

**Universidade do Minho**

Departamento de Informática

Mestrado Integrado em Engenharia Informática

Perfil de Sistemas Inteligentes

Unidade Curricular de Aprendizagem e Extração de Conhecimento

2015/2016

Aprendizagem e Extração de Conhecimento

André Geraldes a67673

Bruno Barbosa a67646

Francisco Dourado pg30458

**Abstract**

This report describes the development process of a multi agent architecture framed in the management and monitoring problem of an intelligent house with sensors. Every decision made throughout the project is presented, and the relevant implementation details explored.

**Resumo**

Neste relatório é descrito o processo de desenvolvimento de uma arquitetura multiagente enquadrada no problema de gestão e monitorização de uma casa inteligente com auxílio de sensores. São apresentadas todas as decisões tomadas e explorados os detalhes de implementação mais relevantes.

Índice

[1 Introdução 4](#_Toc437204478)

[1.1 Enquadramento e objetivos 4](#_Toc437204479)

[1.2 Planeamento 5](#_Toc437204480)

[1.3 Estrutura do documento 6](#_Toc437204481)

[2 Desenvolvimento 7](#_Toc437204482)

[2.1 Arquitetura da Aplicação 7](#_Toc437204483)

[2.1.1 Diagrama da Arquitetura 7](#_Toc437204484)

[2.1.2 Interface Agent 8](#_Toc437204485)

[2.1.3 Controller Agent 10](#_Toc437204486)

[2.2 Sensores 11](#_Toc437204487)

[2.2.1 Temperatura 11](#_Toc437204488)

[2.2.2 Humidade 12](#_Toc437204489)

[2.2.3 Luminosidade 12](#_Toc437204490)

[2.2.4 Fumo 12](#_Toc437204491)

[2.2.5 Movimento 12](#_Toc437204492)

[2.3 Arduino 12](#_Toc437204493)

[2.4 Interface 14](#_Toc437204494)

[2.4.1 Tab Overview 15](#_Toc437204495)

[2.4.2 Tab de Estatísticas 16](#_Toc437204496)

[2.4.3 Tab Painel de Controlo 17](#_Toc437204497)

[3 Conclusões e Trabalho Futuro 19](#_Toc437204498)

[4 Anexos 20](#_Toc437204499)

[4.1 Tutorial da Aplicação 20](#_Toc437204500)

[4.2 Código do Arduino 25](#_Toc437204501)

# Introdução

Um sistema multiagente é um sistema composto por agentes inteligentes definidos por um comportamento, na maioria dos casos simples, cuja interação conjunta é capaz de produzir resultados complexos e poderosos.

No âmbito da unidade curricular Agentes Inteligentes do perfil de Sistemas Inteligentes do Mestrado Integrado em Engenharia Informática, foi proposto o desenvolvimento de uma aplicação baseada numa arquitetura multiagente que fizesse uso de sensores virtuais. Os sensores seriam responsáveis pela recolha de dados, e deveriam simular o comportamento esperado de sensores reais, incluindo erros nas leituras de valores. Deveria ser tido também em conta a possibilidade de *timeouts* e mensagens erróneas.

Era o objetivo que com esta implementação fossem cimentados alguns dos conceitos lecionados nas aulas, e desenvolvidas competências de programação com a plataforma Jade  e de conceção de arquiteturas distribuídas baseadas em agentes.

## Enquadramento e objetivos

Com a liberdade de expansão do trabalho base proposto, foi decidido desenvolver uma aplicação de monitorização e automação de uma casa. Esta decisão baseou-se no facto de uma casa inteligente ser um ambiente com elevada integração de sensores e no interesse geral dos elementos do grupo, uma vez que este é um tópico atual e em constante evolução.

De facto, com a nova onda de mudança no panorama tecnológico, como a iminência da adoção de realidade virtual e realidade aumentada pelas massas ou os avanços em novas matérias como plataformas interativas, a domótica poderá rapidamente ser uma realidade na vida de muitas pessoas. Algumas das tendências futuras apontadas focam-se na poupança de energia, espaços adaptativos e funcionais, objetos inteligentes interligados, monitorização de fatores de saúde e automação que garanta um estilo de vida independente (devido ao envelhecimento da população registado em países de 1º mundo). No fundo, a criação de uma experiência de vivência que se adapta ao *lifestyle* de cada um, respondendo a questões como o quê, como, quando, onde e porquê, de forma a proporcionar mais que conforto físico e funcional, conforto psicológico.

Após discussão conjunta, onde se tentou balancear os recursos disponíveis e a ambição do projeto, foram delineadas as seguintes metas a cumprir no desenvolvimento da aplicação:

* Possibilidade de ativar e desativar sensores;
* Monitorização, em tempo real, de valores fornecidos por vários sensores ativos;
* Existência de vários tipos de sensores, com comportamentos diferentes;
* Possibilidade de adição e remoção de sensores em tempo de execução, permitindo modificar a simulação de instalações físicas dos sensores;
* Apresentação de uma interface gráfica com bom ratio usabilidade/complexidade;
* Visualização da variação de valores de sensores ao longo do tempo com o auxílio de gráficos referentes a cada tipo de sensor;
* Apresentação de registos de atividade pertinentes (falhas, timeouts, mudanças de estado, etc);
* Utilização de perfis de automação compostos por regras e condições que produzam os efeitos previamente selecionados quando estas são validadas;
* Permitir ao utilizador a personalização completa de perfis de automação e a sua importação;
* Uso de um *Arduino* como prova de conceito, quer como sensor, quer como reator de perfil de automação.

## Planeamento

A metodologia de desenvolvimento de *software* utilizada neste projeto foi AGILE. Esta metodologia adequa-se bem a pequenas equipas que necessitam de produzir versões funcionais de *software* rapidamente.



Figura 1. Planeamento de tarefas

Criou-se uma lista de tarefas “TODO” organizadas por ordem de prioridade decrescente onde se atribuía um elemento do grupo como responsável. Quando alguém terminava a tarefa atribuída (sendo esta passada para a lista “Feito”), era-lhe atribuído uma nova tarefa prioritária da lista, dando preferência, se possível, à área da aplicação onde esse elemento do grupo se sentia mais confortável. Usou-se uma terceira lista que guardou notas variadas, desde discussões de *features* a recursos de auxílio à implementação e ainda uma outra que guardou os conjuntos de dúvidas a serem respondidas nas alturas de contacto com a equipa docente.

Assim, e em conjunção com breves reuniões de ponto de situação, foi fácil aproveitar os recursos temporais e humanos disponíveis num ambiente incerteza, onde o planeamento de projetos tradicional, baseado na previsão de datas concretas, é menos eficiente.

## Estrutura do documento

O presente documento está dividido em três secções e um anexo. Na primeira secção fazemos a introdução ao caso de estudo bem como o seu enquadramento no enunciado, os seus objetivos e o seu planeamento.

Na segunda secção apresentamos toda a parte referente ao desenvolvimento deste projeto, desde a arquitetura da aplicação, passando pela descrição dos agentes e os seus comportamentos, pela implementação do *Arduino* até à exploração da interface.

Na terceiro e última secção, apresentamos as conclusões obtidas através da realização deste trabalho.

Nos anexos estão expostos o código desenvolvido para a implementação do *Arduino* e um pequeno tutorial sobre as várias *features* da aplicação.

# Desenvolvimento

## Arquitetura da Aplicação

Após algum debate e troca de ideias delineámos a arquitetura da nossa aplicação.

### Diagrama da Arquitetura

Apresentamos primeiramente o diagrama da arquitetura de agentes implementada.

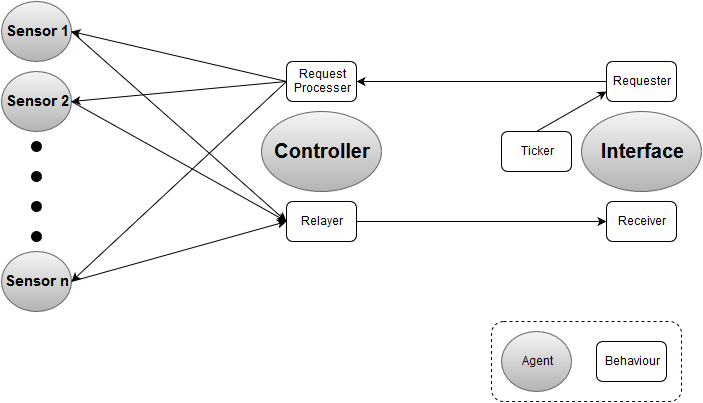


Figura 2. Arquitetura da aplicação

O agente de Interface é responsável pela gestão da informação que deve estar próxima da interface e pela ligação entre esta e os agentes da aplicação, traduzindo as ações do utilizador em mensagens estruturadas a serem enviadas ao agente *Controller,* quando necessário.

