Cadastro automático de novos dispositivos EPOSMoteIII incluídos numa rede IoT

Cesar Smaniotto Júnior[1], **Gilney Nathanael Mathias**[1], **Luis Gustavo Lorgus Decker**[1]

[1] Departamento de Informática e Estatística
Universidade Federal de Santa Catarina (UFSC) – Florianópolis, SC – Brasil

Abstract. *The present project aimes to solve the registration issue of new smart objects in a Internet of Things network, in a way that new aplications can discover the existing smart objects in the network and control them. The registration and command submission is done with a message exchange protocol developed, using the zigbee radio network.*

Resumo. *O presente trabalho solucionou o problema do cadastramento de novos objetos adicionados a uma rede IoT, de modo que outras aplicações possam descobrir os objetos existentes e controlá-los. O cadastramento e envio de comandos é feito a partir do envio de mensagens pela rede zigbee, seguindo um padrão de mensagens desenvolvido pela própria equipe.*

1. Introdução

Um objeto inteligente é um objeto que expande a sua interação não somente com pessoas mas também com outros objetos inteligentes. Ele pode se referir não somente a interação com objetos físicos do mundo real, mas também com objetos virtuais, participantes de um ambiente de computação. Ao criar um objeto inteligente, é natural que queiramos centralizar as informações provenientes dos diversos objetos inteligentes participantes de uma rede de internet das coisas. Visando esta centralização, é necessário que um objeto inteligente informe suas funções e capacidades de sensoriamento a algum servidor central, como uma central de domótica.

Para efetuar esta troca de informações e persistir os dados referentes a um objeto inteligente dentro de uma rede, um sistema de cadastro foi desenvolvido. Este sistema de cadastro recebe as informações enviadas pelos objetos inteligentes, modelados como aplicações utilizando o sistema operacional EPOS [Fröhlich and Schröder-Preikschat 1999] e utilizando como infraestrutura um conjunto de EPOSMoteIII+ 2.0.

2. Limitações do trabalho

Atualmente, existe um aplicativo Android, resultado do trabalho [Nascimento et al. 2015] capaz de buscar os objetos inteligentes pertencentes a um usuário, e montar uma interface gráfica para controle com base em especificações do dispositivo Android. A solução proposta por nossa equipe criou padrões de mensagens próprios para lidar com o cadastramento do objeto e seu controle, visto que são diferentes da solução existente no trabalho de [Nascimento et al. 2015], não é possível integrar diretamente a solução implementada com o aplicativo Android.

3. Funcionamento básico

O cadastramento do objeto no banco de dados é efetuado após sua inicialização. O objeto envia por broadcast uma mensagem informando o identificador do dispositivo, que é interceptada pelo gateway. O gateway repassa ao servidor, que verifica se o objeto com aquele id já foi previamente cadastrado. O servidor responde a solicitação do gateway, e caso a inserção do objeto tenha sido feita, o gateway finaliza a interação com o dispositivo. Caso contrário, o gateway solicita que o objeto envie as informações necessárias para o cadastramento. O dispositivo troca várias mensagens com o gateway, e após a última ser recebida, o gateway as repassa para o servidor que irá decodificar e salvar no banco de dados o objeto inteligente, seus serviços e respectivos parâmetros. A Figura 1 apresenta o diagrama de interação, que explicita as diversas trocas de mensagens entre os atores do sistema para realização do cadastramento do dispositivo. Quando o emissor envia uma mensagem, ele aguarda o receptor confirmar o recebimento para enviar as próximas.

As interações para execução de comandos no objeto inteligente ocorrem de forma similar, com as mensagens partindo do servidor, passando pelo gateway até chegar ao dispositivo. No destino final, as mensagens são decodificadas, o valor dos parâmetros são definidos e o serviço é então executado. Caso o serviço seja a requisição de leitura de algum parâmetro, então uma mensagem contendo o valor do parâmetro é enviada pelo objeto inteligente até chegar ao servidor.

4. Aplicações desenvolvidas

Foi necessário o desenvolvimento de três aplicações distintas, uma sendo executada nos dispositivos, para o envio das mensagens de cadastramento e de resposta aos comandos, uma executando no gateway para intermediar a comunicação entre dispositivo inteligente e servidor e por fim, uma hospedada no servidor conectado ao gateway, que decodifica as mensagens de cadastro e de resposta de comandos. As subseções a seguir detalham as especificações de cada aplicação.

4.1. Objeto inteligente

O objeto inteligente, aqui implementado em um EPOSMoteIII+ 2.0, se constitui de um conjunto de serviços, estes por sua parte compostos de um conjunto de parâmetros. O objeto inteligente é responsável pela administração da atualização dos seus parâmetros, tanto para atuar no caso de escrita de um parâmetro ou na própria atualização de um parâmetro provido por um sensor.

Um objeto inteligente é capaz de atualizar o valor de um de seus parâmetros, desencadeando assim uma função associada a tal parâmetro que execute as funcionalidades associadas a atualização deste parâmetro. Quando o parâmetro for de leitura, esta função associada será responsável por atualizar o dado deste parâmetro com o dado mais atualizado, seja ele proveniente de algum sensor ou algum estado interno do objeto inteligente.

