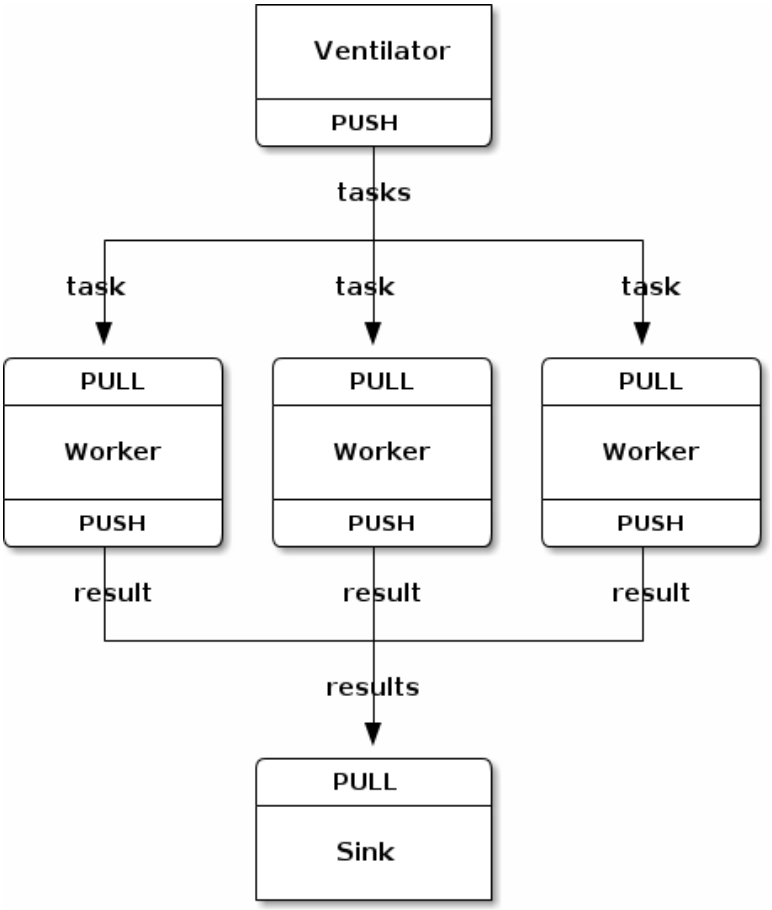

Figura 11 - Esquema ZeroMQ divide and conquer

4.2.2.1. Estrutura de dados na comunicação com outras aplicações

Em seguida, vão ser apresentadas as estruturas dos dados a serem enviados e recebidos pelo simulador. Estruturas estas que foram desenhadas em conjunto com outros estudantes da Universidade de Aveiro com o objectivo de ligar outras aplicações ao simulador apresentado no presente relatório.

Formato de envio

```
<?xml version=\"1.0\" encoding=\"utf-16\"?>
<contextML>
    <action>get</action>
    <sensorid>W0JX:YL58:FCW8:26R7</sensorid>
    <Temperatura>13,92</Temperatura>
    <timestamp>1354705493</timestamp>
</contextML>
```

Tabela 4 - Xml de envio

```
{
  "type": "9",
  "contextML": "<?xml version=\"1.0\" encoding=\"utf-16\"?>
<contextML>
<action>get</action>
<sensorid>W0JX:YL58:FCW8:26R7</sensorid>
<Temperatura>13,92</Temperatura>
<timestamp>1354705493</timestamp>
</contextML>"
}
```

Tabela 5 - JSON de envio

Na Tabela 4, é apresentada a estrutura de dados resultantes da leitura de um sensor, onde é possível visualizar o tipo de dados, que neste caso é "*get"* o que quer dizer que é uma leitura, podendo ainda ficar a conhecer o *Mac Adress* do mesmo a partir do elemento "sensorid". Posteriormente, é apresentado no nome do fenómeno e o seu valor, e, para finalizar, é apresentada a data e hora da leitura no elemento "*timestamp"*.

A Tabela 5 apresenta a estrutura de um ficheiro JSON de envio, onde estão os campos "*type"*, que é sempre 9. Isto serve para informar o sistema que se ligou ao simulador que este ficheiro JSON é proveniente do simulador. Sendo que o campo "ContextML" é o que apresenta o ficheiro xml apresentado na Tabela 6.

Formato de recepção

```
<?xml version=\"1.0\" encoding=\"utf-16\"?>
 <contextML>
     <action>set</action>
    <component>
          <id>2</id>
          <on>10</on>
          <valor>10</valor>
    </component>
</contextML>
```

Tabela 6 - Xml de recepção

```
{
  "type": "9",
  "contextML": "<?xml version=\"1.0\" encoding=\"utf-16\"?><contextML>
<action>set</action><component><id>2</id><on>10</on><valor>10</valor></co
mponent></contextML>"
}
```

Tabela 7 - Json de recepção

A Tabela 6 apresenta a estrutura do ficheiro XML a ser enviado ao simulador para que este active um determinado componente.

Para tal, tem que definir o elemento “action” como set, o elemento “id”, que deve ser o id de identificação do componente que se pretende activar, o elemento “on” é o tempo em minutos que o componente se vai manter em funcionamento e o elemento “valor” é o valor que se vai atribuir ao componente. Por exemplo, no caso do ar condicionado, ao se atribuir o valor 20, ele iria ser programado para que trabalhasse para 20 graus.

A Tabela 7 é semelhante à Tabela 3, diferenciando-se apenas no facto de que varia o Xml que vai ser inserido no campo contextML.

Componente

Classe responsável pelas acções dos componentes durante a simulação, umas das classes mais importantes do sistema.

Durante o seu desenvolvimento foi considerado, por um lado, o dinamismo da classe, de forma a ser possível criar todos os componentes necessários, e, por outro, a reprodução do seu comportamento no mundo real.