O agente *Controller* recebe e analisa as mensagens do agente Interface de forma a decidir que ação tomar. Recebe também mensagens de sensores, reenviando-as ao agente interface se se tratarem de respostas a pedidos.

Cada agente que representa um sensor pode receber mensagens do *Controller*, interagindo com as mesmas de várias formas, que serão explicadas em maior detalhe nas próximas secções.

Todos os agentes referidos podem ser classificados como sendo fracos, estacionários, de serviço, de baixa autonomia, inserindo-se numa arquitetura colaborativa, reativa a estímulos, provocados inicialmente pela interface, que se propagam por mensagens entre os agentes. Em todos os agentes verifica-se a presença, por mais que mínima, de memória.

Vale a pena referir também que todos os agentes no projeto fazem *extend* à classe CyclicBehaviour, uma vez que este comportamento é muito versátil, permitindo capturar eventos temporais e de mensagens de outros sensores.

Como é observável no diagrama, os comportamentos de agentes foram organizados de forma a que se mantivesse um fluxo de mensagens unidirecional. Esta organização revelou-se útil na diminuição de problemas de concorrência e no *debug* da aplicação durante o seu desenvolvimento.

### Interface Agent

A classe InterfaceAgent apresenta os seguintes objetos:

* Menu menu: objeto responsável pela apresentação da interface;
* Integer currentConvoId: valor usado na geração de novos identificadores de requests;
* HashMap<Integer, String> requestMap: guarda os *requests* enviados e ainda não respondidos;
* HashMap<String, String> sensorTypes: guarda a associação entre o nome de um sensor e o seu tipo;
* ArrayList<String> activeSensors: guarda os sensores instalados que se encontram ativos e que, periodicamente, necessitam de fornecer um valor;
* ArrayList<Rule> automationProfile: guarda as regras do perfil de automação ativo a serem avaliadas;

O agente de Interface é composto por três *sub-behaviours* paralelos: InterfaceRequesterBehaviour, InterfaceReceiverBehaviour e InterfaceTickerBehaviour.

A classe InterfaceRequesterBehaviour limita-se a consumir mensagens com a performativa *REQUEST*, enviadas pelo objeto menu, criando uma nova mensagem que é enviada ao agente *Controller*. Esta mensagem possui o mesmo conteúdo e performativa da mensagem recebida, mas um id único gerado a partir de currentConvoId através do método getNewConvoId(). Depois de enviada a mensagem, o seu id e conteúdo é guardado no requestMap do agente Interface.

A classe InterfaceReceiverBehaviour consome mensagens com as performativas *NOT\_UNDERSTOOD, INFORM, CONFIRM* e *FAILURE* provenientes do *Controller*. Dependendo da performativa, do conteúdo da mensagem e do conteúdo original do pedido (guardado no requestMap do agente Interface) é feita invocação da função correta que irá a processar o conteúdo da mensagem.

Destacamos as seguintes funções presentes na classe InterfaceReceiverBehaviour:

* printLog(String txt): Cria um log na interface com a hora atual concatenada com a *String* passada como parâmetro.
* processStatus(String content): Atualiza a mudança de estado (*online/offline*) de agentes, atualizando o ArrayList activeSensors e a interface gráfica. É também responsável por garantir que todos os agentes são desativados antes do fecho da interface de forma a não gerar erros nas próximas execuções da aplicação;
* processScan(String content): Atualiza a tabela de agentes instalados na interface gráfica.
* processSensorValue(String content, Integer id): Atualiza o valor atual do sensor na interface e procura nas regras do perfil de automação ativo por condições referentes ao sensor atualizado. Estas condições são avaliadas com o novo valor.

A classe InterfaceTickerBehaviour é responsável por enviar periodicamente pedidos de atualização de valores de sensores *online* ao *controller*. Os pedidos são feitos da mesma forma que na InterfaceRequesterBehaviour. Para além disso, este *behaviour* é também responsável por percorrer as regras do perfil de automação e atualizar a interface com os resultados previamente calculados. Desta forma, as regras não são constantemente re-avaliadas, embora se mantenha uma velocidade de resposta rápida. Este *behaviour* pode ainda receber mensagens com a performativa PROPOSE e um valor inteiro, que permite alterar o intervalo de atualização de valores dos sensores ativos e do processamento de regras.

Como os vários *behaviours* do agente Interface acedem a currentConvoId e requestMap são usados *synchronized statements* nas funções do agente Interface que interagem com esses métodos, nomeadamente, em int getNewConvoId(), void saveRequest(int id, String content), String getRequestContent(int id) e void removeRequest(int id), não se registando assim problemas de concorrência.

### Controller Agent

O agente *Controller* é composto por dois *sub-behaviours* paralelos, RequestProcesserBehaviour e RelayerBehaviour, guardando apenas um objeto HashMap<String,Long> convoIds que guarda os ids de pedidos recebidos e um *timestamp* do tempo em que estes chegam ao agente.

A classe RequestProcesserBehaviour consome mensagens com a performativa *REQUEST* vindas do agente Interface, guarda o id de mensagem e processa o pedido. Começa por procurar, com auxílio do agente DirectoryFacilitator, os nomes de agentes disponíveis que tenham como tipo um dos que o *Controller* reconheça. Os tipos reconhecidos são guardados num *enum* neste *behaviour*. Seguidamente, é verificado se o agente alvo se encontra na *string* de todos os agentes disponíveis, sendo enviada uma mensagem  à interface com a performativa *FAILURE* se este não for o caso. Depois é verificado o conteúdo do pedido. Se o conteúdo for scan é enviada uma mensagem ao agente Interface com a *string* que contém o nome dos agentes disponíveis. Se for *online*, *offline* ou *value* é enviada ao agente alvo de forma a ser realizada a mudança de estado ou possa ser obtido o seu valor atual. Se for outro o caso é enviada uma mensagem com a *performative NOT\_UNDERSTOOD* ao agente Interface.

Esta classe é também responsável pela verificação de *timeouts* de pedidos sendo feita uma comparação de *timestamps* com o tempo atual de todos os pedidos ainda não respondidos em convoIds. Essa verificação é feita na função *action* e, para garantir que há evento que faça executar esta função, foi usado block(10000).

A classe RelayerBehaviour consome mensagens com performativas *NOT\_UNDERSTOOD, INFORM, CONFIRM* e *FAILURE* provenientes de sensores, limitando-se a criar uma mensagem idêntica se o seu id corresponder a um id de pedido ainda não respondido. O id do pedido é removido de convoIds e a mensagem é enviada para a Interface.

Como os dois behaviours do agente acedem a currentConvoId são usados *synchronized statements* nas funções do agente Controller que interagem com esses métodos, nomeadamente, boolean hasRequestId(String id), long getTimeStamp(String id), void saveRequestId(String id) e void removeRequestId(String id).

## Sensores

Fazendo um levantamento de quais os sensores a serem integrados na aplicação de forma a ser possível obterem-se configurações variadas, obteve-se a seguinte lista:

* Temperatura;
* Humidade;
* Luminosidade;
* Fumo;
* Movimento.

Todas as classes que representam sensores contêm um método takeDown que serve para retirar o seu serviço dos serviços disponíveis, o método *setup* que inicia o agente e o regista no agente DirectoryFacilitator (DF), e um ReceiveBehaviour que trata os pedidos que o sensor recebe. Se o conteúdo recebido for “shutdown” coloca a *flag* *finished* a *true*, “online” coloca o sensor *online*, “offline” coloca o sensor *offline*. O funcionamento destes pedidos é comum a todos os sensores. A diferença entre os sensores é o seu tipo que pode ser “temp”, “humi”, “lumi”, “smoke” e “move” e o funcionamento do pedido “values”.

Para maior aproximação à realidade, em cada sensor incorre a possibilidade de um erro de leitura. Quando há um pedido “values” é gerado um valor aleatório de 0 a 10 que funcionará como probabilidade de erro, se o valor for maior ou igual a 9 o sensor envia um erro “XXXXX” caso contrário envia um valor normal.

### Temperatura

O seu tipo é “temp” e este sensor foi-nos cedido com o enunciado do trabalho. Estudámos o seu funcionamento e fizemos algumas modificações que nos pareceram relevantes, como a atribuição de uma gama de variação aos valores gerados pelo sensor. Para tal criámos uma variável no sensor (int *last*) que representa o último valor de temperatura, iniciado com um valor aleatório de 0 a 40, e a sua variação é de -5 a +5 a cada pedido do tipo values. Se o valor for inferior a -10 ou superior a 40 o sensor gera um novo valor. É assumido que os valores de temperatura se encontram em graus Celsius.

### Humidade

É do tipo “humi” e neste caso o funcionamento do pedido *values* é idêntico ao do sensor de temperatura, a diferença é que a escala de humidade varia entre 0 e 100. Valores acima de 100 ou abaixo de 0 originam um novo valor. Os valores de humidade são dados em percentagem.

