Pedro Pignaton Negri

Uma Transformação Automática Entre Modelos de Representação de Situações e Modelos de Especificação de Simulação em Ambiente Baseado em Relacionamento Agente-Objeto

Vitória - ES, Brasil

Agosto de 2014

Pedro Pignaton Negri

Uma Transformação Automática Entre Modelos de Representação de Situações e Modelos de Especificação de Simulação em Ambiente Baseado em Relacionamento Agente-Objeto

Projeto de Graduação do Curso de Engenharia de Computação pela Universidade Federal do Espírito Santo.

Universidade Federal do Espírito Santo

Centro Tecnológico

Departamento de Informática

Orientador: Giancarlo Guizzardi

Coorientador: Vítor E. Silva Souza

Vitória - ES, Brasil

Agosto de 2014

Resumo

O desenvolvimento de aplicações sensíveis ao contexto passa pela necessidade de especificação dos contextos e situações de interesse. A SML permite a descrição de modelos conceituais de diferentes tipos de contextos e situações que auxiliam em tal processo através de uma linguagem gráfica de fácil entendimento. A fim de melhor explorar os modelos descritos, este trabalho descreve uma transformação MDA automatizada de modelos de especificação de situações em modelos do ambiente de simulação baseado em relacionamento agente-objeto AORS. Contribui fazendo com que modelos conceituais sejam transformados automaticamente em código a ser executado. Permite, assim, que situações especificadas em SML sejam detectadas em ambiente de simulação AORS. Finalmente, um estudo de caso é implementado no cenário de situações de emergência a fim de exemplificar o possível suporte na gerência de planos de emergência.

Palavras-chaves: Transformação de Modelos. Especificação de Situações. Simulação. Sistemas de Informação Orientados à Agentes. Planos de Emergência.

Abstract

When developing context-aware applications it is necessary to specify context and situation of interest. SML conveys conceptual models for different types of contexts and situations which allow better specification for its easy to understand graphical language. In order to better explore the aforementioned models, this work describes an automatic MDA transformation of situation models into models of AORS simulation environment based on agent-object-relationship. The contribution of this work lies on automatic transformation of conceptual models into a code to be executed. This, in turn, allows SML situation models to be detected in AORS simulation environment. Finally, a case study is implemented based on emergency situations as to exemplify support to management of emergency plans.

Key-words: Model Transformation. Situation Specification. Simulation. Agent-Oriented Information Systems. Emergency Plans.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

ABS	*Agent-Based Simulation*
AOR	*Agent-Object-Relationship*
AORS	*Agent-Object-Relationship Simulation*
AORSL	*Agent-Object-Relationship Simulation Language*
ATL	*ATL Transformation Language*
CIM	*Computational Independent Model*
EAORD	*External AOR Diagram*
EMF	*Eclipse Modeling Framework*
MDA	*Model-Driven-Architecture*
OMG	*Object Management Group*
PIM	*Platform Independent Model*
PSM	*Platform Specific Model*
SML	*Situation Modeling Language*
UML	*Unified Modeling Language*
XML	*eXtensible Markup Language*

Sumário

1 Introdução

1.1 Motivação

A miniaturização dos dispositivos de processamento traz uma nova perspectiva para a computação. Com tais dispositivos a computação torna-se mais presente no mundo, aumentando as possibilidades de percepção e interação com o meio no que Hansmann et al. (2003) definem como computação ubíqua ou pervasiva. As atividades humanas tornam-se permeadas de dispositivos providos de elevado poder de processamento que suportam uma infinidade de aplicações e comportamentos. Desta forma, o mesmo dispositivo e/ou software pode estar envolvido em uma grande diversidade de ambientes e contextos. Para o desenvolvimento de aplicações mais atrativas e funcionais é interessante que se considere esta possibilidade de execução em variados ambientes e condições. A utilização de informações que descrevam tais ambientes pode ser usada para alterar o comportamento da aplicação de acordo com as necessidades fazendo-as, assim, sensíveis ao contexto.

Contexto é definido por Dey (2001) como qualquer informação que pode ser usada para caracterizar a situação das entidades (uma pessoa, um lugar, ou um objeto) que são relevantes para uma aplicação, incluindo o próprio usuário e a própria aplicação. Assim, softwares e dispositivos podem ser desenvolvidos de forma a perceberem e interpretarem o estado do ambiente que os envolve e terem seus comportamentos sensíveis aos diversos contextos e situações. Ou seja, podem apresentar comportamentos diferenciados de acordo com as necessidades dos diversos meios e situações. Desta forma, nota-se um aumento na eficiência na interação usuário-dispositivo já que este último pode ter um comportamento mais direcionado às necessidades do contexto.

Não é necessário saber como determinado software será capaz de detectar as informações sobre o contexto. No escopo deste trabalho a importância recai sobre a definição fiel dos componentes que descrevem o contexto. Para tal, os contextos são então representados através de modelos conceituais. Segundo Guizzardi (2005), modelos conceituais são representações abstratas de um determinado domínio independente de um *design* ou tecnologia específica. Servem, portanto, para descrever determinados domínios e trazer entendimento geral/universal sobre determinados conceitos e, no nosso caso, contextos e situações.

A fase de definição, compreensão e registro do domínio pela modelagem conceitual promove o entendimento sobre o domínio entre os *stakeholders* e, assim, se tornam de suma importância no processo de desenvolvimento da aplicação:

Conceptual specifications are used to support understanding (learning),

> *problem-solving, and communication, among stakeholders about a given subject domain. Once a sufficient level of understanding and agreement about a domain is accomplished, then the conceptual specification is used as a blueprint for the subsequent phases of a system's development process. (GUIZZARDI, 2005)*

Desta forma, é fundamental que softwares sensíveis ao contexto tenham modelos conceituais de qualidade. Segundo Costa (2007), a modelagem conceitual de contexto deve anteceder o *design* detalhado das aplicações sensíveis ao contexto, da mesma forma que a análise deve anteceder o projeto detalhado de um sistema de informação.

Para a modelagem conceitual de contextos e situações, é proposta em (MIELKE, 2013) uma linguagem de modelagem gráfica chamada *Situation Modeling Language* (SML) com o intuito de prover facilidades na especificação de contextos e situações. Em seu trabalho é ressaltada a necessidade de que especialistas de domínio também tenham fácil acesso e compreensão dos modelos gerados para tais domínios.

O modelo construído em SML pode ser explorado de diversas formas. Transportá-lo para um ambiente de simulação torna-se relevante já que pode-se produzir de forma controlada eventos e variações nas informações e variáveis que compõem o domínio e, assim, observar o comportamento da simulação baseada no modelo. Com isso, pode-se melhor avaliá-lo diante os requisitos propostos e aumentar o entendimento sobre o mesmo. É desenvolvido neste trabalho, então, uma transformação de modelos visando transportar os modelos produzidos em SML para um ambiente de simulação com o objetivo de serem testados e melhor estudados.

Para tal, utiliza-se a plataforma de simulação proposta em (WAGNER, 2004) chamada *Agent-Object-Relationship Simulation* (AORS). Os modelos de simulação AORS são especificados com uma linguagem declarativa de alto nível (AORSL - *Agent-Object-Relationship Simulation*) que permite a modelagem conceitual do ambiente de simulação. Por ser um simulador multiagente, permite também especificar individualmente o comportamento dos atores que se relacionam com o domínio possibilitando a criação de modelos de simulação mais próximos da realidade.

1.2 Objetivos

Este trabalho busca contribuir no processo de modelagem conceitual de contextos e situações pela possibilidade de simulação dos modelos melhorando, assim, o entendimento e validação dos mesmos através da simulação de condições que envolvem tais modelos. Para isso, se propõe aos seguintes objetivos:

Objetivo 1: Definir o mapeamento entre o metamodelo de contexto e situações proposto em (MIELKE, 2013) com a SML (*Situation Modeling Language*) e o metamodelo

AORS (*Agent-Object-Relationship Simulation*) descrito em (WAGNER, 2004).

Objetivo 2: Descrever a transformação entre modelos SML e AORS baseada no correlacionamento entre os metamodelos dos mesmos.

Objetivo 3: Realizar estudo de caso para avaliar a contribuição deste trabalho modelando contextos e situações associados ao domínio de Planos de Emergência já que este é um domínio que envolve situações críticas e de especial interesse e, após a transformação de modelo descrita no objetivo 2, simular tal domínio no ambiente AORS.

1.3 Metodologia

Inicialmente foi feito um estudo dos temas que permeiam a proposta deste trabalho como modelagem conceitual, contextos e situações, sistemas de simulação e sistemas baseados em relacionamento agente-objeto. Depois, foi preparado o ambiente computacional para uso das ferramentas de suporte à SML e AORSL e, assim, explorada as funcionalidades para aumento da familiaridade com cada uma delas.

Posteriormente entrou-se na fase da transformação entre os modelos SML e AORS. Buscou-se aprofundar o conhecimento sobre os metamodelos da linguagem de descrição de situações e da linguagem de suporte a simulação AORS para fazer o mapeamento entre as classes dos metamodelos relacionando e transportando os conceitos de um modelo para o outro. Como as linguagens têm propósitos diferentes, o mapeamento requer especial atenção pois determinados conceitos não são transportados diretamente de um modelo para o outro - como por exemplo a detecção da ocorrência de situações que está intimamente ligada ao funcionamento do *framework* de simulação.

Com o mapeamento, foram pesquisadas formas de se programar a transformação automática entre tais modelos. Algumas linguagens declarativas foram estudadas como a ATL (*ATL Transformation Language*). Por fim optou-se por programar a transformação automática em linguagem imperativa pela liberdade que tais linguagens dispõem para tratar as informações uma vez que as linguagens declarativas propostas por ferramentas como a ATL podem conter restrições referentes às funcionalidades pré-programadas das mesmas.

Com a transformação feita, um estudo de caso é implementado para validação da ferramenta de transformação. Como este trabalho visa ser de aplicação prática, foi escolhido um domínio em voga atualmente por conta dos grandes eventos esportivos no país. Assim, algumas situações específicas possíveis em caso de ocorrência de incidentes emergenciais em estádios ou grandes eventos são modeladas em SML, transformadas automaticamente para AORSL e por fim simuladas.

1.4 Estrutura

Os capítulos deste trabalho estão estruturados como visto a seguir:

Capítulo 2: Aborda os principais conceitos utilizados neste trabalho como modelagem conceitual, contexto e situação, SML (*Situation Modeling Language*), modelagem e simulação AOR (*Agent-Object-Relationship*) e transformação entre modelos.

Capítulo 3: Apresenta o mapeamento entre as classes dos modelos SML e AORSL (*Agent-Object-Relationship Simulation Language*), trata de aspectos relativos a simulação em ambiente AOR e especifica como a transformação automática entre modelos SML e AORSL foi especificada.

Capítulo 4: Propõe um estudo de caso para validação da transformação entre modelos simulando situações relacionadas ao gerencimento de situações de emergência.

Capítulo 5: Apresenta as conclusões deste trabalho bem como as propostas de trabalhos futuros.

2 Fundamentação Teórica

2.1 Contexto e Situação

De acordo com Dey (2001), contexto é qualquer informação que pode ser usada para caracterizar a situação das entidades (uma pessoa, um lugar, ou um objeto) que são relevantes para uma aplicação, incluindo o próprio usuário e a própria aplicação. Assim, se determinada informação é relevante para caracterizar o usuário, a aplicação ou os estados onde estes estão inseridos, interagindo, então esta informação descreve parte do contexto.

Alguns exemplos simples de contexto são a localização do usuário, a data e hora, as condições climáticas do local onde o usuário se encontra ou até do local para o destino aonde o usuário está, por ventura, se deslocando, o perfil sonoro de alertas de um dispositivo, entre outros.

A partir dessas informações sobre o próprio usuário e sobre o meio que o cerca, podem-se definir combinações específicas de circunstâncias, acontecimentos ou estados. Essas são as situações. Dadas as informações de contexto exemplificadas anteriormente, podemos, por exemplo, definir uma situação na qual a data é um dia útil da semana, e é cedo, 7h. Assim, se for um dia útil da semana e é cedo, precisa-se, por exemplo, acordar para ir ao trabalho ou à faculdade e o perfil sonoro do celular deve ser ativado para tocar o alarme despertador.

Este tipo de comportamento baseado no contexto e situação é característica de aplicativos sensíveis ao contexto. Ademais, Dey (2001) também descreve que um sistema é sensível ao contexto se ele utiliza contexto para prover informação e/ou serviços relevantes para o usuário, em que a relevância depende das tarefas dos usuários. Assim, uma aplicação utiliza informações de contexto para perceber situações específicas de interesse e alterar seu comportamento provendo funcionalidades mais adequadas ou mais interessantes para o usuário. Não necessariamente a aplicação estará interessada nos estados/valores de cada informação de contexto e sim no estado que tais valores juntos representam. A localização geográfica do usuário combinada com o horário atual, por exemplo, pode ser bastante útil para a aplicação prover informações sobre restaurantes próximos caso esteja em horário de almoço.

2.2 Modelagem Conceitual

Modelagem Conceitual é uma forma de representar conhecimento sobre um determinado domínio, uma parte do mundo, e capturar aspectos relevantes através de esquemas

objetivando uma melhor compreensão e representação deste domínio.

Segundo Mylopoulos (1992), Modelagem Conceitual é a atividade de descrever formalmente alguns aspectos do mundo físico e social ao nosso redor com o propósito de entendimento e comunicação. Modelagem conceitual suporta facilidades de estruturação e inferência que são psicologicamente fundamentadas. Afinal, as descrições resultantes do processo de modelagem conceitual se direcionam à utilização humana, não por máquinas. Ou seja, a modelagem conceitual tem, antes de tudo, o objetivo de ser usada por pessoas com o propósito de entendimento comum e comunicação sobre o domínio em questão.

Durante o desenvolvimento do modelo, é importante a participação de técnicos e especialistas no domínio. Técnicos são os que conhecem o ferramental para construção de modelos conceituais, sabem como representar determinado conhecimento em modelo conceitual. Os especialistas são os que entendem o domínio a fundo e trarão informações sobre domínio a ser modelado.

Modelos Conceituais são, segundo Guizzardi (2005), representações abstratas de um determinado domínio independentemente do *design* ou tecnologias específicas. Assumem grande importância no processo de design de uma aplicação já que são a base para construção de entendimento sobre o domínio em questão e, subsequentemente, o desenvolvimento da aplicação. Assim, para que o modelo criado seja utilizado com eficência, os envolvidos devem debatê-lo exaustivamente até que se atinja um grau de entendimento e conformidade com a realidade suficiente para seu uso.

2.3 Modelagem de Contexto

Um modelo de contexto, ou modelo conceitual de contexto, é a representação das características que compõem determinado contexto de interesse. Em (COSTA et al., 2005) é definida fundamentação para criação de modelos conceituais (segundo Mylopoulos (1992)) de contexto para embasar aplicações sensíveis ao contexto. Desta forma, todas as características e aplicações de modelos conceituais descritos nas Seção 2.2 também se aplicam para modelos de contexto.

2.3.1 Fundamentação Ontológica

Em (COSTA, 2007) são propostas abstrações de modelagem de contexto. São definidos os conceitos de fundamentação Entidade (*Entity*) e Contexto (*Context*). Essa proposta extende os conceitos propostos em (GUIZZARDI, 2005) e são base para classificação dos conceitos específicos do domínio de interesse de maneira fundamentada. Assim, segundo essas propostas, o conceito Pessoa é representado como uma *Entity* e o conceito Localização é representado como *Context*. Esses dois conceitos são Tipos. Se falamos em José e sua atual localização, temos instâncias desses tipos, respectivamente.

A Figura 1 mostra as classes *Entity* e *Context* representando respectivamente os conceitos de entidade e uma condição de contexto particular a ela associado (por exemplo localização ou data/hora) através das relações *isContextOf* (é contexto de) e *hasContext* (tem o contexto).

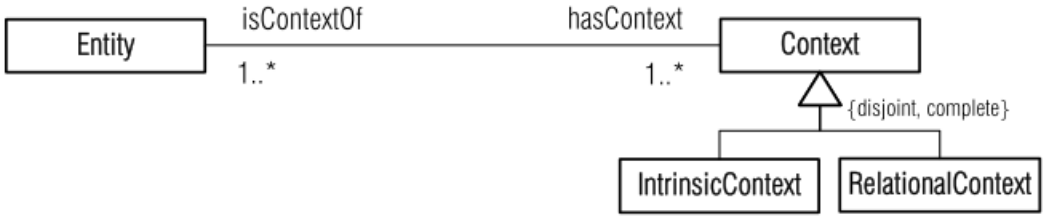


Figura 1: Conceitos Fundamentais de Modelagem de Contexto (COSTA, 2007)

Ainda na Figura 1 observamos que Contexto pode ser caracterizado como Contexto Intrínseco (*IntrinsicContext*) e Contexto Relacional (*RelationContext*). A diferença entre eles é que o Contexto Intrínseco é um tipo de contexto que é definido por uma simples entidade e não depende da relação com outras entidades (como temperatura e data/hora). Já o Contexto Relacional é aquele que depende da relação entre entidades distintas. Por exemplo, a localização geográfica de uma pessoa é um contexto intrínseco. Já a informação de que uma pessoa está em um restaurante define um contexto relacional que relaciona a entidade pessoa com a entidade restaurante.

2.3.2 Modelagem de Situação

Como já falado na Seção 2.1, uma situação é uma configuração específica dos estados das informações de um contexto. Assim, o modelo de situações deve definir restrições sobre as instâncias do modelo de contexto para gerar estados de interesses - situações de interesse.

Tais situações podem ser de interesse não somente pelo valor das variáveis de informação contextual que a descrevem mas sim pelo que elas juntas representam, significam. Pode-se citar, por exemplo, a situação de proximidade entre duas entidades. A aplicação pode não estar interessada no valor da distância entre tais entidades mas somente na informação que indica se estão próximas ou não. A seção a seguir descreve uma linguagem específica para modelagem de contexto e situações.

2.4 *Situation Modeling Language* (SML)

Na fase de modelagem conceitual é de suma importância que os especialistas do domínio participem da construção do modelo pois são eles que melhor o entendem e, assim, podem melhor defini-lo. É proposta, então, em (MIELKE, 2013) a *Situation Modeling*

Language (SML - Linguagem de Modelagem de Situação), uma linguagem de modelagem gráfica de situações e contextos que prima pelo fácil entendimento e manipulação dos modelos para permitir que especialistas do domínio e especialistas técnicos se comuniquem e se expressem com mais facilidade melhorando, então, o desenvolvimento dos modelos.

A SML utiliza um metamodelo de contexto e um metamodelo de situações próprios (detalhados respectivamente nas Seções 2.4.1 e 2.4.2). Tais metamodelos foram criados para ser possível representar contextos e situações de diversos domínios. Segundo Almeida (2006), modelo é um artefato criado por meio de uma linguagem de modelagem para representar uma determinada informação, conhecimento. Um metamodelo, segundo Miller e Mukerji (2003), define a sintaxe abstrata da linguagem de modelagem utilizada para criação de modelos. Assim, os modelos (tanto de contextos quanto de situações) criados usando SML são baseados nos metamodelos em questão.

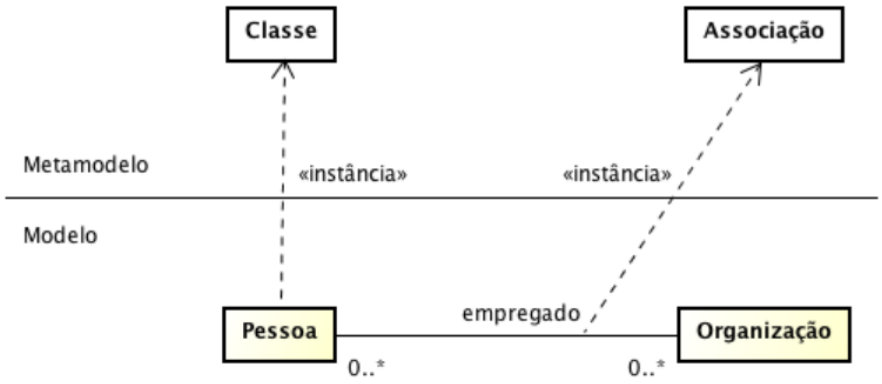


Figura 2: Modelo e Metamodelo em UML (MIELKE, 2013)

A Figura 2 ilustra os conceitos de modelo e metamodelo com um exemplo de alguns elementos do metamodelo UML - *Unified Modeling Language* (OMG, 2011). No nível de metamodelo temos os elementos sintáticos para composição do modelo. Assim, neste exemplo, o modelo contém duas instâncias do elementos Classe (Pessoa e Organização) e uma instância do elemento Associação (empregado).

A SML utiliza-se de uma notação visual gráfica própria para representar modelos de tipos de situação. Isso porque este tipo de notação contém vantagens principalmente quanto à usabilidade e facilidade de entendimento. Segundo Moody (2009), a notação visual desempenha um papel crítico na comunicação com os usuários finais, uma vez que acredita-se que estas transmitam informação de maneira mais efetiva para não técnicos do que texto. O autor afirma também que representações visuais são eficazes porque exploram as capacidades do poderoso e altamente paralelizável sistema visual humano.

2.4.1 Metamodelo de Contexto

Um modelo de contexto descreve os elementos do domínio de interesse e forma a base para a especificação dos tipos de situações de interesse. Ou seja, define as entidades, os contextos relacionais e intrínsecos que podem existir em um determinado domínio de interesse.

