Uma Abordagem Dinâmica para Descoberta de Recursos na IoT Explorando Processamento Semântico*

Renato Dilli[2], Huberto Filho[2], Maurício de Azevedo[1], Gerson Cavalheiro[2], Ana Marilza Pernas[2], Adenauer Yamin[1,2]

[1]Universidade Católica de Pelotas (UCPel)
Rua Gonçalves Chaves, 373 - Centro - CEP 96015-560 – Pelotas - RS - Brasil

[2]Universidade Federal de Pelotas (UFPel)
Rua Gomes Carneiro, 1 - Centro - CEP 96010-610 – Pelotas – RS – Brasil

{renato.dilli, hkaiser, gerson.cavalheiro, marilza, adenauer}@inf.ufpel.edu.br, mmadrugadeazevedo@gmail.com

Abstract. *The Internet of Things considers a scenario in which resources dynamically enter and exit the computing environment. The discovery procedure in this scenario, whose quantity of resources is increasing, characterizes an open research challenge. This article presents EXEHDA-RD, a discovery service integrated with the EXEHDA middleware that employs Semantic Web technologies and the REST architectural standard. The main components of the architecture were implemented considering characteristics relevant to IoT, such as: presence control, self-discovery and increase of expressiveness in the queries. The results obtained are promising and point to the continuity of study and research efforts.*

Resumo. *A Internet das Coisas considera um cenário no qual recursos dinamicamente entram e saem do ambiente computacional. O procedimento de descoberta neste cenário, cuja quantidade de recursos é crescente, caracteriza um desafio de pesquisa em aberto. Este artigo apresenta o EXEHDA-RD, um serviço de descoberta de recursos integrado ao middleware EXEHDA que emprega tecnologias de Web Semântica e o padrão arquitetural REST. Os principais componentes da arquitetura foram implementados considerando características relevantes para IoT, tais como: controle de presença, auto-descoberta e aumento de expressividade nas consultas. Os resultados obtidos se mostram promissores e apontam para continuidade dos esforços de estudo e pesquisa.*

1. Introdução

A IoT (*Internet of Things*) vem ganhando destaque como o novo paradigma de evolução da Internet [Perera et al. 2014]. Esta evolução deve-se ao fato que a IoT preconiza a ideia do tudo conectado, ou seja, qualquer "coisa" pode possuir sensores capazes de coletar informações as quais podem ser compartilhadas através da Internet.

A Internet das Coisas trouxe vários desafios a serem superados, entre eles, questões de identificação dos inúmeros dispositivos levando a uma necessidade de transição para IPv6 e a descoberta dos dispositivos e seus recursos [Gubbi et al. 2013].

*O presente trabalho foi realizado com apoio do Programa Nacional de Cooperação Acadêmica da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – CAPES/Brasil.

A Descoberta de Recursos é um aspecto importante na IoT devido ao elevado número de dispositivos distribuídos dinamicamente e com características heterogêneas, o que constitui um desafio na disponibilização de seus recursos. O sistema computacional precisa ter conhecimento dos recursos disponíveis e quais atendem as necessidades de uma determinada aplicação, para que possam ser alocados com a mínima interferência do usuário [Marin-Perianu et al. 2005]. Resumidamente, pode-se dizer que o processo de descoberta consiste na localização de recursos e serviços existentes que são relevantes para uma determinada requisição baseado na descrição de sua semântica [Klusch 2014].

Neste sentido, os principais desafios da Descoberta de Recursos, na perspectiva da Internet das Coisas, são os seguintes [Al-Fuqaha et al. 2015][Miorandi et al. 2012]: (i) Heterogeneidade de dispositivos; (ii) Escalabilidade; (iii) Auto-organização e (iv) Interoperabilidade semântica.

Com a premissa de atender estes desafios, o objetivo geral deste trabalho é a concepção de um mecanismo para descoberta de recursos, visando contribuir com os esforços de pesquisa do LUPS (*Laboratory of Ubiquitous and Parallel Systems*) da Universidade Federal de Pelotas em Internet das Coisas. Deste modo, este trabalho irá remodelar a arquitetura de software existente no serviço de descoberta do *middleware* EXEHDA para dar suporte à IoT.

O artigo está organizado da seguinte forma: na Seção 2 é apresentada a arquitetura e as funcionalidades do serviço de descoberta de recursos EXEHDA-RD, na Seção 3 a validação do EXEHDA-RD através de Cenário de Uso, na Seção 4 são discutidos os Trabalhos Relacionados e na Seção 5 estão as Considerações Finais.