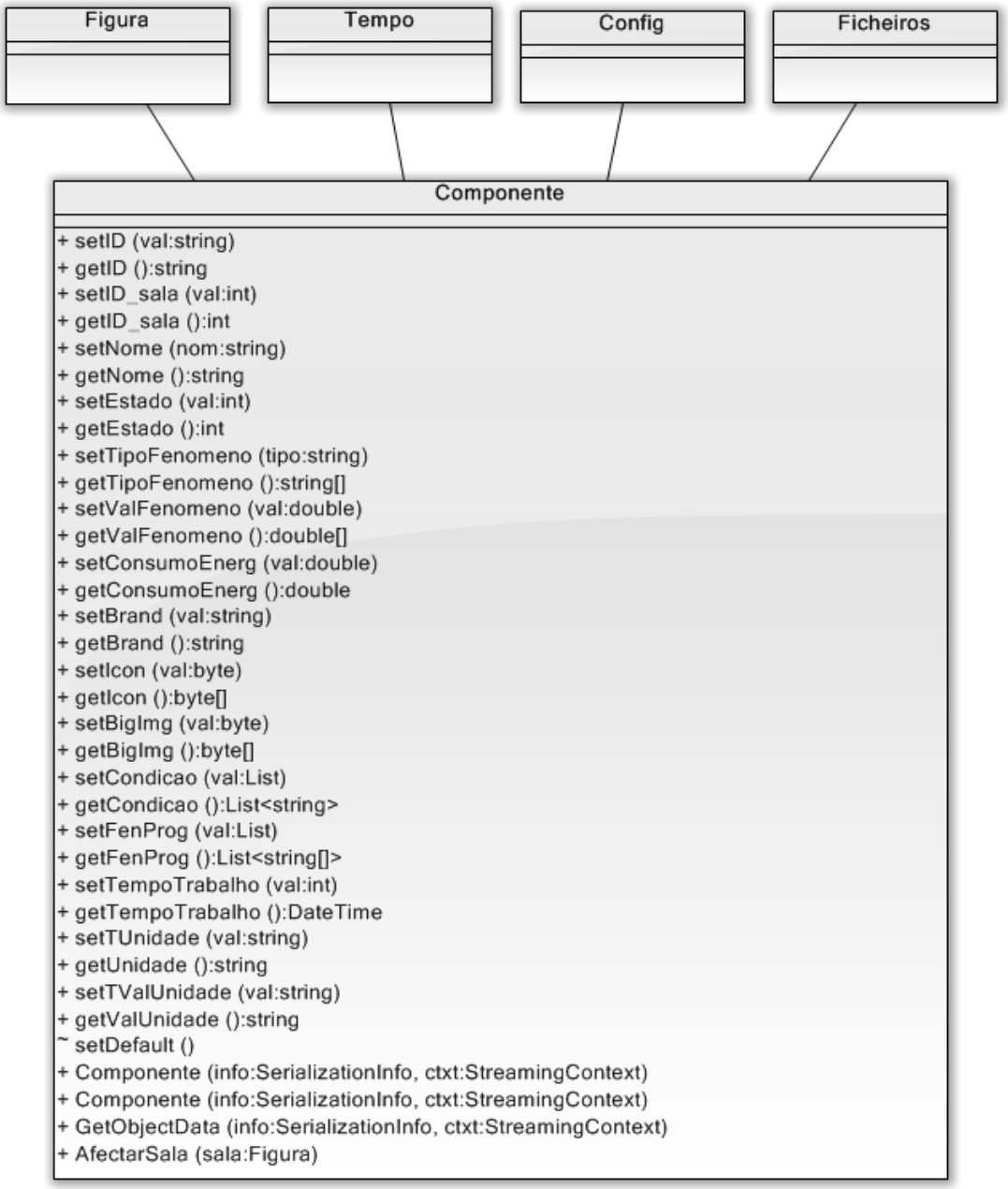

Figura 12 - Classe componente

Variáveis	
Operações	
setID (val:string)	Definir o Id do componente.
getID ():string	Ver id do componente.
setID_sala (val:int)	Definir id da sala em que esta instalado.
getID_sala ():int	Ver id da sala em que está instalado.
setNome (nom:string)	Definir nome de componente
getNome ():string	Ver o nome do componente.
setEstado (val:int)	Definir estado do componente, 0 = desligado, 1= Ligado
getEstado ():int	Ver se o componente está ligado ou não.
setTipoFenomeno (tipo:string[])	Definir array de fenómenos que o componente vai afectar
getTipoFenomeno ():string[]	Ver lista de nomes dos fenómenos que interagem com o componente.
setValFenomeno (val:double)	Definir array de valores afectados pelo componente. Os valores estão relacionados pela posição do nome do tipo de fenómeno no índice.
getValFenomeno ():double[]	Ver valores dos fenómenos que o componente interfere.
setConsumoEnerg (val:double)	Definir consumo energético do componente
getConsumoEnerg ():double	Ver consumo energético.
setBrand (val:string)	Definir marca do componente
getBrand ():string	Ver marca.
setIcon (val:byte)	Definir ícone do componente.
getIcon ():byte[]	Ver ícone.
setBigImg (val:byte)	Definir imagem do componente na planta.
getBigImg ():byte[]	Ver imagem.
setCondicao (val:List)	Definir condições do funcionamento do componente.
getCondicao ():List<string>	Ver lista de condições.
setFenProg (val:List)	Definir Lista de fenómenos programados. Isto é o comportamento do fenómeno quando adicionado o componente. (Mais informações no anexo A)
getFenProg ():List<string[]>	Ver lista de fenómenos programados.
setTempoTrabalho (val:int)	Definir tempo de trabalho do componente. Em

	minutos.
getTempoTrabalho ():DateTime	Ver o tempo de trabalho do componente. (ver até q horas o componente vai trabalhar.)
setTUnidade (val:string)	Definir unidade utilizada no componente. (exemplo: numa tv a unidade é o canal.)
getUnidade ():string	Ver unidade utilizada pelo componente.
setTValUnidade (val:string)	Definir valor da unidade (segundo o exemplo anterior, o valor seria a posição do canal na grelha, por exemplo 1)
getValUnidade ():string	Ver valor da unidade do componente.
Componente (info:SerializationInfo, ctxt:StreamingContext)	serialização do componente
GetObjectData (info:SerializationInfo, ctxt:StreamingContext)	Operação de des-serialização do componente.

Tabela 8- Descrição da classe Componente

4.3.3. Sensor

Classe responsável pelos dados dos sensores a serem apresentados na simulação. A presente classe não é considerada independente no ponto de vista da gestão dos próprios dados. Isto porque, para obter os dados, estes terão de ser enviados pela classe Figura. Esta foi a abordagem escolhida no início do projecto, de forma a evitar problemas de concorrência.