### Luminosidade

O tipo deste sensor é “lumi”, e a gama de valores é de 0 a 1000. O valor aleatório gerado inicial está entre 0 e 1000 e a sua variação é de -50 a +50 a cada pedido. Valores de luminosidade em lux abaixo de 400 representam luminosidade noturna e acima de 400 luminosidade diurna.

### Fumo

Sensor do tipo “smoke” com uma gama de valores de 0 a 100. Valores em percentagem.

### Movimento

O sensor de movimento é do tipo “move” e gera um valor aleatório entre 0 e 120. Se o valor estiver entre 0 e 30 ele envia o valor “1”, se estiver entre 30 e 100 envia “0” e se for superior a 100 envia um erro (XXXXX).

## Arduino

Todos os sensores anteriormente referidos são virtuais, e, como tal, não são suscetíveis a problemas reais. De acordo com as metas de projeto, incluímos a possibilidade de se usar um *Arduino* na aplicação, um controlador de baixo custo que nos permite utilizar ferramentas para simular o que acontece na realidade. Para o nosso projeto utilizamos um sensor de temperatura e dois leds, um verde e um vermelho.

A integração do *Arduino* funciona como um sensor de temperatura normal, enviando valores de temperatura lidos e é ainda capaz de alternar qual o led ligado como resultado de um perfil de automação.

Para utilizar o *Arduino* no projeto foi preciso fazer as ligações necessárias a nível de *hardware*. As ligações feitas utilizaram um *Arduino*, uma *breadboard*, duas resistências (220 ohms) e alguns *jumper wires*, componentes visíveis no seguinte esquema.

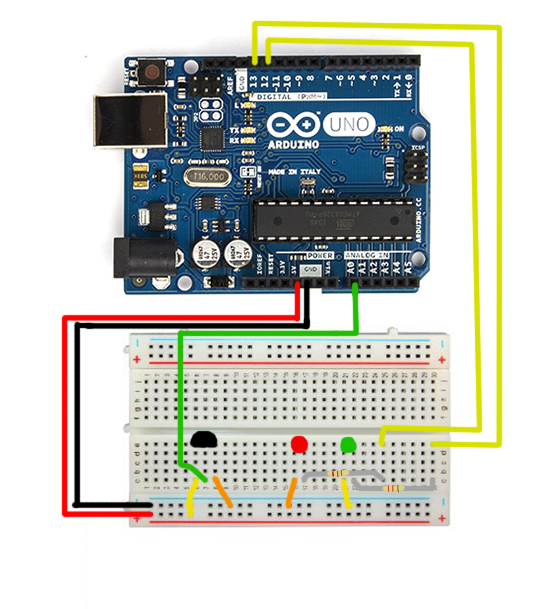


Figura 3. Configuração do Arduino

No esquema podemos ver que a porta analógica utilizada pelo sensor de temperatura é a A0 e as portas digitais utilizadas pelos leds são a 12 e 13 para o led vermelho e led verde, respetivamente.

Após esta configuração foi necessário desenvolver o *software* que fizesse a gestão dos eventos do hardware. Para isto, foi utilizando o IDE oficial do *Arduino* para criar o ficheiro temptest.in. Este cria uma serial *port* 9600 e envia para lá os valores de temperatura lidos, assim como os valores recebidos neste caso (101 e 102) para alterar o funcionamento dos leds. (O código do arduino pode ser consultado nos anexos)

Foi depois necessário proceder à integração na aplicação Java. Criou-se uma classe chamada SerialPortCommunication que serve para comunicar com o *Arduino* utilizando a mesma porta em que ele está a escrever os valores de temperatura e a ler os valores para os leds. Seguidamente foi criada a classe arduinoTemp, com o tipo “arduino” e os mesmos métodos dos sensores previamente referidos. Adicionalmente está presente uma instância SerialPortCommunication *serial* usada para comunicar com o *Arduino*. O pedido “value” utiliza um “serial”.getData para ler o valor da temperatura e enviar para o *controller*, o pedido “turnon” faz um “serial”.writeData(102) para ligar o led verde e desligar o led vermelho e o pedido “turnoff” faz um “serial”.writeData(101) para ligar o led vermelho e desligar o led verde.

## Interface

A interface é o instrumento de ligação entre o utilizador e o sistema. Para podermos dar ao utilizador uma boa experiência de utilização, optámos por desenvolver uma interface gráfica onde é possível acompanhar o estado da aplicação, isto é, verificar a monitorização, assim como configurar as suas preferências de utilização e configurações dos sensores.

Relativamente ao *layout*, do lado esquerdo temos uma tabela (agente, tipo e estado) com a lista de sensores presentes no sistema e um botão (na coluna estado) que os permite ligar ou desligar **(ver retângulo vermelho)**. Para a criação de novos agentes, ao lado desta tabela existe um painel que permite isso mesmo, associando um tipo disponível de sensores a um nome desejado. **(ver retângulo verde)**

Para que seja possível ao utilizador acompanhar as alterações de comportamento do sistema, existe uma área de texto na região inferior onde são registados esses acontecimentos **(ver retângulo amarelo)**. Por fim, no lado direito da janela da aplicação encontram-se três separadores de painéis alusivos à *overview* da casa, às estatísticas e ao painel de controlo, respetivamente. No caso do separador das estatísticas, este subdivide-se ainda em seis separadores, um para cada tipo de sensor. Cada um destes painéis serão explicados com maior detalhe nas seguintes secções. **(ver retângulo azul)**

Veja a imagem 4 na página seguinte.

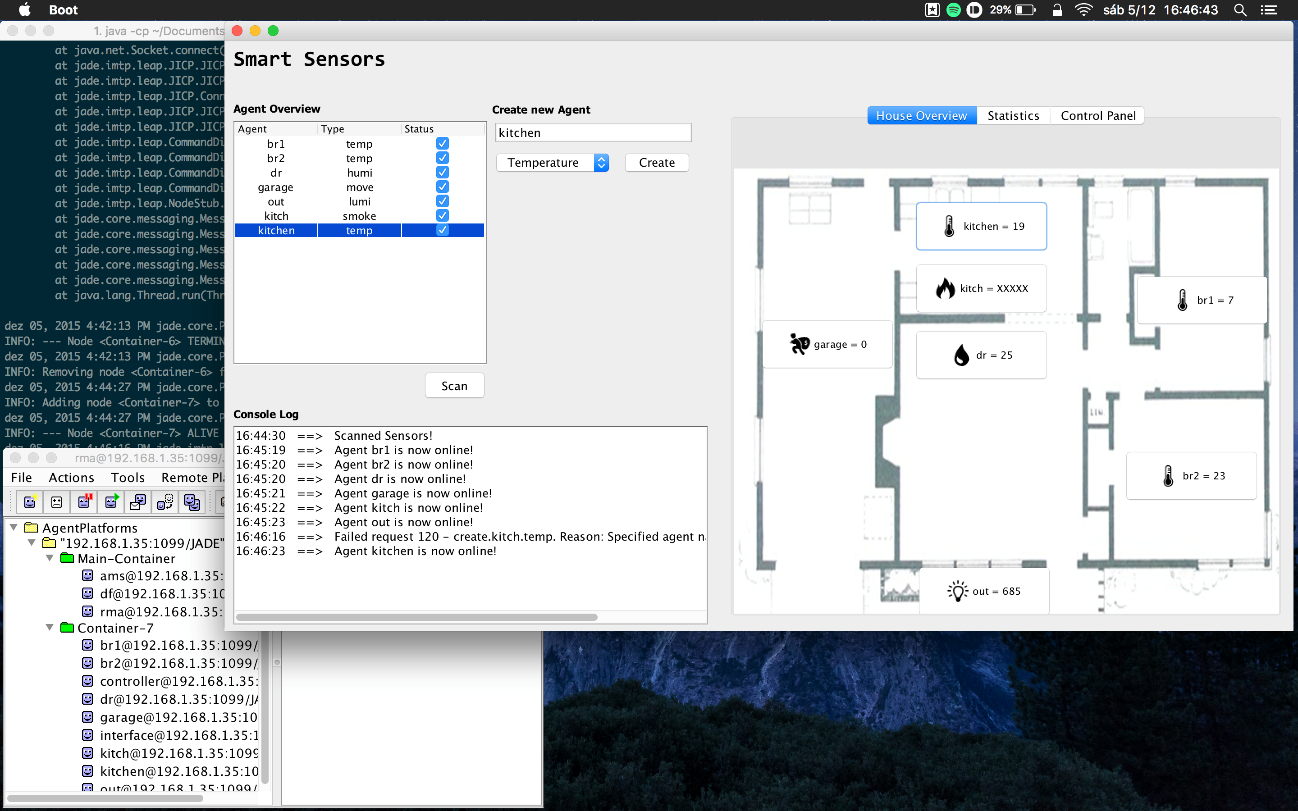


Figura 4. Interface gráfica (layout)

### Tab Overview

A aplicação irá suportar a gestão de uma casa inteligente e como tal, para que o utilizador possa ter uma visão geral do que está acontecer na casa, neste separador existe uma representação da planta da casa interativa.