4.2. Gateway

O gateway, também implementado em um EPOSMoteIII+ 2.0, é responsável por intermediar a comunicação entre os objetos inteligente e o servidor onde está hospedado o banco de dados e de emissão de comandos.

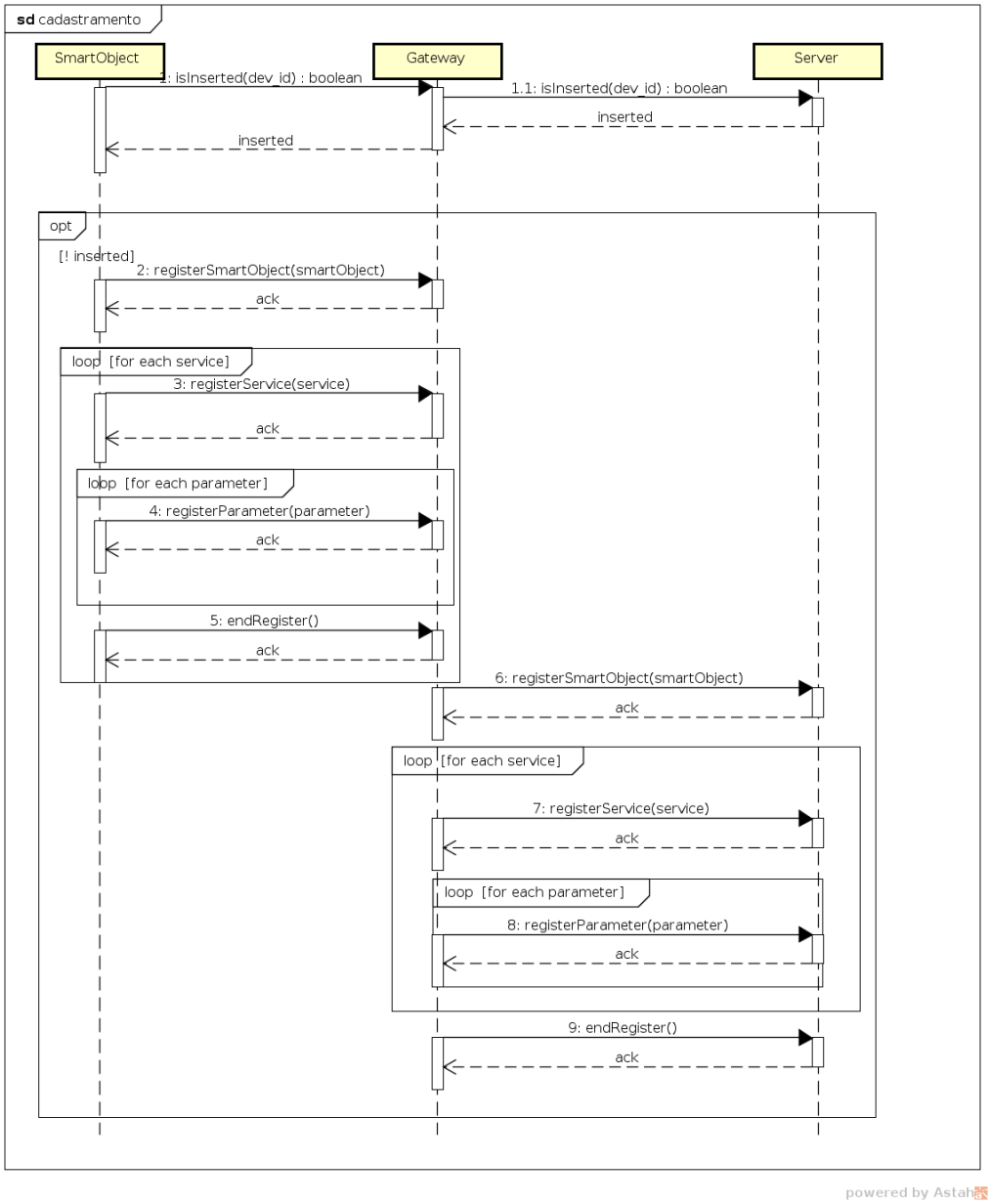

Figura 1. Diagrama de interação do cadastramento do objeto inteligente no banco de dados

4.3. Protocolo de Comunicação

Para realizar a comunicação de cadastro e controle dos objetos inteligentes, foi desenvolvido um protocolo de mensagens. Este protocolo precisou ser desenvolvido devido ao limite de transmissão dos EPOSMote.

Neste protocolo, os parâmetros são enviado um por vez, junto com o id do objeto inteligente que gerou a mensagem. Esta fragmentação permite que enviemos objetos de valor arbitrário para o cadastro, respeitando o limite máximo de bytes por mensagem do rádio zigbee do EPOSMoteIII+ 2.0.

No mesmo modelo, as mensagens de comando que são enviadas aos objetos inteli-

gentes também seguem este protocolo, aproveitando o mesmo mecanismo de serialização e deserialização.