2. EXEHDA-RD: Arquitetura e Funcionalidades

O EXEHDA [Davet 2015] [Lopes et al. 2014] é o *middleware*, foco deste trabalho, responsável por prover a infraestrutura computacional básica para a IoT. Este ambiente computacional é constituído por células de execução, nas quais os dispositivos computacionais são distribuídos. Cada célula é constituída dos seguintes componentes: (i) EXEHDABase, o elemento central da célula, sendo responsável por todos serviços básicos e constituindo referência para os demais elementos; (ii) o EXEHDANodo que corresponde aos dispositivos computacionais responsáveis pela execução das aplicações; (iii) o EXEHDANodo móvel, um subcaso do anterior, que corresponde aos dispositivos tipicamente móveis que podem se deslocar entre as células do ambiente ubíquo, (iv) o EXEHDABorda, responsável por fazer a interoperação entre os serviços do *middleware* e os diversos tipos de *gateways*; e (v) o EXEHDAGateway, que consiste no elemento responsável por setorizar pontos de coleta e/ou atuação distribuídos, disponíveis no meio físico, realizando a interação destes com os outros componentes do *middleware*.

O EXEHDA-RD (*Resource Discovery*) é um novo serviço para descoberta de recursos proposto para o *middleware* EXEHDA. Sua arquitetura de software foi modelada considerando a dinamicidade em que os recursos entram e saem do ambiente, portanto, controla a presença de dispositivos, inclusive com baixo poder computacional, conectados à gateways. O aumento da expressividade na descrição e consulta por recursos foi provida através de tecnologias de Web Semântica com intuito de potencializar a descoberta de recursos no ambiente. A arquitetura de software do EXEHDA-RD possui três componentes distintos: (CD) Componente Diretório, (CR) Componente Recurso e (CC)

Componente cliente (vide Figura 1).

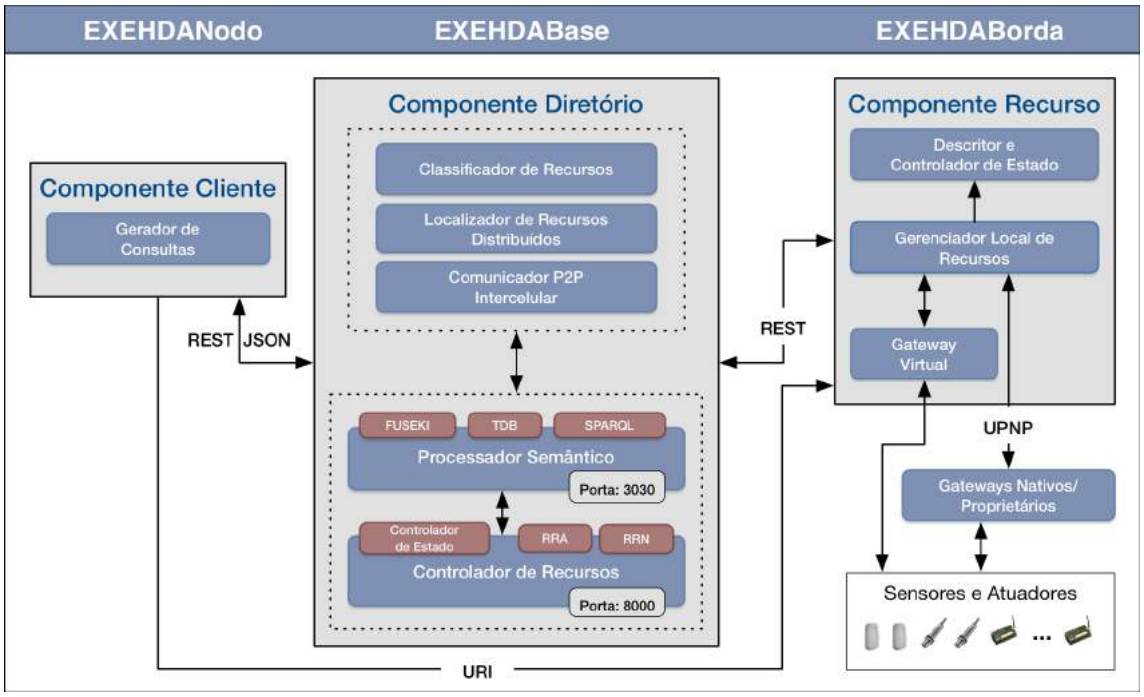

Figura 1. Arquitetura de software do EXEHDA-RD

2.1. CC - Componente Cliente

O componente CC é responsável pelo disparo da consulta por recursos, empregando critérios pertinentes as demandas do cliente. O CC é composto do módulo Gerador de Consultas.