Inspirado pela abordagem de modelagem apresentada em (COSTA, 2007), que propõe a utilização de um modelo conceitual de contexto em UML (OMG, 2011) com a adição de estereótipos (definidos em um perfil UML), na SML são definidos também conceitos de Entidade e Contexto, que servem de base para que o desenvolvedor classifique os conceitos específicos do domínio da aplicação de maneira fundamentada, de acordo com as teorias propostas em (GUIZZARDI, 2005).

Um Modelo de Contexto é construído baseado em um metamodelo de contexto. A SML usa um metamodelo de contexto próprio porque, segundo Mielke (2013), outras linguagens de modelagem podem não oferecer estruturas nativas para expressão de alguns conceitos do domínio de contextos (o que traria a necessidade de sobrecarga semântica para especificação de tais conceitos).

O Metamodelo de Contexto completo é encontrado no Anexo A. As Figuras 3, 4 e 5 apresentam visões parciais deste metamodelo com as descrições dos elementos baseadas nas definições encontrada em (MIELKE, 2013).

A Figura 3 exibe os principais elementos do metamodelo com destaque para o elemento inicial, a metaclasse *ContextModel*. Esta metaclasse é composta pelos elementos que podem fazer parte de um modelo de contexto por meio da metaclasse abstrata *ContextModelElement* que é especializada em:

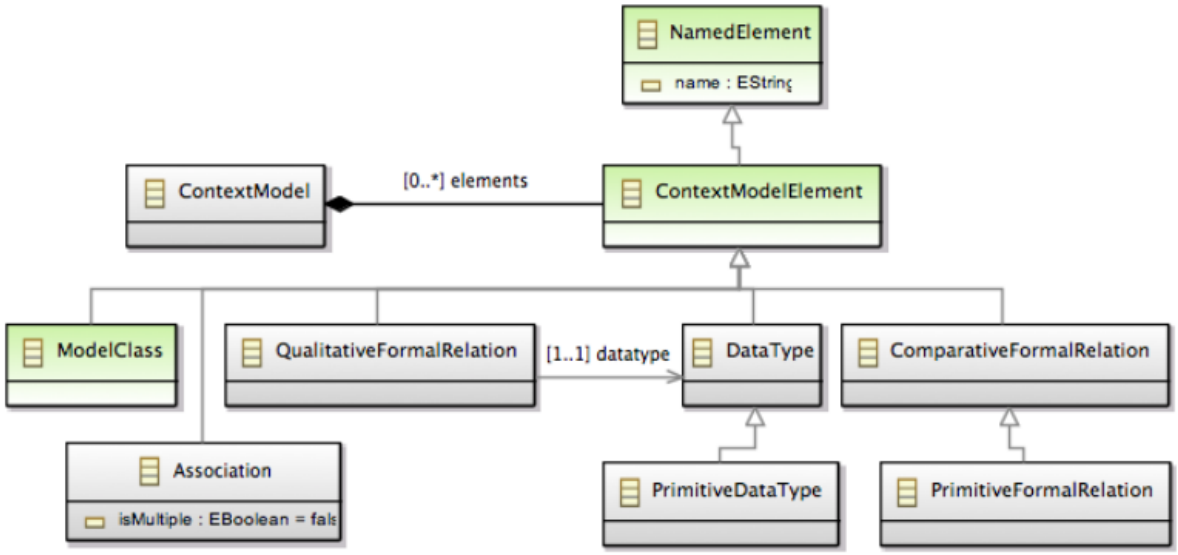


Figura 3: Principais elementos do metamodelo de contextos da SML (MIELKE, 2013)

ModelClass Metaclasse abstrata utilizada para representar entidades e contextos relacionais no modelo de contexto (detalhada na Figura 4).

DataType Representa os possíveis tipos de dados que podem ser utilizados na construção de um modelo de contexto. Metaclasse especializada em *PrimitiveDataType* - tipos de dados primitivos para distinguir os tipos internos (e.g. inteiros e *strings*) dos criados pelo usuário (e.g. localização geográfica).

ComparativeFormalRelation Especifica as possíveis relações formais de comparação (GUIZZARDI, 2005) que podem existir entre os diferentes elementos que compõem um modelo de contexto (e.g. a relação *maior que* pode ser especificada entre duas temperaturas). É especializada na metaclasse *PrimitiveFormalRelation* para distinção entre elementos primitivos e elementos definidos pelo usuário. Detalhes na Figura 5.

QualitativeFormalRelation Representa relações formais utilizadas para derivação de informações como por exemplo a relação *distância* entre duas localizações.

Association Representa a associação entre um contexto relacional e as entidades que estão relacionadas, associadas, a este contexto.

Na Figura 4 é explorada a metaclasse *ModelClass* especializada em *EntityClass*, que representa os tipos de entidades que compõem o modelo de contexto, e *RelatorClass*, que representa os tipos de contexto relacionais entre as entidades.

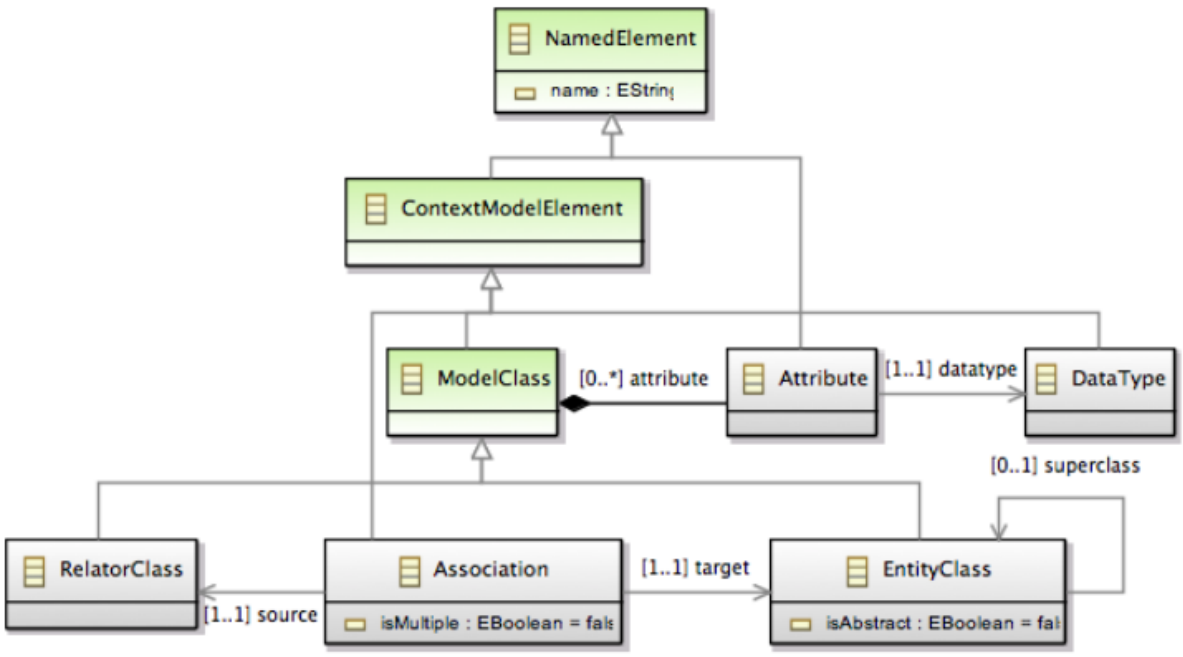


Figura 4: Elementos estruturais do metamodelo de contextos da SML (MIELKE, 2013)

Contextos intrínsecos são especificados pelo uso da metaclasse *Attribute* pertencente a uma *ModelClass*. Como pode ser observado, tanto as entidades quanto os contextos relacionais podem possuir contextos intrínsecos associados.

Ainda na Figura 4, observamos a possibilidade de associação entre um contexto relacional (*RelatorClass*) e as entidades relacionadas (*EntityClass*) por meio da metaclasse *Association*. As restrições de cardinalidade especificadas para as associações são realizadas pelo atributo *isMultiple*, que indica se a associação à entidade é múltipla ou simples.

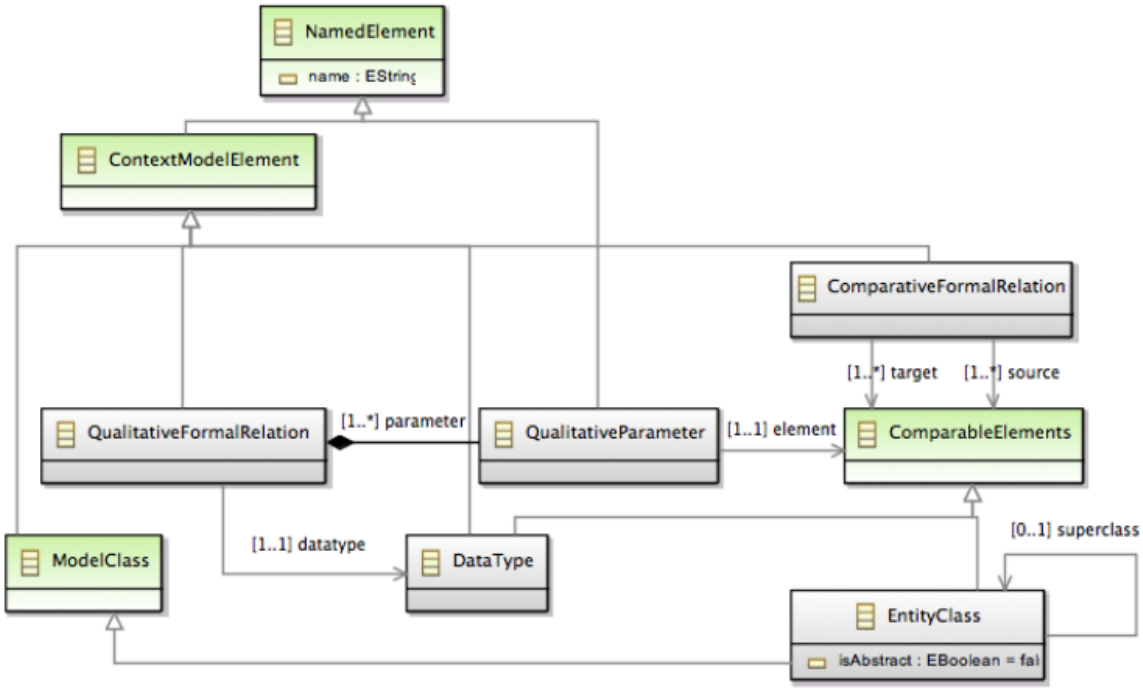


Figura 5: Elementos representando a especificação de relações formais do metamodelo de contextos da SML (MIELKE, 2013)

A Figura 5 contém os detalhes das metaclasses utilizadas para a especificação de relações formais de comparação e relações formais de derivação de informação para o modelo de contexto. Tais relações formais de comparação (*ComparativeFormalRelation*) podem ser especificadas entre entidades (*EntityClass*) e tipos de dados (*DataType*).

Ainda na Figura 5 tem-se a metaclasse *QualitativeFormalRelation* que é utilizada para representar relações formais de derivação de informação através de parâmetros *QualitativeParameter*. Tais parâmetros podem ser entidades (*EntityClass*) e tipos de dados (*DataType*).

2.4.2 Metamodelo de Situações

O Modelo de Contexto é um modelo que define as entidades e os contextos intrínsecos e relacionais que podem existir em um determinado domínio de interesse. O Modelo de Situações traz, por sua vez, restrições sobre as instâncias do modelo de contexto para

representar estados específicos das informações de contexto, situações específicas de interesse. Assim, o metamodelo de situações definido na SML faz referência direta aos conceitos do metamodelo de contexto da SML.

O metamodelo de situações completo encontra-se no Anexo B. A Figura 6 destaca o elemento principal do metamodelo de situações (*SMLModel*) que é composta por tipos de situação (*SituationType*).

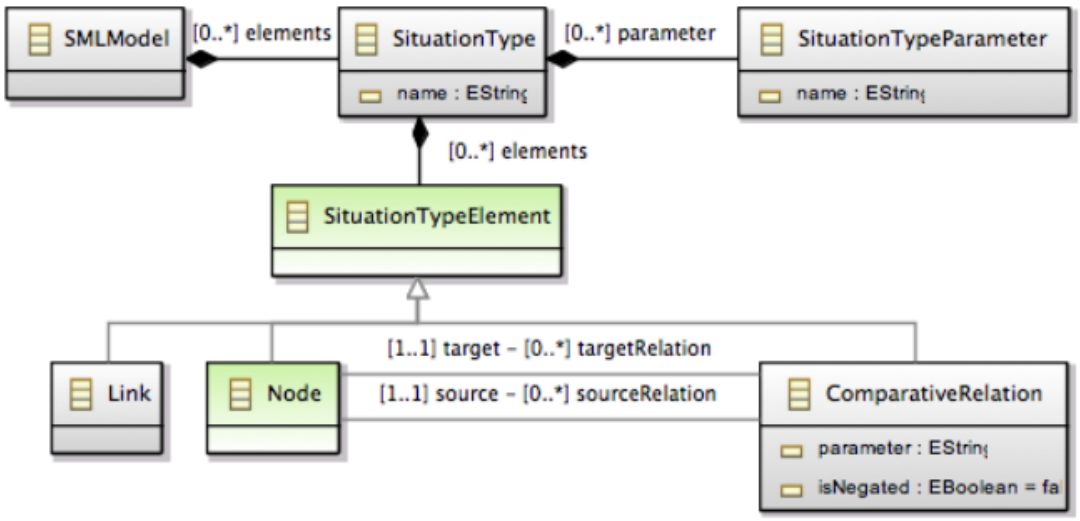


Figura 6: Principais elementos do metamodelo de situações da SML (MIELKE, 2013)

Um tipo de situação é constituído por elementos de situação, representados pela metaclasse abstrata *SituationTypeElement* especializada em:

Node Metaclasse abstrata que representa os diferentes elementos que participam da especificação de um tipo de situação como entidades e contextos relacionais. Detalhada na Figura 7.

Link Representa uma associação entre uma entidade e um contexto relacional. Como exemplo, Detalhes na Figura 8.

ComparativeRelation Representa a ocorrência de relações formais entre diferentes elementos que compõem a especificação de um tipo de situação. Como exemplo, a relação de proximidade entre a localização de duas entidades.

Como mostrado na Figura 6, um tipo de situação pode possuir diversos parâmetros de situação (*SituationTypeParameter*) que identificam quais são as entidades que caracterizam o tipo de situação (denominadas de participantes da situação) dentre as entidades que compõem a especificação da situação.

A Figura 7 mostra os elementos que podem ser definidos como parâmetros de situação além da hierarquia de especialização da metaclasse abstrata *Node*. São destacadas as seguintes metaclasses:

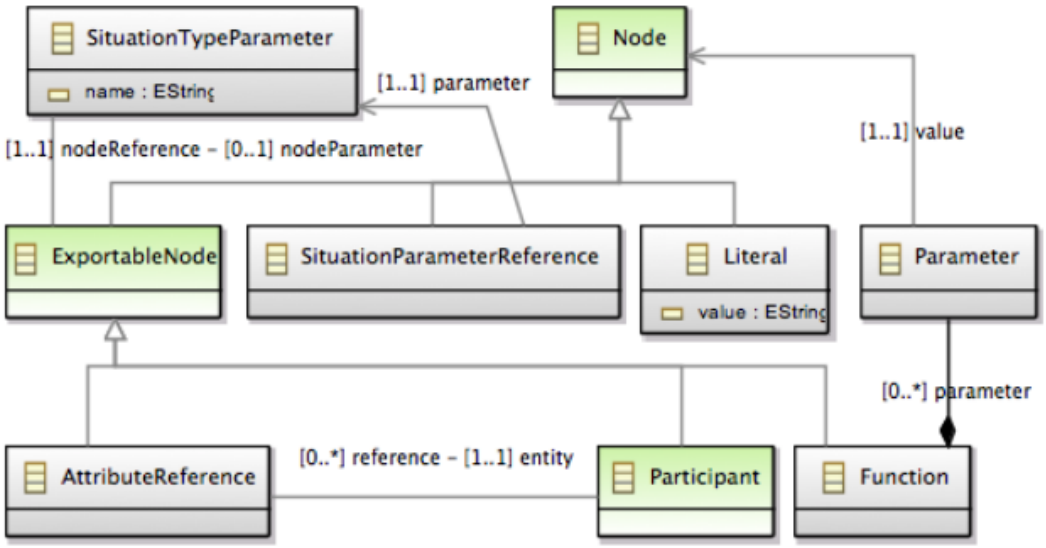


Figura 7: Detalhes da hierarquia da metaclasse abstrata *Node* do metamodelo de situações SML (MIELKE, 2013)

Literal Representa um valor literal utilizado para a especificação de tipos de situação.

Function Representa a derivação de dados baseada em diferentes elementos que compõem a situação, tais como valores associados a contextos intrínsecos, literais, resultados de outras funções e etc.

Participant Representa os principais elementos (entidades, contextos relacionais) que compõem a situação. Detalhada na Figura 8.

AttributeReference Representa valores associados a atributos (contextos intrínsecos) das entidades que compõem uma situação como por exemplo o atributo *Localização* de uma entidade *Pessoa*.

A Figura 8 mostra detalhes da hierarquia da metaclasse abstrata *Participant* e exibe a relação das subclasses com outros elementos do Metamodelo de Situações. As principais metaclasses exibidas nesta figura são:

EntityParticipant Representa a ocorrência de uma entidade na especificação de um tipo de situação.

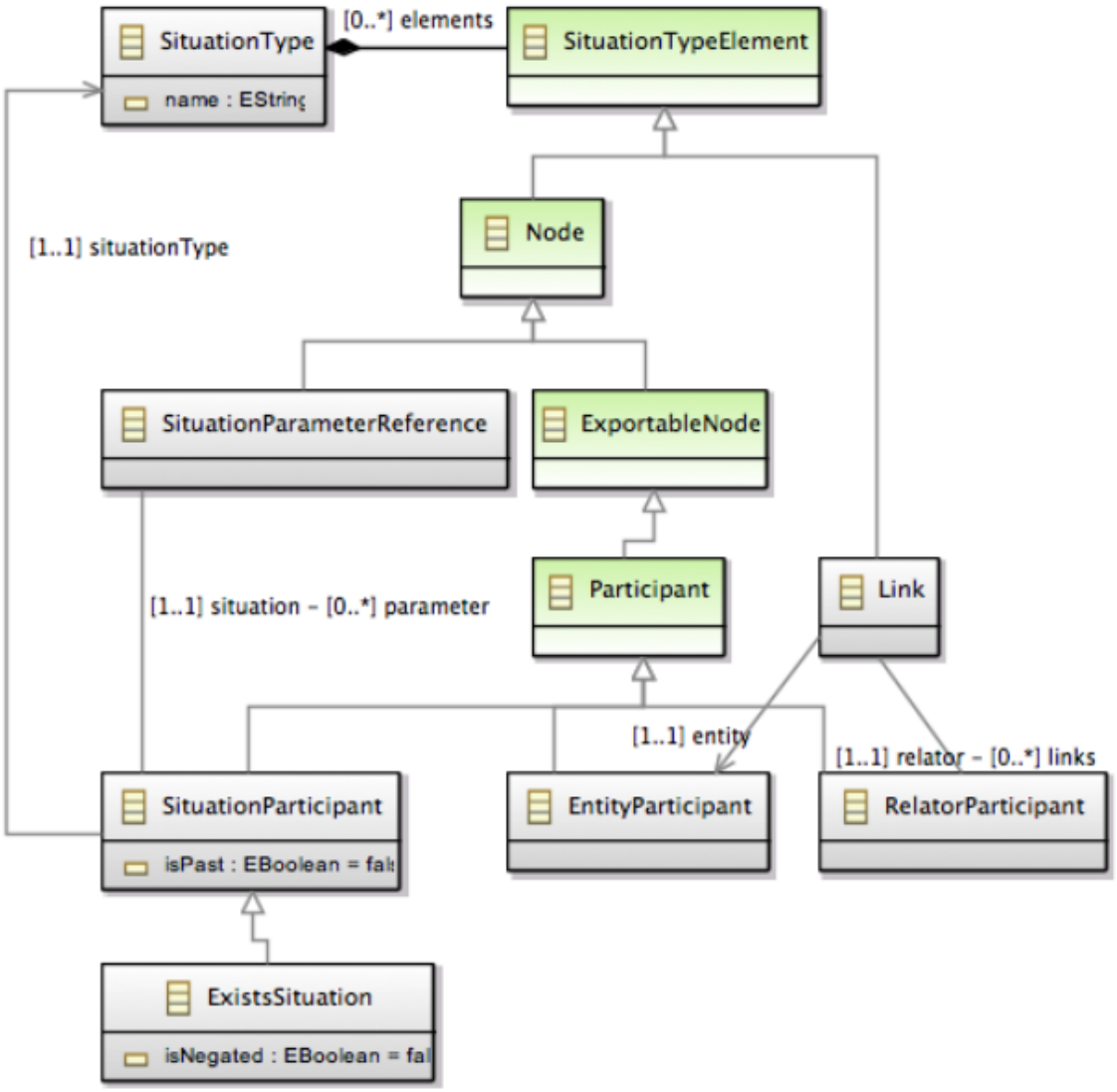


Figura 8: Detalhes da hierarquia da metaclasse abstrata *Participant* do metamodelo de situações da SML (MIELKE, 2013)

RelatorParticipant Representa a ocorrência de um contexto relacional na especificação de uma situação. A ligação entre o contexto relacional e as entidades é feita por meio da metaclasse *Link*.

SituationParticipant Representa a ocorrência de uma situação na especificação de outra situação (situação composta). Esta metaclasse faz referência a um tipo específico de situação no modelo (*SituationType*), podendo considerar apenas instâncias de situações passadas (já finalizadas) ou atuais por meio do atributo *isPast*. É possível referenciar os participantes da situação durante a composição por meio da metaclasse *SituationParameterReference* (também exibida na Figura 7).

