Simulando o Acesso a Recursos Compartilhados em um Cenário de Computação Ubíqua

Marcos Alves Vieira[1,2], **Sergio T. Carvalho**[2]

[1]Instituto Federal de Educação, Ciência e Tecnologia Goiano (IF Goiano)
Iporá – GO – Brasil

[2]Instituto de Informática – Universidade Federal de Goiás (UFG)
Goiânia – GO – Brasil

marcos.vieira@ifgoiano.edu.br, sergio@inf.ufg.br

Abstract. *New computing paradigms like the Internet of Things (IoT) and the Web of Things (WoT) are making smart spaces more and more common. One of the problems in this context is to enable access to the resources of the smart spaces by the different entities that compose it, considering different access levels. This paper aims to present and simulate the use of an algorithm and an access policies specification language for shared resources in a ubiquitous computing scenario. The simulation was performed under the formalities of the Discrete Event Systems Specification (DEVS) in the MS4 Modeling Environment (MS4 Me). We were able to conclude that both the language and the algorithm are valid approaches to handle concurrent access to smart objects resulting from the smart spaces overlapping.*

Resumo. *Novos paradigmas de computação como a Internet das Coisas e a Web das Coisas estão tornando os espaços inteligentes cada vez mais comuns. Um dos problemas nesse contexto é garantir o acesso aos recursos dos espaços inteligentes pelas diferentes entidades que o compõem, considerando diferentes prioridades e permissões de acesso. Este trabalho tem como objetivo apresentar e simular a utilização de um algoritmo e uma linguagem de especificação de políticas de acesso a recursos compartilhados em um cenário de computação ubíqua. A simulação foi feita sob os formalismos da Especificação de Sistemas de Eventos Discretos (DEVS) no ambiente de modelagem MS4 Me (MS4 Modeling Environment). Pôde-se concluir que tanto a linguagem quanto o algoritmo apresentados são abordagens válidas para tratar o acesso concorrente aos recursos de um cenário de computação ubíqua.*

1. Introdução

A convergência, disseminação e popularização das tecnologias de rádio, dos microprocessadores e dos dispositivos eletrônicos digitais pessoais, aliadas aos novos paradigmas computacionais, como a Internet das Coisas (*Internet of Things* - IoT) e a Web das Coisas (*Web of Things* - WoT) estão concretizando o conceito de ubiquidade vislumbrado por Mark Weiser. Nesse contexto, dispositivos inteligentes, tanto móveis quanto fixos, coordenam-se entre si para prover aos usuários acesso imediato e universal a novos serviços que visam aumentar as capacidades humanas. A computação ubíqua permite incorporar tecnologia de forma transparente a ambientes físicos, possibilitando auxiliar as

pessoas na realização de suas tarefas diárias de forma contínua e onipresente. Os dispositivos podem ser integrados aos espaços físicos cotidianos, transformando-os, assim, em espaços inteligentes [Weiser 1991].

Com os novos e emergentes paradigmas de computação, como a Internet das Coisas (*Internet of Things* - IoT) e a Web das Coisas (*Web of Things* - WoT), os espaços inteligentes estão se tornando cada vez mais comuns e, com isso, surgem novos desafios no que diz respeito ao acesso dos recursos que os compõem. É necessário possibilitar formas de controlar níveis e prioridades de acesso, com o objetivo de garantir o acesso imediato aos recursos às entidades com maior prioridade. Em um trabalho prévio [Vieira and Carvalho 2016b], uma linguagem de modelagem de cenários de computação ubíqua foi proposta para, dentre outras coisas, facilitar a modelagem de espaços inteligentes e possibilitar vocabulário e representação gráfica comuns entre os pesquisadores da área para os conceitos específicos do domínio de espaços inteligentes. Essa linguagem se baseia em um metamodelo próprio que serviu como base para construção do metamodelo que define a linguagem de regras de acesso a recursos de um cenário de computação ubíqua. O termo *cenário de computação ubíqua* refere-se a:

> *Um conjunto modelado de espaços inteligentes tanto pessoais quanto fixos, pessoas, aplicações ubíquas, objetos inteligentes e suas inter-relações.*

Este trabalho se apoia nos formalismos da Especificação Sistemas de Eventos Discretos (*Discrete Event System Specification* - DEVS) para simular a utilização de *(i)* uma linguagem para definição de regras de acesso a objetos inteligentes, considerando um cenário de computação ubíqua e *(ii)* um algoritmo para seu processamento, os quais foram construídos sob os conceitos de Engenharia Dirigida por Modelos (*Model-Driven Engineering* - MDE) e apresentados em trabalhos anteriores [Vieira 2016, Vieira and Carvalho 2016a, Vieira and Carvalho 2016b]. Como resultado, pode-se observar que ambas as abordagens são complementares e se mostraram viáveis para tratar o acesso concorrente aos recursos de um cenário de computação ubíqua.