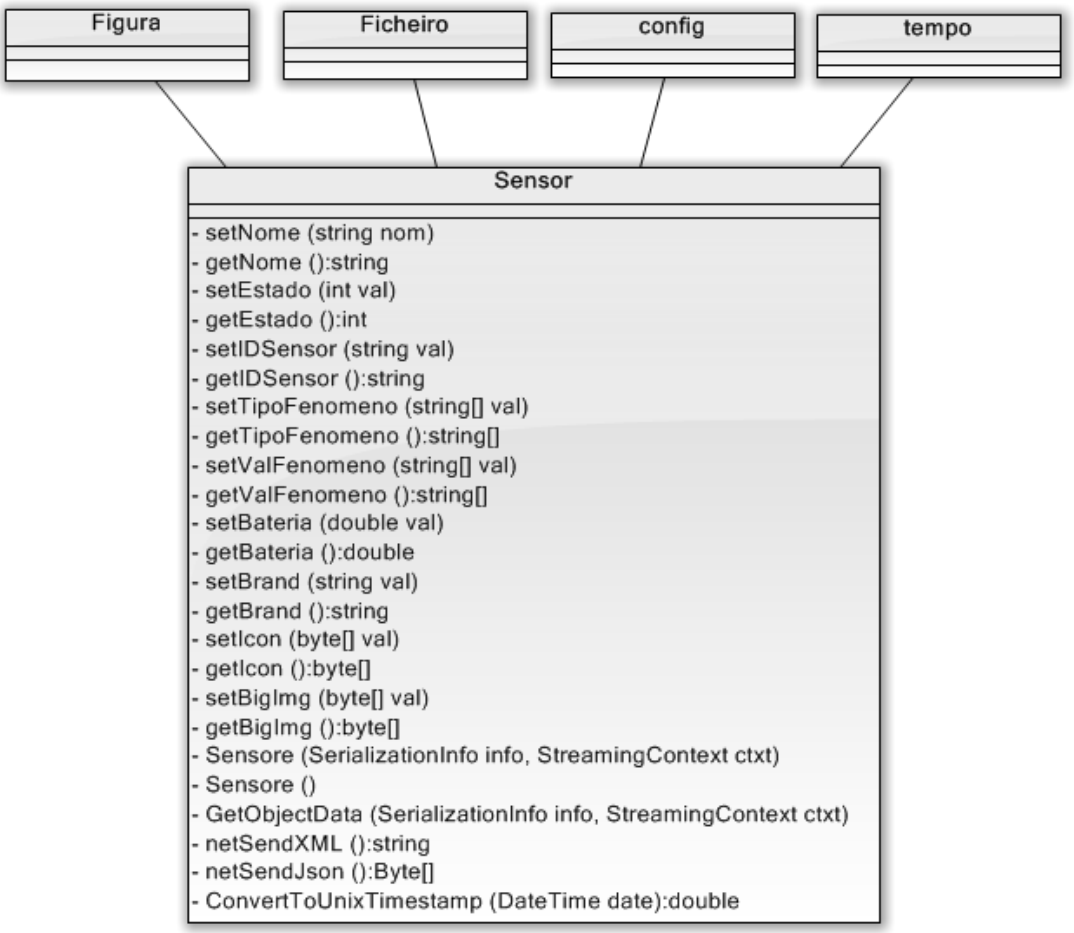

Figura 13 - Classe Sensor

Variáveis	
Operações	
setNome(string nom)	Atribuir nome ao sensor
getNome() : string	Ver nome do sensor
setEstado(int val)	Definir estado do sensor, 0=desligado e 1= ligado.
getEstado() : int	Ver estado do sensor
setIDSensor(string val)	Definir id do sensor
getIDSensor() : string	Ver id do sensor
setTipoFenomeno(string[] val)	Definir lista de nomes dos fenómenos que o sensor vai ler.
getTipoFenomeno() : string[]	Ver lista de nomes de fenómenos lidos pelo sensor
setValFenomeno(string[] val)	Atribuir os valores dos fenómenos lidos pelos sensores.
getValFenomeno() : string[]	Ver leituras efectuadas pelo sensor.
setBrand(string val)	Definir marca do sensor
getBrand() : string	Ver marca do sensor
setIcon(byte[] val)	Definir icon do sensor
getIcon() : byte[]	Ver icon
setBigImg(byte[] val)	Definir imagem que representa o sensor na planta
getBigImg() : byte[]	Ver imagem
Sensore(SerializationInfo info, StreamingContext ctxt)	Serialização do sensor
Sensore()	Inicializador do objecto
GetObjectData(SerializationInfo info, StreamingContext ctxt)	Des- Serialização do objecto sensor
-netSendXML() : string	Reponde string XML com os valores das leituras do sensor
netSendJson() : Byte[]	Converte o ficheiro Json com informações do leitor em bytes
ConvertToUnixTimestamp(DateTime date) : double	Operação que converte a data hora do formato UNIX para Timestamp. A resposta é dada no formato double.

Tabela 9 - Descrição da tabela Sensor

4.3.4. Figuras

É uma das classes fundamentais do simulador, pois é nela que se prende uma grande parte da simulação. A classe Figuras é a responsável pelo que é apresentado no ecrã, é ela que actualiza os valores dos sensores numa divisória da casa, assim como os valores dos fenómenos que estão a ocorrer em cada divisão da casa.

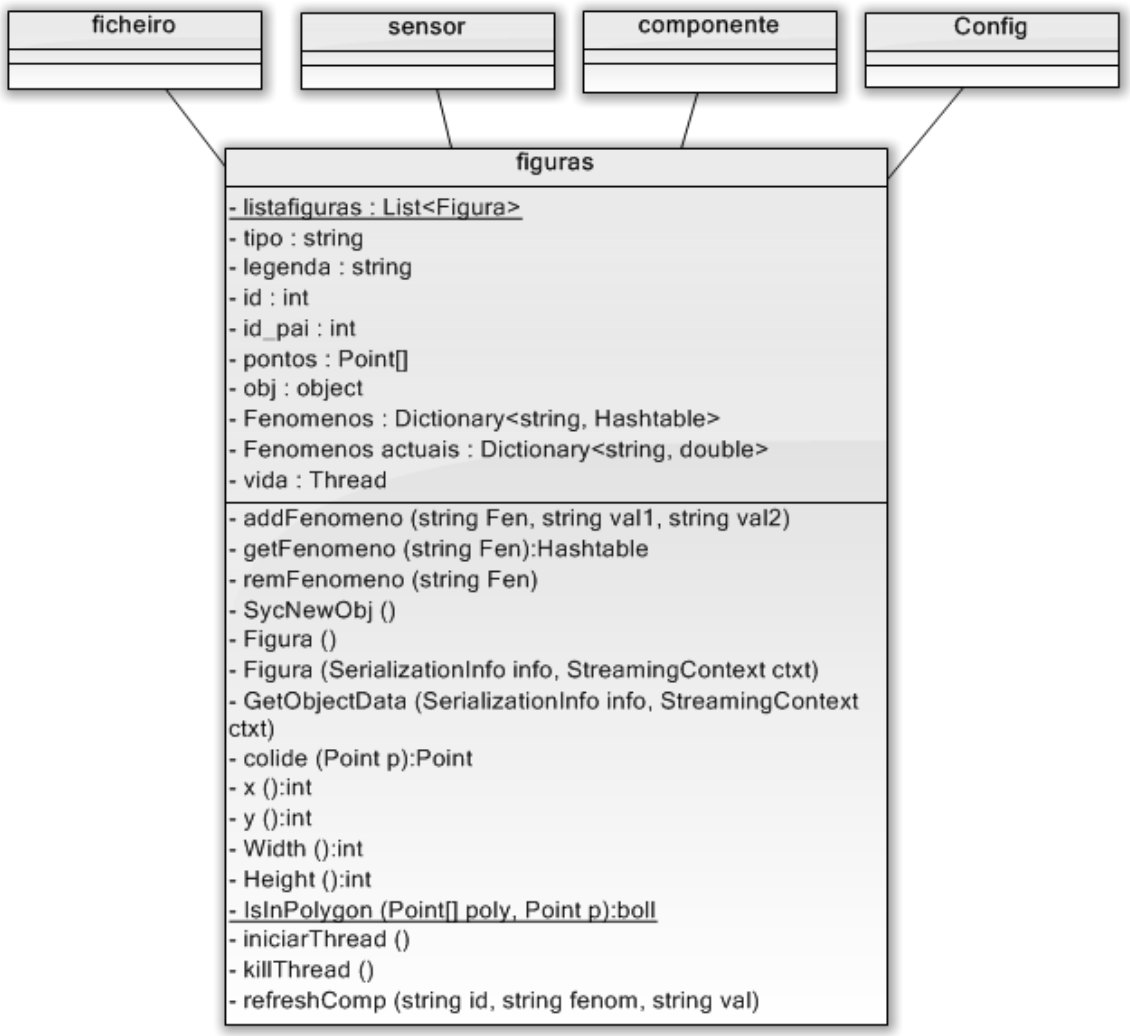

Figura 14 - Classe Figuras

Variáveis	
listafiguras : List<Figura>	Lista de figuras utilizadas na simulação
tipo : string	Tipo de figura (componente ou sensor)
legenda : string	Legenda da divisão da casa ou do componente ou do sensor.
id : int	Id da figura
id_pai : int	Id do poligno a que a imagem está associada.
pontos : Point[]	Pontos do objecto
obj : object	Objecto para guardar objectos do tipo sensor ou componentes. (No futuro se houver a necessidade de criar o homem, seria aqui que o seu objecto iria ficar associado. Optou-se por este tipo de abordagem para o programa ficar mais dinâmico, mas em contra partida pode ser mais propicio a erros).
Fenomenos : Dictionary<string, Hashtable>	Caso seja uma figura geométrica esta variável define os nomes e valores dos fenómenos existentes na divisória da casa. (valore gerado em função do valores obtido no “exterior da casa ”)
Fenomenos actuais : Dictionary<string, double>	Lista do nome e valores actuais dos fenómenos que afectam a divisão da casa.
vida : Thread	Thread responsável pela “vida do objecto”.
Operações	
addFenomeno(string Fen, string val1, string val2)	Operação que adiciona os fenómenos às divisões da casa. Val1 corresponde à hora (0-23) e o val2 corresponde ao valor do fenómeno na hora referida no val1.
getFenomeno(string Fen) : Hashtable	Operação que responde à lista de valores do fenómeno ao longo do dia da simulação.
remFenomeno(string Fen)	Remove fenómeno da divisão da casa.
SycNewObj()	Verifica componente ou sensor e insere o objecto analisado na variável obj. Caso do sensor cria o numero de Mac adress.
Figura()	Inicializador da classe
Figura(SerializationInfo info, StreamingContext ctxt)	Serialização da classe
GetObjectData(SerializationInfo info, StreamingContext ctxt)	Des-serialização da classe
colide(Point p) : Point	Verifica colisões de pontos em figuras.
x() : int	Ver Coordenada X
y() : int	Ver Coordenada Y
Width() : int	Ver largura

Height() : int	Ver altura
IsInPolygon(Point[] poly, Point p) : boll	Verifica se o ponto inserido esta no interior do polígono inserido.
iniciarThread()	Inicia Thread q corresponde aos eventos ocorridos nas divisão da casa em questão. Caso a figura seja uma geometria (geometria para o sistema é considerada divisão da casa.)
killThread()	“Mata” a thread
refreshComp(string id, string fenom, string val)	Actualiza valor de um fenómeno no componente. Id= id do componente, fenom = Nome do fenómeno, val = valor do componente.