ExistParticipant Tipo especial de composição de situações que faz uso de quantificador existencial afim de verificar a existência de instâncias de tipos de situação.

O metamodelo de situações possui relação com o metamodelo de contexto já que a situação é um estado específico das informações que caracterizam um contexto. Assim, algumas classes do metamodelo de situações fazem referência à classes do metamodelo de contexto através de atributos conforme mostrado na Tabela 1.

Metaclasse no Metamodelo de Situação	Nome do atributo na metaclasse	Metaclasse no Metamodelo de Contexto
SMLModel	contextModel	ContextModel
EntityParticipant	isOfType	EntityClass
RelatorParticipant	isOfType	RelatorClass
Link	isOfType	Association
AttributeReference	attribute	Attribute
ComparativeRelation	relation	ComparativeFormalRelation
Literal	dataType	DataType
Function	function	QualitativeFormalRelation
Parameter	parameter	QualitativeParameter

Tabela 1: Relações entre os metamodelos de contexto e situações

2.4.3 Sintaxe Concreta de SML

Esta seção apresenta uma visão dos principais símbolos utilizados na notação gráfica da SML e sua relação com o conceito representado e o elemento correspondente no metamodelo de situações.

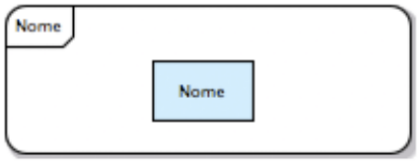
A seguir uma visão completa dos elementos utilizados na construção da notação gráfica e o seu mapeamento para as classes do metamodelo de situações.

SituationType

Notação Retângulo de borda arredondada com nome no canto superior esquerdo.

Descrição Representa um tipo de situação.

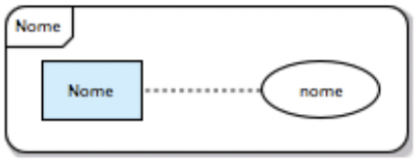
EntityParticipant



Notação Retângulo azul-claro.

Descrição Representa a participação de uma entidade na ocorrência da situação.

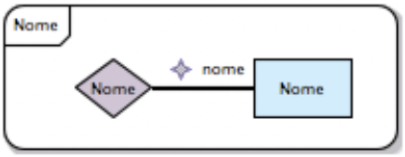
AttributeReference



Notação Elipse branca conectada a uma entidade por meio de uma linha tracejada.

Descrição Representa um contexto intrinseco da respectiva entidade.

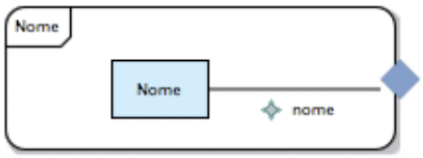
RelatorParticipant



Notação Losango roxo-claro no interior de uma situação.

Descrição Representa participação de um contexto relacional na situação.

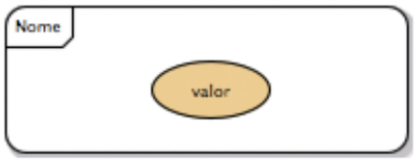
SituationTypeParameter



Notação Losango azul na borda de uma situação.

Descrição Representa que a entidade referenciada pode ser utilizada na composição de outras situações, caracterizando esta entidade como parte essencial da situação.

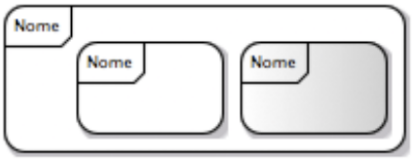
Literal



Notação Elipse alaranjada.

Descrição Representa ocorrencia de literais na definição de tipos de situação.

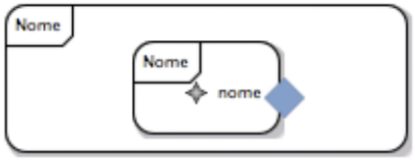
SituationParticipant



Notação Retangulo de borda circular no interior de uma situação.

Descrição Representa a composição de situações. Situações passadas são representadas com cor de fundo branca e situações correntes com a cor de fundo em gradiente de cinza.

SituationParameterReference



Notação Lozango azul na borda de uma situação que está compondo outra situação.

Descrição Representa um participante da situação composta.

ExistisSituation

Notação Retangulo de borda circular no interior de uma sitação juntamente com o símbolo de existencial ou sua negação (∃ ou !∃).

Descrição Representa composição existencial de situações ou sua negação.

A seguir outras metaclasses ou seus atributos representados por linhas na construção da notação gráfica de SML.

Link

Notação Linha preta cheia.

Descrição Representa associação entre contextos relacionais e suas entidades.

ComparativeRelation

Notação Linha cinza direcional.

Descrição Representa a ocorrência de relações formais entre diferentes elementos que compõem a especificação do tipo de situação.

atributo reference (Participant)

Notação Linha pontilhada cinza.

Descrição Liga um contexto intrínseco à sua entidade correspondente. É o atributo *reference* da metaclasse *Participant*.

Parameter

Notação Linha cinza, pontilhada e direcional.

Descrição Representa parâmetros de funções de derivação de dados.

nodeReference da SituationTypeParameter

Notação Linha cinza ligada a um losango na borda de um tipo de situação.

Descrição Identifica quais das entidades que compõem a especificação de uma situação podem ser referenciadasna composição de outras situações. É o atributo *nodeReference* da metaclasse *SituationTypeParameter*.

2.5 Agent-Object-Relationship Modeling and Simulation

Agent-Object-Relationship (AOR) *Modeling and Simulation*, descrita em (WAGNER, 2004), é uma abordagem orientada a agente para modelagem conceitual que provê além de uma linguagem de modelagem ontologicamente fundamentada, um *framework* de simulação também baseado em agente que permite simulações mais próximas da realidade de cenários de eventos discretos complexos.

Simulação baseada em agente (ABS - *Agent-based simulation*) é um paradigma aplicado ao problema de simulação. Em ABS, os agentes participantes (animais, humanos, instituições sociais, sistemas de software ou máquinas) podem realizar ações, perceber o ambiente e reagir a mudanças neles. Agentes também podem conter um estado mental que compreende componentes como conhecimentos e crenças, memórias, compromissos e metas. Assim, Davidsson (2002) expõe que a ABS se destaca de métodos tradicionais de simulação (como equações matemáticas, automato celular, teoria dos jogos, simulador de evento discreto) pois tenta descrever explicitamente o comportamento específico de cada ator do sistema a ser simulado, enquanto outras técnicas de simulação normalmente se baseiam em modelos matemáticos sobre os efeitos médios de comportamento dos indivíduos ou da população inteira.

Assim, AOR *Modeling and Simulation* é um *framework* multi-propósito, ABS e com uma abordagem orientada a agente para modelagem conceitual do domínio através de uma linguagem visual de especificação de alto nível e que permite a especificação de um modelo de simulação em uma linguagem declarativa baseada em XML. Esta linguagem permite expressar a essência da lógica de simulação com constructos de alto nível (como regras de comportamento de agentes) e definir processamentos especiais com expressões de linguagem de programação em trechos de código.

2.5.1 AOR Modelling

A modelagem conceitual do domínio é feita utilizando uma linguagem de especificação de sintaxe visual, declarativa, de alto nível, baseada em UML chamada *Agent-Object-Relationship Modeling Language* (AORML) definida em (WAGNER, 2003).

Em AORML, como observado na Figura 9, uma entidade (*Entity*) é um agente

(*agent*), um evento (*event*), uma ação (*action*), uma reivindicação (*commitment/claim*) ou um objeto (*object*). Somente agentes podem comunicar, perceber, agir, cumprir compromissos e satisfazer reivindicações. Objetos são entidades passivas sem tais capacidades. Além dos agentes humanos e artificiais, AORML também inclui o conceito de agentes institucionais que são compostos por um número de outros agentes que agem em seu nome.

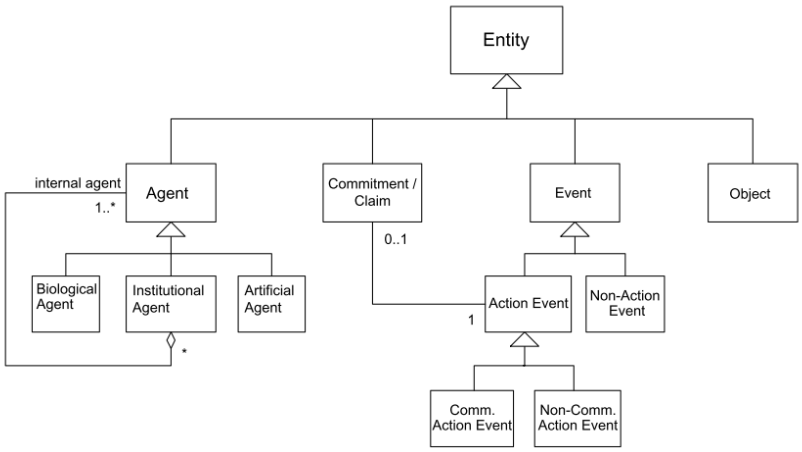


Figura 9: As meta-entidades de um modelo AOR externo (WAGNER, 2003)

Modelos AOR podem ser do tipo *externo* ou *interno*. Um modelo AOR externo adota a perspectiva de um observador que olha externamente, de fora, para os agentes e suas interações no domínio em consideração. Já em um modelo AOR interno, adota-se a visão interna, primeira pessoa, de um agente específico a ser modelado.

Um modelo AOR externo pode ser composto por um ou mais diagramas dentre os listados a seguir (WAGNER, 2003):

Agent Diagram representam os tipos de agente do domínio, determinados tipos de objetos relevantes e a relacionamento entre eles.

Interaction Frame Diagram representam os tipos de evento de ação e tipos de compromisso/reivindicação que determinam as possíveis interações entre dois tipos de agentes (ou instâncias).

Interaction Sequence Diagram representam instâncias de processos de interação.

Interaction Pattern Diagram focada no padrão geral de interação expressada por significados por uma série de regras de reação (*reaction rules*) definindo um tipo de processo de interação.

Os elementos mais importantes de um diagrama AOR externo são vistos nas Figura 10. Este diagrama mostra que há uma diferenciação entre *action events* (eventos de ação) e *non-action events* (eventos sem ação), entre um *communicative action event* (mensagem) e

non-communicative action event e mostra também que um *commitment/claim* é associado com um *action event* que cumpre este *commitment* ou satisfaz esse *claim*.

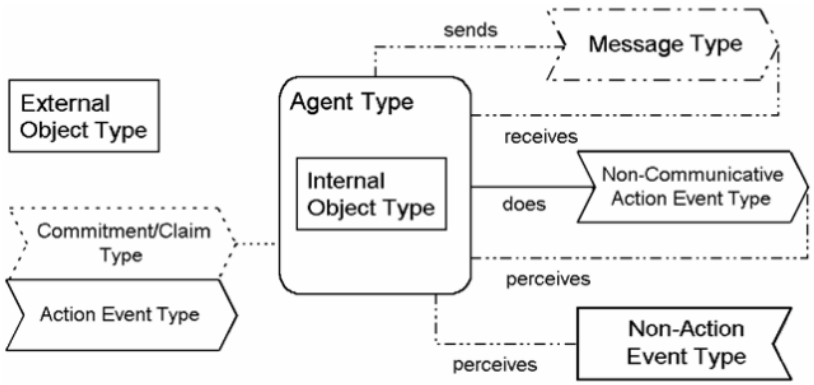


Figura 10: Elementos principais de um modelo AOR externo (WAGNER, 2003)

No diagrama *Interaction Pattern Diagram* tem-se um importante elemento de modelagem de comportamento que descreve os padrões de interação, as regras de reação (*reaction rules*). Uma regra de reação é expressa como um círculo com setas de entrada e saída, desenhada dentro do retângulo do agente cujo o padrão de *reaction* ela representa. Tem-se na Figura 11 um exemplo de uma regra de reação, R1.

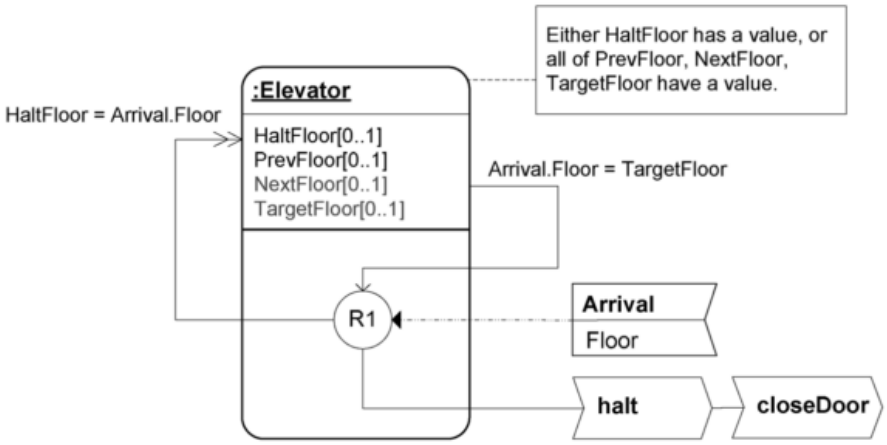


Figura 11: Diagrama de Padrão de Interações com Regras de reação (WAGNER, 2003)

Uma regra de reação tem exatamente uma seta de entrada com uma ponta de seta sólida que especifica o tipo de evento de acionamento. Pode haver também outras setas de entrada (sem a ponta de seta sólida) representando condições de estado (referindo-se às instâncias correspondentes de outros tipos de entidade). Existem dois tipos de setas de saída: uma que especifica efeitos mentais, mudanças em crenças e/ou compromissos(*beliefs / commitments*) e que é representada como uma seta de ponta dupla, e outra para especificar a realização de ações (físicas e comunicativas) que é conectada a tal *action event type* que ela especifica a ser desempenhada.

2.5.2 AOR Simulation

AOR *Simulation* é um *framework* ABS com uma abordagem orientada a agente que se propõe a ser um ambiente de simulação multi-propósitos. As categorias ontológicas de mais alto nível da Simulação AOR, mostradas na Figura 12, são mensagens (*Message*), eventos (*Event)* e objetos (*Object*), o que inclui agentes (*Agent*), objetos físicos (*PhysicalObject*) e agentes físicos (*PhysicalAgent*). O ambiente permite a especificação de um modelo de simulação em uma linguagem declarativa baseada em XML estudada a seguir neste mesmo capítulo.

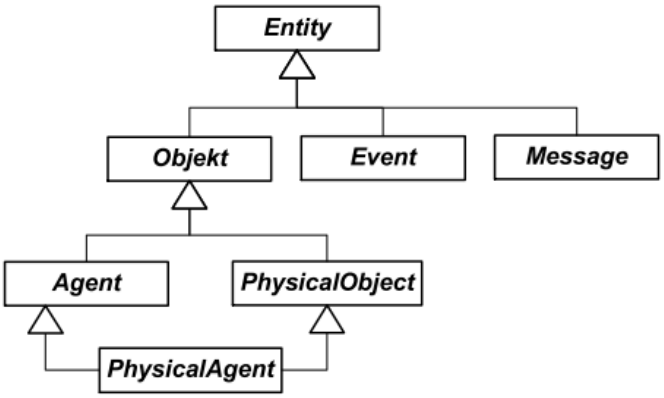


Figura 12: Categorias Ontológicas do ambiente AOR *Simulation*, (WAGNER; DIACONESCU, 2009)

O Simulador AOR (com arquitetura e modelos de execução definidos em (WAGNER, 2003)) pode ser implementado em diferentes linguagens de programação. Estão disponíveis duas implementações do simulador no *website* do projeto [1]: uma em Java e outra em Java Script. A especificação do modelo de simulação (*Simulation Scenario*) é a mesma independente do simulador e, como visto na Figura 13, pode ser traduzido e utilizado em diferentes plataformas.

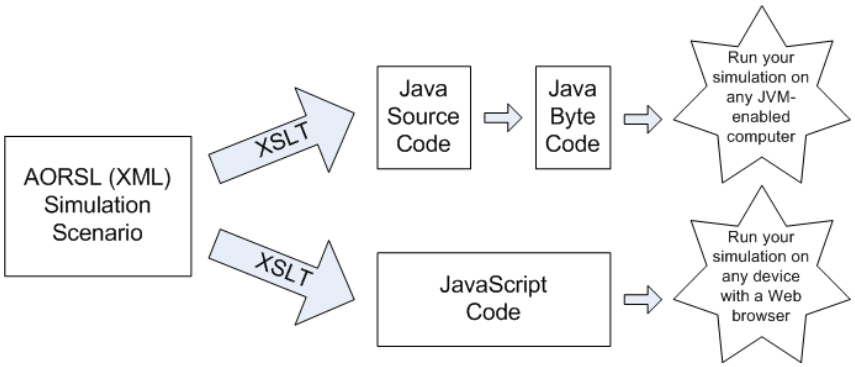


Figura 13: Aspectos da simulação

[1] http://http://oxygen.informatik.tu-cottbus.de/aor/?q=node/2

AORSL - *AOR Simulation Language*

A *Agent-Object-Relationship Simulation Language* (AORSL) é uma linguagem declarativa de alto nível baseada em XML que permite a especificação do modelo de simulação utilizado no *framework* de Simulação AOR. Esta linguagem permite expressar a essência da lógica de simulação com constructos de alto nível (como regras de comportamento de agentes) e definir tratamentos, processamentos, especiais com trechos de código com expressões em linguagem de programação. A documentação completa da AORSL pode ser encontrada em (WAGNER, 2011).

O elemento principal da AORSL é o ***SimulationScenario*** (Cenário de Simulação) que consiste essencialmente dos elementos *Simulation Model* (Modelo de Simulação), *InitialState* (Estado Inicial) e *UserInterface* (Interface com Usuário).

Os principais elementos do modelo de simulação (***SimulationModel***) são um conjunto de definições de tipos de entidades (*EntityTypes*) e um conjunto de regras de ambiente(*EnvironmentRule*) que afetam o ambiente.

EntityType

As *EntityTypes* incluem diferentes categorias de eventos, mensagens, objetos e agentes. Alguns listados a seguir:

ObjectType: Entidades passivas, sem ação, que podem conter atributos e referências à outros objetos.

AgentType: Entidades ativa com capacidade de (re)ação, percepção, crenças e outras.

ExogenousEventType: Evento que acontece com certa periodicidade. Pode conter atributos e referências à outros objetos.

CausedEventType: Evento causal de alguma regra de ambiente.

EnvironmetRule

As *EnvironmentRules* definem leis de causalidade que governam as mudanças de estado do sistema. As regras são expressas como variáveis 6-tuplas (WHEN, FOR, DO, IF, THEN, ELSE) detalhadas a seguir:

When: uma expressão de evento que especifica o tipo de evento que dispara a regra;

For: um conjunto de declaração de variáveis onde cada uma está ligada a um objeto específico ou a um conjunto de objetos;

If: uma fórmula de condição lógica possivelmente contendo variáveis;

Do, Then e Else: são os elementos de execução que consistem de dois blocos:

Update-Env: expressão especificando a atualização de um estado de ambiente;

Schedule-Evt: especifica uma lista de eventos resultantes que serão agendados.

2.6 Transformação de Modelos

No trabalho com modelos, uma ação importante é a manipulação de modelos para construção de outros. Diferentes modelos podem representar aspectos do mesmo domínio e, até por dependência de plataformas, domínios podem ser modelados de formas diferentes. Assim, segundo Almeida (2006), especificações de transformação de modelos especificam como objetos de diferentes modelos se relacionam, podendo, então, produzir modelos-alvos baseados em modelos-fontes transportando seus significados representados.

Para dar suporte à manipulação de modelos, foi criado pelo *Object Management Group* (OMG) o padrão MDA-*Model Driven Architecture* (MILLER; MUKERJI, 2003). Este padrão considera três tipos de visualização de um sistema: Modelo Computacionalmente Independente (CIM - *Computational Independent Model*), Modelo Independente de Plataforma (PIM - *Platform Independent Model*) e Modelo de Plataforma Específica (PCI - *Platform Specific Model*).

O padrão MDA estabelece que o CIM é uma visualização do sistema independente de aspectos computacionais com foco no domínio e, assim, assume um papel importante estabelecendo comunicação entre o especialista de modelagem e especialista do domínio. Também é chamado de modelo de domínio.

Já o PIM é um modelo computacionalmente dependente. Foca na operação do sistema ainda com certa independencia da plataforma a fim de ser utilizada em diferentes plataformas com certa similaridade.

PSM foca no uso de uma plataforma específica. Combina especificações oriundas do PIM com detalhes que especificam como este trabalha em determinado tipo de plataforma.

A Figura 14 mostra o padrão de transformação baseado em MDA. Nota-se que a transformação é especificada no nível de metamodelo. Esse mapeamento entre as entidades relacionadas nos dois modelos pode acontecer no nível de instância, modelo ou metamodelo. Para uma transformação genérica e única para diversos domínios, é considerando o nível de metamodelo já que este traz a significação de todas as possíveis instâncias do metamodelo e do modelo e, assim, qualquer modelo instância deste pode ser transformado.

Ainda na Figura 14, temos os modelos-fonte (modelo de entrada) é do tipo PIM e o modelo-alvo (modelo de saída) é do tipo PSM. O modelo de entrada traz as especificação alto nível do domínio enquanto o modelo de saída já incorpora detalhes da implementação na plataforma específica.