O restante do trabalho está organizado da seguinte forma: a Seção 2 apresenta a fundamentação teórica necessária para contextualização deste trabalho; a Seção 3 detalha a linguagem para definição de regras de acesso para os recursos de um cenário de computação ubíqua, o algoritmo para processamento dos modelos escritos nessa linguagem e a modelagem das regras de acesso para um cenário de computação ubíqua tido como exemplo; a Seção 4 apresenta a simulação da utilização da linguagem de regras de acesso e de seu algoritmo de processamento, a qual foi orientada pelos formalismos de DEVS; por fim, a Seção 5 traz as conclusões e indica caminhos para possíveis trabalhos futuros.

2. Fundamentação Teórica

O termo e a ideia de Computação Ubíqua tomaram forma a partir da visão de Mark Weiser [Weiser 1991], o qual vislumbrou a possibilidade de tornar a utilização da computação invisível ao usuário, fundindo-a com elementos do dia-a-dia, ou seja, fazendo com que o usuário não precise perceber a tecnologia para aproveitar seus benefícios. Segundo este conceito, a computação estaria permeada nos objetos do ambiente físico do usuário, não requerendo dispositivos computacionais tradicionais para a interação, tais como teclado e

mouse. Na computação ubíqua, o foco do usuário sai do dispositivo computacional que ele manipula e passa para a tarefa ou para a ação a ser realizada.

Os espaços inteligentes são ambientes favoráveis à computação ubíqua, pois estendem a computação para os ambientes físicos e permitem que diferentes dispositivos forneçam suporte coordenado aos usuários baseado em suas preferências e na atual situação do ambiente físico (contexto) [Smirnov et al. 2013]. Em outras palavras, em um nível de abstração elevado, os espaços inteligentes podem ser tidos como ambientes de computação ubíqua que entendem e reagem às necessidades humanas [Lupiana et al. 2009].

Os espaços inteligentes tradicionais (*i.e.*, *espaços inteligentes fixos*) focam em fornecer serviços em espaços confinados e geograficamente limitados, e têm uma perspectiva centrada no sistema, na qual os usuários são atores externos que não estão contidos no espaço inteligente, ou seja, os usuários estão meramente localizados nos espaços inteligentes, não fazendo parte deles [Taylor 2011]. Contudo, programar apenas espaços inteligentes fixos pode levar a ilhas de ubiquidade separadas por espaços vazios, onde o suporte à computação ubíqua é limitado, pois estes não possibilitam o compartilhamento de dispositivos e serviços com outros espaços inteligentes.

Objetivando minimizar as ilhas de ubiquidade e em contraste com os espaços inteligentes fixos, um *espaço inteligente pessoal* (*Personal Smart Space* - PSS) é formado com base nos conceitos de computação ubíqua aliados aos diversos objetos inteligentes (*i.e.*, sensores, atuadores e dispositivos) que um usuário carrega consigo. O PSS acompanha o usuário durante sua mobilidade, ou em outras palavras, o PSS acompanha seu dono, estando sempre disponível e permitindo a interação com outros espaços inteligentes, sejam fixos ou mesmo pessoais [Taylor 2008].

Uma das alternativas para se programar espaços inteligentes é o uso de técnicas de Engenharia Dirigida por Modelos (*Model Driven Engineering* - MDE), tais como o Desenvolvimento Dirigido por Modelos (*Model Driven Development* - MDD) e a Arquitetura Dirigida por Modelos (*Model Driven Architecture* - MDA), as quais propõem o uso de abstrações mais próximas do domínio do problema como uma forma de mitigar a distância semântica existente entre o problema a ser solucionado e o ferramental (software) utilizado para tal. Em MDE, considera-se que os modelos são os principais artefatos no desenvolvimento de um sistema.