Tabela 10- Descrição da classe Figuras

4.3.5. Ficheiros

Classe responsável por gravar e abrir ficheiros. Estes podem ser referentes à planta, sensores ou componentes.

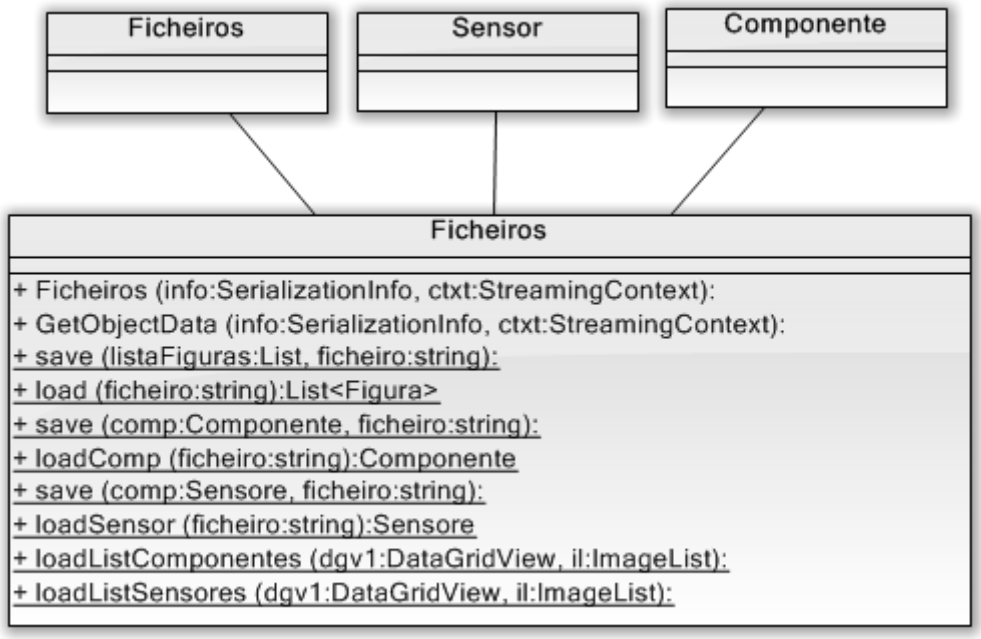

Figura 15 - Classe Ficheiros

Variáveis	
Operações	
Ficheiros(info : SerializationInfo, ctxt : StreamingContext) :	Operação que faz parte do contracto do ISerializable
GetObjectData(info : SerializationInfo, ctxt : StreamingContext) :	Operação que faz parte do contracto do ISerializable
save(listaFiguras : List, ficheiro : string) :	Operação que recebe a lista de figuras da planta, e apos a sua serialização grava as para o ficheiro indicado.
load(ficheiro : string) : List<Figura>	Lê ficheiro indicado, des-serializa a informação e responde uma lista com as figuras obtidas.
save(comp : Componente, ficheiro : string) :	Operação específica para gravar o componente que vai estar disponível na criação da planta.
loadComp(ficheiro : string) : Componente	Operação específica para ler o componente que vai estar disponível na criação da planta.
save(comp : Sensore, ficheiro : string) :	Operação específica para gravar o sensor que vai estar disponível na criação da planta.
loadSensor(ficheiro : string) : Sensore	Operação específica para ler o sensor que vai estar disponível na criação da planta.
loadListComponentes(dgv1 : DataGridView, il : ImageList)	Operação que lê os ficheiros correspondestes aos componentes, e organiza os mesmos numa *datadridview* de forma a ficarem apresentáveis.
loadListSensores(dgv1 : DataGridView, il : ImageList) :	Operação que lê os ficheiros correspondestes aos sensores, e organiza os mesmos numa *datadridview* de forma a ficarem apresentáveis.

Tabela 11 - Descrição da classe Ficheiros

4.3.6. Configuração

A classe Configuração, que na verdade no sistema é apresentada como config, é a única responsável pelos fenómenos. No início do projecto tinha sido projectado esta classe também ser responsável pelo tempo, mas como a gestão da classe tempo tornou-se mais complexa, optou-se por criar uma classe exclusiva, ficando assim a classe config a cargo dos fenómenos.

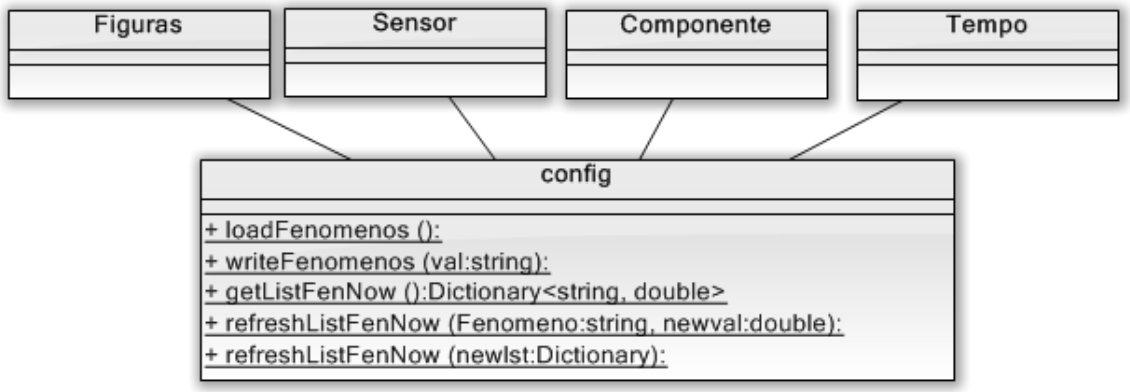

Figura 16 - Classe Config

Variáveis	
Operações	
loadFenomenos() :	Operação que carrega os fenómenos e suas configurações para o sistema.
writeFenomenos(val : string) :	Operação que tem como finalidade a actualização as configurações dos fenómenos.
getListFenNow() : Dictionary<string, double>	Operação que permite visualizar os nomes e valores num dado momento da simulação. Estes dados são os dados referência para os valores gerados na simulação
refreshListFenNow(Fenomeno : string, newval : double) :	Operação que visa a actualização dos dados "referencia" de um dado fenómeno.
refreshListFenNow(newlst : Dictionary) :	Operação que actualiza os valores dos fenómenos referencia da simulação de acordo com a hora da simulação.

Tabela 12 - Descrição da Classe Config.

Os dados de configuração dos fenómenos devem estar guardados num ficheiro com o formato XML, como representado na tabela que se segue.

```
<?xml version="1.0"?>
<Fenomenos>
	<Fenomeno  xtit="C" ytit="temperatura" xvals="0;23" xint="1" v1data="2012-02-05" v1val="1" v2data="2012-07-18" v2val="38">Temperatura</Fenomeno>

	<Fenomeno  xtit="%" ytit="Humidade" xvals="0;23" xint="1" v1data="2012-01-16" v1val="100" v2data="2012-07-18" v2val="2">Humidade</Fenomeno>

	<Fenomeno  xtit="%" ytit="Humidade" xvals="0;23" xint="1">Luminosidade</Fenomeno>
</Fenomenos>
```

Tabela 13- Xml de configuração de fenómenos.

Descrição dos atributos do elemento Fenómeno:

Xtit: medida para medição do fenómeno.

Ytit: nome do Fenómeno.

Xvals: valor das horas de início e fim. Período em que o fenómeno é activo. Os dados inseridos devem apresentar o seguinte formato. “numero inteiro da hora de inicio: número inteiro da hora final”. Os valores devem oscilar entre 0 e 23.

V1data: data onde o valor atinge o valor mínimo.

V1Val: valor mínimo do Fenómeno.

V2data: data onde o valor atinge o valor máximo.

V2val: valor mínimo do Fenómeno.