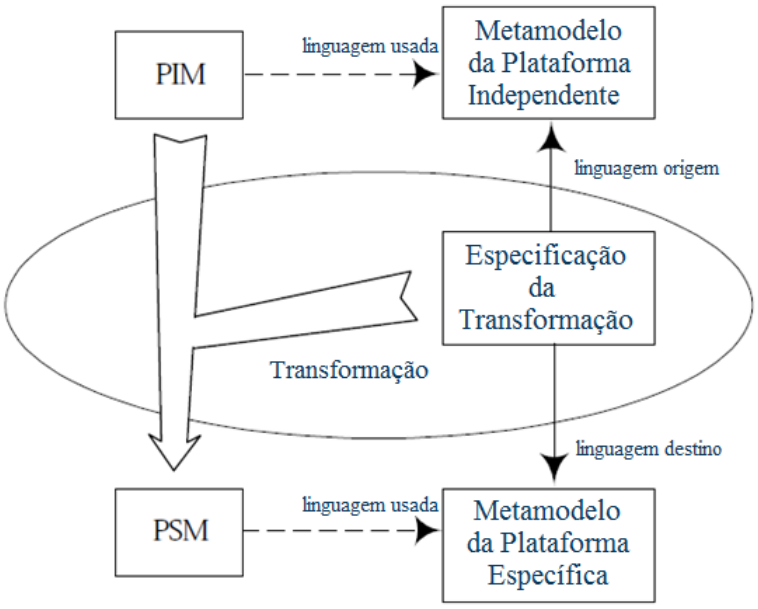


Figura 14: Aspectos da transformação de modelos MDA. Adaptado de (MILLER; MUKERJI, 2003)

No escopo deste projeto, trabalha-se com dois diferentes modelos que podem se relacionar. O primeiro é o modelo-fonte proveniente da especificação de contextos e situações com a SML (Seção 2.4) e o segundo é o modelo-alvo de simulação utilizado pelo ambiente de simulação AOR com a linguagem AORSL (Seção 2.5). Para que um modelo de situação descrito em SML possa ser simulado no ambiente de simulação AOR, é necessário que haja uma transformação de modelos para que as informações descritas em SML sejam traduzidas para a linguagem de simulação AORSL e representem, assim, corretamente o ambiente de simulação para o domínio descrito em SML.

3 Especificação da Transformação Entre Modelos SML e AORSL

A especificação e desenvolvimento de uma trasformação entre modelos começa com o estudo do signficado conceitual das entidades (e das relações entre elas) de cada modelo para, então, relacioná-los de forma a transportar conceitos possíveis de serem expressos em um modelo para o outro. Porém, os metamodelos apresentados no Capítulo 2.4 (Figuras 3,4, 5, 6, 7 e 8) não foram criados focados na representação dos mesmos domínios e possuem até mesmo objetivos de aplicação, de uso, diferentes. Assim, trazem uma perspectiva diferente na interpretação do mundo.

A transformação de modelos desenvolvida neste trabalho funciona no padrão MDA (MILLER; MUKERJI, 2003) no nível de metamodelo, ou seja, o mapeamento das relações entre SML e AORSL acontece com objetos do metamodelo. A SML é uma linguagem para especificação de contextos e situações. Suas entidades representam, ao seu modo, conceitos significativos relativos a contextos e situações. Já a AORSL é uma linguagem de descrição de ambientes genéricos em forma de modelos de simulação e, assim, leva em consideração aspectos tecnológicos de seu ambiente de execução. Além disso, permite a descrição de um domínio maior que a SML. Desta forma, percebe-se que um modelo pode ser mais expressivo para alguns aspectos do domínio do que outro. Enquanto em SML as classes tem significados substanciais, são conceitualmente específicas, em AORSL as classes do metamodelo, pela natureza e proposta do *framework*, tem significados mais genéricos podendo representar uma maior variadade de conceitos.

São trabalhadas neste capítulo as relações entre os objetos dos metamodelos SML para especificação de contextos e situações e do metamodelo AORSL. Sendo a SML uma linguagem de descrição, ao ser transformada para AORSL são trabalhados também outros aspectos além da descrição do domínio, como os relativos à simulação e à tecnologia do ambiente de simulação.

3.1 Mapeamento entre os modelos

Como visto na Seção 2.4, a SML foi desenvolvida especificamente para representar contextos e situações. A AORSL é uma linguagem de um ambiente com o propósito de simulação genérica e, assim, suas entidades tem significados menos específicos do que SML, cujo domínio é restrito. Assim, se faz uso de entidades com significados mais abrangentes em AORSL para representar entidades de significado mais específico em SML.

É exibido na Tabela 2 o mapeamento entre as entidades SML e AORSL com

um resumo do motivo de tal relação. Nas seções a seguir o mapeamento é detalhado e exemplificado.

SML	AORSL
SMLModel	**SimulationScenario**
SituationType	**ObjectType** - *Objeto utilizado para registro dos dados da situação como as entidades que a compõe e informações temporais de ocorrência da mesma.*
EntityParticipant	**ObjectType** - *Em SML as entidades são modeladas sem comportamento ativo e, assim, são mapeados para objetos passivos em AORSL.*
RelatorParticipant	**ObjectType**
SituationParticipant	*Não é necessário mapea-la em AORSL pois o ObjectType originado pela SituationType da situação em questão atende o propósito de SituationParticipant.*
ComparativeRelation	**EnvironmentRule.IF** - *Como a ComparativeRelation expressa restrição para caracterização da situação, seu mapeamento em AORSL se dá na construção da condicional da tag IF da EnvironmentRule criada para a detecção de tal situação.*
Link	**ObjectType.ReferenceProperty** - *A classe Link associa contextos relacionais com as entidades relacionadas. Esta classe é mapeada em AORSL, então, como uma ReferenceProperty da classe ObjectType do contexto relacional para fazer referências às entidades associadas a este.*
Literal	**EnvironmentRule.IF** - *A classe Literal é utilizada para especificar restrições nos valores associados às entidades e, assim, participa da construção do condicional IF.*
SituationType **Parameter**	*Não é necessário mapea-la em AORSL pois o ObjectType originado pela SituationType atende o propósito de SituationTypeParameter.*
SituationParameter **Reference**	*Não é necessário mapea-la em AORSL pois o ObjectType originado pela SituationType atende o propósito de SituationParameter.*
Function	*Não implementada*
ExistSistuation	*Não implementada*

Tabela 2: Resumo do mapeamento entre as entidades SML para entidades AORSL

3.1.1 SML:SituationType

A classe *SituationType* em SML caracteriza a ocorrência de uma determinada situação. Em transformação em AORSL são registradas as propriedades padrão da ocorrência

de tal situação (momento de início, momento de fim e duração) e, sendo a *situação* a denominação dada ao estado específico de algumas entidades, são registradas também as entidades responsáveis pela ocorrência de tal situação.

SituationType é mapeada, então, como *ObjectType* no modelo AORSL. Tal entidade permite a criação de atributos (*Attributes*) que podem armazenar as propriedades básicas padrão de todas as situações (momento de início, fim e duração) e permite, também, fazer referência a outros objetos.

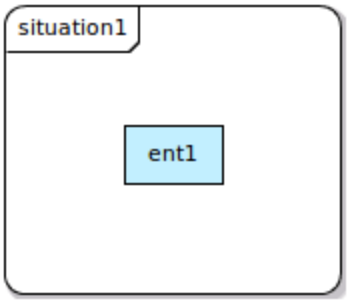


Figura 15: Representação em SML de uma situação (*situation1*) composta por uma entidade (*ent1*)

No modelo SML da Figura 15 há a descrição de uma situação pela classe *SituationType* instanciada com nome *situation1* composta por uma entidade participante (classe *EntityParticipant*) com nome *ent1*. Para tal situação, a transformação resultante para AORSL, vista no Código 3.1, contém um *ObjetType* relativo à situação com os atributos padrões das situações - duração (*duration*), momento de fim (*finalTime*) e momento de início da situação (*startTime*). A AORSL contém suporte a tipos de dados de data e hora mas como este é um simulador discreto, baseado em passos (*steps*), opta-se pelo tipo de dados Inteiro (*Integer*) para os três atributos mencionados já que assim, seguindo o padrão de número de passos, a referência ao tempo de simulação é mais direta e intuitiva.

Código 3.1: Especificação da situação *situation1* do modelo da Figura 15 em AORSL

```
1    </EntityTypes>
2      <ObjectType name="sit1">
3        <Attribute name="duration" type="Integer"/>
4        <Attribute name="finalTime" type="Integer"/>
5        <Attribute name="startTime" type="Integer"/>
6        <ReferenceProperty name="ent1" type="ent1"/>
7      </ObjectType>
8    </EntityTypes>
```

Para fazer referência às outras entidades que compõem esta situação utiliza-se a propriedade *ReferenceProperty*. No exemplo da Figura 15 temos somente uma entidade compondo a situação, *ent1*, e, então, é somente esta entidade que é referenciada pelo objeto representativo da situação através do atributo *ReferenceProperty*.

Convém ressaltar que esta seção aborda o registro de ocorrência das situação. A detecção das mesmas é explicada adiante na Seção 3.1.4.

3.1.2 SML:Participant

A classe *Participant* em SML corresponde aos principais elementos que compõem uma situação. Pode ser especializada em *EntityParticipant*, *RelatorParticipant* and *SituationParticipant*.

Em SML, as classes *Participant* são entidades que possuem características de estado - parâmetros de contexto - e não possuem comportamento ativo. Assim, mesmo tendo modelado uma entidade que no mundo real possa ter comportamento ativo, em SML ele não é assim modelado e, então, somente são registradas suas propriedades de descrição de contexto. Em AORSL essas entidades são mapeadas em *ObjectTypes* como detalhado nas seções seguintes.

3.1.2.1 SML:EntityParticipant

EntityParticipant é uma especialização de *Participant* e representa a ocorrência de uma entidade na especificação de um tipo de situação. Como discutido na Seção 3.1.2, não representa uma entidade de comportamento ativo mas sim um objeto ordinário, passivo. Sua representação em AORSL então se faz representando seu tipo e seus atributos respectivos em um *ObjectType*.

A Figura 15 mostra a situação *situation1* composta pela declaração de uma entidade *EntityParticipant* chamada *ent1*. Este modelo de situação em SML é transformado no Código 3.2 em AORSL, em que se observa que é criada uma entidade do tipo *ObjectType* com o nome *ent1*. É também criado um atributo *ReferenceProperty* para fazer referência à situação a que aquela entidade pode fazer parte. Isso porque instâncias deste objeto podem fazer parte de diferentes ocorrências deste tipo de situação.

Código 3.2: Especificação da entidade *ent1* do modelo da Figura 15 em AORSL

```
1 <EntityTypes>
2    <ObjectType name="ent1">
3      <Attribute name="ent1attribute1" type="Integer"/>
4      <ReferenceProperty name="situation1" type="situation1"/>
5    </ObjectType>
6  </EntityTypes>
```

Nota-se que mesmo não tendo explicitado os atributos da entidade *ent1* no modelo de situações, o objeto criado contém o atributo *ent1attribute1*. Na Seção 2.4.2 se discutiu que um modelo de situações em SML é baseado em um modelo de contexto já que situações são estados específicos das entidades que compõem o contexto. Desta forma, as *EntityParticipants* (modelo de situações) têm seu tipo definido por classes do modelo

de contexto (*EntityClass*) carrengando, assim, todas as características da *EntityClass* relacionada. No caso em questão, mesmo não utilizando o atributo da entidade para definição da situação, ele faz parte do significado da entidade e, assim, é modelado para o ambiente de simulação.

Ainda no Código 3.2, o objeto *ent1* contém a *tag ReferenceProperty*. Esta propriedade faz referência a outro objeto que neste caso é o objeto referente à situação que esta entidade pode compor.

3.1.2.2 SML:RelatorParticipant

A *RelatorParticipant* é usada para descrever contextos relacionais e representa o relacionamento entre diferentes entidades. Em AORSL, então, sua representação se dá por um *ObjectType*, passivo, com registro de seus atributos e referência aos tipos de entidades que ele relaciona.

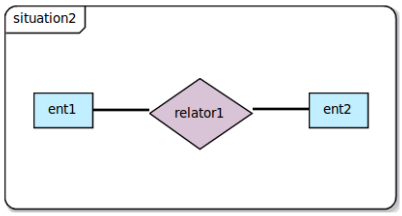


Figura 16: Representação de uma situação com uma entidade RelatorParticipant em SML

A Figura 16 traz um exemplo do uso da classe *RelatorParticipant* em SML compondo a situação *situation2*. Essa situação é caracterizada quando há uma relação *relator1* entre as entidadaes *ent1* e *ent2*. Este modelo transformado para AORSL é demonstrado no Código 3.3. Em tal código são declaradas as *EntityParticipants ent1* e *ent2* como *ObjectType* e posteriormente a *RelatorParticipant relator1* também como *ObjectType*.

Assim como para *EntityParticipant*, a classe *RelatorParticipant* tem seu tipo descrito no modelo de contexto - neste caso, pela classe *RelatorClass*. O atributo *relatorAttribute1* presente no código AORSL, apesar de não especificado no modelo de situação, é parte do tipo *RelatorClass* a que pertence este *RelatorParticipant* e, assim, é especificado em AORSL pela transformação desta entidade.

Código 3.3: Representação das entidades *RelatorParticipant* e *EntityParticipants* do modelo da Figura 16 em AORSL

```
1  <EntityTypes>
2    <ObjectType name="ent1">
3      <Attribute name="ent1attrbiute1" type="Integer"/>
4      <ReferenceProperty name="situation2" type="situation2"/>
5    </ObjectType>
6    <ObjectType name="ent2">
7      <Attribute name="ent2attribute1" type="Integer"/>
```

```
8       <ReferenceProperty name="situation2" type="situation2"/>
9     </ObjectType>
10    <ObjectType name="relator1">
11      <Attribute name="relatorAttribute1" type="Integer"/>
12      <ReferenceProperty name="ent2" type="ent2"/>
13      <ReferenceProperty name="ent1" type="ent1"/>
14    </ObjectType>
15  <EntityTypes>
```

3.1.2.3 SML:SituationParticipant

Situações podem ser compostas pela ocorrência de outras situações. Para tal, especializa-se a classe *Participant* em *SituationParticipant* fazendo referência a uma situação (SituationType) já existente. Na linguagem concreta de SML, a composição de situações é feita como mostrado na Figura 17 onde temos a definição da situação *sit2* utilizando a situação *sit1* para ser caracterizada.

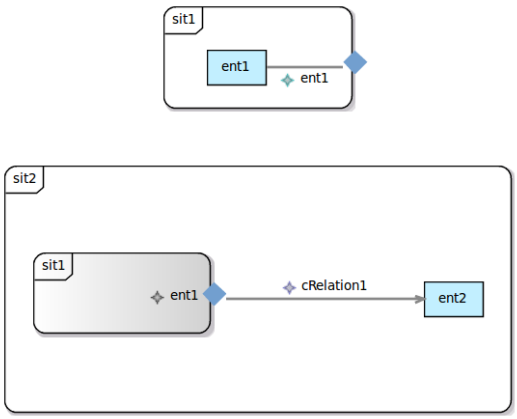


Figura 17: Representação de situações em SML para representar, em *sit2*, o uso de situações compostas.

No Código 3.4 temos a declaração da entidade referente à situação *sit2* pela transformação de SML para AORSL. Neste objeto temos um atributo de referência (*PropertyReference*) à situação participante *sit1*. A inserção desta referência no objeto da situação *sit2* é suficiente para a modelagem em AORSL. Isto porque assim pode-se verificar a existência da instância daquela situação *sit1* e, com referência à própria, acessar seus atributos e entidades.

No exemplo em questão, a entidade *ent1* da situação *sit1* é acessada e utilizada em uma comparação pela relação de comparação *cRelation1*. Esta possibilidade de acesso a determinadas entidades de uma situação (caracterizada pelo losango na borda da situação ligado à entidade que se quer dar acesso) é feita em SML com o uso das classes *SituationTypeParameter* na situação *sit1* para indicar a possibilidade desta entidade ser utilizada na composição de outras situações, e *SituationParameterReference* na situação *sit2* para fazer referência à *SituationTypeParameter* da situação participante.

Código 3.4: Representação em AORSL da situação *sit2* do modelo da Figura 17

```
<EntityTypes>
  <ObjectTypename="sit2">
    <Attribute name="duration" type="Integer"/>
    <Attribute name="finalTime" type="Integer"/>
    <Attribute name="startTime" type="Integer"/>
    <ReferenceProperty name="ent2" type="ent2"/>
    <ReferenceProperty name="sit1" type="sit1"/>
  </ObjectType>
</EntityTypes>
```

3.1.3 SML:ComparativeRelation

Esta entidade SML explicita restrições das entidades na caracterização de situações e sua transformação para AORSL resulta em condicionais na detecção da situação à qual pertence. É implementada a transformação de *ComparativeRelations* para relação entre os tipos *AttributeReference* (das classes *Participant*) e *Literal*, e entre duas entidades *AttributeReference*. Sua transformação em condicionais é mostrada na Seção 3.1.4 onde é especificada a criação de regras de ambiente (*EnvironmentRule*) na transformação para detecção de situações.

3.1.4 Detecção de Situações

Cada situação descrita em SML gera em AORSL outros elementos além do *ObjectType* que é utilizado para registro da ocorrência da situação (Seção 3.1.1). Em AORSL, para fins de execução da simulação, é necessário descrever como tais situações serão detectadas. Assim, para cada situação especificada no modelo SML, é criado em AORSL um elemento que verifica as condições que caracterizam aquela situação. Este elemento é o tipo *EnvironmentRule*, uma regra de ambiente que é executada toda vez que determinadas condições são atingidas. Tais condições, no escopo desta transformação, são justamente as restrições sobre o modelo para que seja caracterizada determinada situação.

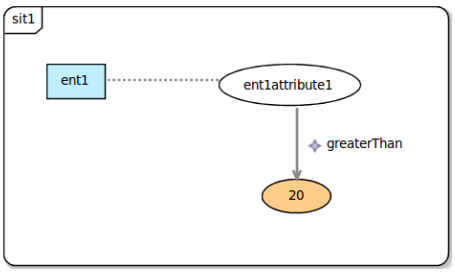


Figura 18: Especificação da situção *sit1* em SML

O modelo da Figura 18 apresenta uma entidade (*ent1*) com seu atributo *ent1attribute1* explicitado. Existe uma relação comparativa (*ComparativeRelation*) chamada *greaterThan*

(maior que) entre o atributo da entidade e uma outra entidade do tipo *Literal* cujo valor é 20. Esta é a descrição da situação *sit1*.

Esta situação gera em AORSL as entidades já descritas anteriormente (*ObjectTypes* para a *ent1* e *sit1*) e, agora, uma regra de ambiente (*EnvironmentRule*) para detecção de tal situação.

A *EnvironmentRule* é programada a reagir quando as regras de restrições daquela situação são satisfeitas. Para detecção da situação descrita na Figura 18, cria-se o Código 3.5 em AORSL. Nota-se o *IF* para verificar a condição de o atributo *ent1attribute1* da entidade *ent1* ser maior (*greaterThan*) que o valor da entidade *Literal* (20). As classes do tipo *ComparativeRelation* da Seção 3.1.3 são transformadas em condicionais desta regra de ambiente pois representam, justamente, restrições para caracterização da situação.

Código 3.5: Representação da regra de ambiente (*EnvironmentRule*) para detecção da situação modelada na Figura 18

```
<EnvironmentRule name="sit1Situation">
  <WHEN eventType="Step" eventVariable="stepEvent"/>
  <FOR-ObjectVariable variable="ent1" objectType="ent1" />
  <IF language="Java JavaScript"><![CDATA[ this.ent1.getEnt1attribute1() > 20 ]]></IF>
  <THEN>
    <SCHEDULE-EVT>
      <CausedEventExpr eventType="sit1Detected">
        <Slot property="ent1">
          <ObjectValueExpr objectVariable="ent1"/>
        </Slot>
      </CausedEventExpr>
    </SCHEDULE-EVT>
  </THEN>
</EnvironmentRule>
```

A *tag FOR-ObjectVariable* faz com que se percorra todas as instâncias de tipo *ent1* para verificação de atendimento às condições da situação. Ou seja, é verificado em todas as instâncias do objeto especificado se a condicional descrita na *tag IF* é satisfeita. Caso positivo, é agendado um evento (*SCHEDULE-EVT*) do tipo *sit1Detected* reportando que a situação *sit1* foi detectada. Tal evento (*CausedEventExpr*) guarda referência às instâncias das entidades envolvidas na situação.

Esse evento disparado pela regra de ambiente é descrito como uma *EntityType* (Código 3.6). Esta entidade existe para mostrar que a *sit1* foi detectada e quais entidades fazem parte da ocorrência daquela situação pelo atributo *ReferenceProperty*. Como na situação da Figura 18 há somente uma entidade, no *CausedEventType* há somente o registro de uma propriedade de referência (*ReferenceProperty*).

Código 3.6: Representação em AORSL do evento disparado quando a situação modelada na Figura 18 é detectada

```
<EntityTypes>
```

```
2   <CausedEventType name="sit1Detected">
3     <ReferenceProperty name="ent1" type="ent1"/>
4   </CausedEventType>
5 <EntityTypes>
```

Uma especificação de situação gera também uma outra regra de ambiente mostrada no Código 3.7. Esta regra é ativada quando ocorre um evento do tipo *sit1Detected* que é gerado quando a situação é detectada. A condição *IF* desta regra verifica se esta situação é nova. Isto é feito conferindo se não existe ainda situação para aquela entidade ou se para aquela entidade o último registro daquela situação é antigo. Caso nova (*THEN*), precisa-se criar uma nova instância da entidade *sit1* - através da *tag Create*. Caso esta seja uma situação que já esteja acontecendo (*ELSE*), então a duração e o tempo final da situação são atualizados para o tempo (*step*) atual.