Um modelo é uma representação gráfica ou textual de alto nível de um sistema, onde cada um de seus elementos é uma representação virtual de um componente presente no sistema real. Os modelos servem não apenas para descrever ou documentar um software, mas também para atuar no seu desenvolvimento, manutenção e operação [Seidewitz 2003, Schmidt 2006]. Os relacionamentos e as abstrações utilizadas em um modelo são descritos por um metamodelo [Völter et al. 2013]. Um metamodelo pode ser considerado uma Linguagem de Modelagem Específica de Domínio (*Domain-Specific Modeling Language* - DSML). Uma DSML, por sua vez, é "uma linguagem textual ou gráfica que oferece, por meio das notações e abstrações apropriadas, poder de expressividade com foco em um domínio de problema particular, para visualizar, especificar, construir e documentar artefatos de um sistema de software" [Van Deursen et al. 2000, Chiprianov et al. 2014].

Uma das teorias de modelagem e simulação amplamente aplicáveis é a Especi-

ficação de Sistemas de Eventos Discretos (*Discrete Event System Specification* - DEVS) [Zeigler et al. 2000]. DEVS é um formalismo modular e hierárquico de sistemas, os quais podem ser descritos por meio de eventos discretos, estados e funções de transições entre os estados.

No formalismo DEVS, especifica-se os modelos básicos e como esses modelos são acoplados de forma hierárquica. No modelo básico, chamado *modelo atômico*, são descritas as dinâmicas internas de um modelo, definindo o comportamento real do sistema. Um *modelo acoplado* conecta os modelos básicos para formar um novo modelo. Este modelo pode ser empregado como componente em um modelo acoplado maior, permitindo a construção hierárquica de modelos complexos. Ambos modelos gerenciam suas operações de entrada/saída por meio de portas. Essas portas são ligadas por acoplamentos que definem a relação entre os componentes [Hong et al. 1997, Henares et al. 2019].

A partir dos objetivos da simulação, é possível construir um modelo de sistema por meio de sua base de conhecimento. A base de conhecimento tem um modelo base e regras de modelo. O modelo base contém um conjunto de modelos que descrevem um domínio específico. As regras ajudam a escolher as estruturas de sistema necessárias a partir do modelo base. Após a construção do modelo, a etapa de validação diz respeito à correspondência entre os objetivos e o modelo construído [Hong et al. 1997].

3. Tratando o Acesso aos Objetos Inteligentes

Um cenário de computação ubíqua é um ambiente dinâmico, composto por diversos recursos e entidades. Exemplos de dinamismo incluem a mobilidade de um usuário, que pode entrar e sair desses ambientes ou uma aplicação ubíqua que pode necessitar acessar um determinado objeto ubíquo em uma situação (*i.e.*, abrir uma porta mediante um acesso autorizado) e outro objeto inteligente em situação diferente (*i.e.*, acionar o alarme mediante um acesso negado). Contudo, um determinado recurso pode já estar em uso por outra entidade no momento em que uma nova requisição de acesso lhe é feita. Para lidar com o acesso concorrente aos objetos inteligentes de um cenário de computação ubíqua, considerando diferentes níveis de prioridade, foram propostos, em trabalhos anteriores, *(i)* uma linguagem para definição de regras de acesso aos objetos inteligentes e aplicações ubíquas de um cenário de computação ubíqua [Vieira and Carvalho 2016b]; e *(ii)* um algoritmo para o processamento dos modelos construídos com essa linguagem [Vieira and Carvalho 2016a]. Ambos serão brevemente apresentados nesta seção.

A linguagem de definição de regras de acesso permite a definição de um conjunto de regras de acesso para cada um dos objetos inteligentes e aplicações ubíquas que compõem um cenário de computação ubíqua. O algoritmo, por sua vez, se baseia no modelo de regras de acesso previamente definido para determinar se uma entidade requisitante terá permissão de acesso a um dado recurso.

A linguagem para definição de regras de acesso é definida por um metamodelo próprio, construído no *Eclipse Modeling Framework* (EMF)[1] e ilustrado na Figura 1. Sua sintaxe concreta é representada por meio de um esquema XML. Sendo assim, um arquivo XML que segue esse esquema representa uma instância do metamodelo da linguagem de regras de acesso. Essa representação permite modelar as regras de acesso para um cenário completo em um único arquivo XML, facilitando sua serialização e processamento.

[1] `http://www.eclipse.org/modeling/emf/`

A sintaxe abstrata do metamodelo de regras de acesso é detalhada a seguir. As metaclasses `SmartObject` e suas constituintes (`Actuator`, `Sensor` e `InputOutput`), além de `UbiquitousApplication` e `Person` são oriundas do metamodelo para modelagem de cenários de computação ubíqua, proposto em [Vieira and Carvalho 2016b].