De referir que os atributos têm os nomes apresentados, porque estava previsto o sistema apresentar gráficos. E estes eram os termos mais indicados para uma melhor compreensão do significado dos dados.

Capítulo 5 – Conclusão

Este projecto teve como objectivo desenvolver um simulador que demonstre a utilização dos vários componentes existentes numa casa, e como estes vão afectar os diferentes fenómenos que ocorrem nas diferentes divisões.
O simulador foi concebido de forma a garantir uma organização funcional coesa, tornando-o dinâmico e expansível, sendo assim possível maximizar a sua utilização.

5.1. Conclusões

Antes de mais queria mencionar que foi muito importante e gratificante para mim a realização deste projecto. Consequentemente, tive oportunidade de trabalhar com pessoas fantásticas que me proporcionaram uma excelente orientação e me deram sempre um conjunto de requisitos concretos e bem definidos para a realização do projecto.

Após a conclusão deste projecto, posso afirmar que o simulador mostrou ser uma ferramenta extremamente eficaz para a previsão de resultados onde ainda não é possível obter dados reais. Porém, ficou evidente durante o desenvolvimento que, por não ser algo trivial, os seus resultados sejam satisfatórios, e em alguns casos ainda não é possível obter os dados pretendidos.

Ao longo do seu desenvolvimento, a aprendizagem foi por vezes lenta, muito devido à falta de conhecimento de algumas tecnologias utilizadas. Porém, é possível afirmar que se conseguiu ultrapassar estes obstáculos.

É necessário referir que um dos obstáculos que sobressaem prende-se com a utilização do mono. No decorrer do desenvolvimento do projecto recorreu-se ao NuGet para obter bibliotecas. E as obtidas não foram compiladas de forma a serem compatíveis com o mono. Mas, na verificação da informação disponível no site do ZeroMQ e do NewTonSoft.JSON, é referido que as mesmas são compatíveis com o mono depreendendo-se que, aquando a utilização das mesmas nos diferentes compiladores, tem que se alterar as referências.

Um aspecto não referido neste relatório é que, durante o processo de desenvolvimento do simulador, houve uma tentativa de utilizar outras bibliotecas, de entre as quais destaco o Zedgraph, uma biblioteca gratuita para desenvolvimento de gráficos, cuja única desvantagem é a descontinuidade do seu desenvolvimento há vários anos.

A segunda biblioteca de destaque mas não implementada é a taoframework-2.1.0, a qual tem como objectivo o desenvolvimento 3D com recurso ao openGL. Não foi dado seguimento à sua implementação porque não estava a ser possível obter os resultados esperados dentro do tempo esperado.

Ainda que a aplicação criada necessite de mais funcionalidades e melhorias no seu comportamento, é possível afirmar que os objectivos principais propostos foram cumpridos.

5.2. Trabalho futuro

Ao longo do desenvolvimento da aplicação, foram identificadas várias possibilidades de melhoramento desta, pelo que seria de todo pertinente dar continuidade ao trabalho iniciado neste projecto.

As possíveis melhorias a introduzir na aplicação prendem-se essencialmente com as funcionalidades disponibilizadas pela interface gráfica. Seria, sem dúvida alguma, uma grande mais-valia que esta permitisse a edição da planta em tempo de execução.

Um outro aspecto importante a implementar no futuro seria a possibilidade de utilização de *Layers* na planta. Desta forma, seria possível simular casas com mais do que um piso, ou até mesmo prédios habitacionais. Também seria importante simular o comportamento de seres humanos no interior da casa.

O ponto mais importante a ser realizado num possível trabalho futuro passaria pela melhoria do código e correcção de *bugs*, pois, actualmente, este possui muito “lixo” e alguma desorganização, mas as classes criadas bem organizadas teriam potencial para desenvolvimento de uma API para simulação de vários cenários.

Referências bibliográficas

Bonino, D., & Corno, F. (2011). *AUTOMATION IN CONSTRUCTION*.

Ecma-international. (2006). Standard ECMA-334 C# Language Specification. Retrieved from http://www.ecma-international.org/publications/standards/Ecma-334.htm

HARREL, C. R., GHOSH, B. K., & BOWDEN, R. (2000). *Simulation Using ProModel*®.

Hintjens, P. (2011). *ZeroM zguide*.

KELLNER, M. I., MADACHY, R. J., & RAFFO, D. M. (1998). *Software process simulation modeling: Why? What? How?*

LAW, A. M., & KELTON, W. D. (1991). *Simulation Modeling and Analysis*.

Loureiro, H. (n.d.). *C# 4.0 com Visual Studio 2010 - Curso Completo*.

Murach, J. (n.d.). *Murach's C# 2010* (p. 2012).

Paulo Marques, Pedroso, H., & Figueira, R. (n.d.). *C# 4.0*.

Portaleducacao. (2008). História e características da linguagem C#. Retrieved from http://www.portaleducacao.com.br/informatica/artigos/6137/historia-e-caracteristicas-da-linguagem-c

Prof. Borges, R. C. de M. (n.d.). Cálculo do Nascer e Pôr do Sol. *Instituto de Informática - UFRGS*. Retrieved from http://www.inf.ufrgs.br/~cabral/Nascer_Por_Sol.html

PROMODEL. (2000). *User's Guide Online*.

SAKURADA, N., MIYAKE, & D.I. (2002). *Softwares de Simulação de Eventos Discretos*.

Anexo A

Manual de Utilizador

Simulador Context

2013

Índice

Utilização Geral

1. Visão geral

A aplicação *simulador context* como se pode verificar na figura abaixo, e pelos exemplos dados ao longo do manual, foi desenhada com o cuidado de ser fácil de usar por qualquer tipo de utilizador.

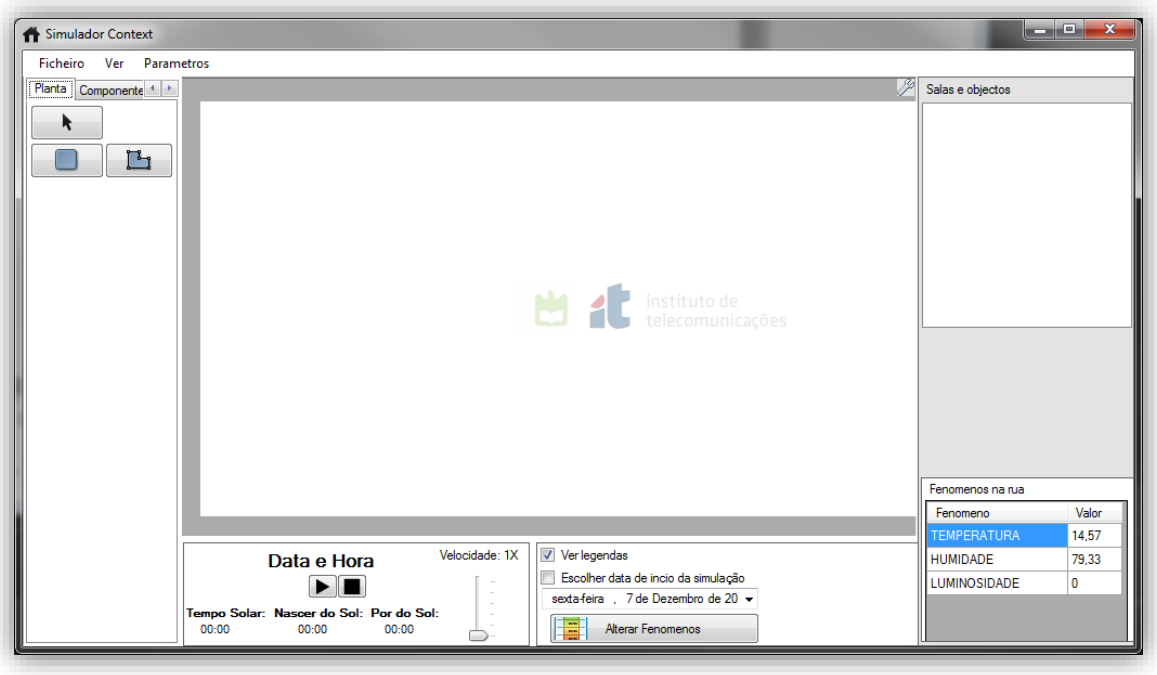

Imagem 1 - Área de trabalho

2. Criar novos Componentes e Sensores

2.1. Componente

Ir a parâmetros e escolher componentes, no caso de pretender criar novos componentes.