Código 3.7: Representação em AORSL da regra de ambiente (*EnvironmentRule*) para atualização das informações do registro da situação modelada na Figura 18

```
1 <EnvironmentRule name="sit1SituationUpdate">
2   <WHEN eventType="sit1Detected" eventVariable="sit1Detected"/>
3   <FOR objectVariable="ent1" objectType="ent1" >
4     <ObjectRef language="Java␣JavaScript"><![CDATA[this.sit1Detected.getEnt1()]]></
          ObjectRef>
5   </FOR>
6   <IF language="Java␣JavaScript"> <![CDATA[ sit1Detected.getEnt1().getSit1() == null  ||
        this.sit1Detected.getOccurrenceTime() > sit1Detected.getEnt1().getSit1().
        getFinalTime() + 10 ]]>  </IF>
7   <THEN>
8     <UPDATE-ENV>
9       <Create>
10        <Object type="sit1" objectVariable="sit1"/>
11      </Create>
12      <UpdateObject>
13        <ObjectRef language="Java␣JavaScript" objectType="ent1"><![CDATA[this.
              sit1Detected.getEnt1()]]></ObjectRef>
14        <Slot property="sit1">
15          <ObjectValueExpr objectVariable="sit1"/>
16        </Slot>
17      </UpdateObject>
18      <UpdateObject objectVariable="sit1">
19        <Slot property="ent1">
20          <ObjectValueExpr objectVariable="ent1"/>
21        </Slot>
22        <Slot property="finalTime">
23          <ValueExpr language="Java␣JavaScript"/><![CDATA[this.sit1Detected.
                getOccurrenceTime()]]></ValueExpr>
24        </Slot>
25        <Slot property="startTime">
26          <ValueExpr language="Java␣JavaScript"/><![CDATA[this.sit1Detected.
                getOccurrenceTime()]]></ValueExpr>
27        </Slot>
28      </UpdateObject>
29    </UPDATE-ENV>
30  </THEN>
31  <ELSE>
32    <UPDATE-ENV>
```

```
33        <UpdateObject>
34          <ObjectRef language="Java␣JavaScript" objectType="sit1"><![CDATA[this.
                sit1Detected.getent1().getsit1()]]></ObjectRef>
35          <Slot property="duration">
36            <ValueExpr language="Java␣JavaScript"/><![CDATA[this.sit1Detected.
                  getOccurrenceTime() - this.sit1Detected.getent1().getsit1().getStartTime()
                  ]]></ValueExpr>
37          </Slot>
38          <Slot property="finalTime">
39            <ValueExpr language="Java␣JavaScript"/><![CDATA[this.sit1Detected.
                  getOccurrenceTime()]]></ValueExpr>
40          </Slot>
41        </UpdateObject>
42      </UPDATE-ENV>
43    </ELSE>
44 </EnvironmentRule>
```

O código completo desta transformação é encontrado no Apêndice A.

3.2 Implementação / Especificação de Transformação

A transformação foi desenvolvida em Java em ambiente Eclipse EMF - *Eclipse Modeling Framework* [1]. Na implementação da transformação é interessante que se tenha disponível as classes de domínio baseadas nos metamodelos em que se trabalha para melhor manipulação das entidades dos modelos e especificação do mapeamento.

As ferramentas desenvolvidas para utilização da SML, seus metamodelos e o código fonte estão disponíveis no site do projeto *Situation Modeling Language* [2]. Para a SML, os metamodelos e as classes de domínio com os métodos de acesso das mesmas já estão implementadas e prontas para uso. Assim, precisa-se somente importá-las para uso na aplicação.

No caso da AORSL, apesar da especificação da sintaxe da linguagem, não há um metamodelo disponível. Assim, foi necessário especificá-lo baseado na própria sintaxe da linguagem disponível em (WAGNER, 2011). Criado o metamodelo no próprio ambiente Eclipse EMF, foi gerado código Java das classes de domínio do metamodelo AORSL. Tais classes de domínio são importadas pela aplicação para uso na especificação da transformação.

É importante ressaltar que AORSL é uma linguagem abrangente e permite a especificação de um domínio maior na SML. Assim, não se faz uso de todas as entidades disponíveis no metamodelo AORSL para especificação de domínios SML. Desta forma, o metamodelo criado para dar suporte à implementação da transformação é restrito às entidades necessárias para transportar os conceitos presentes em SML.

[1] http://www.eclipse.org/modeling/emf/
[2] http://izontm.github.com/SML/

A especificação de um modelo de situação pode ser realizada graficamente pelas ferramentas disponíveis do projeto SML. Mas, por fim, o modelo é salvo em um arquivo com a extensão *.sml* baseado em XML. Este arquivo contém toda a descrição do modelo de situações e as referências ao modelo de contexto. Este arquivo é acessado pela aplicação e seus elementos são lidos. Consulta-se o tipo do elemento no metamodelo de situações e contexto e, pelo mapeamento, é transformado para classes referentes no metamodelo AORSL.

Assim, o cenário de simulação AORSL primeiro é carregado em memória nas classes Java para depois a geração do arquivo de saída contendo o código AORSL no formato *.xml*.

É necessário ainda que alguns aspectos relativos à simulação sejam levados em consideração para execução do modelo AORSL. Essas considerações são feitas na Seção 3.3.

3.3 Aspectos práticos da simulação

Neste trabalho foi mapeada e posteriormente implementada a transformação de situações representadas em modelo SML para o modelo de simulação em linguagem AORSL levando em consideração a detecção de tais situações no ambiente de simulação.

Não é implementado, porém, um ambiente para interação com o usuário em tempo de execução da simulação. Este ambiente tem importância já que possibilitaria a alteração dos estados das entidades da simulação dinamicamente pelo usuário. A plataforma de simulação AOR em Java (AOR-JavaSim) não provê suporte a esta funcionalidade. Já o ambiente de simulação online baseado em *JavaScript* (Simurena) suporta o objeto *AgentControlUI* que poderia ser usado para este tipo de interação com o usuário apesar de não ter especificamente esta finalidade. Porém, a implementação deste objeto é recente e mesmo o ambiente *online* de simulação ainda não suporta todas as funcionalidades deste objeto. Desta forma, apesar de a transformação gerar código de especificação de modelo de simulação pronto para funcionamento e detecção de situações, a interação com o usuário é prejudicada já que essas funcionalidades ainda não foram implementadas integralmente no simulador.

Uma forma de executar e testar a especificação do ambiente de simulação é programando previamente de forma manual testes com rotinas de modificações dos estados dos objetos em AORSL para que sejam criados os estados necessários para caracterização da situação em teste.

Para teste da situação expressa na Figura 18, por exemplo, pode ser gerada uma regra de ambiente (*EnvironmentRule*) como especificado no Código 3.8, que incrementa a

cada passo da simulação o valor do atributo *ent1attribute1*. Assim, pela regra de ambiente já especificada no Código 3.5, quando o valor do atributo for maior que o *Literal* de valor 20, a situação será detectada.

Código 3.8: Exemplo de regra de ambiente (*EnvironmentRule*) em AORSL utilizada para testar a simulação da situação modelada na Figura 18

```
<EnvironmentRule name="IncrementAttribute">
  <ON_EACH_SIMULATION_STEP/>
  <FOR_ObjectVariable variable="ent1" objectType="ent1" objectIdRef="1"/>
  <DO>
    <UPDATE_ENV>
      <UpdateObject objectVariable="ent1">
        <Increment property="ent1attribute1" value="1"/>
      </UpdateObject>
    </UPDATE_ENV>
  </DO>
</EnvironmentRule>
```

Essas regras de ambiente para variação das informações de contexto visando a simulação do modelo são utilizadas e novamente exemplificadas no estudo de caso deste trabalho, na Seção 4.4.

4 Estudo de Caso: Situações de Emergência

Neste capítulo, a fim de expor e exemplificar o funcionamento e a contribuição deste trabalho, é feito um estudo de caso com situações de emergência. Com a motivação da realização de grandes eventos esportivos de escopo internacional no Brasil nos anos de 2014 e 2016, o domínio escolhido para estudo assume grande importância e aplicabilidade.

Grandes eventos merecem especial atenção pois concentram um número elevado de pessoas em ambientes de tal maneira que incidentes como incêndios, problemas de infraestrutura ou outros distúrbios da ordem podem ter suas consequências potencializadas. Esforços para prevenir e minimizar os efeitos de possíveis incidentes são feitos e este estudo de caso visa a contribuir na gestão de incidentes com o processamento e criação de conhecimento sobre as situações de emergência.

Para a tomada de decisões na atuação de combate às situações de emergência, Diniz (2006) cita que são utilizados três tipos de conhecimento: conhecimento pessoal prévio, conhecimento formal prévio e conhecimento contextual atual. O conhecimento pessoal prévio é aquele que diz respeito à própria experiência das pessoas envolvidas. O conhecimento formal prévio é o conhecimento documentado, como por exemplo um plano de gerenciamento de emergências. O conhecimento contextual atual é o gerado durante a ocorrência da situação de emergência sobre a própria situação de emergência. A contribuição deste trabalho e estudo de caso está relacionada com o conhecimento contextual atual.

A detecção de alguns tipos de situações específicas pode ser feita de forma sistemática e a descrição da caracterização dessas situações podem ser feitas usando a SML. É de interesse, levando em consideração até o aspecto crítico do domínio de emergências, onde a possibilidade de falhas deve ser surpimida ao máximo para minimizar os danos, que essas descrições, esses modelos de situações sejam amplamente testados e validados. Isto para se ter segurança quando tais modelos forem utilizados na prática em situações reais e até para melhorar o entendimento explorando os comportamentos do modelo para suporte na fase de elaboração do mesmo. É neste momento que a possibilidade de simulação em ambiente computacional se torna relevante.

Assim, as tecnologias estudadas e a transformação de modelos entre elas feita neste trabalho viabiliza a descrição e posterior simulação de contextos e situações de emergência.

4.1 Planos de Gerenciamento de Emergências

Situações de emergência são acontecimentos críticos que envolvem uma grande quantidade de variáveis. Diversas causas podem iniciar um estado de emergência como incêndios, condições meteorológicas severas, problemas estruturais do ambiente e ameaças criminosas. Esforços para a prevenção de tais incidentes são grandes mas é inevitável que estes ocorram. Pela grande responsabilidade de preservar principalmente a vida humana, essas situações podem criar um ambiente de *stress* emocional nos organizadores e responsáveis pela segurança do evento dificultando a tomada de decisões para atuação no incidente que precisam ser rápidas e assertivas a fim de minimizar o impacto de tais acontecimentos.

Desta forma, além de existir recursos (humanos e materiais) disponíveis para o resposta ao incidente como saídas de emergência, sinalização adequada, equipes para socorro médico e equipes de apoio para suporte na evacuação de prédios, é necessário também um grande esforço no planejamento prévio da atuação no combate aos incidentes fazendo bom proveito de tais recursos. Assim, um estudo sobre as possibilidades de incidentes e as ações necessárias para mitigá-lo, relacionando as equipes envolvidas, os recursos necessários, as responsabilidades e ações para resposta ao incidente e se faz de extrema importância. Neste contexto surgem os planos de gerenciamento de emergência. Tais planos tem intenção de proteger a vida e minimizar danos. Detalham como a comunidade envolvida, os órgãos e equipes irão responder em caso de emergências. Contém informações e instruções de atuação para uma dada situação emergencial específica auxiliando nas tomadas de decisões e na indicação das ações a serem tomadas em situações críticas.

4.2 Especificação de Situações de Emergência

Para pôr em prática o que é especificado no plano de gerencimento de emergência, é necessário identificar corretamente e rapidamente a situação específica de emergência, já que a resposta a cada situação pode ter ações diferentes e até envolver atuadores diferentes. Por isso, é importante que se tenha muitas informações disponíveis para ajudar no processo de entendimento da situação que está a ocorrer. Alguns tipos de conhecimento são inerentes à capacidade cognitiva e intuitiva do ser-humano mas outros podem ser acessados e manipulados por máquinas para um processamento prévio das informações de maneira que se possa estabelecer conhecimentos mais refinados dos acontecimentos.

Desta forma, a descrição prévia de contextos e situações de emergência possíveis se faz importante para o suporte no processo de planejamento e execução de reposta às situações e, até mesmo, para a detecção de tais situações incluindo seu entendimento. Para tal, a SML provê ferramental importante para descrição de contextos e situações.

Importante ressaltar que uma situação de emergência como as citadas até então neste capítulo, engloba uma série de outras sub-situações mais específicas. Estas situações específicas, mais pontuais, que compõem a situação de estado de emergência, são deveras importantes para o entendimento de toda situação emergencial já que elas são detalhes do acontecimento geral. Assim, conhecer as mini situações é de muita importância para a atuação no problema porque podem indicar necessidades específicas.

4.2.1 Descrição das Situações

Considerando um evento em estádio de futebol ou em outra construção que comporte um grande público, trabalha-se com a possibilidade de uma situação de incêndio. Esta situação está associada com outras situações e seu conhecimento detalhado pode ser muito útil.

Assim, a primeira situação de interesse trata-se de um detector de incêndio ativado indicando um possível foco de incêndio (*PossibleFire*). Identificada esta situação, os gerentes podem iniciar algumas ações como um contato prévio com as equipes de bombeiros e resgate para que estes estejam a postos para a possível necessidade.

Como esta é uma situação grave é interessante que seja confirmada por um ser humano para que sejam iniciados os procedimentos necessários como a evacuação do prédio. Um agente deve verificar, então, no local ou proximidades do sensor de incêndio alarmado algum indício de fogo. Se confirmado, temos a situação de confirmação de incêndio (*FireDetected*) e, assim, uma indicação verdadeira para que os responsáveis possam dar inicio aos procedimentos de resposta à emergência (descritos no Plano de Gerenciamento de Emergência).

Uma outra situação relacionada é o caso de o foco de incêndio estar impedindo o acesso a alguma saída de emergência (*EmExitObstructed*). Esta é uma situação de extrema relevância pois se não observada pode agravar o problema no caso de necessidade de evacuação. Quando detectada, os responsáveis podem remanejar a saída do público por vias de emergência seguras através de agentes ou até mesmo automaticamente através das indicações visuais de saída de emergência.

4.2.2 Especificação das situações em SML

A modelagem dessas três situações em SML começa pela modelagem do contexto na qual elas estão inseridas. A Figura 19 apresenta esta modelagem. Nota-se a descrição das entidades presentes no contexto (*Entity Class*) e seus atributos (*Attribute*), as relações formais de comparação (*Comparative Formal Relation*) as definições dos tipos de dados utilizados neste contexto (*Data Type*), os tipos de contextos relacionais entre as entidades (*Relator Class*) e a definição das associações do contexto relacional com as entidades

(*Association*).

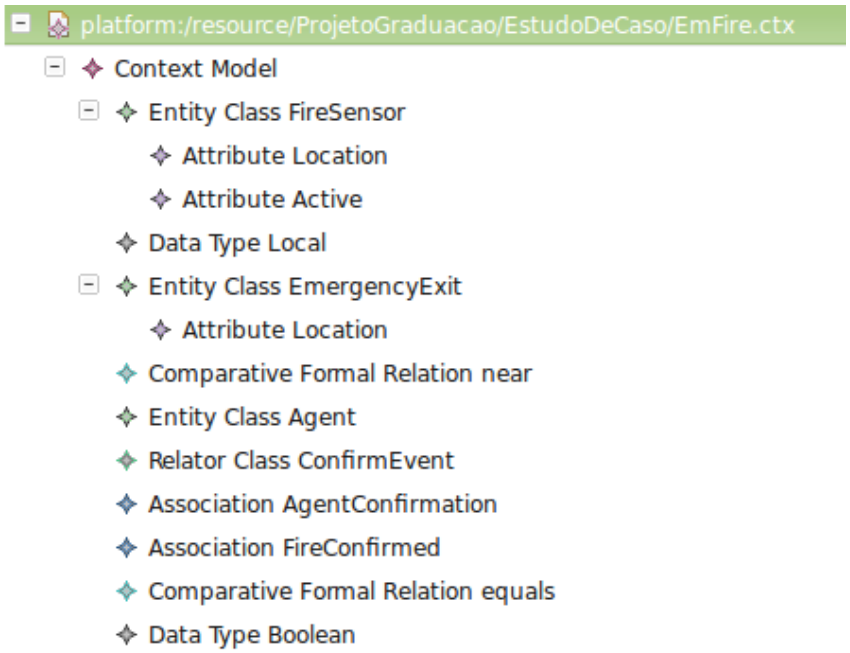


Figura 19: Especificação em SML do modelo de contexto das situações de emergência

Essas classes serão utilizadas para definição das situações de interesse como visto a seguir e, assim, o modelo de contexto precisa ser importado no modelo de situações.

A situação *PossibleFire* (Possível Fogo), Figura 20, indica uma possibilidade de incêndio. Esta é uma situação simples que ocorre no caso de alguma instância da entidade (*EntityParticipant*) para sensores de incêndio (*FireSensor*) ter o atributo *Active* (que indica detecção de incêndio) com valor verdadeiro (*true*). Nota-se que a entidade *FireSensor* pode ser acessada por outras situações devido o uso da classe *SituationTypeParamenter* representada pelo losango na borda da situação. Isto será importante para o uso desta situação na composição de outras.

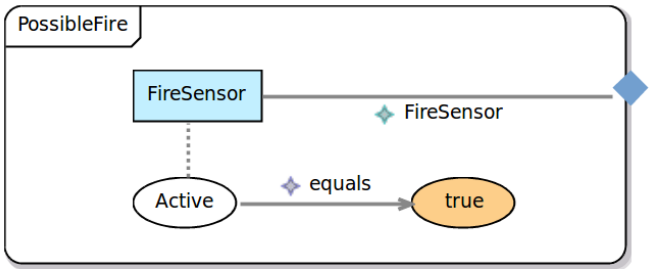


Figura 20: Especificação em SML da situção *PosibleFire* indicando possibilidade de incêndio

Na Figura 21 temos a modelagem da situação *FireDetected* (Fogo Detectado). Nesta situação, caso exista a ocorrência de uma situação do tipo *PossibleFire* e exista um agente humano através da entidade *EntityParticipant Agent* que confirme o evento acionado pelo

sensor de incêndio que compõe a situação *Possible Fire* pela entidade *RelatorParticipant ConfirmEvent*, é caracterizada a situação que indica a presença confirmada de fogo.

Esta situação pode servir para a tomada de decisões diversas como o acionamento de equipe de combate à incêndio e, sabendo a localização do fogo através do atributo *Location* da entidade sensor, até remanejar/indicar à equipe de combate o incêndios por qual entrada do estádio sua chegada deve ser feita.

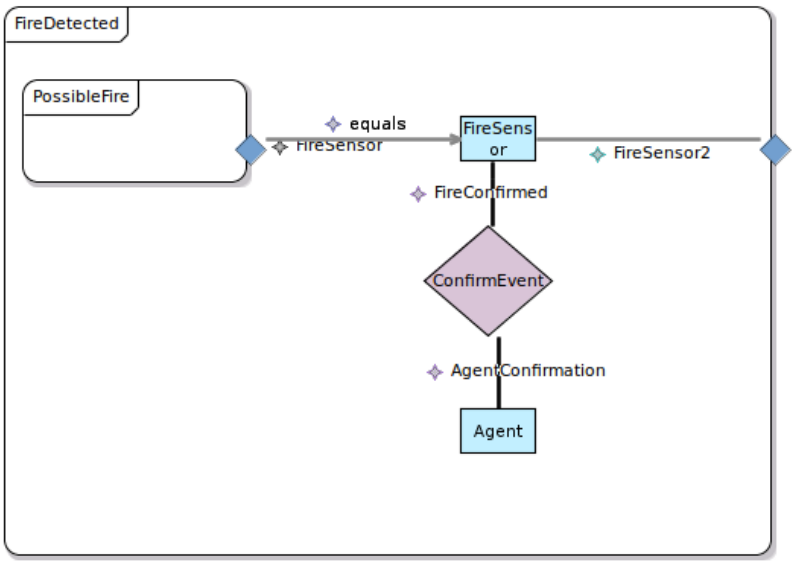


Figura 21: Especificação em SML da situção *FireDetected* indicando a confirmação de incêndio

Esta situação de indicação de confirmação de incêndio é ainda utilizada na composição de outra situação como visto na Figura 22. Esta figura indica uma situação de obstrução de alguma saída de incêndio devido ao fogo. Esta averiguação é feita comparando a localização do sensor de incêndio com a localização de alguma saída de emergência. Caso haja alguma saída de emergência próxima ao sensor aonde que detectou o incêndio, é caracterizada a situação de Saída de Emergência Obstruída pelo Incêndio (*EmExitObstructed*).

A indicação desta situação é importante para a tomada de decisão de resposta ao incidente. Pode, por exemplo, indicar a necessidade de um remanejamento simples na evacuação optando pelas portas e rotas de saída não obstruídas.

4.3 Transformação das Situações de Emergência

Com as situações de emergência definidas, podemos simulá-las transformando-as para o modelo AORSL. A transformação automática definida no Capítulo 3 gera o modelo AORSL que pode ser acessado por completo no Apêndice B.