Existe um botão associado a cada sensor registado que representa o valor enviado **(ver retângulos vermelhos)***.* É possível movimentar este botão pelo painel para que seja possível representar a posição exata dele na casa e apenas é visível caso o sensor esteja *online.* Para além disso, para que seja possível identificar melhor os diferentes tipos de sensor cada botão tem também um ícone intuitivo associado.

Assim, garantimos que o utilizador pode observar com clareza o que acontece dentro da casa de uma forma simples e bastante intuitiva.

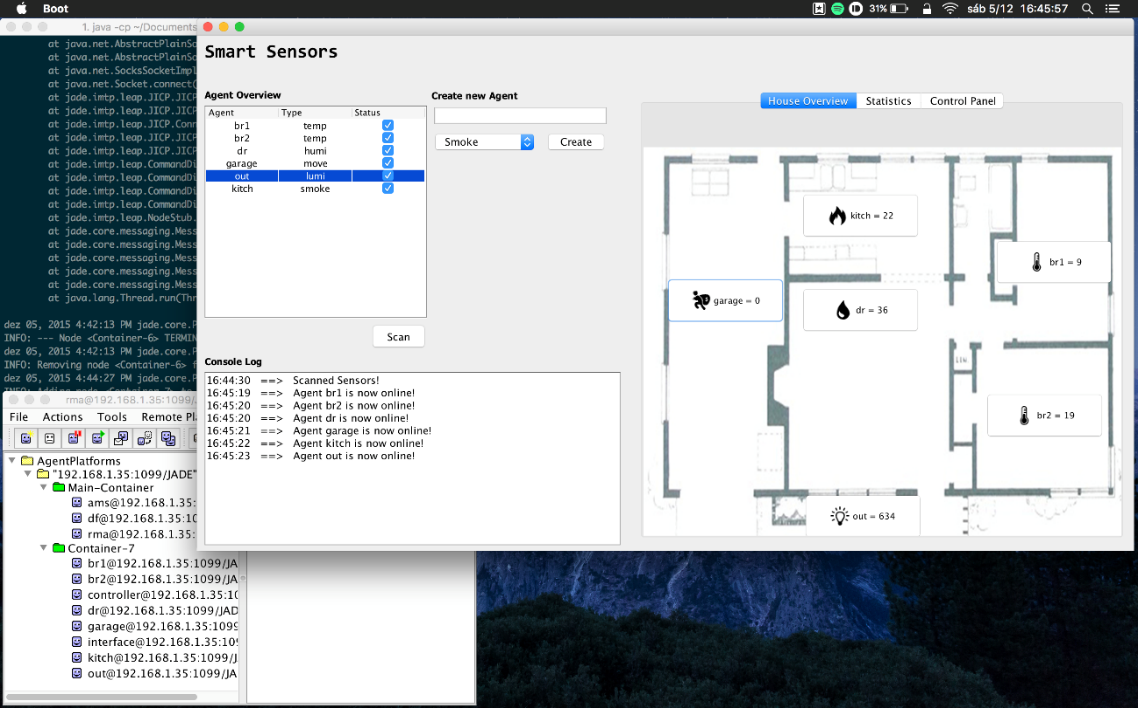


Figura 5. Tab overview

### Tab de Estatísticas

A *tab* de estatísticas permite ao utilizador ver graficamente os dados recolhidos pelos sensores nos últimos instantes. O procedimento de criação dos gráficos resume-se em iniciar os painéis com o gráfico a que lhe diz respeito e iniciar todos os *datasets* a eles respetivo. Depois de serem lidos pelos sensores, os dados são inseridos nos *datasets* classificados de acordo com o tipo de sensor.

Para a obtenção dos gráficos foi usada uma biblioteca auxiliar designada *JFreeChart* que inclui as ferramentas necessárias e adequadas para esse efeito. Todos os gráficos gerados correspondem a gráficos de linhas pelo que, por exemplo, no gráfico das temperaturas cada linha representa um sensor desse tipo.

Para além disso, as escalas dos eixos verticais estão em concordância com a gama de valores possíveis dependendo do sensor (ver secção dos sensores). Em relação à escala horizontal, usamos os cinco mais recentes *timestamps* dos instantes em que foram feitas leituras de dados do ambiente. Desta forma, à medida que inserem novos dados, o gráfico atualiza automaticamente ajustando-se sempre ao instante da última leitura. **(ver retângulos vermelhos)**

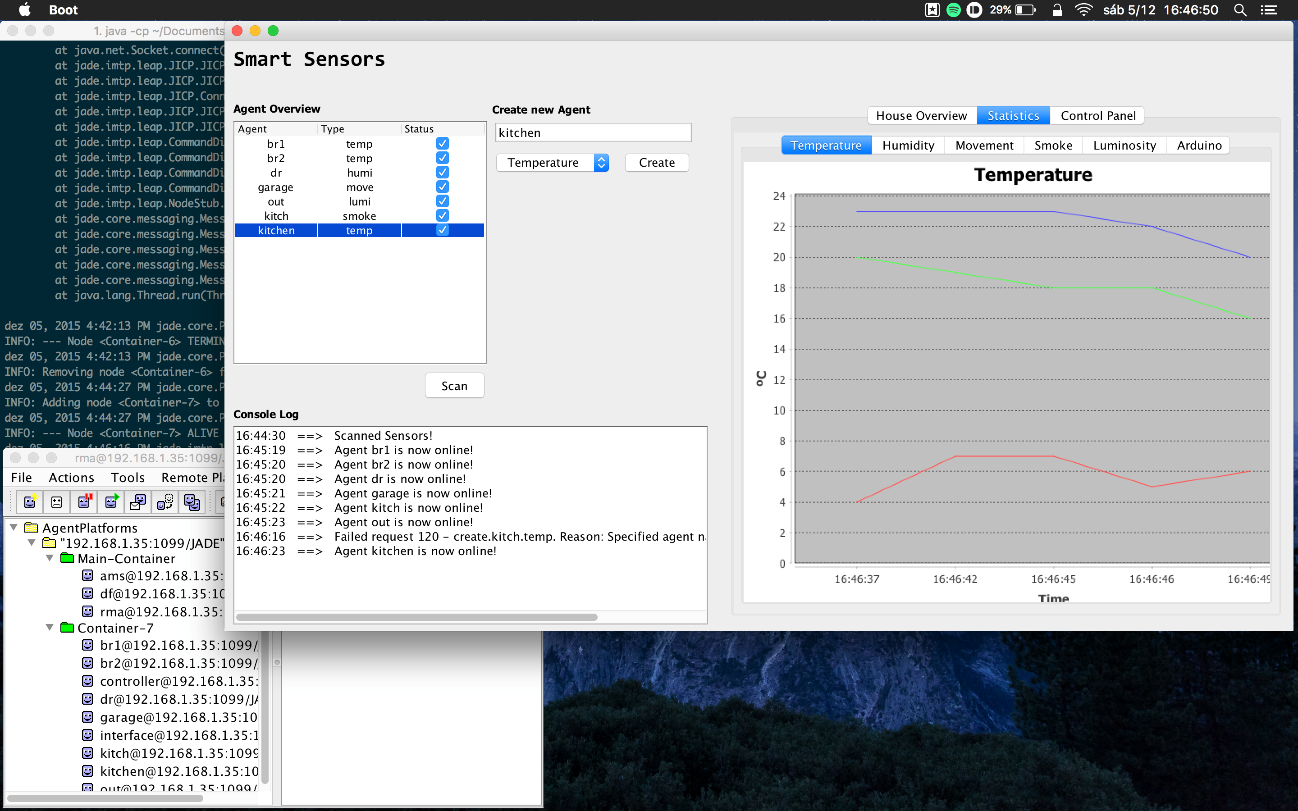


Figura 6. Tab estatísticas (temperatura)