2.2. CR - Componente Recurso

O Componente Recurso (CR) é responsável por notificar a presença do recurso. Isto é feito por troca de mensagens com o CD, dentro de um intervalo de tempo. Quando o recurso ingressa pela primeira vez em uma célula, o CR envia a descrição dos seus recursos para o CD da célula onde está localizado o recurso. O CD gera uma instância do recurso na ontologia local. O CR é composto pelos módulos Descritor e Controlador de Estado e Gerenciador Local de Recursos.

O módulo Descritor e Controlador de Estado é responsável por enviar mensagens ao CD com as características do recurso e anunciar sua presença no ambiente. O módulo Gerenciador Local de Recursos tem por objetivo realizar a descoberta de gateways através do protocolo UPnP.

O CR é ativado em nodos EXEHDABorda, cuja finalidade é anunciar a presença dos dispositivos confinados que estão conectados aos gateways da célula.

2.3. CD - Componente Diretório

O Componente Diretório, instanciado no EXEHDABase é formado por cinco módulos:

1. Controlador de Recursos: o qual tem 5 objetivos; (i) realizar a manutenção de estado dos recursos. Os estados possíveis são: Ativo, Inativo e Negado; (ii) gerenciar o Repositório de Recursos Ativos (RRA); (iii) gerenciar o Repositório de Recursos Negados (RRN); (iv) receber as consultas dos Componentes Cliente; (v) receber as mensagens enviadas pelos Componentes Recurso. O *lease* é um intervalo de tempo que é gerenciado pelo Controlador de Recursos. O Componente Recurso precisa renovar o *lease* do recurso localizado no CD periodicamente, caso contrário será removido do Repositório de Recursos Ativos (RRA).
2. Processador Semântico: responsável pelo processamento das consultas em linguagem SPARQL. O processamento semântico segue o padrão arquitetural REST com a integração do Apache Jena Fuseki. O componente TDB realiza a persistência de dados e o motor de inferência foi configurado para utilizar o raciocinador genérico do Apache Jena. É possível definir regras adicionais para aumentar a expressividade nas consultas.
3. Comunicador P2P Intercelular: através deste módulo o mecanismo de descoberta processa a consulta de clientes em outros células do ambiente utilizando protocolo P2P.
4. Localizador de Recursos Distribuídos: responsável por receber e organizar as respostas das consultas realizadas em outras células, recebidas pelo módulo Comunicador P2P Intercelular.
5. Classificador de Recursos: tem por finalidade realizar a classificação dos recursos mais adequados à solicitação do cliente.

2.4. Fluxo Operacional

Todo recurso possui um único UUID (*Universal Unique Identifier*) que é anunciado pelo CR ao CD, junto com as demais características do recurso. Esta mensagem é enviada a cada 30 segundos ao CD. Ao receber esta mensagem, o módulo Controlador de Recursos do CD realiza as seguintes operações para realizar o gerenciamento deste recurso:

- Na primeira mensagem recebida pelo CD, o Processador Semântico dispara o método de inserção de um novo objeto na ontologia e o Controlador de Recursos insere as informações do recurso no Repositório de Recursos Negados (RRN);
- Quando o Componente Diretório recebe a mensagem em um segundo momento, o mesmo se encontrando com estado Negado na Ontologia, ele ignora todos os métodos;
- Se o Componente Diretório recebeu a mensagem e o recurso já foi autorizado pelo administrador, o Processador Semântico realiza o disparo do método responsável em modificar o estado do recurso para Ativo na Ontologia em linguagem SPARQL;
- Confirmada a atualização de mudança de estado na ontologia, o módulo Controlador de Recursos do Componente Diretório executa o método responsável por inserir as informações do recurso na base de dados relacionais, no Repositório de Recursos Ativos (RRA);
- Ao receber uma mensagem, e o recurso em questão já estiver com seu estado configurado como Ativo na ontologia e o mesmo já estiver presente no RRA, o módulo Controlador de Recursos se responsabiliza em disparar o método responsável em atualizar a data de última atualização deste recurso no RRA.