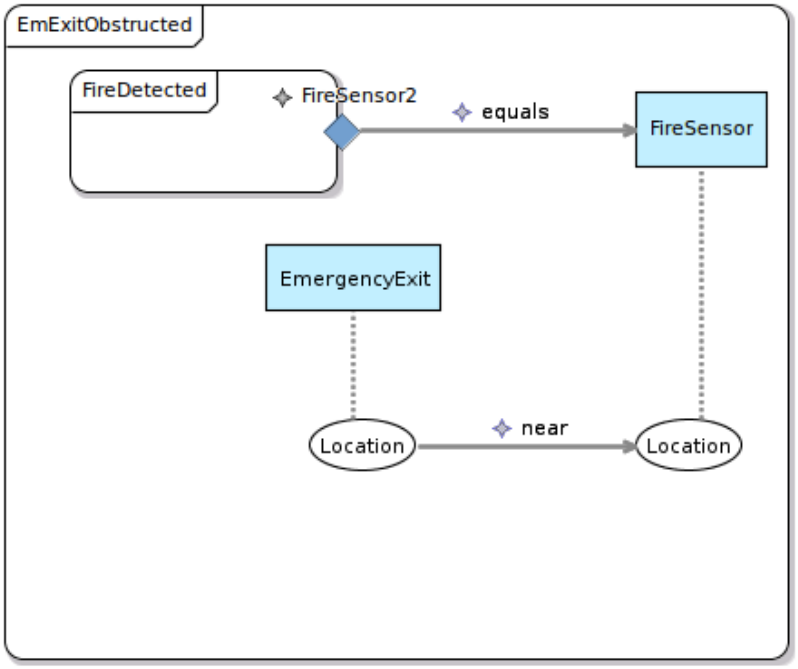


Figura 22: Especificação em SML da situção *EmExitObstructed* indicando a obstrução de uma saída de emergência por um incêndio

4.3.1 Situação *PossibleFire*

A situação *PossibleFire* indica a possibilidade de incêndio e é modelada na Figura 20. Para esta situação, são gerados basicamentes dois importantes trechos de código AORSL. No Código 4.1 são definidas as entidades relacionadas a esta situação.

A primeira delas é um evento causal (*CausedEventType*) chamado de *PossibleFire-Detected*. Este evento é instanciado por uma regra de ambiente para indicar a detecção de uma situação do tipo *PossibleFire*. Tal regra de ambiente é explicada posteriormente nesta seção.

A entidade seguinte é a que modela o objeto *FireSensor* (Sensor de Incêndio). Esta entidade contém dois atributos que descrevem se está ativo (detectando incêndio) e sua localização (*Active* e *Location*, respectivamente). Finalmente, três propriedades do tipo *ReferenceProperty* fazem referência às situações que este objeto pode fazer parte.

A última entidade modelada é o objeto que descreve a ocorrência de uma instância da situação *PossibleFire*. Além dos atributos padrões para tipos 'situação' (duração, tempo inicial e tempo final), este objeto faz referência à instância das entidades que o compõem, que o caracterizam - neste caso, somente *FireSensor*.

Código 4.1: Código AORSL das entidades oriundas da transformação da Situação *PossibleFire* da Figura 20

```
1  <EntityTypes>
2   <CausedEventType name="PossibleFireDetected">
```

```
3     <ReferenceProperty name="FireSensor1" type="FireSensor"/>
4    </CausedEventType>
5     <ObjectType name="FireSensor">
6     <Attribute name="Active" type="Boolean"/>
7     <Attribute name="Location" type="Integer"/>
8     <ReferenceProperty name="FireDetected" type="FireDetected"/>
9     <ReferenceProperty name="PossibleFire" type="PossibleFire"/>
10    <ReferenceProperty name="EmExitObstructed" type="EmExitObstructed"/>
11   </ObjectType>
12    <ObjectType name="PossibleFire">
13    <Attribute name="duration" type="Integer"/>
14    <Attribute name="finalTime" type="Integer"/>
15    <Attribute name="startTime" type="Integer"/>
16    <ReferenceProperty name="FireSensor1" type="FireSensor"/>
17   </ObjectType>
18  </EntityTypes>
```

No Código 4.2, tem-se a especificação em AORSL da regra de ambiente (*EnvironmentRule*) que detecta a ocorrência da situação *PossibleFire*. Esta regra percorre todos os objetos do tipo *FireSensor* através da *tag FOR-ObjectVariable* a cada passo da simulação e verifica se tal sensor de incêndio está ativo (*tag IF*). Caso esteja, é feito o disparo do evento causal *PossibleFireDetected* indicando a detecção da situação em questão e com referência à instância do objeto *FireSensor* que caracterizou esta ocorrência.

Tal evento é detectado por outra regra de ambiente que tem a função de tratar as informações referentes a ocorrência desta situação tal como vista na Seção 3.1.4.

Código 4.2: Código AORSL da regra de ambiente para detecção da situação oriundo da transformação da Situação *PossibleFire* da Figura 20

```
1   <EnvironmentRules>
2    <EnvironmentRule name="PossibleFireSituation">
3     <WHEN eventType="Step" eventVariable="stepEvent"/>
4     <FOR-ObjectVariable variable="FireSensor1" objectType="FireSensor" />
5     <IF language="Java␣JavaScript">
6      <![CDATA[ FireSensor1.isActive() == true ]]>
7     </IF>
8     <THEN>
9      <SCHEDULE-EVT>
10      <CausedEventExpr eventType="PossibleFireDetected">
11       <Slot property="FireSensor1">
12        <ObjectValueExpr objectVariable="FireSensor1"/>
13       </Slot>
14      </CausedEventExpr>
15     </SCHEDULE-EVT>
16    </THEN>
17   </EnvironmentRule>
18  </EnvironmentRules>
```

4.3.2 Situação *FireDetected*

O Código 4.3 revela a transformação para AORSL da entidade que representa a ocorrência da situação *FireDetected* - modelada em SML na Figura 21. As entidades

Agent, *ConfirmEvent* e *FireDetectedDetected* são resultantes do processo de transformação e podem ser encontradas no código completo desta transformação no Apêndice B.

Código 4.3: Trecho do Código AORSL das entidades oriundas da transformação da Situação *FireDetected* da Figura 21

```
1  <ObjectType name="FireDetected">
2   <Attribute name="duration" type="Integer"/>
3   <Attribute name="finalTime" type="Integer"/>
4   <Attribute name="startTime" type="Integer"/>
5   <ReferenceProperty name="FireSensor2" type="FireSensor"/>
6   <ReferenceProperty name="ConfirmEvent1" type="ConfirmEvent"/>
7   <ReferenceProperty name="PossibleFire" type="PossibleFire"/>
8   <ReferenceProperty name="Agent1" type="Agent"/>
9  </ObjectType>
```

Nota-se no Código 4.3 uma referência para a entidade que representa a situação *PossibleFire*. Isto por que *FireDetected* é uma situação composta e faz parte da sua significação a existência de uma outra situação - no caso, *PossibleFire*.

O Código 4.4 revela um trecho da regra de ambiente criada para detectar a situação *FireDetected*. É interessante notar que na expressão da condicional da linha 11, é verificada a ocorrência da situação *PossibleFire* relacionando o sensor de incêdio desta com o modelado na *FireDetected*.

Código 4.4: Código AORSL da regra de ambiente para detecção da situação oriundo da transformação da Situação *PossibleFire* da Figura 21

```
1  <EnvironmentRule name="FireDetectedSituation">
2   <WHEN eventType="Step" eventVariable="stepEvent"/>
3   <FOR-ObjectVariable variable="Agent1" objectType="Agent" />
4   <FOR-ObjectVariable variable="PossibleFire" objectType="PossibleFire" />
5   <FOR-ObjectVariable variable="ConfirmEvent1" objectType="ConfirmEvent" />
6   <FOR-ObjectVariable variable="FireSensor2" objectType="FireSensor" />
7   <IF language="Java␣JavaScript">
8    <![CDATA[ ConfirmEvent1.getAgent().getId() == Agent1.getId() &&
9     ConfirmEvent1.getFireSensor().getId() == FireSensor2.getId() &&
10    PossibleFire.getFireSensor1().getId() == FireSensor2.getId() ]]>
11  </IF>
12
13   [...]
14
15 </EnvironmentRule>
```

4.3.3 Situação *EmExitObstructed*

A Situação *EmExitObstructed* (Figura 22) verifica se uma saída de emergência está obstruída pelo incêndio. A transformação de tal situação para AORSL apresenta a entidade para registro da situação como mostrado no Código 4.5. O objeto *EmExitObstructed* apresenta, além dos atributos padrões para situação, referência aos participantes que caracterizam esta função: *EmergenceExit1*, *FireSensor3* e a situação *FireDetected*.

Código 4.5: Código AORSL das entidades oriundas da transformação da Situação *PossibleFire* da Figura 21

```
1       <ObjectType name="EmExitObstructed">
2         <Attribute name="duration" type="Integer"/>
3         <Attribute name="finalTime" type="Integer"/>
4         <Attribute name="startTime" type="Integer"/>
5         <ReferenceProperty name="EmergencyExit1" type="EmergencyExit"/>
6         <ReferenceProperty name="FireSensor3" type="FireSensor"/>
7         <ReferenceProperty name="FireDetected" type="FireDetected"/>
8       </ObjectType>
```

O Código 4.6 apresenta um trecho da regra de ambiente criada pela transformação para detecção da situação em questão. Neste código, nota-se na linha 9 que há um condicional para verificar se a situação utilizada para composição desta é uma situação que ainda está a ocorrer.

Código 4.6: Código AORSL da regra de ambiente para detecção da situação oriundo da transformação da Situação *PossibleFire* da Figura 21

```
1   <EnvironmentRule name="EmExitObstructedSituation">
2     <WHEN eventType="Step" eventVariable="stepEvent"/>
3     <FOR-ObjectVariable variable="FireDetected" objectType="FireDetected" />
4     <FOR-ObjectVariable variable="FireSensor3" objectType="FireSensor" />
5     <FOR-ObjectVariable variable="EmergencyExit1" objectType="EmergencyExit" />
6     <IF language="Java␣JavaScript">
7       <![CDATA[ FireDetected.getFireSensor2().getId() == FireSensor3.getId() &&
8       EmergencyExit1.getLocation() == FireSensor3.getLocation() &&
9       this.stepEvent.getOccurrenceTime() < FireDetected.getFinalTime() + 2
10
11      ]]>
12
13    </IF>
14
15    [...]
16
17  </EnvironmentRule>
```

4.4 Simulação da Situação de Emergência

Com a transformação do modelo de descrição de situaçãos de SML para AORSL feito nas seções anteriores, foi criado o ambiente de simulação com o domínio modelado. Para simulá-lo, como a AORSL ainda não provê uma interface para manipulação das entidades como discutido na Seção 3.3, é feito um plano de simulação. Este plano é exibido na Figura 23.

Em tal plano, temos a instanciação de 4 sensores de incêndio. Os sensores 1 e 2 se tornam ativos no tempo 10 e se tornam desativos no tempo 60. Já os sensores 3 e 4 se tornam ativos no tempo 30 e assim continuam até o final da simulação. Esses eventos são suficientes para caracterizarem a situação de possibilidade de incêndio (*PossibleFire*).

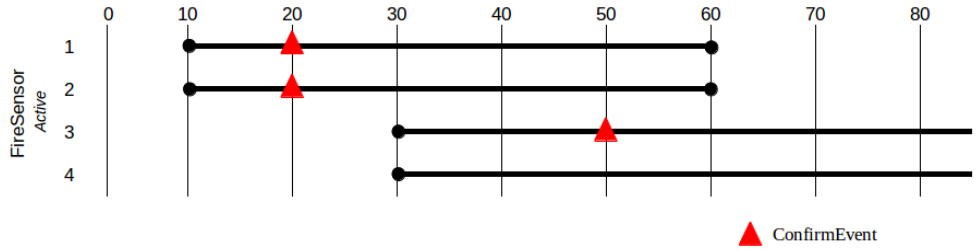


Figura 23: Especificação do plano de simulação com a ativação dos sensores de incêndio e confirmação do evento

Ainda na Figura 23, representa-se a ocorrência do contexto relacional *ConfirmEvent* com um triângulo sobre a linha de tempo dos respectivos sensores. *ConfirmEvent* relaciona uma entidade do tipo *Agent* com o sensor de incêndio que criou a situção de possibilidade de incêndio. Se existe essa relação, se algum agente confirma aquela possibilidade de incêndio, então há confirmação do incêndio.

Com este plano de simulação descrito, transcrevemos manualmente tais ações para a linguagem AORSL. O Código 4.7 mostra a regra de ambiente responsável pelas ações ocorridas no tempo 10. Tal regra percorre os sensores de incêndio com *IDs* 11 e 12 (sensores 1 e 2) para serem usados posteriormente dentro desta própria regra. Caso estejamos no tempo de ocorrência 10, faz-se a atualização dos atributos *Active* dos sensores *FireSensor1* e *FireSensor2* simulando, assim, que tais sensores estão alarmados, detectando fogo.

Código 4.7: Código AORSL de regra de ambiente para alteração de atributos de objetos do tipo *FireSensor* no passo 10 de simulação.

```
1 <EnvironmentRule name="count10">
2  <WHEN eventType="Step" eventVariable="Step"/>
3  <FOR objectVariable="FireSensor1" objectType="FireSensor" objectIdRef="11"/>
4  <FOR objectVariable="FireSensor2" objectType="FireSensor" objectIdRef="12"/>
5  <IF language="Java␣JavaScript">
6   <![CDATA[ this.Step.getOccurrenceTime() == 10]]>
7  </IF>
8  <THEN>
9  <UPDATE-ENV>
10   <UpdateObject objectVariable="FireSensor1">
11    <Slot property="Active" value="true"/>
12   </UpdateObject>
13   <UpdateObject objectVariable="FireSensor2">
14    <Slot property="Active" value="true"/>
15   </UpdateObject>
16  </UPDATE-ENV>
17  </THEN>
18 </EnvironmentRule>
```

Para os tempos de 20, 30, 50 e 60, temos outras regras de ambiente como a vista anteriormente. Tais regras podem ser encontradas no código integral da transformação no Apêndice B. Para o tempo de 20, como pode ser visto na Figura 23, ocorre a confirmação por algum agente (*Agent*) de que o sensor de incêndio está mesmo indicando um incêndio.

Esta indicação de confirmação de incêndio, feita pelo objeto *ConfirmEvent* ocorre na regra. Nesta regra de ambiente procura-se os dois sensores em questão e dois agentes e cria-se, então, duas instâncias do objeto *ConfirmEvent* com referências aos agentes e aos sensores.

A situação *FireDetected* é utilizada na composição da situação *EmExitObstructed*. Com as ações do tempo 20, é caracterizada a situação *FireDetected* e, com isso, verifica-se se também não está a ocorrer a situação que indica obstrução de alguma saída de emergência. No nosso estudo de caso, é modelado que somente o sensor de incêndio 1 está próximo a uma saída de emergência. Desta forma, deve ocorrer o evento *EmExitObstructed* na localização do sensor de incêndio 1.

4.4.1 Resultados da Simulação

A execução da simulação é feita utilizando a ferramenta AORS-JavaSim. O resultado da simulação é acessado via arquivo de *log* gerado pelo ambiente apresentado nesta seção. O *log* contém as ações ocorridas em cada passo da simulação. Por se tratar de um arquivo muito extenso, é selecionado aqui o resultado de alguns passos relevantes. Ao final desta seção a Figura 24 sintetiza o acontecimento das situações previstas neste estudo de caso.

No Código 4.8 temos o trecho do *log* que diz respeito aos acontecimentos no passo 10 e 11. No passo 10 é observada uma mudança de estado em propriedades de dois objetos. Os objetos com *ID* 11 e 12, que são os sensores de incêndio (*FireSensor*) 1 e 2, tem suas propriedades *Active* alteradas para o valor booliano *true*. Esta alteração faz com que seja caracterizada a situação *PossibleFire* e, no passo 11 (linha 17), temos que a situação foi detectada e o evento causal *PossibleFireDetected* com as referências aos sensores é criado.

Código 4.8: Trecho do *log* da simulação nos passos 10 e 11 com a alteração de atributos de objetos do tipo *FireSensor* e posterior detecção da situação *PossibleFire* devido a regra de ambiente exibida no Código 4.7.

```
1  <SimulationStep stepTime="10" xmlns="http://aor-simulation.org/log">
2      <EnvironmentSimulatorStep>
3          <ExogenousEvent nextOccurrenceTime="11.0" type="Step">
4              <ResultingStateChanges resultingFromRule="count10">
5                  <Objects>
6                      <Obj id="11">
7                          <Slot property="Active" value="true"/>
8                      </Obj>
9                      <Obj id="12">
10                         <Slot property="Active" value="true"/>
11                     </Obj>
12                 </Objects>
13             </ResultingStateChanges>
14         </ExogenousEvent>
15     </EnvironmentSimulatorStep>
16 </SimulationStep>
17 <SimulationStep stepTime="11" xmlns="http://aor-simulation.org/log">
18     <EnvironmentSimulatorStep>
19         <ExogenousEvent nextOccurrenceTime="12.0" type="Step">
```

```
20              <ResultingEvents resultingFromRule="PossibleFireSituation">
21                  <CausedEvent delay="1" type="PossibleFireDetected">
22                      <Slot property="FireSensor1" refId="11"/>
23                  </CausedEvent>
24                  <CausedEvent delay="1" type="PossibleFireDetected">
25                      <Slot property="FireSensor1" refId="12"/>
26                  </CausedEvent>
27              </ResultingEvents>
28          </ExogenousEvent>
29      </EnvironmentSimulatorStep>
30 </SimulationStep>
```

Por conta do evento causal criado na detecção da situação *PossibleFire*, no passo seguinte da simulação é criada uma instância do objeto *PossibleFire* e suas propriedades editadas. O Código 4.9 contém o trecho do *log* do passo 12 que mostra essas alterações.

Código 4.9: Trecho do *log* da simulação no passo 12 com a criação dos objetos *ConfirmEvent*.

```
1 <CausedEvent type="PossibleFireDetected">
2       <ResultingStateChanges resultingFromRule="PossibleFireSituationUpdate">
3         <Objects>
4           <Obj id="11">
5             <Slot property="PossibleFire" refId="-1"/>
6           </Obj>
7           <Obj id="-1">
8             <Slot property="FireSensor1" refId="11"/>
9           </Obj>
10          <Obj id="-1">
11            <Slot property="finalTime" value="12"/>
12          </Obj>
13          <Obj id="-1">
14            <Slot property="startTime" value="12"/>
15          </Obj>
16        </Objects>
17        <Create>
18          <Objects>
19            <Obj type="PossibleFire" id="-1"/>
20          </Objects>
21        </Create>
22      </ResultingStateChanges>
23 </CausedEvent>
```

No passo 20, conforme o plano de simulação, a situação de possibilidade de incêndio é confirmada através da instanciação de duas entidades do tipo *ConfirmEvent*. No Código 4.10 vê-se o trecho arquivo de *log* referente. Dois objetos do tipo *ConfirmEvent* são criados e posteriormente tem suas propriedades alteradas indicando a confirmação do alerta de um sensor de incêndio por um agente.

Código 4.10: Trecho do *log* da simulação no passo 20 com a criação dos objeto *PossibleFire* para a ocorrência da situação de mesmo nome.

```
1 <Create>
2   <Objects>
3     <Obj type="ConfirmEvent" id="51"/>
4     <Obj type="ConfirmEvent" id="52"/>
```

```
5   </Objects>
6 </Create>
7 <Objects>
8   <Obj id="51">
9     <Slot property="Agent" refId="31"/>
10  </Obj>
11  <Obj id="51">
12    <Slot property="FireSensor" refId="11"/>
13  </Obj>
14  <Obj id="52">
15    <Slot property="Agent" refId="32"/>
16  </Obj>
17  <Obj id="52">
18    <Slot property="FireSensor" refId="12"/>
19  </Obj>
20 </Objects>
```

Com a confirmação do incêndio, é caracterizada a situação *FireDetected* que é detectada no passo seguinte, 21, como o demostrado no Código 4.11. No passo 22 a instância do objeto que descrever a ocorrência da situação é criada.

Código 4.11: Trecho do *log* da simulação no passo 21 indicando a detecço da situaço *FireDetected* através do evento causal *FireDetectedDetected*

```
1 <ResultingEventsresultingFromRule="FireDetectedSituation">
2   <CausedEventdelay="1"type="FireDetectedDetected">
3     <Slotproperty="FireSensor2"refId="11"/>
4     <Slotproperty="ConfirmEvent1"refId="51"/>
5     <Slotproperty="Agent1"refId="31"/>
6     <Slotproperty="PossibleFire"refId="-1"/>
7   </CausedEvent>
8   <CausedEventdelay="1"type="FireDetectedDetected">
9     <Slotproperty="FireSensor2"refId="12"/>
10    <Slotproperty="ConfirmEvent1"refId="52"/>
11    <Slotproperty="Agent1"refId="32"/>
12    <Slotproperty="PossibleFire"refId="-2"/>
13  </CausedEvent>
14 </ResultingEvents>
```

O sensor de fogo número 1 localiza-se próximo a uma saída de emergência. Sendo a instância do objeto *FireDetected* criada no passo 22, no passo 23 é detectada a ocorrência da situação *EmExitObstructed* por conta da proximidade do sensor 1 com uma saída de emergência. Por conta disso, é criado o evento *EmExitObstructedSituation* indicando a ocorrência desta situação. O Código 4.12 refere-se à este acontecimento.

Código 4.12: Trecho do *log* da simulação no passo 23 com o evento causal de indicaço da detecço da situaço *EmExitObstructed*

```
1 <ResultingEvents resultingFromRule="EmExitObstructedSituation">
2   <CausedEvent delay="1" type="EmExitObstructedDetected">
3     <Slot property="EmergencyExit1" refId="21"/>
4     <Slot property="FireSensor3" refId="12"/>
5     <Slot property="FireDetected" refId="-4"/>
6   </CausedEvent>
7 </ResultingEvents>
```

Para melhor visualização da simulação, a ocorrência das situações são sintetizadas manualmente na Figura 24. As situações estão relacionadas à cada um dos quatro sensores.