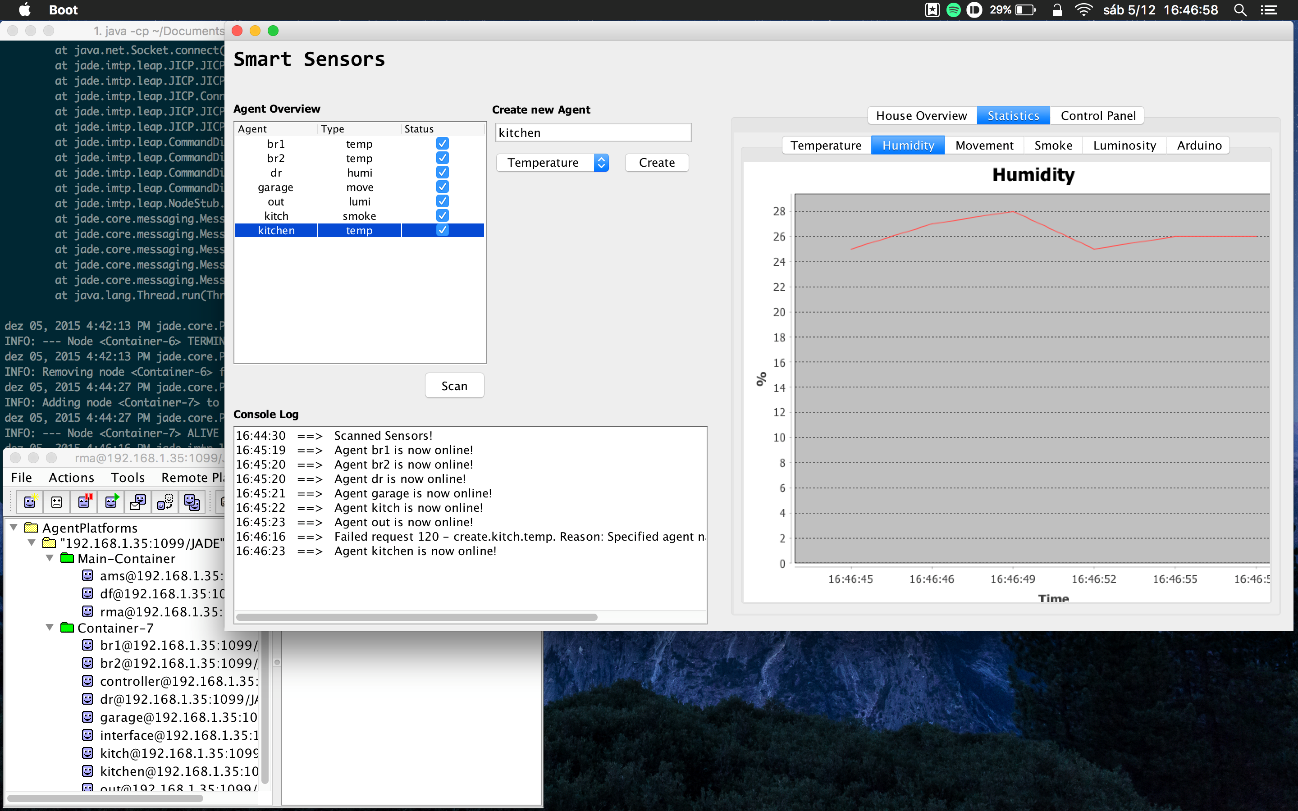


Figura 7. Tab estatísticas (humidade)

### Tab Painel de Controlo

Como vimos anteriormente, os sensores terão regras associadas a si, as quais o sistema irá gerir e criar comportamentos reativos para cada. Neste separador existem os controlos disponíveis para a criação das regras e para a associação destas com os sensores.

O funcionamento é muito simples, na primeira tabela desenvolvemos as condições das regras (1). Estas podem ser relacionadas com o valor de um determinado sensor (2) ou do tipo temporal (“a partir das 10h faz X”) (3), para a criação de cada uma existe um botão com um ícone representativo do tipo da condição.

Assim que as condições estiverem criadas, podemos também adicionar uma mensagem para quando a regra se verifica e outra para quando fica desativada (4). Posto isto, com a regra e as suas condições definidas podemos adiciona-la ao perfil de automação. Este é representado numa tabela na parte inferior do separador, tem uma coluna para ativar e desativar as regras registadas e uma outra para que saibamos as condições da regra.

É ainda permitido importar e exportar perfis de automação e para tal temos os botões respetivos junto da tabela. Quando pretendemos exportar o perfil, essa operação seleciona todas as regras registadas e guarda-as num ficheiro que mais tarde pode ser importado,  assim que é carregado substitui as regras em vigor até aquele momento pelas do perfil importado. (5)

Veja a imagem 8 na página que se segue.

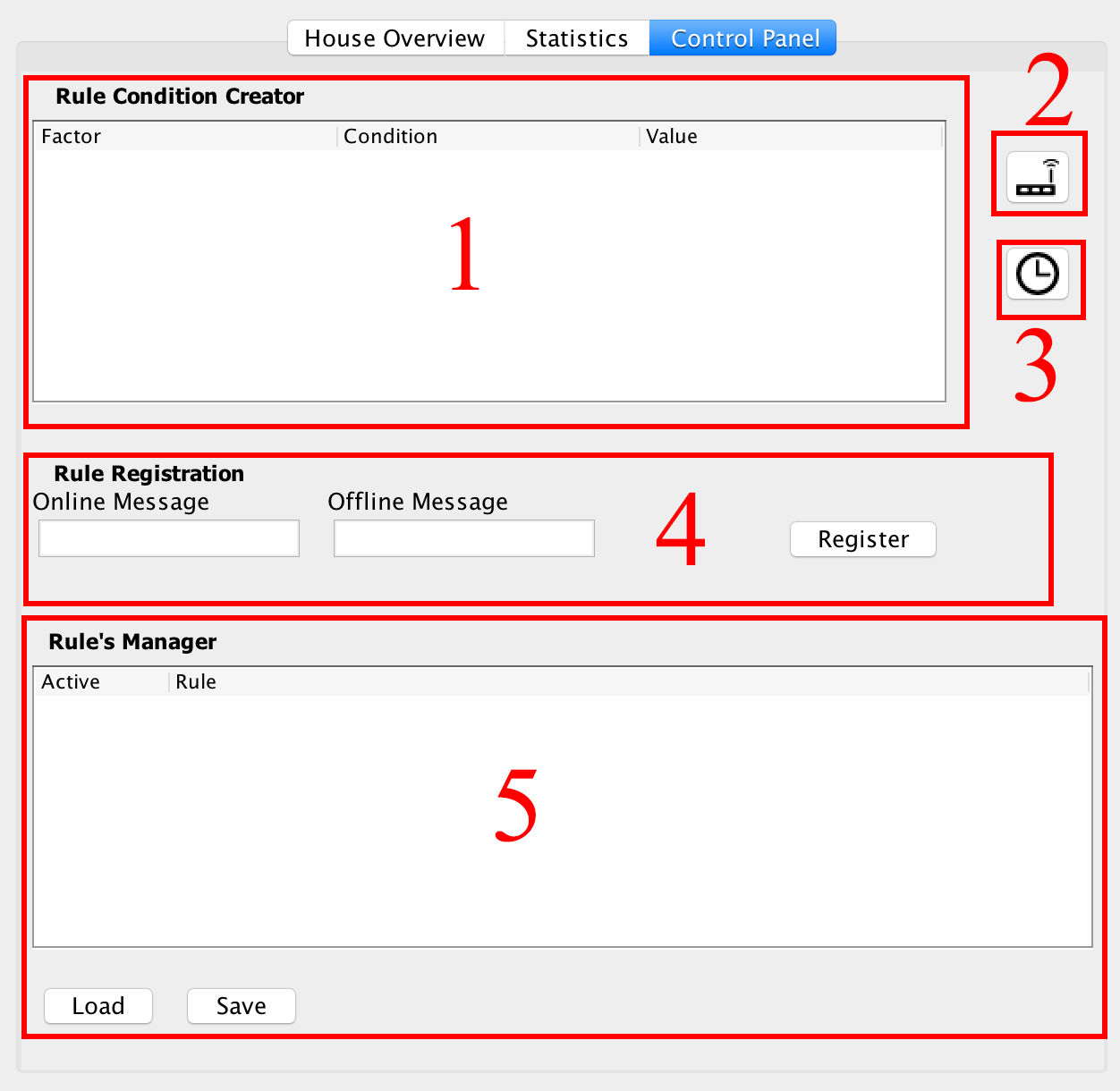


Figura 8. Tab de controlo

# Conclusões e Trabalho Futuro

Com este trabalho pudemos aumentar a nossa experiência com a plataforma JADE e cimentar alguns conceitos sobre agentes e sistemas multi-agente lecionados nas aulas. Apreciámos também o facto de podermos ter expandido os objetivos base propostos e enquadrá-los num contexto real, como foi o caso de uma casa inteligente.

Tendo em conta as limitações de tempo e recursos disponíveis, é óbvia a impossibilidade de que a qualidade e robustez da aplicação desenvolvida se equipare à de um projeto de real de domótica. Existem muitas possibilidades de expansão das quais destacamos:

* a integração de mais sensores;
* a utilização do padrão Observador para melhor separar a GUI de um agente, permitindo simular múltiplos ecrãs para um mesmo agente de Interface;
* Adição de mais tipos de condições no perfil de automação;
* a sugestão de perfis de automação baseada na recolha de informação de estilo de vida;

No entanto, o grupo conseguiu cumprir em tempo útil todos os objetivos a que se propôs, sendo a maioria destes adições proactivas aos presentes no enunciado. Mais particularmente, a implementação de perfis de automação permite exemplificar interações de uma aplicação de domótica. A integração com sensores reais foi também um passo bastante importante no desenvolvimento do trabalho. Ao introduzir um sensor Arduino no nosso sistema podemos reforçar a nossa prova de conceito e mostrar que a nossa aplicação estaria preparada para funcionar num ambiente real.