- O módulo Controlador de Estado é responsável por verificar de 30 em 30 segundos se existem recursos no Repositório de Recursos Ativos (RRA) que não tenham sido atualizados a mais de 60 segundos da data e hora atual. Este intervalo de tempo, inicialmente configurado para 60 segundos pode ser modificado passando-se um novo valor para o atributo *lease*, em segundos, na inicialização do serviço. Ao ser executado o serviço, todos os recursos que não corresponderem aos requisitos terão seu estado modificado para Inativo na Ontologia e são removidos do Repositório de Recursos Ativos (RRA), por meio de um método executado pelo módulo Processador Semântico do Componente Diretório em linguagem SPARQL.

3. EXEHDA-RD: Cenário de Uso

O cenário proposto tem por objetivo explorar as funcionalidades dos Componentes Cliente, Diretório e Recurso, quando atuando no controle da presença de recursos e na qualificação das respostas das requisições através do aumento da expressividade promovido pelo Processador Semântico.

Na validação do EXEHDA-RD foi empregado o emulador CORE [Ahrenholz et al. 2008] para prover uma infraestrutura distribuída contendo vários recursos. Os Componentes Cliente e Recurso foram instanciados dentro do emulador CORE, o Componente Diretório, por exigir mais recursos computacionais ficou instanciado na máquina hospedeira.

Os testes foram realizados em uma máquina virtual Linux com os módulos dos Componentes implementados na linguagem Python, servidor Apache Jena Fuseki, banco de dados Postgres para manutenção dos repositórios RRA e RRN. O Fuseki foi ativado na porta 3030 e o módulo Controlador de Recursos na porta 8000. Com o servidor Apache Jena Fuseki em execução, foi feito o upload da ontologia (Figura 2) e inicializado o módulo Controlador de Estado.

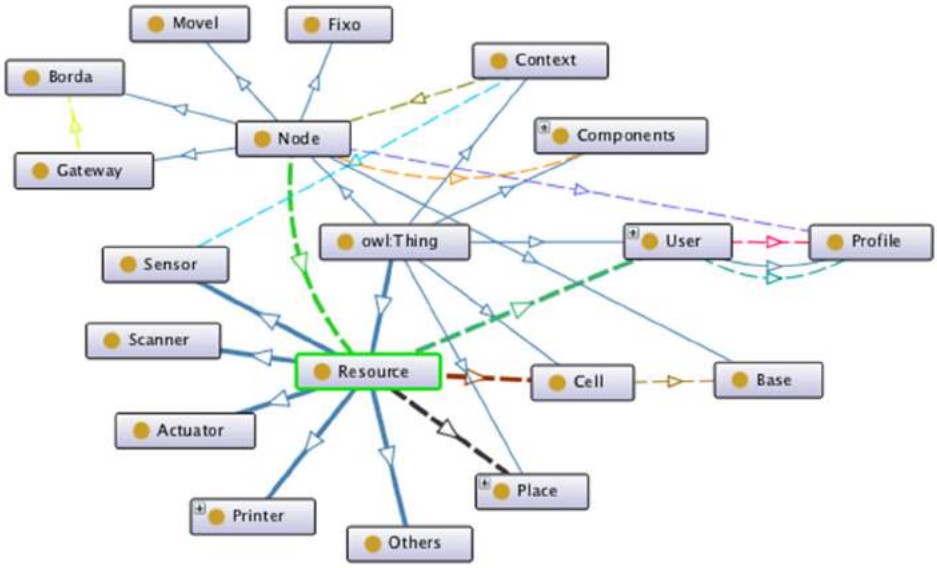

Figura 2. Ontologia desenvolvida para o EXEHDA-RD

Os recursos a serem descobertos no ambiente da IoT gerenciado pelo *middleware*

EXEHDA estão conectados em gateways. Neste cenário estão sendo utilizados Gateways Virtuais, localizados nos EXEHDABorda.

Após a inicialização dos serviços na máquina hospedeira, foram ativados os Componentes Recurso, em nodos do emulador CORE. Este procedimento tem por objetivo simular o Componente Recurso presente em um Gateway Virtual através de mensagens de anúncio no ambiente emulado.