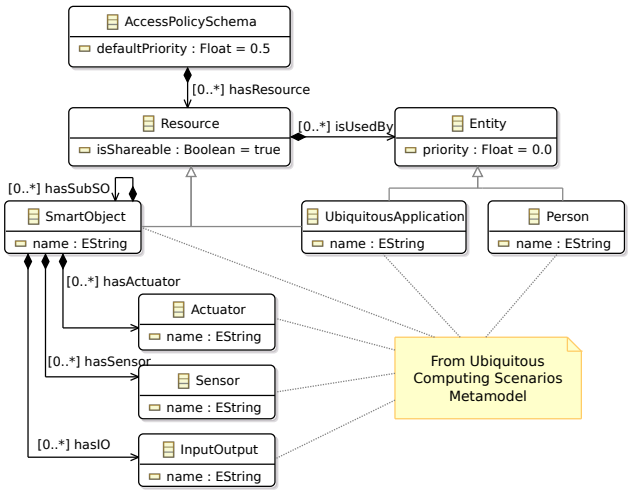

Figura 1. Metamodelo da linguagem de regras de acesso.

- `AccessPolicySchema`: este é o nó raiz e representa o modelo de regras de acesso de um cenário. O `AccessPolicySchema` é constituído por recursos (`Resource`) e estes são acessados por entidades (`Entity`). Por meio do atributo `defaultPriority` é possível definir o valor de prioridade de acesso padrão para todas as entidades ou recursos que não tenham sido modelados. Esse valor é determinado inicialmente em `0.5`, em uma escala que varia de `0` a `1`, onde `0` indica sem permissão de acesso e `1` indica prioridade máxima de acesso. É importante notar que se o valor da prioridade de acesso padrão for definido em `0`, todos os recursos não modelados não poderão ser acessados, a não ser que a entidade requisitante seja uma pessoa (`Person`) dona do recurso, isto é, associada ao mesmo por meio da relação `isOwnerOf`. O mesmo ocorre caso o valor de `defaultPriority` seja `0` e uma determinada entidade requisitante não modelada em um determinado recurso tentar usá-lo.
- `Resource`: um recurso define um objeto inteligente (`SmartObject`) ou uma aplicação ubíqua (`UbiquitousApplication`). Recursos são acessados por entidades (`Entity`). Um recurso é compartilhável por padrão, sendo possível modificar essa característica por meio de seu atributo `isShareable`. Para cada recurso, é possível detalhar a prioridade que as demais entidades do cenário terão ao usá-lo. Caso um recurso não tenha sido modelado, todas as entidades irão usá-lo com valor de prioridade de acesso padrão (`defaultPriority`).
- `Entity`: uma entidade define uma pessoa (`Person`) ou aplicação ubíqua (`UbiquitousApplication`). Entidades acessam recursos (`Resource`). O atributo `priority` determina o valor da prioridade de acesso da entidade ao recurso no qual ela está modelada. Este número varia de `0` a `1`, sendo que `0` indica que a entidade não possui permissão de acesso ao recurso e `1` indica prioridade máxima de acesso. Caso uma entidade não esteja modelada em um de-

terminado recurso esta irá acessá-lo com valor de prioridade de acesso padrão (`defaultPriority`), definido em `AccessPolicySchema`.

A lógica do algoritmo de processamento de modelos de regras de acesso é apresentada na Figura 2. Este algoritmo toma por base o modelo de regras de acesso do cenário de computação ubíqua, na forma de um arquivo XML, para analisar as requisições de acesso, permitindo ou negando o acesso aos recursos desse cenário.

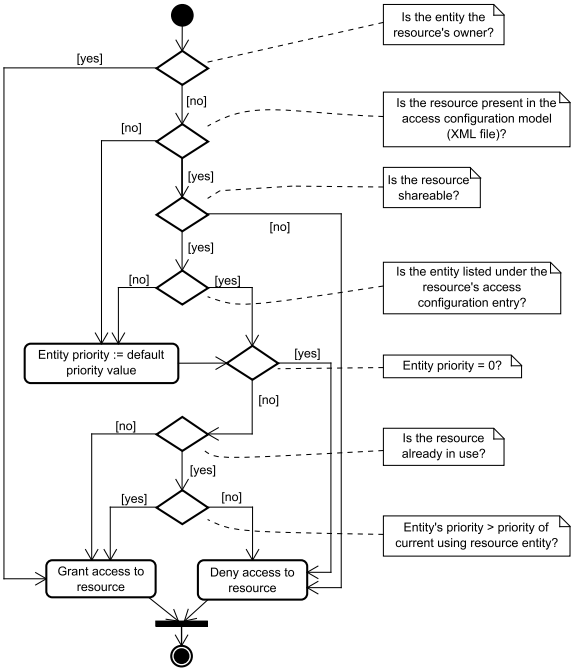

Figura 2. Algoritmo para definição da prioridade de acesso.