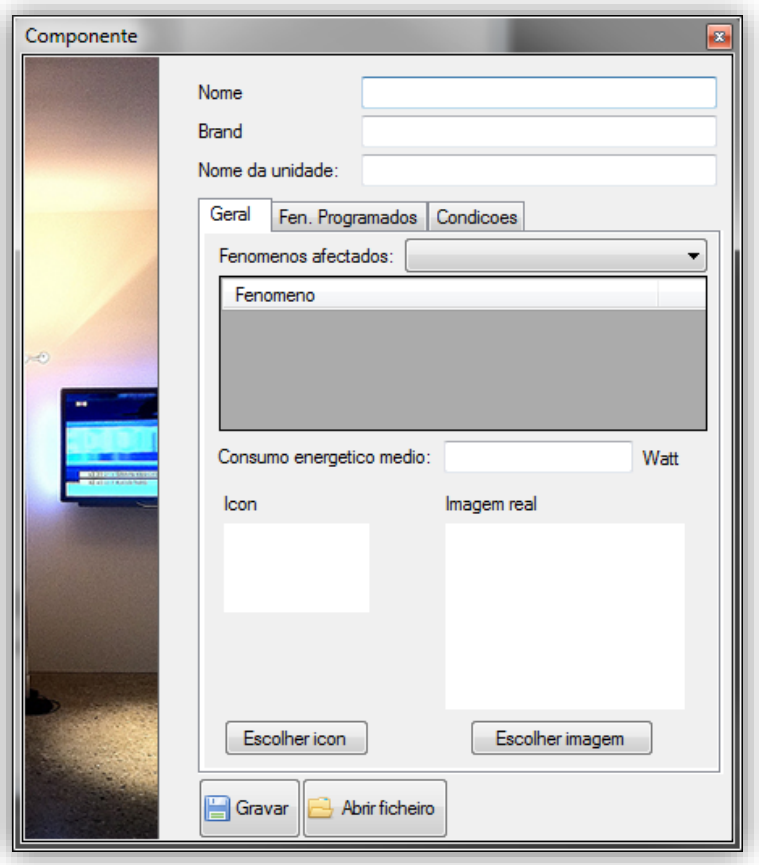

Imagem 2 - Formulário de criação de Componentes

Depois de aberta a janela representada acima, pode iniciar a edição ou criação de um novo componente, para tal todos os campos apresentados na imagem são obrigatórios.

De referir que o campo unidade, é a unidade do equipamento, por exemplo se estivermos a falar de um televisor, a unidade será canal.

Icon: é a imagem que vai aparecer na barra de selecção dos componentes.

Imagem: imagem que vai aparecer na planta.

Os componentes criados deve ser guardados na seguinte pasta @\Data\Componentes

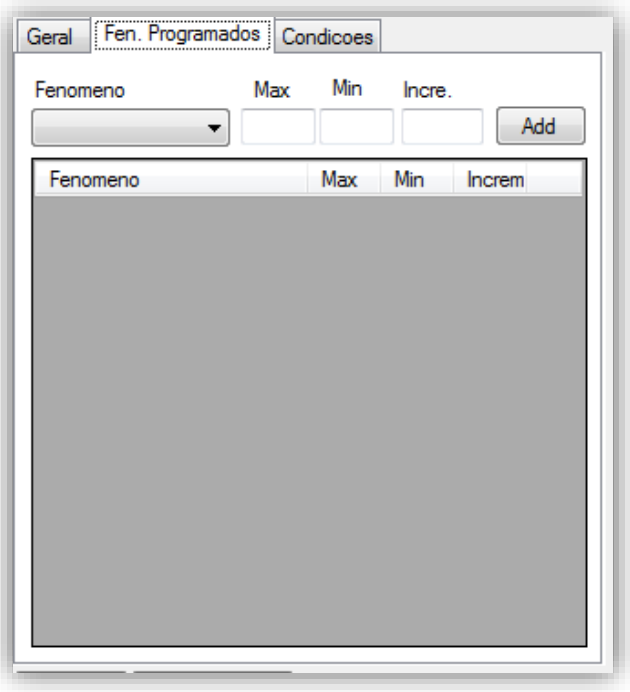

Imagem 3 - separador de Fenómenos programados

No separador apresentado na imagem 3 é onde se vai introduzir a forma de como o componente vai interagir com os fenómenos.

Temos uma lista de fenómenos para escolher o fenómeno que pretendemos afectar, depois tempos o valor máximo, mínimo e incrementador hora. Por exemplo, no caso de um ar condicionado podemos por os seguintes valores:

Fenómeno: Temperatura

Maximo:35

Minimo: 0

Incrementador 20

Isto na pratica quer dizer que o ar condicionado só tem a capacidade de afetar a temperatura entre 0 e 35 graus, quando se pede para a sua acção saia deste intervalo o que vai acontecer é que o equipamento não a capacidade de fazer o fenómeno temperatura descer a baixo dos 0 graus, assim como acima dos 35 graus, e o incrementador quer dizer que ele em uma hora só consegue subir ou descer 20 graus. Se tivermos a temperatura divisão a 0 graus, o ar condicionado vai demorar aproximadamente hora e meia a colocar a temperatura da divisão nos 35 graus.

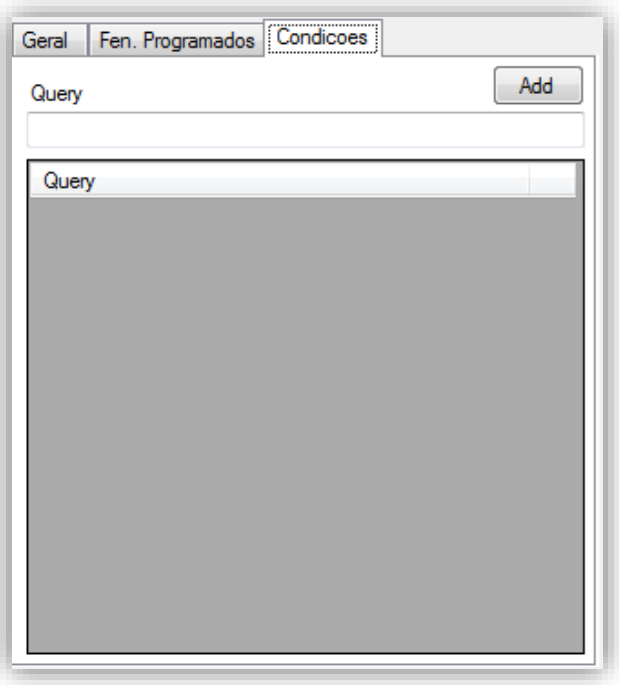

Imagem 4 - Separador condições

O separador condições é o que vai da "vida" ao equipamento, isto porque são as condições que devem alterar o funcionamento do componente.

O componentes tem uma condição obrigatória, que tem como objectivo indicar qual é o fenómeno que o componente vai afetar.
Voltando ao exemplo do ar condicionado. Quando interagimos com o ar condicionado o objectivo é interagir com a temperatura.
Então ao ser criado o componente tem que se inserir a seguinte condição:

Valor Objectivo Temperatura

Exemplo numero dois:

Se pretendermos dizer que o ar condicionado estiver quase a atingir o seu objectivo, então este vai consumir menos energia. Para tal temos a seguinte condição:

Se objectivo – valor < 5 então consumoenergetico = 20%

Nota: Como se pode constatar o fenómeno aparece numa única palavra, isto deve-se ao facto do sistema não reconhecer espaços nos fenómenos.