As mensagens enviadas do CR ao CD contém o UUID do recurso e sua URI. As mensagens são enviadas de 30 em 30 segundos utilizando o padrão REST. Este intervalo de tempo pode ser alterado no CR.

Após receber a mensagem do CC, o CD verifica na ontologia a existência do UUID do recurso, caso seja um novo recurso, ele será inserido na ontologia com estado definido como Negado e armazenado no Repositório de Recursos Negados (RRN), Figura 3.

Data Output | Explain | Messages | History

added timesta...	uuid [PK] uuid	status characte...	resource text	gateway text	category text
2017-01-...	4cef01da-bfa4-11e6-a4a6-cec0c932ce12	1	Sensor	GW02	Temperatura
2017-01-...	4cef01da-bfa4-11e6-a4a6-cec0c932ce14	1	Sensor	GW02	Temperatura

Figura 3. RRN - Repositório de Recursos Negados

O administrador deverá liberar os recursos através da interface WEB disponível no endereço http://localhost/denieds/all/ para que os recursos fiquem disponíveis no ambiente. A Figura 4 apresenta o código em SPARQL utilizado na inserção de um novo recurso na ontologia pelo EXEHDA-RD.

```
sparql = SPARQLWrapper("http://127.0.0.1:3030/inf/update")
sparql.setQuery("""
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ont: <http://www.owl-ontologies.com/Ontology1251223167.owl#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>

INSERT DATA {
        ont:""" + data['uuid'] + """ rdf:type ont:""" + data['resource'
            ↪ ] + """, owl:NamedIndividual .
        ont:""" + data['uuid'] + """ ont:Resource_Status 'Negado' .
        ont:""" + data['uuid'] + """ ont:Resource_Type '""" + data['
            ↪ category'] + """' .
        ont:""" + data['uuid'] + """ ont:Resource_Node ont:""" + data['
            ↪ gateway'] + """.
}
""")

sparql.method = 'POST'
sparql.query()
```

Figura 4. Procedimento para inserção de recurso em SPARQL

O procedimento de liberação do recurso pelo administrador envia uma requisição via POST para a URL http://localhost:8000/resources/renew/ com os dados do recurso

selecionado, sendo removido este recurso do Repositório de Recursos Negados (RRN), tendo seu estado modificado para Inativo na ontologia, o bloco de código da Figura 5 caracteriza a troca do estado do recurso na ontologia.

```
    PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
    PREFIX ont: <http://www.owl-ontologies.com/Ontology1251223167.owl#>
    PREFIX owl: <http://www.w3.org/2002/07/owl#>

        DELETE {?resource ont:Resource_Status 'Negado'}
        INSERT {?resource ont:Resource_Status 'Inativo'}

        WHERE { ?resource owl:sameAs ont:%s . }
```

Figura 5. Procedimento para atualização de estado do recurso de Negado para Inativo

Após ser liberado o acesso do recurso pelo administrador, no próximo anúncio realizado pelo CR, o seu estado é modificado para Ativo na ontologia e o recurso é inserido na Repositório dos Recursos Ativos, conforme Figura 6. Assim, toda vez que o CR dispara uma mensagem de anúncio, é atualizada a data da última atualização do recurso no Repositório de Recursos Ativos, data esta que é utilizada pelo serviço de manutenção de estado dos recursos no módulo Controlador de Recursos do CD.

	added timestamp with time zone	updated timestamp with time zone	uuid [PK] uuid	uri character varying(200)
1	2017-01-27 02:29:20.933345-02	2017-01-27 02:29:31.099201-02	4cef01da-bfa4-11e6-a4a6-cec0c932ce14	http://teste.com/sensor/sensor5
*				

Figura 6. RRA - Repositório de Recursos Ativos

As requisições por recursos foram originadas através de um nodo do emulador CORE com o Componente Cliente instanciado. O CC envia a requisição ao CD em formato SPARQL (Figura 7) solicitando sensores que estão sendo gerenciados por gateways que empregam o sistema operacional Unix.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ont: <http://www.owl-ontologies.com/Ontology1251223167.owl#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>