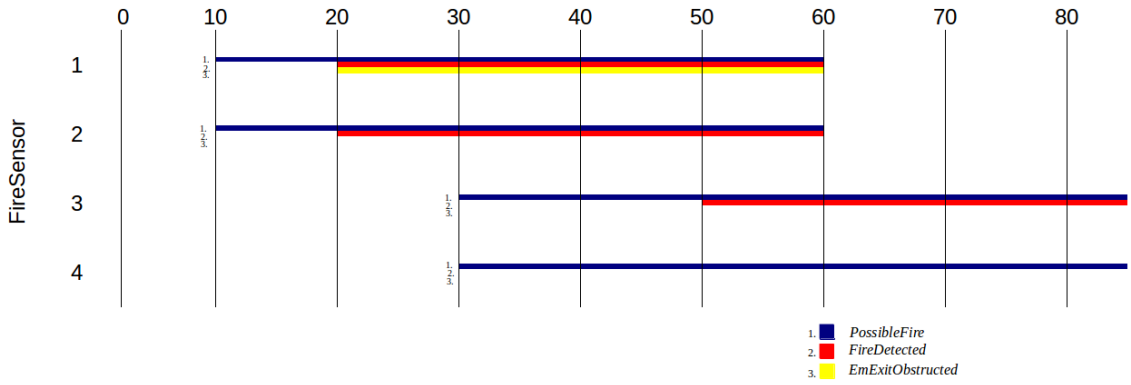


Figura 24: Ocorrência das situações de acordo com o número do sensor de incêndio associado.

5 Conclusão

Este trabalho define uma transformação MDA automatizada de SML para AORSL relacionando, assim, uma ferramenta para modelagem conceitual com um ambiente de simulação. Tal transformação possibilita a execução de modelos de situações descritos em SML no ambiente de simulação AORS contribuindo, assim, no processo de desenvolvimento e modelagem uma vez que permite explorar o modelo de novas formas possibilitando melhor compreensão sobre o mesmo.

A AORSL é uma linguagem compatível para simulação dos modelos SML uma vez que consegue expressar os conceitos desta para os fins de execução. Importante ressaltar, porém, que os modelos AORSL não trazerem explicitamente a semântica presente nos modelos SML já que a finalidade das mesmas é diferente.

Aspectos importantes para uma simulação interativa não foram implementadas. Assim, a transformação de modelos ainda demanda um esforço do desenvolvedor no sentido de que este precisa explicitar o plano de simulação antes da execução da mesma via código AORSL. A leitura dos resultados também demanda um esforço adicional para a leitura dos *logs* de comportamento do ambiente de simulação.

A fim de avaliar a proposta, foi realizado estudo de caso no cenário de planos de emergência. Este domínio tem singular relevância já que trata de situações críticas. As situações propostas modeladas em SML foram transformadas para AORSL e simuladas. A simulação executada do modelo trouxe os resultados esperados de comportamento do simulador validando o modelo bem como o intencionado com a modelagem SML.

Este trabalho, então, assume importância uma vez que transforma modelos conceituais, que tem objetivo de comunicação e entendimento, em código a ser executado em ambiente computacional. Permite, assim, que o modelador se concentre na modelagem sem se preocupar com a implementação para simulação.

Como possiveis trabalhos futuros, pode ser citado a implementação de mecanismos para propiciar uma simulação participativa do usuário. Isto inclue o controle em tempo de execução da criação das entidades e alteração das informações de ambiente e também a vizualização dos resultados específicos de detecção de situações baseado no *log* ou dentro do próprio ambiente AORS. Cita-se também a implementação da transformação para toda SML já que algumas entidades não foram contempladas neste trabalho como a classe *Function*, a verificação de existência de determinada situação pela classe *ExistsSituation*, a verificação de uma situação passada pelo atributo *isPast*da classe *SituationParticipant*.

Referências

ALMEIDA, J. *Model-driven design of distributed applications*. Tese (Doutorado), 2006. Disponível em: <http://www.springerlink.com/index/p0u3wrqqkud40c1y.pdf>. Citado 2 vezes nas páginas 17 e 32.

COSTA, P. D. *Architectural support for context-aware applications: from context models to services platforms*. Tese (Doutorado), 2007. Disponível em: <http://doc.utwente.nl-/58357/>. Citado 5 vezes nas páginas 4, 11, 15, 16 e 18.

COSTA, P. D. et al. Towards Conceptual Foundations for Context-Aware Applications. 2005. Citado na página 15.

DAVIDSSON, P. Agent based social simulation: A computer science view. *J. Artificial Societies and Social Simulation*, v. 5, 2002. Disponível em: <http://jasss.soc.surrey.ac.uk-/5/1/7.html>. Citado na página 27.

DEY, A. K. Understanding and Using Context. p. 4–7, 2001. Citado 2 vezes nas páginas 10 e 14.

DINIZ, V. B. Uma Abordagem para Definição de Sistemas de Gestão de Conhecimento no Tratamento de Emergências. p. 194, 2006. Citado na página 46.

GUIZZARDI, G. *Ontological foundations for structural conceptual models*. [s.n.], 2005. ISBN 9075176813. Disponível em: <http://doc.utwente.nl/50826>. Citado 5 vezes nas páginas 10, 11, 15, 18 e 19.

HANSMANN, U. et al. *Pervasive Computing*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2003. ISBN 3540002189. Citado na página 10.

MIELKE, I. T. *Uma Abordagem baseada em modelos para especificação e detecção de situações em sistemas sensíveis ao contexto*. Dissertação (Mestrado), 2013. Citado 13 vezes nas páginas 4, 5, 11, 16, 17, 18, 19, 20, 21, 22, 23, 75 e 76.

MILLER, J.; MUKERJI, J. MDA Guide Version 1.0. 1. *Object Management Group*, n. June, 2003. Citado 5 vezes nas páginas 4, 17, 32, 33 e 34.

MOODY, D. L. The "physics" of notations: toward a scientific basis for constructing visual notations in software engineering. *Software Engineering, IEEE Transactions on*, v. 35, n. 5, p. 756–778, 2009. Disponível em: <http://ieeexplore.ieee.org/xpls/abs_all-.jsp?arnumber=5353439>. Citado na página 17.

MYLOPOULOS, J. Conceptual Modelling and Telos 1. p. 1–20, 1992. Citado na página 15.

OMG. *OMG Unified Modeling Language (OMG UML), Infrastructure, Version 2.4.1*. 2011. Disponível em: <http://www.omg.org/spec/UML/2.4.1>. Citado 2 vezes nas páginas 17 e 18.

WAGNER, G. The Agent–Object-Relationship metamodel: towards a unified view of state and behavior. *Information Systems*, v. 28, n. 5, p. 475–504, jul. 2003. ISSN 03064379. Disponível em: <http://linkinghub.elsevier.com/retrieve/pii/S0306437902000273>. Citado 5 vezes nas páginas 4, 27, 28, 29 e 30.

WAGNER, G. AOR modelling and simulation: Towards a general architecture for agent-based discrete event simulation. *AgentOriented Information Systems*, v. 3030, p. 174–188, 2004. Disponível em: <http://www.springerlink.com/index/q9497uf4mxk8cyqu.pdf>. Citado 3 vezes nas páginas 11, 12 e 27.

WAGNER, G. *Reference Manual for AORSL 0.8.4*. 2011. Disponível em: <http://-/oxygen.informatik.tu-cottbus.de/aors/AORSL.html>. Citado 2 vezes nas páginas 31 e 43.

WAGNER, G.; DIACONESCU, M. Aor-simulation. org: cognitive agent simulation. *... Conference on Autonomous Agents and ...*, p. 3–4, 2009. Disponível em: <http://dl.acm.org/citation.cfm?id=1558317>. Citado 2 vezes nas páginas 4 e 30.

A Código completo Exemplo SituationType Detection

Código AORSL completo da transformação do modelo SML da Figura 25.

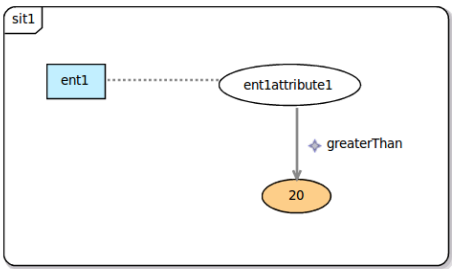


Figura 25: Representação da *sit1* em SML

Código A.1: Código AORSL oriundo da transformação do modelo expresso na Figura 25

```
<SimulationModel modelName="MODELNAME" modelTitle="MODEL TITLE">
  <EntityTypes>
    <CausedEventType name="sit1Detected">
      <ReferenceProperty name="ent1" type="ent1"/>
    </CausedEventType>
    <ObjectType name="ent1">
    <Attribute name="ent1attribute1" type="Integer"/>
      <ReferenceProperty name="sit1" type="sit1"/>
      </ObjectType>
      <ObjectType name="sit1">
        <Attribute name="duration" type="Integer"/>
        <Attribute name="finalTime" type="Integer"/>
        <Attribute name="startTime" type="Integer"/>
        <ReferenceProperty name="ent1" type="ent1"/>
      </ObjectType>
    </EntityTypes>

    <EnvironmentRules>
      <EnvironmentRule name="sit1Situation">
        <WHEN eventType="Step" eventVariable="stepEvent"/>
        <FOR-ObjectVariable variable="ent1" objectType="ent1" />
        <IF language="Java JavaScript"> this.ent1.getEnt1attribute1() > 20 </IF>
        <THEN>
          <SCHEDULE-EVT>
            <CausedEventExpr eventType="sit1Detected">
               <Slot property="ent1">
                 <ObjectValueExpr objectVariable="ent1"/>
              </Slot>
            </CausedEventExpr>
          </SCHEDULE-EVT>
        </THEN>
      </EnvironmentRule>

      <EnvironmentRule name="sit1SituationUpdate">
```

```
35          <WHEN eventType="sit1Detected" eventVariable="sit1SituationDetected"/>
36          <FOR objectVariable="ent1" objectType="ent1" >
37            <ObjectRef language="Java␣JavaScript"><![CDATA[this.sit1SituationDetected.
                  getEnt1()]]></ObjectRef>
38          </FOR>
39          <IF language="Java␣JavaScript"> <![CDATA[ sit1SituationDetected.getent1().getsit1
                ()  == null   || this.sit1SituationDetected.getOccurrenceTime() >
                sit1SituationDetected.getent1().getsit1().getFinalTime() + 10 ]]>  </IF>
40          <THEN>
41            <UPDATE-ENV>
42              <Create>
43                <Object type="sit1" objectVariable="sit1"/>
44              </Create>
45              <UpdateObject>
46                <ObjectRef language="Java␣JavaScript" objectType="ent1"><![CDATA[this.
                      sit1SituationDetected.getEnt1()]]></ObjectRef>
47                <Slot property="sit1">
48                  <ObjectValueExpr objectVariable="sit1"/>
49                </Slot>
50              </UpdateObject>
51              <UpdateObject objectVariable="sit1">
52                <Slot property="ent1">
53                  <ObjectValueExpr objectVariable="ent1"/>
54                </Slot>
55                <Slot property="finalTime">
56                  <ValueExpr language="Java␣JavaScript"/><![CDATA[this.
                        sit1SituationDetected.getOccurrenceTime()]]></ValueExpr>
57                </Slot>
58                <Slot property="startTime">
59                  <ValueExpr language="Java␣JavaScript"/><![CDATA[this.
                        sit1SituationDetected.getOccurrenceTime()]]></ValueExpr>
60                </Slot>
61              </UpdateObject>
62            </UPDATE-ENV>
63          </THEN>
64          <ELSE>
65            <UPDATE-ENV>
66              <UpdateObject>
67                <ObjectRef language="Java␣JavaScript" objectType="sit1"><![CDATA[this.
                      sit1SituationDetected.getent1().getsit1()]]></ObjectRef>
68                <Slot property="duration">
69                  <ValueExpr language="Java␣JavaScript"/><![CDATA[this.
                        sit1SituationDetected.getOccurrenceTime() - this.
                        sit1SituationDetected.getent1().getsit1().getStartTime()]]></
                        ValueExpr>
70                </Slot>
71                <Slot property="finalTime">
72                  <ValueExpr language="Java␣JavaScript"/><![CDATA[this.
                        sit1SituationDetected.getOccurrenceTime()]]></ValueExpr>
73                </Slot>
74              </UpdateObject>
75            </UPDATE-ENV>
76          </ELSE>
77        </EnvironmentRule>
78
79      </EnvironmentRules>
80    </SimulationModel>
```

B Código AORSL Completo da Transformação das Situações de Emergência do Estudo de Caso

Código AORSL completo da transformação do modelo SML da Figura 26.

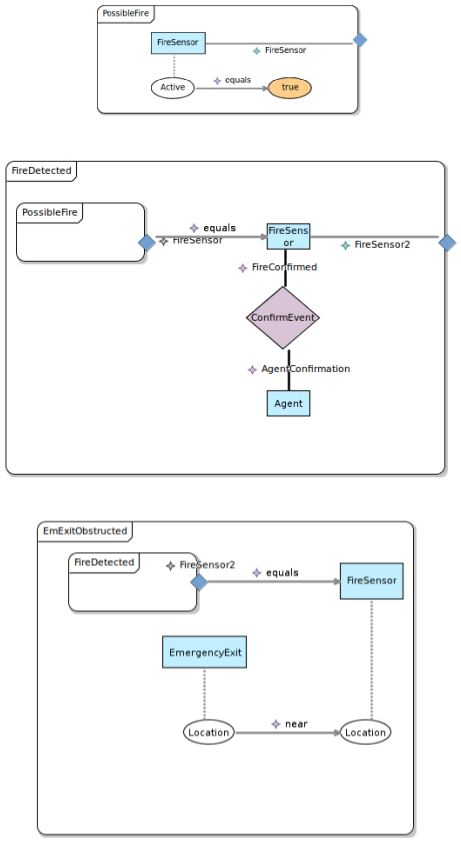


Figura 26: Representação das situações de emergência do estudo de caso do Capítulo 4 em SML

Código B.1: Código AORSL oriundo da transformação do modelo expresso na Figura 25

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <?xml-stylesheet type="text/xsl" href="prettyprint.xsl"?>
3 <SimulationScenario xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://
      aor-simulation.org"
4   xmlns:aors="http://aor-simulation.org" xmlns:dc="http://purl.org/dc/elements/1.1/"
5   xmlns:h="http://www.w3.org/1999/xhtml" version="0.9"
6   xsi:schemaLocation="http://aor-simulation.org ../../../ext/aorsl/AORSL-0-8-3.xsd"
7   scenarioName="SingleServiceQueueWithStatistics"
8   scenarioTitle="A single service queue with service utilization and maximum queue length
        statistics"
9   simulationManagerDirectory="../../..">
10
```

```
<SimulationParameters simulationSteps="6000" stepDuration="1" timeUnit="min"/>

<SimulationModel modelName="MODELNAME" modelTitle="MODEL TITLE">
  <EntityTypes>
    <ExogenousEventType name="Step" periodicity="1"/>
    <CausedEventType name="EmExitObstructedDetected">
      <ReferenceProperty name="EmergencyExit1" type="EmergencyExit"/>
      <ReferenceProperty name="FireSensor3" type="FireSensor"/>
      <ReferenceProperty name="FireDetected" type="FireDetected"/>
    </CausedEventType>
    <CausedEventType name="FireDetectedDetected">
      <ReferenceProperty name="FireSensor2" type="FireSensor"/>
      <ReferenceProperty name="ConfirmEvent1" type="ConfirmEvent"/>
      <ReferenceProperty name="Agent1" type="Agent"/>
      <ReferenceProperty name="PossibleFire" type="PossibleFire"/>
    </CausedEventType>
    <CausedEventType name="PossibleFireDetected">
      <ReferenceProperty name="FireSensor1" type="FireSensor"/>
    </CausedEventType>
    <ObjectType name="EmergencyExit">
      <Attribute name="Location" type="Integer"/>
      <ReferenceProperty name="EmExitObstructed" type="EmExitObstructed"/>
    </ObjectType>
    <ObjectType name="EmExitObstructed">
      <Attribute name="duration" type="Integer"/>
      <Attribute name="finalTime" type="Integer"/>
      <Attribute name="startTime" type="Integer"/>
      <ReferenceProperty name="EmergencyExit1" type="EmergencyExit"/>
      <ReferenceProperty name="FireSensor3" type="FireSensor"/>
      <ReferenceProperty name="FireDetected" type="FireDetected"/>
    </ObjectType>
    <ObjectType name="Agent">
      <ReferenceProperty name="FireDetected" type="FireDetected"/>
    </ObjectType>
    <ObjectType name="ConfirmEvent">
      <ReferenceProperty name="FireSensor" type="FireSensor"/>
      <ReferenceProperty name="Agent" type="Agent"/>
      <ReferenceProperty name="FireDetected" type="FireDetected"/>
    </ObjectType>
    <ObjectType name="FireSensor">
      <Attribute name="Active" type="Boolean"/>
      <Attribute name="Location" type="Integer"/>
      <ReferenceProperty name="FireDetected" type="FireDetected"/>
      <ReferenceProperty name="PossibleFire" type="PossibleFire"/>
      <ReferenceProperty name="EmExitObstructed" type="EmExitObstructed"/>
    </ObjectType>
    <ObjectType name="FireDetected">
      <Attribute name="duration" type="Integer"/>
      <Attribute name="finalTime" type="Integer"/>
      <Attribute name="startTime" type="Integer"/>
      <ReferenceProperty name="FireSensor2" type="FireSensor"/>
      <ReferenceProperty name="ConfirmEvent1" type="ConfirmEvent"/>
      <ReferenceProperty name="PossibleFire" type="PossibleFire"/>
      <ReferenceProperty name="Agent1" type="Agent"/>
    </ObjectType>
    <ObjectType name="PossibleFire">
      <Attribute name="duration" type="Integer"/>
      <Attribute name="finalTime" type="Integer"/>
      <Attribute name="startTime" type="Integer"/>
      <ReferenceProperty name="FireSensor1" type="FireSensor"/>
```

```
71        </ObjectType>
72      </EntityTypes>
73
74      <EnvironmentRules>
75        <EnvironmentRule name="EmExitObstructedSituation">
76          <WHEN eventType="Step" eventVariable="stepEvent"/>
77          <FOR-ObjectVariable variable="FireDetected" objectType="FireDetected" />
78          <FOR-ObjectVariable variable="FireSensor3" objectType="FireSensor" />
79          <FOR-ObjectVariable variable="EmergencyExit1" objectType="EmergencyExit" />
80          <IF language="Java␣JavaScript">
81            <![CDATA[ FireDetected.getFireSensor2().getId() == FireSensor3.getId() &&
82            EmergencyExit1.getLocation() == FireSensor3.getLocation() &&
83            this.stepEvent.getOccurrenceTime() < FireDetected.getFinalTime() + 2
84
85            ]]>
86
87          </IF>
88          <THEN>
89            <SCHEDULE-EVT>
90              <CausedEventExpr eventType="EmExitObstructedDetected">
91                 <Slot property="FireDetected">
92                   <ObjectValueExpr objectVariable="FireDetected"/>
93                 </Slot>
94                 <Slot property="FireSensor3">
95                   <ObjectValueExpr objectVariable="FireSensor3"/>
96                 </Slot>
97                 <Slot property="EmergencyExit1">
98                   <ObjectValueExpr objectVariable="EmergencyExit1"/>
99                 </Slot>
100             </CausedEventExpr>
101           </SCHEDULE-EVT>
102         </THEN>
103       </EnvironmentRule>
104
105       <EnvironmentRule name="EmExitObstructedSituationUpdate">
106         <WHEN eventType="EmExitObstructedDetected" eventVariable="
              EmExitObstructedDetected"/>
107         <FOR objectVariable="FireSensor" objectType="FireSensor" >
108           <ObjectRef language="Java␣JavaScript"><![CDATA[this.EmExitObstructedDetected.
                getFireSensor3()]]></ObjectRef>
109         </FOR>
110         <FOR objectVariable="EmergencyExit" objectType="EmergencyExit" >
111           <ObjectRef language="Java␣JavaScript"><![CDATA[this.EmExitObstructedDetected.
                getEmergencyExit1()]]></ObjectRef>
112         </FOR>
113         <IF language="Java␣JavaScript">
114 <![CDATA[ EmExitObstructedDetected.getFireSensor3().getEmExitObstructed() == null || this
      .EmExitObstructedDetected.getOccurrenceTime() > EmExitObstructedDetected.
      getFireSensor3().getEmExitObstructed().getFinalTime() + 10 ]]>
115         </IF>
116         <THEN>
117           <UPDATE-ENV>
118             <Create>
119               <Object type="EmExitObstructed" objectVariable="EmExitObstructed"/>
120             </Create>
121             <UpdateObject>
122               <ObjectRef language="Java␣JavaScript" objectType="FireSensor"><![CDATA[this
                    .EmExitObstructedDetected.getFireSensor3()]]></ObjectRef>
123               <Slot property="EmExitObstructed">
124                 <ObjectValueExpr objectVariable="EmExitObstructed"/>
```