2.2. Criar um sensor

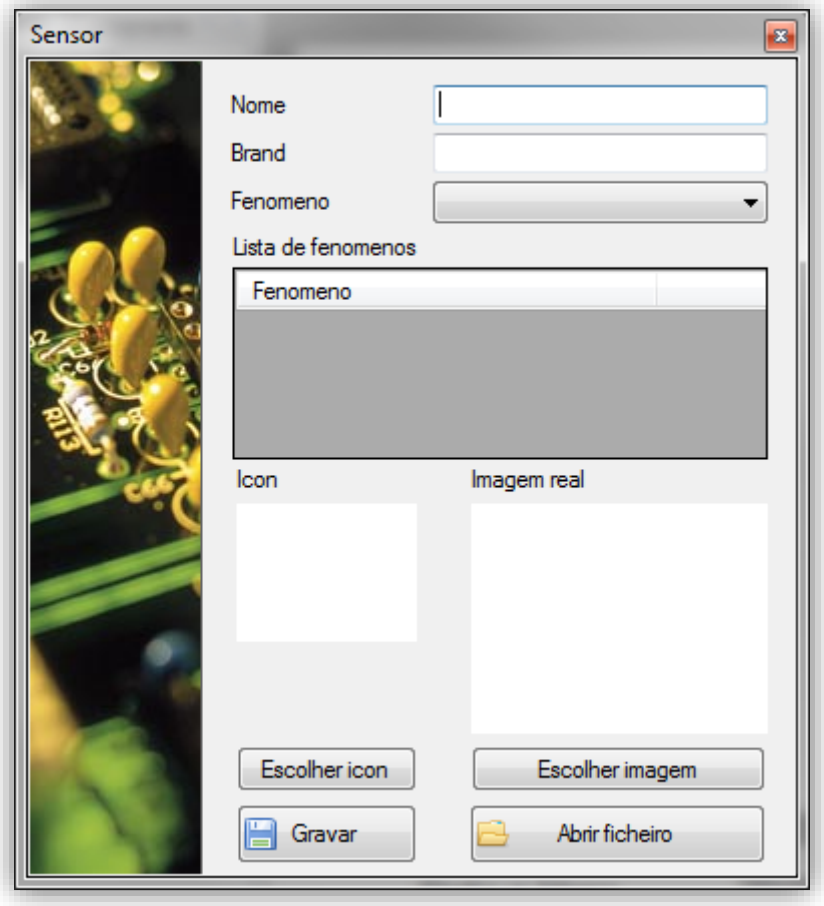

Imagem 5 - Formulário para criação de novo sensor

Nome: Nome do sensor.

Brand: Marca do sensor.

Fenomenos: Um Sensor tem a capacidade de ler um ou mais fenómenos.

Icon: é a imagem que vai aparecer na barra de selecção dos componentes.

Imagem: imagem que vai aparecer na planta.

Os Sensores criados devem ser guardados na seguinte pasta @\Data\ Sensor

3. Criar uma nova planta

Imagem 6 - Opções para desenhar planta

O separador planta é onde o utilizador pode escolher as geometrias a utilizar na construção da planta.

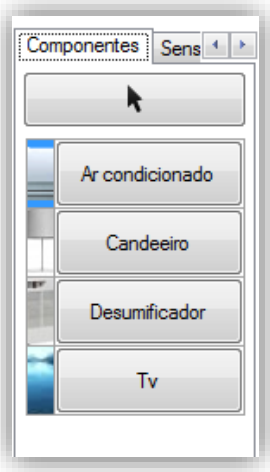

Imagem 7 - Barra de componentes

O separador componente é a área destinada à selecção dos componentes a serem introduzidos no interior das divisões da casa.

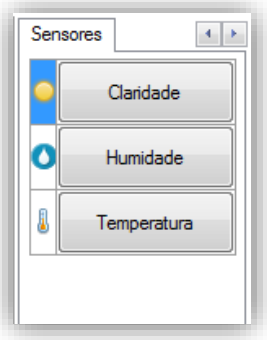

Imagem 8 - Barra de Sensores

O separador Sensores é a área destinada à selecção dos sensores a serem introduzidos no interior das divisões da casa. Para obter as leituras dos fenómenos.

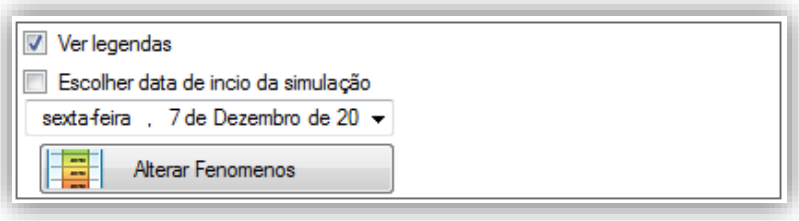

Imagem 9 - Barra de edição de simulação

Área onde utilizador deve indicar a data de início da simulação, caso o deseje efectuar a alteração da data, este procedimento dever ser efectuado antes do inico da construção da planta. Assim desta forma todos os valores gerados, tem como referencia a época do ano escolhida.

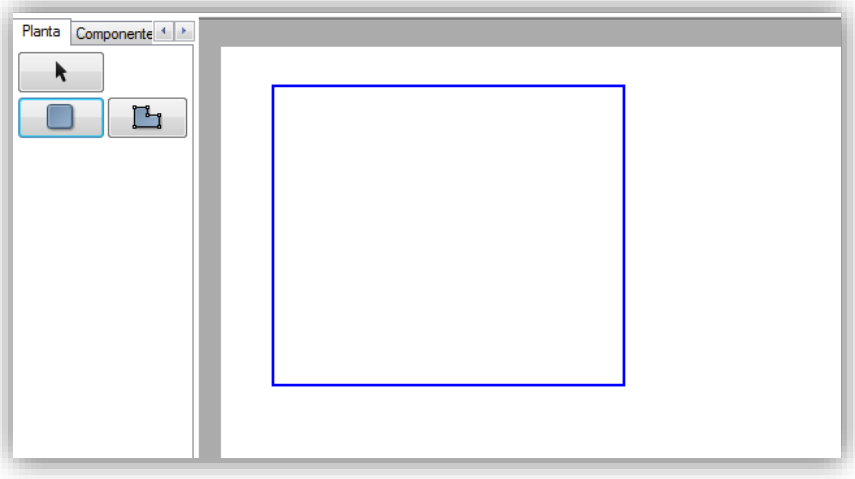

Imagem 10 - Desenho de geometria na area de trabalho

Para iniciar a construção da planta, têm que seleccionar a geometria pretendida, e depois clicar uma vez na zona de desenho e dar a forma pretendida à divisão da casa. Para terminar a geometria deve pressionar 2X o rato (duplo clique).
(representado na Imagem 10).

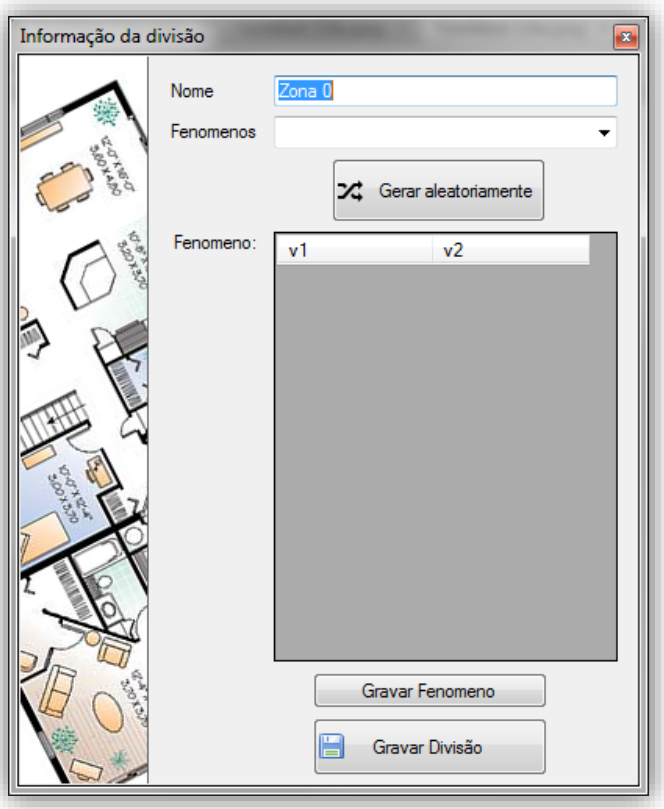

Imagem 11 - Formulário de edição de divisão

Após concluir a figura, vai ser apresentado um formulário onde pode atribuir o nome à divisão da casa, assim como alterar ou gerar novos valores dos fenómenos.