SELECT *
WHERE {
        ?resource rdf:type ont:Sensor .
        ?resource ont:Resource_Node ?node .
        ?node rdf:type ont:Gateway .
        ?node ont:Node_Components ?sysope .
        ?sysope rdf:type ont:Unix .
}
```

Figura 7. Especificação de Consulta em SPARQL

O CD recebe a requisição e o Processador Semântico por sua vez retorna o resultado em formato JSON, que é devolvido ao Componente Cliente.

Para validar a expressividade com o uso de ontologias, foram realizadas duas consultas buscando sensores que estão vinculados a gateways com sistema operacional Unix. Sem o uso de raciocinador a consulta trouxe apenas um gateway com sistema operacional Solaris, enquanto ao ser utilizado o raciocinador, os resultados trouxeram também gateways com sistema operacional Linux, retornando gateways Ubuntu e Debian. As Figuras 8 e 9 apresentam os resultados obtidos diretamente da interface gráfica do servidor Fuseki. O aumento da expressividade através do raciocinador genérico do Apache Jena permitiu maximizar as respostas, retornando gateways com sistema operacional Linux, que é uma subclasse do conceito Unix.

	resource	type	node	sysope
1	ont:4cef01da-bfa4-11e6-a4a6-cec0c932ce02	"Umidade"	ont:GW02	ont:Solaris

Showing 1 to 1 of 1 entries

Figura 8. Resultado de Busca sem Raciocinador

	resource	type	node	sysope
1	ont:4cef01da-bfa4-11e6-a4a6-cec0c932ce01	"Temperatura"	ont:GW01	ont:Ubuntu
2	ont:4cef01da-bfa4-11e6-a4a6-cec0c932ce02	"Umidade"	ont:GW02	ont:Solaris
3	ont:4cef01da-bfa4-11e6-a4a6-cec0c932ce03	"Temperatura"	ont:GW03	ont:Debian

Showing 1 to 3 of 3 entries

Figura 9. Resultado de Busca com Raciocinador

A Figura 10 apresenta o resultado da consulta em JSON enviada ao Componente Cliente, com o raciocinador desativado.