No final deste projeto orgulhamo-nos no trabalho desenvolvido e avaliamos toda a experiência a que fomos expostos muito positivamente. Acreditamos que o produto final pode facilmente ser usado como um primeiro protótipo de uma verdadeira solução de inteligência na habitação ou como um simulador interativo de configurações de sensores a serem instalados em casas cliente.

# Anexos

## Tutorial da Aplicação

Este tutorial corresponde a uma explicação da utilização geral da aplicação com alguns exemplos das suas funcionalidades. Eis o aspeto inicial da aplicação no ecrã inicial.

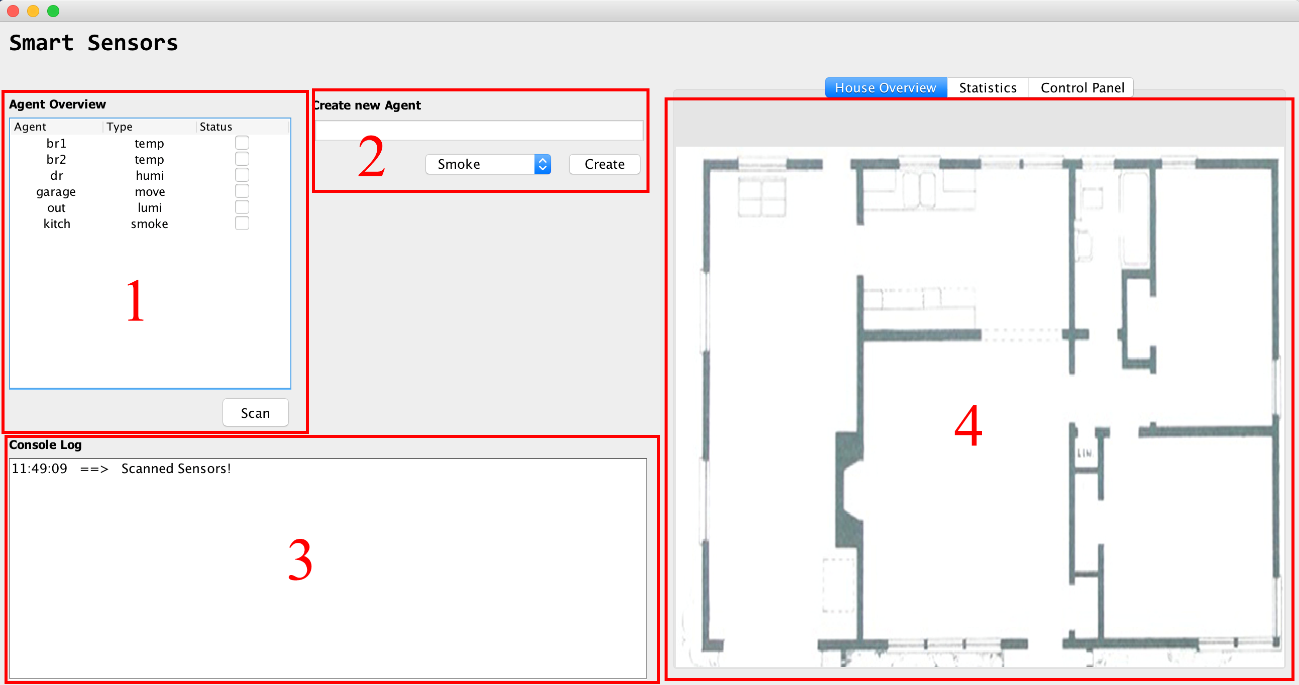


Figura 9. Ecrã inicial da aplicação (layout)

Ao inicializar a aplicação é feito um scan aos sensores disponíveis e estes são listados na tabela Agent Overview (1). No início os sensores encontram-se todos *offline* e o utilizador pode escolher quais os que pretende colocar *online* para fazer a monitorização. O utilizador pode também criar novos sensores em Create new Agent (2) que serão listados em 1 após um novo scan. Temos também uma Console Log (3) onde é possível ver um log dos eventos que ocorrem durante a utilização da aplicação. Em (4) termos disponíveis 3 tabs distintas: na House Overview onde podemos ver a planta da casa e os sensores em cada divisão e os seus valores em tempo real, a tab Statistics onde podemos ver os gráficos da variação dos valores dos sensores.

Veja a imagem 10 na página seguinte.

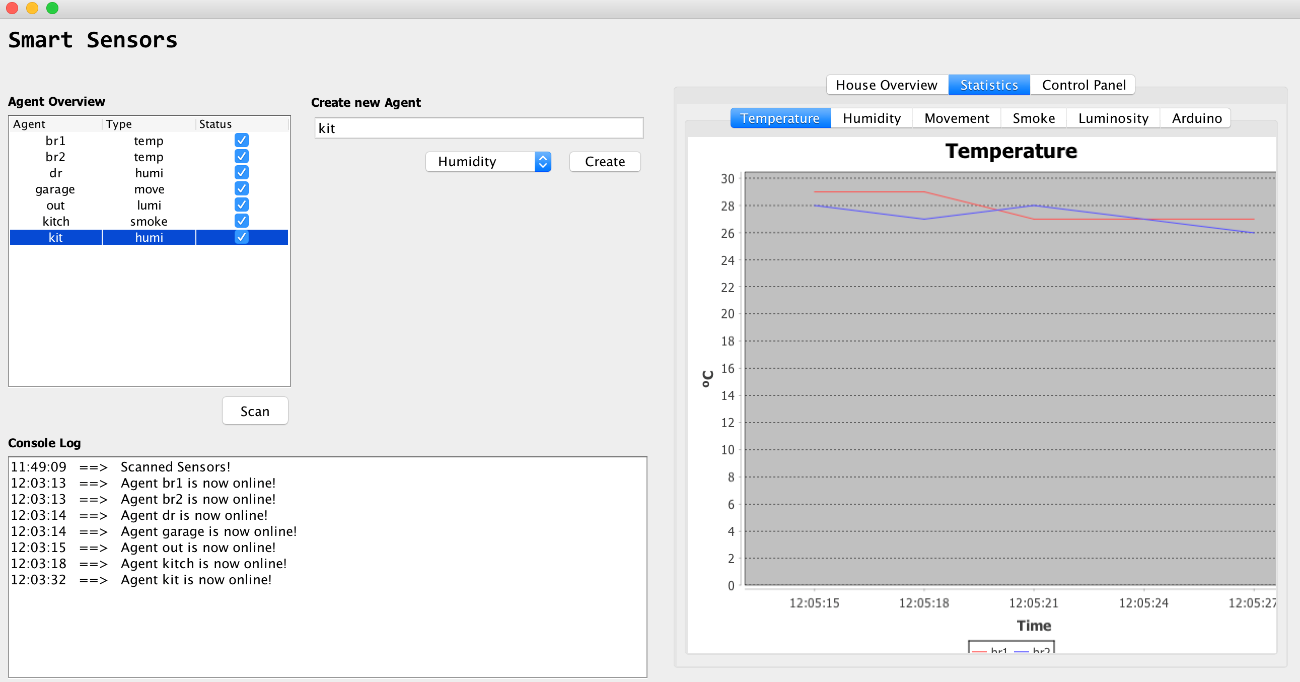


Figura 10. Tab estatísticas

A tab Control Panel onde é possível definir regras, em 1 vemos as condições para a criação da nova regra, em 2 é possível adicionar uma condição conforme o sensor selecionado, em 3 uma condição temporal, em 4 definimos o que fazer quando as condições forem verdadeiras (Online Message) ou falsas (Offline Message) e 5 temos as regras já criadas, que podem ser ativadas ou desativadas e apagadas, podemos também carregar um perfil já criado ou gravar o perfil atual de regras.