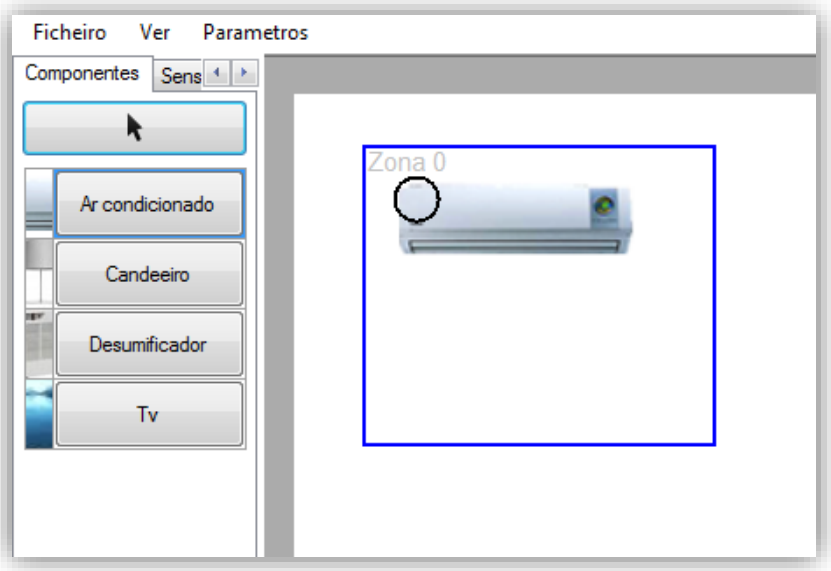

Imagem 12 - Inserir Componentes na planta

Para inserir um componente na planta tem que o seleccionar e clicar na divisão onde pretende colocar o mesmo. Sempre que pretende inserir um novo componente numa divisão da planta, tem de repetir o processo.

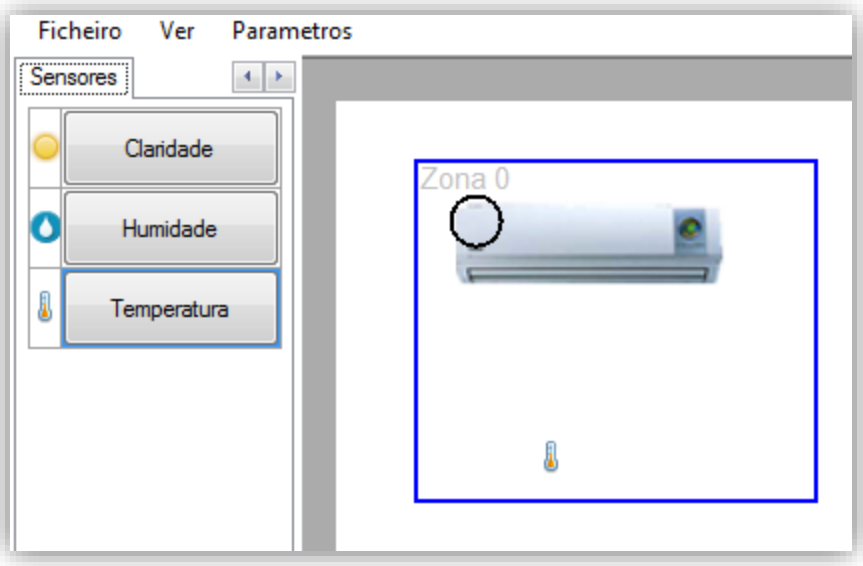

Imagem 13 - Inserir sensor na planta

Para inserir um Sensor na planta tem que o seleccionar e clicar na divisão onde pretende colocar o mesmo. Sempre que pretende inserir um novo sensor numa divisão da planta, tem de repetir o processo.

Imagem 14 - Controlador de simulação

Painel que o informa da data e hora em que se encontra a simulação, assim como o tempo solar, hora em que nasce o sol e hora do por do sol. Neste painel também se pode visualizar a velocidade da simulação assim como alterar a mesma.

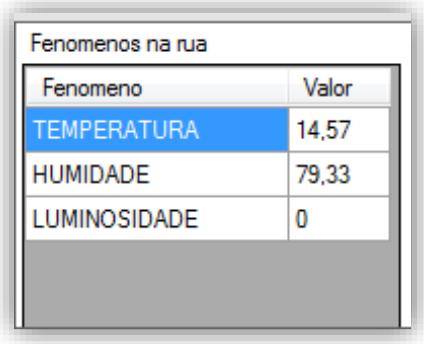

Imagem 15 - Fenómenos referencia do simulador

A imagem 15 apresenta o mostrador de valores referencia dos fenómenos. Estes são actualizados automaticamente quando a data se altera.

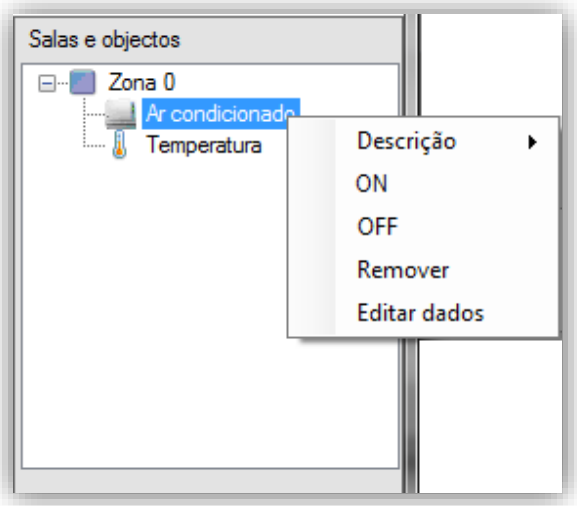

Imagem 16 - Arvore de objecto presentes na simulação

A figura 16 apresentanos a arvore dos objectos que estão presentes na plata. Tambem nos permite ver quais os sensores e componentes que estão dentro de cada divisão da planta.

Ao seleccionar o componente e pressionar na tecla esquerda do rato, é-nos apresentado um menu com varias opções, onde podemos ligar e desligar o equipamento, assim como remover o mesmo. Também podemos ver a descrição do mesmo. Na opção “Descrição” pode-se ver em tempo real as leituras de sensores, assim como os valores seleccionados nos componentes.

4. Ligação à aplicação via rede

Para efectuar a ligação ao simulador as portas a que se deve ligar são as seguintes:

Leitura: 8051:tcp

Envio: 8052:tcp

A estortura de dados a ser enviada deve ser a seguinte:

```
{
  "type": "9",
  "contextML": "<?xml version=\"1.0\" encoding=\"utf-16\"?>
                    <contextML>
                    <action>set</action>
                    <component>
                    <id>id do componente</id>
                    <on>valor inteiro correspondentes aos minutos que vai estar ligado</on>
                    <valor>valor a atribuir ao componente</valor>
                    </component>
                    </contextML>"
}
```

A estortura dos dados recebidos deve ser a seguinte:

```
 {
   "type": "9",
   "contextML": "<?xml version=\"1.0\" encoding=\"utf-16\"?>
 <contextML>
 <action>get</action>
 <sensorid>mac adress do sensor</sensorid>
 <Fenomeno>valor do fenomeno</Fenomeno>
 <timestamp>valor</timestamp>
 </contextML>"
}
```

Estes anexos só estão disponíveis para consulta através do CD-ROM.
Queira por favor dirigir-se ao balcão de atendimento da Biblioteca.

Serviços de Biblioteca, Informação Documental e Museologia
Universidade de Aveiro