3.1. Modelando as Regras de Acesso de um Cenário de Computação Ubíqua

Para exemplificar o uso da linguagem de políticas de acesso, é apresentada a seguir a descrição de um cenário, o qual é considerado na simulação descrita na Seção 4:

> *Uma instituição de ensino é composta por três tipos de pessoas:* `teacher`, `student` *e* `technician`. *Nessa instituição, existem três tipos de recursos, sendo todos objetos inteligentes:* `teacherComputer`, `secretaryComputer` *e* `labComputer`.

Para esse cenário de exemplo, definiu-se, arbitrariamente, que as políticas de acesso devem ser modeladas conforme consta a seguir.

- Para o objeto inteligente *teacherComputer*, *teacher* é seu dono, além de possuir prioridade máxima de acesso (*i.e.*, `1.0`), enquanto que *student* não possui permissão de acesso a esse objeto inteligente (*i.e.*, `0.0`).
- Para o objeto inteligente *secretaryComputer*, *technician* é seu dono, além de possuir prioridade máxima de acesso (*i.e.*, `1.0`), enquanto que *teacher* possui permissão de acesso mediana (*i.e.*, `0.5`) e *student* não possui permissão de acesso a esse objeto inteligente (*i.e.*, `0.0`).
- Para o objeto inteligente *labComputer*, *student* possui prioridade mediana de acesso (*i.e.*, `0.5`), enquanto que *technician* e *teacher* possuem prioridade máxima de acesso (*i.e.*, `1.0`).

4. Simulação da Linguagem de Regras de Acesso e seu Algoritmo de Processamento

A simulação descrita neste trabalho foi realizada em um computador do tipo PC equipado com processador Intel Core i5-7200U com quatro núcleos de 2,5GHz, 8GB de memória RAM e sistema operacional Microsoft Windows 10. As diretrizes da simulação foram guiadas pelo trabalho de [de França and Travassos 2016], sumarizadas na Tabela 1.

Tabela 1. Diretrizes da simulação, conforme [de França and Travassos 2016].

Diretriz	Definição da diretriz	Cumprimento da diretriz neste trabalho
SG01	Identificação do relatório	Simulando o Acesso a Recursos Compartilhados em um Cenário de Computação Ubíqua
SG02 a SG06	Do contexto às questões de pesquisa	Validar uma linguagem de definição de regras de acesso aos recursos de um cenário de computação ubíqua e seu algoritmo de processamento.
SG08	Background e trabalhos relacionados	Apresentados na Seção 3.
SG09 e SG10	Modelo de simulação e validação	O modelo foi construído com base na linguagem e algoritmos publicados em trabalhos anteriores.
SG11	Assuntos envolvidos	Computação ubíqua, acesso concorrente, metamodelagem.
SG16	Dados de suporte	Arquivo de texto contendo 108 requisições de acesso geradas de forma aleatória, garantindo todas as permutações possíveis entre entidades e recursos do cenário de exemplo.
SG17	Ambiente de suporte à simulação	MS4 Modeling Environment (MS4 Me)
SG18	Análise de saída	Observação do visualizador do simulador e leitura do arquivo de registro (*log*) gerado.
SG19	Ameaças à validade	O conjunto de estímulos limitado pode ser uma ameaça à validade, contudo essa ameaça foi mitigada ao se garantir todas as permutações de tipos de requisições possíveis, conforme relatado na Seção 4.2. Outra ameaça diz respeito à correta implementação do algoritmo de processamento da linguagem de regras de acesso.
SG20 a SG22	Conclusões e trabalhos futuros	Apresentados na Seção 5.
SG23 a SG25	Validação do modelo	O modelo da simulação foi construído com base em trabalho prévio publicado pelos próprios autores.

4.1. Realização da Simulação

O formalismo DEVS separa conceitualmente os modelos do simulador, tornando possível simular o mesmo modelo usando diferentes simuladores trabalhando em modos de execução centralizados, paralelos ou distribuídos. A modelagem e simulação apresentada neste trabalho foi realizada no ambiente de modelagem MS4 Me [2] (*MS4 Modeling Environment*).