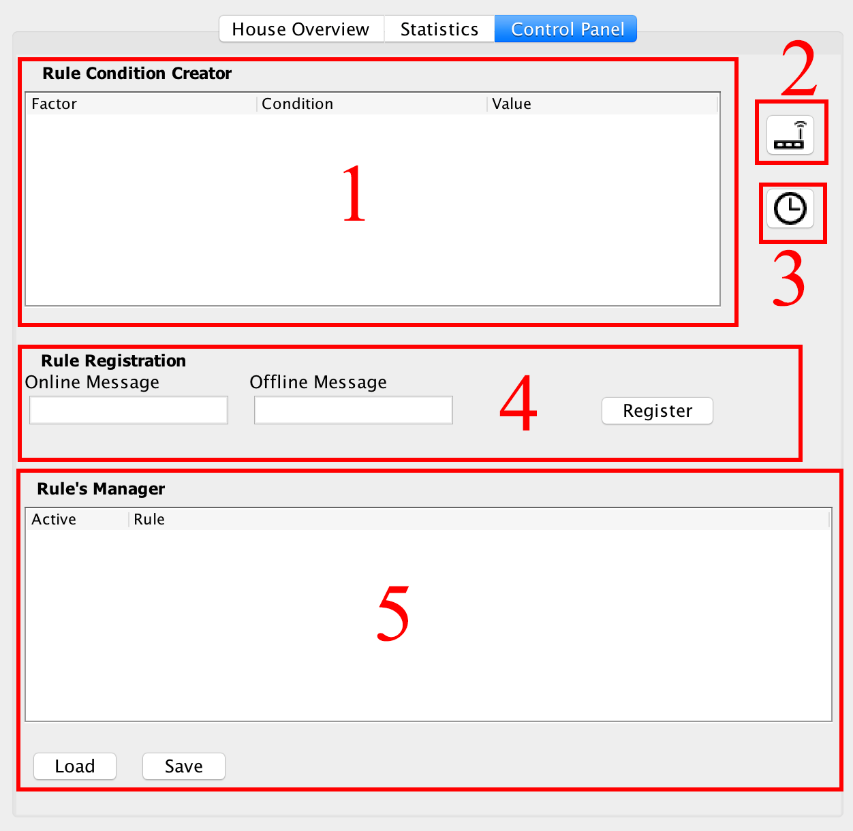


Figura 11. Tab de controlo

Por último, nas figuras seguintes podemos ver o que acontece após colocar os sensores *online* e criação de um novo sensor.

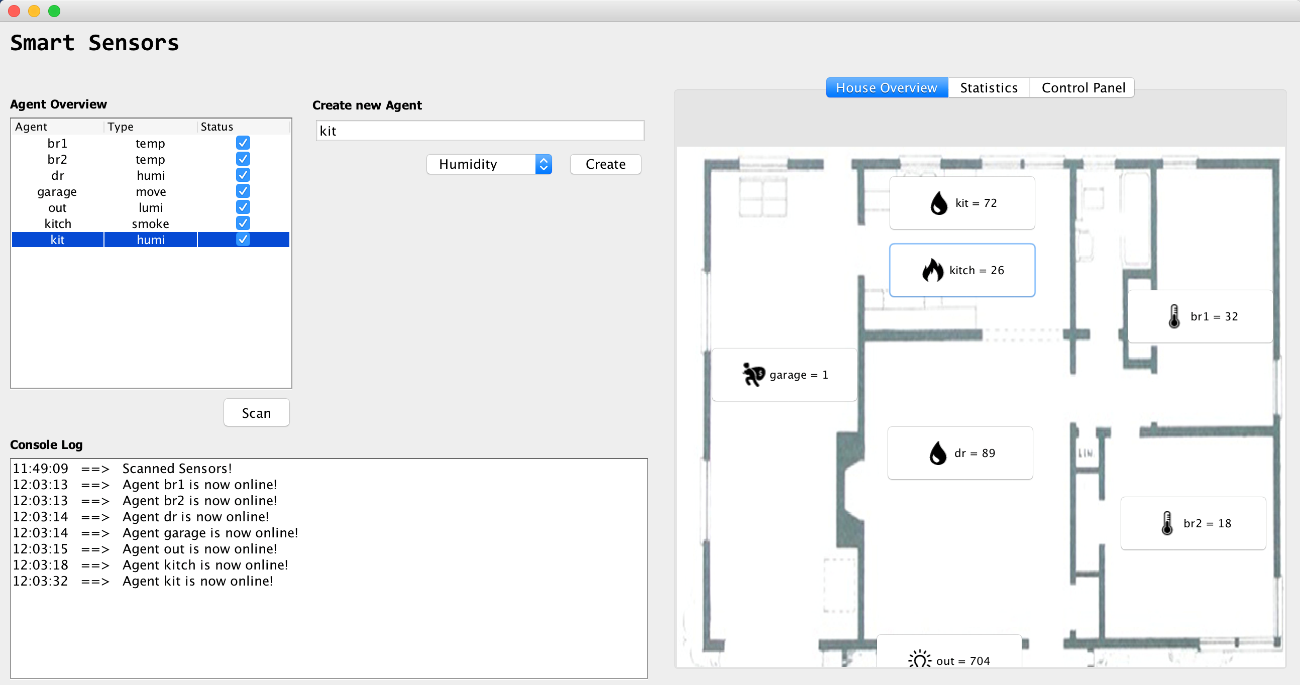


Figura 12. Colocar os sensores online e criação de um novo sensor

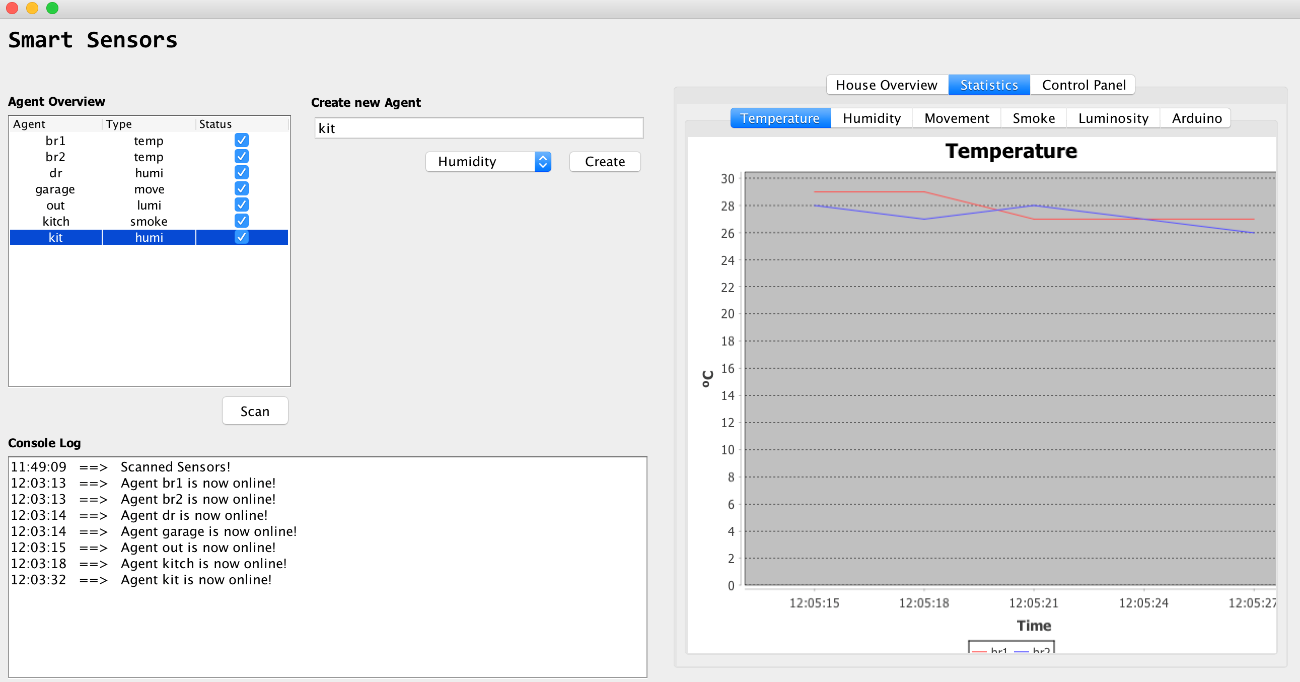


Figura 13. Gráfico dos sensores de temperatura

Na próxima página encontram-se três imagens que demonstram como se efetua a criação de regras na aplicação.