```
{
  "head": {
    "vars": [ "resource" , "type" , "node" , "sysope" ]
  } ,
  "results": {
    "bindings": [
      {
        "resource": { "type": "uri" , "value": "http://www.owl-ontologies.com/
            ↪ Ontology1251223167.owl#4cef01da-bfa4-11e6-a4a6-cec0c932ce02" } ,
        "type": { "type": "literal" , "value": "Umidade" } ,
        "node": { "type": "uri" , "value": "http://www.owl-ontologies.com/
            ↪ Ontology1251223167.owl#GW02" } ,
        "sysope": { "type": "uri" , "value": "http://www.owl-ontologies.com/
            ↪ Ontology1251223167.owl#Solaris" }
      }
    ]
  }
}
```


Figura 10. Resultado JSON s/ Raciocinador

4. Trabalhos Relacionados

Esta seção discute abordagens utilizadas pelos trabalhos relacionados envolvendo a estratégia empregada para manutenção e controle da presença de recursos e o mecanismo de consultas utilizado. Na discussão são priorizados os aspectos relacionados ao gerenciamento da presença de recursos e expressividade, por estes serem aspectos centrais na proposta do EXEHDA-RD.

A arquitetura apresentada por [Talal Ashraf Butt 2012], foi modelada usando web services RESTful, onde cada recurso possui um *leasing* adaptativo que aumenta no momento em que este recurso vem se tornado menos utilizado. Desta forma, um recurso pode retornar em resultado de buscas e estar Inativo, caso tenha um *leasing* muito grande. Os recursos são agrupados de acordo com sua localização, onde um dos recursos desenvolve o papel de gerente do grupo, realizando sub-consultas, onde a complexidade das consultas são afetadas diretamente pelo poder computacional do recurso que atua com gerente do grupo de recursos.

O trabalho de [Cirani et al. 2014] propõe uma arquitetura baseada em P2P, autoconfigurável, escalável para mecanismos de descoberta de recursos e serviços, sem intervenção humana. O protocolo Zeroconf foi utilizado para a descoberta de recursos dentro de redes locais. Ao ingressar no ambiente computacional, dispositivos se anunciam através do Zeroconf e o gateway percebe sua presença. A camada de serviço busca as portas de servidores CoAP em rede, enquanto a camada de descoberta permite descobrir dispositivos gerenciados por um servidor CoAP. Não há preocupação com expressividades, pois são realizadas apenas consultas do método GET para cada nó, retornando os serviços presentes e disponíveis em cada um destes nós.

Em [Andrade Junior et al. 2014] a descoberta de recursos se dá pelo conjunto de componentes utilizando funcionalidades Zeroconf com Bonjour, onde o recurso se conecta diretamente à arquitetura. Uma vez conectado e identificado o recurso, caso ele esteja presente no Servidor de Modelos, um serviço só se torna disponível caso existam modelos na arquitetura para trabalhar com o recurso conectado, e não existe uma forma de controle de *leasing* dos recursos. Questões relacionadas à expressividade não são consideradas no presente trabalho relacionado.

5. Considerações Finais

A principal contribuição do serviço proposto quando confrontado aos modelos existentes é a expressividade na representação e consulta de recursos e sua integração ao EXEHDA, expandindo a capacidade do serviço de descoberta de recursos do *middleware* para localizar dispositivos confinados, limitados, distribuídos, cuja disponibilidade pode variar ao longo do tempo.

O EXEHDA-RD visa garantir a descoberta de recursos no cenário da IoT e assegurar que as informações de estado se mantenham sempre atualizadas. Para tanto, o Serviço de Descoberta proposto possui uma linguagem de descrição e consulta com alta expressividade através da linguagem OWL, prevê a escalabilidade entre células do *middleware* EXEHDA através do protocolo P2P, possui uma arquitetura híbrida, sendo o processo de descoberta centralizado em cada célula, a descoberta em redes confinadas é realizada através do protocolo UPnP configurado nos gateways do ambiente e a invocação do recurso é feito através do protocolo HTTP considerando o padrão arquitetural REST.

Com as simulações realizadas, constatou-se a conformidade da modelagem desenvolvida com os objetivos propostos, e sua adequação com padrões e premissas que regem soluções direcionadas para a Internet das Coisas.

Na continuidade deste trabalho será desenvolvido o módulo Classificador de Recursos com critérios e algoritmos para seleção dos recursos mais adequados dentre os que forem descobertos, considerando as demandas do cliente.

Referências

Ahrenholz, J., Danilov, C., Henderson, T. R., and Kim, J. H. (2008). CORE: A real-time network emulator. In *Proceedings - IEEE Military Communications Conference MILCOM*.

Al-Fuqaha, A., Guizani, M., Mohammadi, M., Aledhari, M., and Ayyash, M. (2015). Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications. *IEEE Communications Surveys and Tutorials*, 17(4):2347–2376.

Andrade Junior, N. V. d., Bastos, D. B., and Prazeres, C. V. S. (2014). Web of Things: Automatic Publishing and Configuration of Devices. In *Proceedings of the 20th Brazilian Symposium on Multimedia and the Web*, WebMedia '14, pages 67–74, New York, NY, USA. ACM.

Cirani, S., Davoli, L., Ferrari, G., Leone, R., Medagliani, P., Picone, M., and Veltri, L. (2014). A scalable and self-configuring architecture for service discovery in the internet of things. *IEEE Internet of Things Journal*, 1(5):508–521.

Davet, P. T. (2015). Uma Contribuicao à Inclusão Digital dos Métodos de Pesquisa Agropecuário no Cenário da IoT.

Gubbi, J., Buyya, R., Marusic, S., and Palaniswami, M. (2013). Internet of Things (IoT): A vision, architectural elements, and future directions. *Future Generation Computer Systems*, 29(7):1645–1660.

Klusch, M. (2014). Service Discovery. In *Encyclopedia of Social Network Analysis and Mining*, pages 1707–1717.

Lopes, J., Souza, R., Geyer, C., Costa, C., Barbosa, J., Pernas, A., and Yamin, A. (2014). A Middleware Architecture for Dynamic Adaptation in Ubiquitous Computing. *J-Jucs*, 20(9):1327–1351.

Marin-Perianu, R., Hartel, P., and Scholten, H. (2005). A classification of service discovery protocols. *Centre for Telematics and Information Technology, Univ. of Twente, The Netherlands*, (June):5–25.

Miorandi, D., Sicari, S., De Pellegrini, F., and Chlamtac, I. (2012). Internet of things: Vision, applications and research challenges. *Ad Hoc Networks*, 10(7):1497–1516.

Perera, C., Zaslavsky, A., Christen, P., and Georgakopoulos, D. (2014). Context aware computing for the internet of things: A survey. *IEEE Communications Surveys and Tutorials*, 16(1):414–454.

Talal Ashraf Butt, Iain Phillips, L. G. G. O. (2012). TRENDY- An Adaptive and Context-Aware Service Descovery protocol for 6LoWPANs. *Wot*.