O MS4 Me permite que o ambiente de simulação seja complementado com códigos escritos em linguagem Java. Dessa forma, foram definidas classes para representar cada uma das metaclasses do metamodelo de regras de acesso. Assim sendo, foi possível programar a instanciação dinâmica dos objetos que representam o cenário de computação ubíqua, com atributos configurados definidos conforme descrito no arquivo XML que representa suas regras de acesso. Esses objetos são então utilizados pelo algoritmo para determinar a prioridade de acesso de uma determinada entidade no momento que esta tenta utilizar um recurso do cenário.

A Figura 3 apresenta a tela do simulador construído no MS4 Me, onde pode-se ver o modelo acoplado `UbiquitousScenario` e os modelos atômicos que compõem, além de suas associações. À esquerda temos o modelo atômico do gerador de estímulos (`StimulosGenerator`) e do gerente de políticas de acesso (`AccessPolicyManager`). Em *(a)* temos os modelos atômicos das entidades (`Student`, `Teacher` e `Technician`) e em *(b)* os modelos atômicos dos recursos (`LabComputer`, `SecretaryComputer` e `TeacherComputer`).

[2] `http://ms4systems.com`

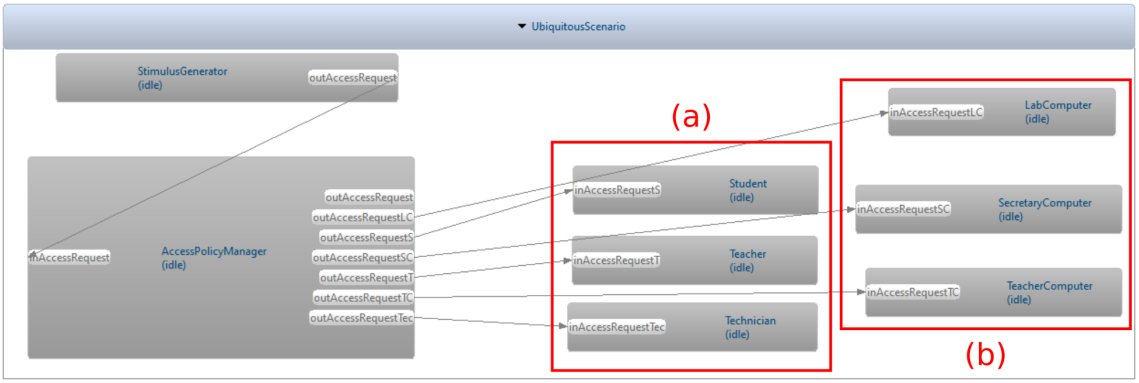

Figura 3. Tela do simulador construído.

O `StimulusGenerator` faz a leitura de um arquivo de texto, daqui em diante chamado de arquivo de estímulos, cujo conteúdo são linhas no formato `entidadeX:recursoY`, que descreve a tentativa de acesso de `entidadeX` ao `recursoY`. O arquivo de estímulos da simulação executada contém 108 sequências de requisições de acesso geradas aleatoriamente, garantindo todas as permutações possíveis de todos as entidades e recursos que compõem o cenário de exemplo. Para cada linha do arquivo de estímulos, o `StimulusGenerator` instancia um objeto do tipo `AccessRequest`, definido em classe Java própria, e então encaminha esse objeto via mensagem para o `AccessPolicyManager`.

O `AccessPolicyManager` contém a implementação do algoritmo de processamento regras de políticas de acesso, apresentado na Figura 2, e o executa a cada estímulo enviado pelo `StimulusGenerator`. A função do algoritmo é determinar, com base no modelo de regras de acesso (arquivo XML), se determinada entidade terá ou não acesso ao recurso requisitado. Para cada requisição, caso o acesso seja concedido ao final do processamento do algoritmo, uma mensagem é envidada para a entidade requisitante e o recurso requisitado. Independentemente de ter acesso permitido ou negado, para cada requisição de acesso um registro é gravado em um arquivo de registro (*log*). O `AccessPolicyManager` mantém o estado de cada uma das entidades e recursos, em tempo de simulação. Para tal, é realizada a instanciação de objetos, definidos em classes Java próprias, que correspondem a cada um dos recursos e entidades constantes no arquivo XML, que por sua vez representa o modelo de políticas de acesso. Isso é necessário para que seja possível confrontar o nível de prioridade da entidade que requisita o acesso a um determinado recurso com o nível de prioridade da entidade que já está utilizando esse mesmo recurso, caso haja.