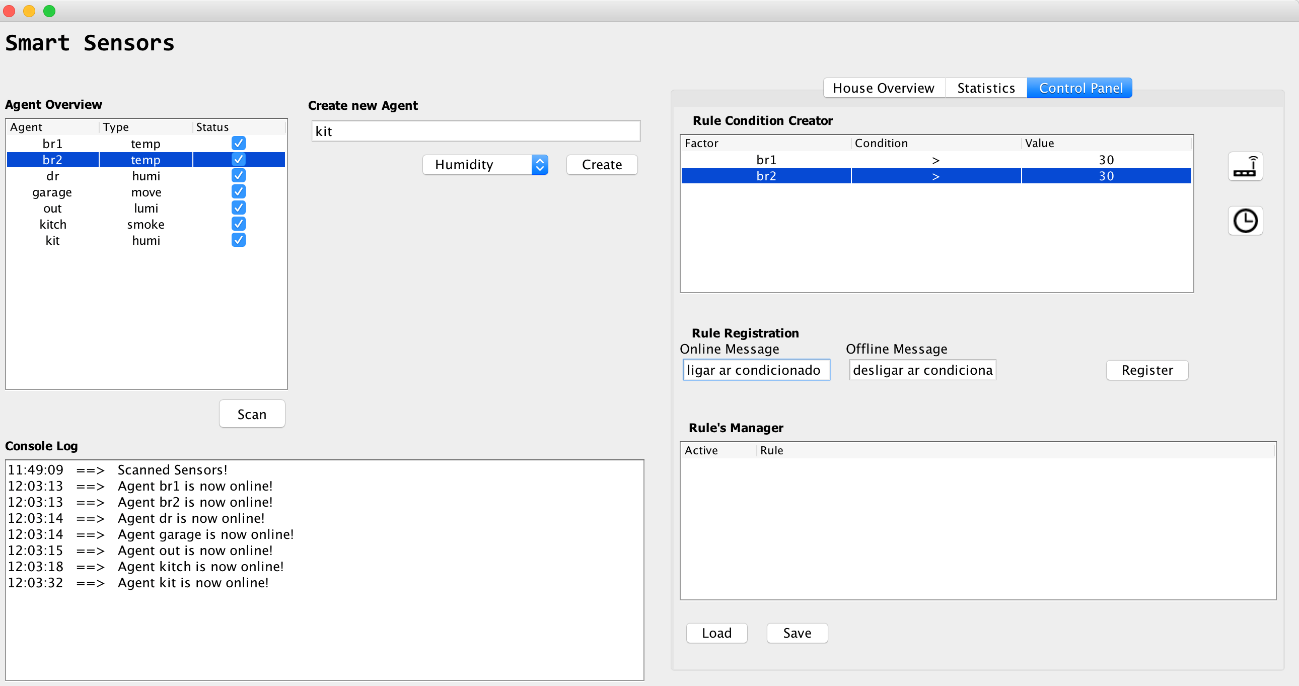


Figura 14. Criação de regras para um sensor

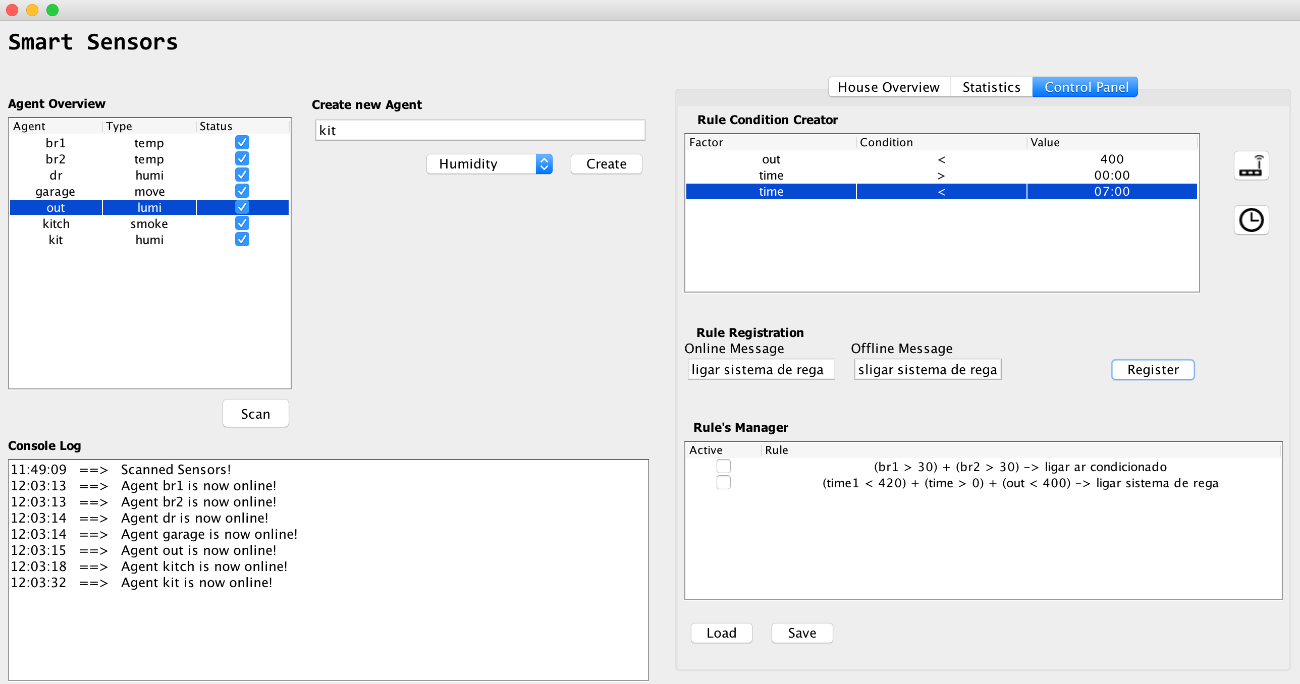


Figura 15. Registo das regras (sensores + temporais)

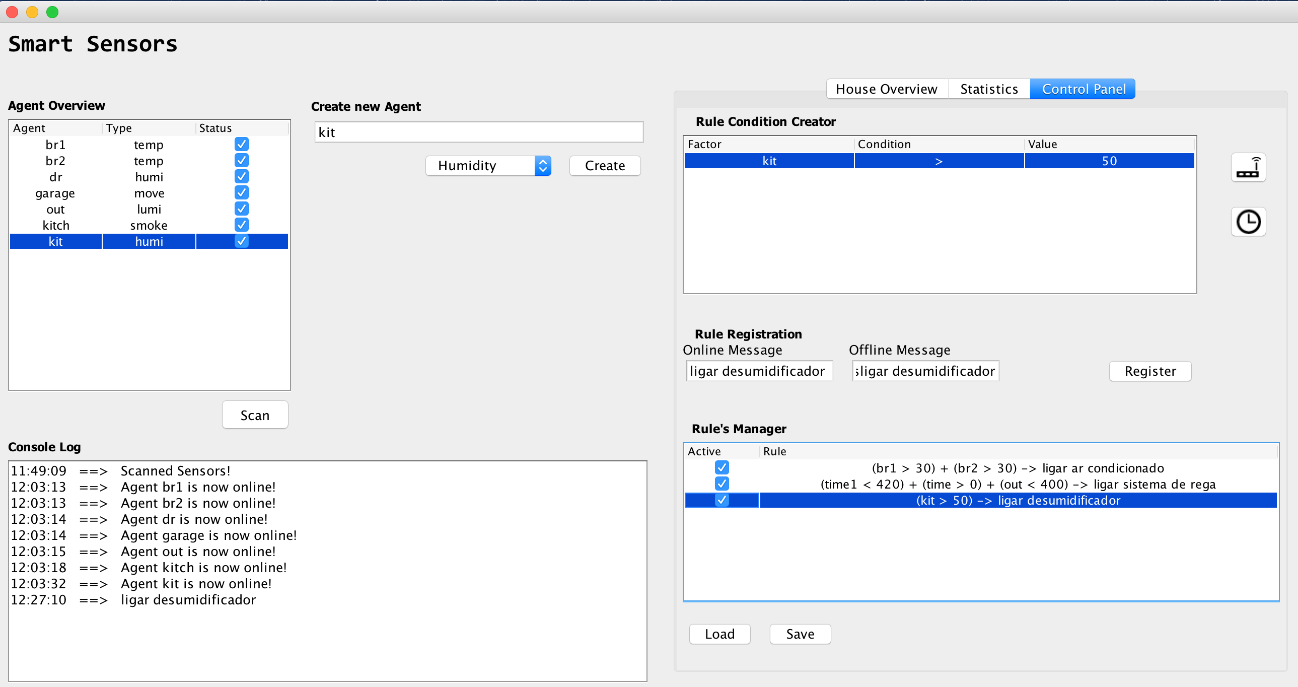


Figura 16. Ativação das regras

## Código do Arduino

const int sensorPin = A0;

const int ledPin = 12;

const int ledPin2 = 13;

int data = 0;

void setup() {

Serial.begin(9600);

pinMode(ledPin, OUTPUT);

pinMode(ledPin2, OUTPUT);

digitalWrite(ledPin2,HIGH);

}

void loop() {

if(Serial.available() > 0){ //verifica se existe comunicação com a porta serial

data = Serial.read();//lê os dados da porta serial

switch(data){

case 101:

digitalWrite(ledPin2,LOW); //desliga o led verde

digitalWrite(ledPin,HIGH); //liga o led vermelho

break;

case 102:

digitalWrite(ledPin,LOW); // desliga o led vermelho

digitalWrite(ledPin2,HIGH); //liga o led verde

break;

}

}

// leitura do sensor

int sensorVal = analogRead(sensorPin);

// conversão para volts

float voltage = (sensorVal / 1024.0) \* 5.0;

// conversão para graus Celsius

float temperature = (voltage - .5) \* 100;

Serial.print(temperature);

Serial.println();

delay(3000);

}