4.4. Servidor

A aplicação responsável por interagir com o gateway e o sistema de banco de dados foi implementada em Python. Essa escolha deve-se ao fato de ser uma linguagem dominada pelos membros do grupo além de possuir bibliotecas que tornem mais fácil a tarefa de manipular as portas seriais do computador e realizar o mapeamento entre os objetos do modelo e as tabelas do banco de dados, como é o caso do pySerial[1] e SQLAlchemy[2] respectivamente.

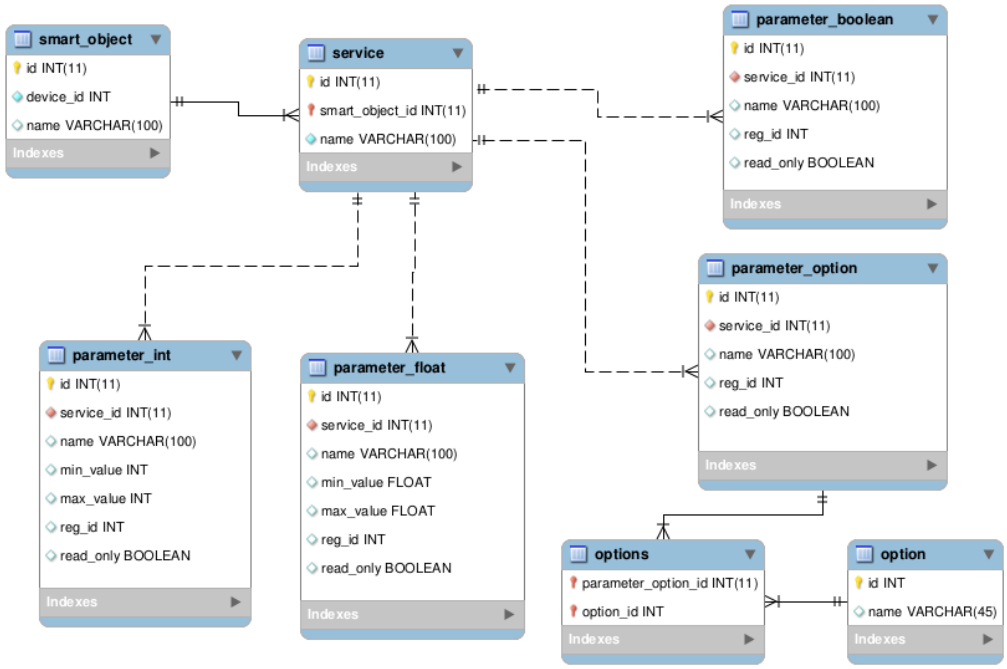

Figura 2. Modelagem do banco de dados da aplicação

Para facilitar a instalação das dependências, foi utilizado a ferramenta virtualenv[3], que cria um ambiente Python isolado para a aplicação. Isto traz duas vantagens: não é necessário ter permissão de root para a instalação das dependências do projeto e por ser um ambiente isolado, não há conflito entre versões de dependências instaladas anteriormente, que servem outros projetos em Python.

Em termos de implementação, a classe SerialPortManager é responsável por ler os bytes oriundos da portal serial conectada ao gateway, identificar o início e fim de uma mensagem e repassá-la para a classe adequada que trata o tipo de mensagem recebida. Foi utilizado o padrão de projeto Observador/Observado, onde os observadores são as classes que tratam as mensagens e o observado é a classe SerialPortManager, que notifica os observadores sempre que uma nova mensagem é recebida. São três grupos de mensagens

[1] https://pythonhosted.org/pyserial/
[2] https://www.sqlalchemy.org/
[3] https://virtualenv.pypa.io/en/stable/

existentes, mensagens relacionadas ao cadastramento do objeto, tratadas pela classe RegisterManager; relacionadas à execução dos serviços, tratadas pela classe CommandManager e mensagens de debug enviadas pelo gateway, tratados pela classe DebugManager.

A Figura 2 apresenta a modelagem do banco de dados da solução implementada. Optou-se por criar uma tabela para cada tipo de parâmetro associado aos serviços, de modo a tornar o banco de dados normalizado.

5. Conclusão

Uma rede de objetos inteligentes exige que várias tecnologias de diferentes níveis sejam utilizadas em conjunto, de maneira harmoniosa, para que as funções providas por tais objetos sejam realmente úteis e utilizáveis. As especificações e ferramentas desenvolvidas durante este trabalho podem ser utilizadas e estendidas, aumentando ainda mais a sua abrangência e permitindo o desenvolvimento de objetos inteligentes cada vez mais populares e acessíveis.

Referências

Fröhlich, A. A. and Schröder-Preikschat, W. (1999). EPOS: an object-oriented operating system. In *Object-Oriented Technology, ECOOP'99 Workshop Reader, ECOOP'99 Workshops, Panels, and Posters, Lisbon, Portugal, June 14-18, 1999, Proceedings*, page 27.

Nascimento, E. G., Crocomo, B. M., and Cancian, R. L. (2015). Geração automática de guis para objetos inteligentes em dispositivos móveis. *Anais do Computer on the Beach*, pages 199–208.