Os modelos atômicos dos recursos e entidades do cenário de exemplo possuem uma porta de entrada para receber do `AccessPolicyManager` uma mensagem informando o recurso que está sendo utilizado (no caso das entidades) ou a entidade que o está utilizando (em caso dos recursos).

O código completo[3] do projeto do MS4 Me, incluindo os arquivos de estímulos e de registro da simulação estão disponíveis para consulta.

[3] `http://bit.ly/UbicompSimulationProject`

4.2. Resultados da Simulação e Discussões

Foram consideradas na simulação as 108 entradas do arquivo de estímulos, representando tentativas de acesso das entidades aos recursos do cenário de computação ubíqua. A simulação levou 98,332 segundos para ser realizada e a análise dos resultados se deu por meio da observação do visualizador do simulador e posterior leitura do arquivo de registro (*log*) gerado.

Foi possível identificar que todas as tentativas de acesso foram adequadamente tratadas pelo `AccessPolicyManager`, garantido a corretude da implementação do algoritmo de processamento das regras de acesso, resultando em 108 entradas no arquivo de registro, cujo trecho pode ser visto no Código 1. Cada linha desse arquivo representa uma tentativa de acesso de uma entidade a um determinado recurso e é gravada no formato: `timestamp entity -> resource? GRANTED/DENIED! Reason`.

Código 1. Trecho do arquivo de registro da simulação executada.

```
1  23/06/2019 13:53:11.956 teacher -> teachercomputer? GRANTED! Entity is Resource's owner.
2  23/06/2019 13:53:11.980 teacher -> secretarycomputer? GRANTED! Resource was idle.
3  23/06/2019 13:53:12.000 technician -> labcomputer? GRANTED! Resource was idle.
4  23/06/2019 13:53:12.028 teacher -> teachercomputer? GRANTED! Entity is Resource's owner.
5  23/06/2019 13:53:12.052 student -> teachercomputer? DENIED! Entity priority is ZERO.
6  23/06/2019 13:53:12.076 technician -> teachercomputer? DENIED! Lower priority: 1.0 vs 0.5
7  23/06/2019 13:53:12.100 teacher -> labcomputer? DENIED! Lower priority: 1.0 vs 1.0
8  23/06/2019 13:53:12.120 student -> secretarycomputer? DENIED! Entity priority is ZERO.
9  23/06/2019 13:53:12.140 technician -> secretarycomputer? GRANTED! Entity is Resource's owner.
10 23/06/2019 13:53:12.160 student -> labcomputer? GRANTED! Entity is Resource's owner.
11 23/06/2019 13:53:12.183 technician -> teachercomputer? DENIED! Lower priority: 1.0 vs 0.5
```

Para a simulação realizada, que contou com três entidades e três recursos, existem apenas nove tipos de requisições de acesso possíveis ($totalDeEntidades \times totalDeRecursos$). Contudo, com um cenário mais complexo, o número de possíveis requisições cresceria e seria bastante difícil acompanhar manualmente e decidir se as requisições de acesso seriam permitidas ou negadas. Dito isso, essa simulação serviu não apenas para garantir que a linguagem de regras de acesso e seu algoritmo de processamento são abordagens válidas, mas também mostra que é possível estendê-la para qualquer número de entidades e recursos de um cenário de computação ubíqua mais complexo para acompanhar o comportamento do cenário mediante as requisições de acesso.

5. Conclusão

Neste trabalho foram utilizados os formalismos da Especificação Sistemas de Eventos Discretos (*Discrete Event System Specification* - DEVS) para simular a utilização de uma linguagem para definição de regras de acesso a objetos inteligentes de um cenário de computação ubíqua e um algoritmo para seu processamento, os quais foram propostos em trabalhos anteriores.

Para realizar a simulação, utilizou-se o ambiente de modelagem e simulação MS4 Me, que permite complementação dos modelos e da simulação utilizando-se linguagem Java. Os resultados da simulação mostraram que tanto a linguagem quanto o algoritmo são abordagens válidas para tratar o acesso concorrentes aos recursos de um cenário de computação ubíqua. A simulação realizada pode ser estendida para qualquer número de entidades e recursos caso se deseje observar o comportamento do cenário mediante as requisições de acesso.

Como trabalhos futuros, pretende-se estender a simulação utilizando arquivos XML de modelos de cenários de computação ubíqua e sua sequência de estímulos de utilização gerados aleatoriamente. Outro trabalho futuro é a geração automatizada do simulador no ambiente MS4 Me mediante leitura dos arquivos XML e de estímulos.

Referências

[Chiprianov et al. 2014] Chiprianov, V., Kermarrec, Y., Rouvrais, S., and Simonin, J. (2014). Extending enterprise architecture modeling languages for domain specificity and collaboration: application to telecommunication service design. *Software & Systems Modeling*, 13(3):963–974.

[de França and Travassos 2016] de França, B. B. N. and Travassos, G. H. (2016). Experimentation with dynamic simulation models in software engineering: planning and reporting guidelines. *Empirical Software Engineering*, 21(3):1302–1345.

[Henares et al. 2019] Henares, K., Risco-Martín, J. L., and Zapater, M. (2019). Definition of a transparent constraint-based modeling and simulation layer for the management of complex systems. In *2019 Spring Simulation Conference (SpringSim)*, pages 1–12. IEEE.

[Hong et al. 1997] Hong, J. S., Song, H.-S., Kim, T. G., and Park, K. H. (1997). A real-time discrete event system specification formalism for seamless real-time software development. *Discrete Event Dynamic Systems*, 7(4):355–375.

[Lupiana et al. 2009] Lupiana, D., O'Driscoll, C., and Mtenzi, F. (2009). Taxonomy for ubiquitous computing environments. In *Networked Digital Technologies, 2009. NDT '09. First International Conference on*, pages 469–475.

[Schmidt 2006] Schmidt, D. C. (2006). Guest editor's introduction: Model-driven engineering. *Computer*, 39(2):0025–31.

[Seidewitz 2003] Seidewitz, E. (2003). What models mean. *IEEE software*, 20(5):26–32.

[Smirnov et al. 2013] Smirnov, A., Kashevnik, A., Shilov, N., and Teslya, N. (2013). Context-based access control model for smart space. In *Cyber Conflict (CyCon), 2013 5th International Conference on*, pages 1–15. IEEE.

[Taylor 2008] Taylor, N. (2008). Personal eSpace and Personal Smart Spaces. In *Self-Adaptive and Self-Organizing Systems Workshops, 2008. SASOW 2008. Second IEEE International Conference on*, pages 156–161.

[Taylor 2011] Taylor, N. (2011). Personal Smart Spaces. In Ferscha, A., editor, *Pervasive Adaptation: The Next Generation Pervasive Computing Research Agenda*, pages 79–80. Institute for Pervasive Computing, Johannes Kepler University Linz, Linz, AUS.

[Van Deursen et al. 2000] Van Deursen, A., Klint, P., and Visser, J. (2000). Domain-Specific Languages: An Annotated Bibliography. *Sigplan Notices*, 35(6):26–36.

[Vieira 2016] Vieira, M. A. (2016). Modelagem de espaços inteligentes pessoais e espaços inteligentes fixos no contexto de cenários de computação ubíqua. Master's thesis, Universidade Federal de Goiás, Goiânia, Goiás, Brazil.

[Vieira and Carvalho 2016a] Vieira, M. A. and Carvalho, S. T. (2016a). Addressing the concurrent access to smart objects in ubiquitous computing scenarios. In *Proceedings of the 22Nd Brazilian Symposium on Multimedia and the Web*, Webmedia '16, pages 79–82, New York, NY, USA. ACM.

[Vieira and Carvalho 2016b] Vieira, M. A. and Carvalho, S. T. (2016b). (Meta)Modelagem de Espaços Inteligentes Pessoais e Espaços Inteligentes Fixos para Aplicações Ubíquas. In *XXXVI Congresso da Sociedade Brasileira de Computação (CSBC) - VIII Simpósio Brasileiro de Computação Ubíqua e Pervasiva (SBCUP)*, pages 1056–1065, Porto Alegre-RS, Brazil. Sociedade Brasileira de Computação (SBC).

[Völter et al. 2013] Völter, M., Stahl, T., Bettin, J., Haase, A., and Helsen, S. (2013). *Model-driven software development: technology, engineering, management*. John Wiley & Sons.

[Weiser 1991] Weiser, M. (1991). The computer for the 21st century. *Scientific american*, 265(3):94–104.

[Zeigler et al. 2000] Zeigler, B. P., Kim, T. G., and Praehofer, H. (2000). *Theory of modeling and simulation*. Academic press.