```
125             </Slot>
126           </UpdateObject>
127           <UpdateObject>
128             <ObjectRef language="Java␣JavaScript" objectType="EmergencyExit"><![CDATA[
                    this.EmExitObstructedDetected.getEmergencyExit1()]]></ObjectRef>
129             <Slot property="EmExitObstructed">
130               <ObjectValueExpr objectVariable="EmExitObstructed"/>
131             </Slot>
132           </UpdateObject>
133           <UpdateObject objectVariable="EmExitObstructed">
134             <Slot property="FireSensor3">
135               <ObjectValueExpr objectVariable="FireSensor"/>
136             </Slot>
137             <Slot property="EmergencyExit1">
138               <ObjectValueExpr objectVariable="EmergencyExit"/>
139             </Slot>
140             <Slot property="finalTime">
141               <ValueExpr language="Java␣JavaScript"><![CDATA[this.
                      EmExitObstructedDetected.getOccurrenceTime()]]></ValueExpr>
142             </Slot>
143             <Slot property="startTime">
144               <ValueExpr language="Java␣JavaScript"><![CDATA[this.
                      EmExitObstructedDetected.getOccurrenceTime()]]></ValueExpr>
145             </Slot>
146           </UpdateObject>
147         </UPDATE-ENV>
148       </THEN>
149       <ELSE>
150         <UPDATE-ENV>
151           <UpdateObject>
152             <ObjectRef language="Java␣JavaScript" objectType="EmExitObstructed"><![
                    CDATA[this.EmExitObstructedDetected.getFireSensor3().
                    getEmExitObstructed()]]></ObjectRef>
153             <Slot property="duration">
154               <ValueExpr language="Java␣JavaScript"><![CDATA[this.
                      EmExitObstructedDetected.getOccurrenceTime() - this.
                      EmExitObstructedDetected.getFireSensor3().getEmExitObstructed().
                      getStartTime()]]></ValueExpr>
155             </Slot>
156             <Slot property="finalTime">
157               <ValueExpr language="Java␣JavaScript"><![CDATA[this.
                      EmExitObstructedDetected.getOccurrenceTime()]]></ValueExpr>
158             </Slot>
159           </UpdateObject>
160         </UPDATE-ENV>
161       </ELSE>
162     </EnvironmentRule>
163
164
165
166     <EnvironmentRule name="FireDetectedSituation">
167       <WHEN eventType="Step" eventVariable="stepEvent"/>
168       <FOR-ObjectVariable variable="Agent1" objectType="Agent" />
169       <FOR-ObjectVariable variable="PossibleFire" objectType="PossibleFire" />
170       <FOR-ObjectVariable variable="ConfirmEvent1" objectType="ConfirmEvent" />
171       <FOR-ObjectVariable variable="FireSensor2" objectType="FireSensor" />
172       <IF language="Java␣JavaScript">
173         <![CDATA[ ConfirmEvent1.getAgent().getId() == Agent1.getId() &&
174         ConfirmEvent1.getFireSensor().getId() == FireSensor2.getId() &&
175         PossibleFire.getFireSensor1().getId() == FireSensor2.getId() &&
```

```
176             this.stepEvent.getOccurrenceTime() < PossibleFire.getFinalTime() + 2
177
178
179           ]]>
180         </IF>
181
182
183
184         <THEN>
185           <SCHEDULE-EVT>
186             <CausedEventExpr eventType="FireDetectedDetected">
187               <Slot property="Agent1">
188                 <ObjectValueExpr objectVariable="Agent1"/>
189               </Slot>
190               <Slot property="PossibleFire">
191                 <ObjectValueExpr objectVariable="PossibleFire"/>
192               </Slot>
193               <Slot property="ConfirmEvent1">
194                 <ObjectValueExpr objectVariable="ConfirmEvent1"/>
195               </Slot>
196               <Slot property="FireSensor2">
197                 <ObjectValueExpr objectVariable="FireSensor2"/>
198               </Slot>
199             </CausedEventExpr>
200           </SCHEDULE-EVT>
201         </THEN>
202       </EnvironmentRule>
203
204       <EnvironmentRule name="FireDetectedSituationUpdate">
205         <WHEN eventType="FireDetectedDetected" eventVariable="FireDetectedDetected"/>
206         <FOR objectVariable="Agent" objectType="Agent" >
207           <ObjectRef language="Java␣JavaScript"><![CDATA[this.FireDetectedDetected.
                getAgent1()]]></ObjectRef>
208         </FOR>
209         <FOR objectVariable="ConfirmEvent" objectType="ConfirmEvent" >
210           <ObjectRef language="Java␣JavaScript"><![CDATA[this.FireDetectedDetected.
                getConfirmEvent1()]]></ObjectRef>
211         </FOR>
212         <FOR objectVariable="FireSensor" objectType="FireSensor" >
213           <ObjectRef language="Java␣JavaScript"><![CDATA[this.FireDetectedDetected.
                getFireSensor2()]]></ObjectRef>
214         </FOR>
215         <IF language="Java␣JavaScript">
216 <![CDATA[ FireDetectedDetected.getAgent1().getFireDetected() == null  || this.
      FireDetectedDetected.getOccurrenceTime() > FireDetectedDetected.getAgent1().
      getFireDetected().getFinalTime() + 10 ]]>
217         </IF>
218         <THEN>
219           <UPDATE-ENV>
220             <Create>
221               <Object type="FireDetected" objectVariable="FireDetected"/>
222             </Create>
223             <UpdateObject>
224               <ObjectRef language="Java␣JavaScript" objectType="Agent"><![CDATA[this.
                    FireDetectedDetected.getAgent1()]]></ObjectRef>
225               <Slot property="FireDetected">
226                 <ObjectValueExpr objectVariable="FireDetected"/>
227               </Slot>
228             </UpdateObject>
229             <UpdateObject>
```

```
230             <ObjectRef language="Java␣JavaScript" objectType="ConfirmEvent"><![CDATA[
                    this.FireDetectedDetected.getConfirmEvent1()]]></ObjectRef>
231             <Slot property="FireDetected">
232               <ObjectValueExpr objectVariable="FireDetected"/>
233             </Slot>
234           </UpdateObject>
235           <UpdateObject>
236             <ObjectRef language="Java␣JavaScript" objectType="FireSensor"><![CDATA[this
                    .FireDetectedDetected.getFireSensor2()]]></ObjectRef>
237             <Slot property="FireDetected">
238               <ObjectValueExpr objectVariable="FireDetected"/>
239             </Slot>
240           </UpdateObject>
241           <UpdateObject objectVariable="FireDetected">
242             <Slot property="Agent1">
243               <ObjectValueExpr objectVariable="Agent"/>
244             </Slot>
245             <Slot property="ConfirmEvent1">
246               <ObjectValueExpr objectVariable="ConfirmEvent"/>
247             </Slot>
248             <Slot property="FireSensor2">
249               <ObjectValueExpr objectVariable="FireSensor"/>
250             </Slot>
251             <Slot property="finalTime">
252               <ValueExpr language="Java␣JavaScript"><![CDATA[this.FireDetectedDetected.
                      getOccurrenceTime()]]></ValueExpr>
253             </Slot>
254             <Slot property="startTime">
255               <ValueExpr language="Java␣JavaScript"><![CDATA[this.FireDetectedDetected.
                      getOccurrenceTime()]]></ValueExpr>
256             </Slot>
257           </UpdateObject>
258         </UPDATE-ENV>
259       </THEN>
260       <ELSE>
261         <UPDATE-ENV>
262           <UpdateObject>
263             <ObjectRef language="Java␣JavaScript" objectType="FireDetected"><![CDATA[
                    this.FireDetectedDetected.getAgent1().getFireDetected()]]></ObjectRef>
264             <Slot property="duration">
265               <ValueExpr language="Java␣JavaScript"><![CDATA[this.FireDetectedDetected.
                      getOccurrenceTime() - this.FireDetectedDetected.getAgent1().
                      getFireDetected().getStartTime()]]></ValueExpr>
266             </Slot>
267             <Slot property="finalTime">
268               <ValueExpr language="Java␣JavaScript"><![CDATA[this.FireDetectedDetected.
                      getOccurrenceTime()]]></ValueExpr>
269             </Slot>
270           </UpdateObject>
271         </UPDATE-ENV>
272       </ELSE>
273     </EnvironmentRule>
274
275
276
277     <EnvironmentRule name="PossibleFireSituation">
278       <WHEN eventType="Step" eventVariable="stepEvent"/>
279       <FOR-ObjectVariable variable="FireSensor1" objectType="FireSensor" />
280       <IF language="Java␣JavaScript">
281         <![CDATA[ FireSensor1.isActive() == true ]]>
```

```
282         </IF>
283         <THEN>
284           <SCHEDULE-EVT>
285             <CausedEventExpr eventType="PossibleFireDetected">
286               <Slot property="FireSensor1">
287                 <ObjectValueExpr objectVariable="FireSensor1"/>
288               </Slot>
289             </CausedEventExpr>
290           </SCHEDULE-EVT>
291         </THEN>
292       </EnvironmentRule>
293
294       <EnvironmentRule name="PossibleFireSituationUpdate">
295         <WHEN eventType="PossibleFireDetected" eventVariable="PossibleFireDetected"/>
296         <FOR objectVariable="FireSensor" objectType="FireSensor" >
297           <ObjectRef language="Java␣JavaScript"><![CDATA[this.PossibleFireDetected.
                getFireSensor1()]]></ObjectRef>
298         </FOR>
299         <IF language="Java␣JavaScript">
300 <![CDATA[ PossibleFireDetected.getFireSensor1().getPossibleFire() == null || this.
      PossibleFireDetected.getOccurrenceTime() > PossibleFireDetected.getFireSensor1().
      getPossibleFire().getFinalTime() + 10 ]]>
301         </IF>
302         <THEN>
303           <UPDATE-ENV>
304             <Create>
305               <Object type="PossibleFire" objectVariable="PossibleFire"/>
306             </Create>
307             <UpdateObject>
308               <ObjectRef language="Java␣JavaScript" objectType="FireSensor"><![CDATA[this
                    .PossibleFireDetected.getFireSensor1()]]></ObjectRef>
309               <Slot property="PossibleFire">
310                 <ObjectValueExpr objectVariable="PossibleFire"/>
311               </Slot>
312             </UpdateObject>
313             <UpdateObject objectVariable="PossibleFire">
314               <Slot property="FireSensor1">
315                 <ObjectValueExpr objectVariable="FireSensor"/>
316               </Slot>
317               <Slot property="finalTime">
318                 <ValueExpr language="Java␣JavaScript"><![CDATA[this.PossibleFireDetected.
                      getOccurrenceTime()]]></ValueExpr>
319               </Slot>
320               <Slot property="startTime">
321                 <ValueExpr language="Java␣JavaScript"><![CDATA[this.PossibleFireDetected.
                      getOccurrenceTime()]]></ValueExpr>
322               </Slot>
323             </UpdateObject>
324           </UPDATE-ENV>
325         </THEN>
326         <ELSE>
327           <UPDATE-ENV>
328             <UpdateObject>
329               <ObjectRef language="Java␣JavaScript" objectType="PossibleFire"><![CDATA[
                    this.PossibleFireDetected.getFireSensor1().getPossibleFire()]]></
                    ObjectRef>
330               <Slot property="duration">
331                 <ValueExpr language="Java␣JavaScript"><![CDATA[this.PossibleFireDetected.
                      getOccurrenceTime() - this.PossibleFireDetected.getFireSensor1().
                      getPossibleFire().getStartTime()]]></ValueExpr>
```

```
332                 </Slot>
333                 <Slot property="finalTime">
334                   <ValueExpr language="Java JavaScript"><![CDATA[this.PossibleFireDetected.
                          getOccurrenceTime()]]></ValueExpr>
335                 </Slot>
336               </UpdateObject>
337             </UPDATE-ENV>
338           </ELSE>
339         </EnvironmentRule>
340
341
342
343
344 <EnvironmentRule name="count10">
345   <WHEN eventType="Step" eventVariable="Step"/>
346   <FOR objectVariable="FireSensor1" objectType="FireSensor" objectIdRef="11"/>
347   <FOR objectVariable="FireSensor2" objectType="FireSensor" objectIdRef="12"/>
348   <IF language="Java JavaScript">
349     <![CDATA[ this.Step.getOccurrenceTime() == 10]]>
350   </IF>
351   <THEN>
352   <UPDATE-ENV>
353     <UpdateObject objectVariable="FireSensor1">
354       <Slot property="Active" value="true"/>
355     </UpdateObject>
356     <UpdateObject objectVariable="FireSensor2">
357       <Slot property="Active" value="true"/>
358     </UpdateObject>
359   </UPDATE-ENV>
360   </THEN>
361 </EnvironmentRule>
362
363 <EnvironmentRule name="count20">
364   <WHEN eventType="Step" eventVariable="Step"/>
365   <FOR objectVariable="FireSensor1" objectType="FireSensor" objectIdRef="11"/>
366   <FOR objectVariable="FireSensor2" objectType="FireSensor" objectIdRef="12"/>
367   <FOR objectVariable="Agent1" objectType="Agent" objectIdRef="31"/>
368   <FOR objectVariable="Agent2" objectType="Agent" objectIdRef="32"/>
369   <IF language="Java JavaScript">
370     <![CDATA[ this.Step.getOccurrenceTime() == 20]]>
371   </IF>
372   <THEN>
373   <UPDATE-ENV>
374     <Create>
375       <Object type="ConfirmEvent" id="51" objectVariable="NewConfirmEvent1"/>
376     </Create>
377     <UpdateObject objectVariable="NewConfirmEvent1">
378       <Slot property="Agent">
379          <ObjectValueExpr objectVariable="Agent1"/>
380       </Slot>
381       <Slot property="FireSensor">
382          <ObjectValueExpr objectVariable="FireSensor1"/>
383       </Slot>
384     </UpdateObject>
385     <Create>
386       <Object type="ConfirmEvent" id="52" objectVariable="NewConfirmEvent2"/>
387     </Create>
388     <UpdateObject objectVariable="NewConfirmEvent2">
389       <Slot property="Agent">
390          <ObjectValueExpr objectVariable="Agent2"/>
```

```
391         </Slot>
392         <Slot property="FireSensor">
393            <ObjectValueExpr objectVariable="FireSensor2"/>
394         </Slot>
395       </UpdateObject>
396     </UPDATE-ENV>
397     </THEN>
398 </EnvironmentRule>
399
400 <EnvironmentRule name="count30">
401     <WHEN eventType="Step" eventVariable="Step"/>
402     <FOR objectVariable="FireSensor3" objectType="FireSensor" objectIdRef="13"/>
403     <FOR objectVariable="FireSensor4" objectType="FireSensor" objectIdRef="14"/>
404     <IF language="Java␣JavaScript">
405       <![CDATA[ this.Step.getOccurrenceTime() == 30]]>
406     </IF>
407     <THEN>
408     <UPDATE-ENV>
409       <UpdateObject objectVariable="FireSensor3">
410         <Slot property="Active" value="true"/>
411       </UpdateObject>
412       <UpdateObject objectVariable="FireSensor4">
413         <Slot property="Active" value="true"/>
414       </UpdateObject>
415     </UPDATE-ENV>
416     </THEN>
417 </EnvironmentRule>
418
419 <EnvironmentRule name="count50">
420     <WHEN eventType="Step" eventVariable="Step"/>
421     <FOR objectVariable="FireSensor3" objectType="FireSensor" objectIdRef="13"/>
422     <FOR objectVariable="Agent3" objectType="Agent" objectIdRef="33"/>
423     <IF language="Java␣JavaScript">
424       <![CDATA[ this.Step.getOccurrenceTime() == 50]]>
425     </IF>
426     <THEN>
427     <UPDATE-ENV>
428       <Create>
429         <Object type="ConfirmEvent" id="53" objectVariable="NewConfirmEvent3"/>
430       </Create>
431       <UpdateObject objectVariable="NewConfirmEvent3">
432         <Slot property="Agent">
433            <ObjectValueExpr objectVariable="Agent3"/>
434         </Slot>
435         <Slot property="FireSensor">
436            <ObjectValueExpr objectVariable="FireSensor3"/>
437         </Slot>
438       </UpdateObject>
439     </UPDATE-ENV>
440     </THEN>
441 </EnvironmentRule>
442
443 <EnvironmentRule name="count60">
444     <WHEN eventType="Step" eventVariable="Step"/>
445     <FOR objectVariable="FireSensor1" objectType="FireSensor" objectIdRef="11"/>
446     <FOR objectVariable="FireSensor2" objectType="FireSensor" objectIdRef="12"/>
447     <IF language="Java␣JavaScript">
448       <![CDATA[ this.Step.getOccurrenceTime() == 60]]>
449     </IF>
450     <THEN>
```

```
451   <UPDATE-ENV>
452     <UpdateObject objectVariable="FireSensor1">
453       <Slot property="Active" value="false"/>
454     </UpdateObject>
455     <UpdateObject objectVariable="FireSensor2">
456       <Slot property="Active" value="false"/>
457     </UpdateObject>
458   </UPDATE-ENV>
459   </THEN>
460 </EnvironmentRule>
461
462     </EnvironmentRules>
463   </SimulationModel>
464
465   <InitialState>
466     <ExogenousEvent type="Step" occurrenceTime="1"/>
467
468     <Object type="FireSensor" id="11">
469       <Slot property="Active" value="false"/>
470       <Slot property="Location" value="1"/>
471     </Object>
472     <Object type="FireSensor" id="12">
473       <Slot property="Active" value="false"/>
474       <Slot property="Location" value="2"/>
475     </Object>
476     <Object type="FireSensor" id="13">
477       <Slot property="Active" value="false"/>
478       <Slot property="Location" value="5"/>
479     </Object>
480     <Object type="FireSensor" id="14">
481       <Slot property="Active" value="false"/>
482       <Slot property="Location" value="6"/>
483     </Object>
484
485     <Object type="EmergencyExit" id="21">
486       <Slot property="Location" value="2"/>
487     </Object>
488     <Object type="EmergencyExit" id="22">
489       <Slot property="Location" value="3"/>
490     </Object>
491     <Object type="EmergencyExit" id="23">
492       <Slot property="Location" value="8"/>
493     </Object>
494
495     <Object type="Agent" id="31"/>
496     <Object type="Agent" id="32"/>
497     <Object type="Agent" id="33"/>
498
499   </InitialState>
500     <UserInterface supportedLanguages="en">
501     </UserInterface>
502 </SimulationScenario>
```

ANEXO A – Metamodelo de Contexto da SML

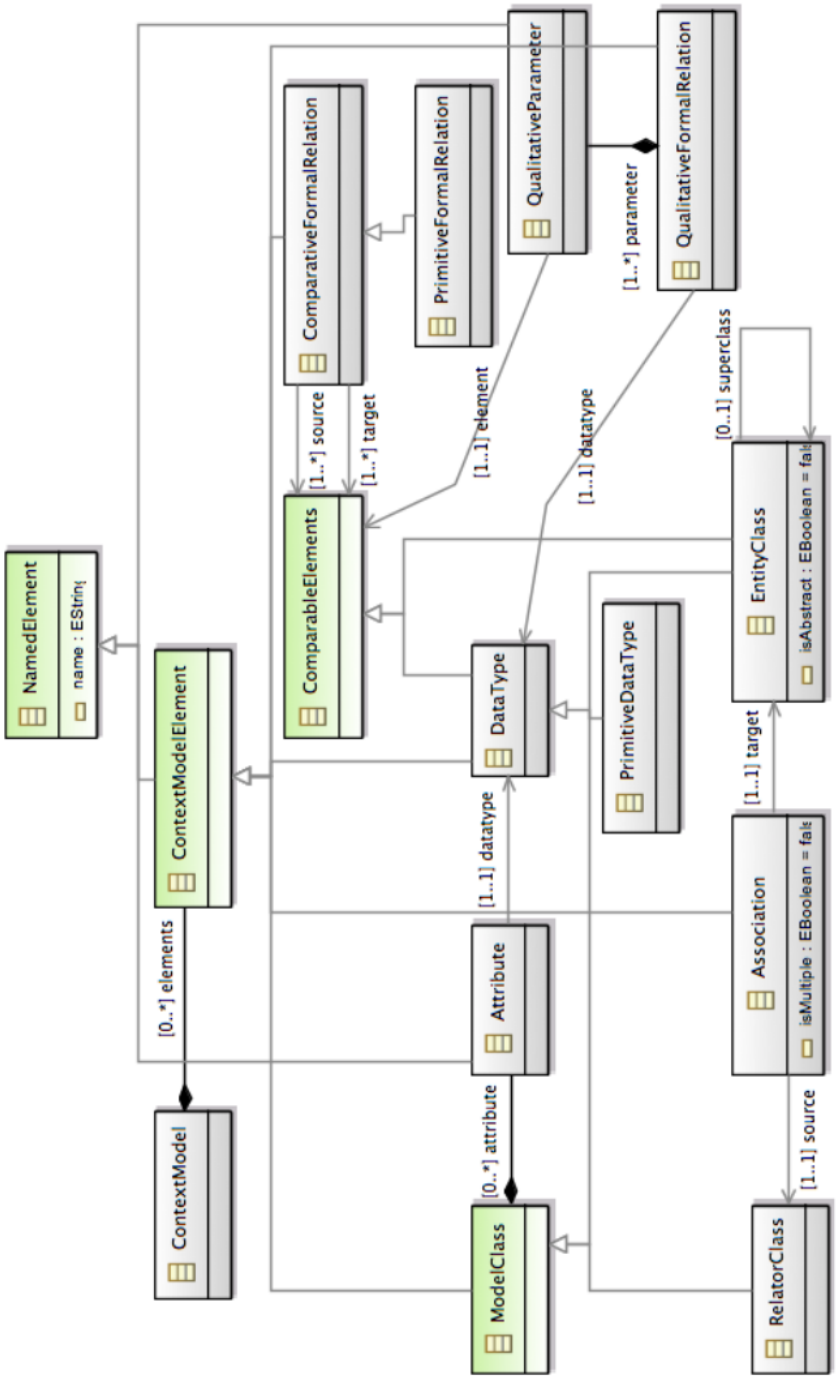


Figura 27: Metamodelo de Contexto da SML (MIELKE, 2013)

ANEXO B – Metamodelo de Situações da SML

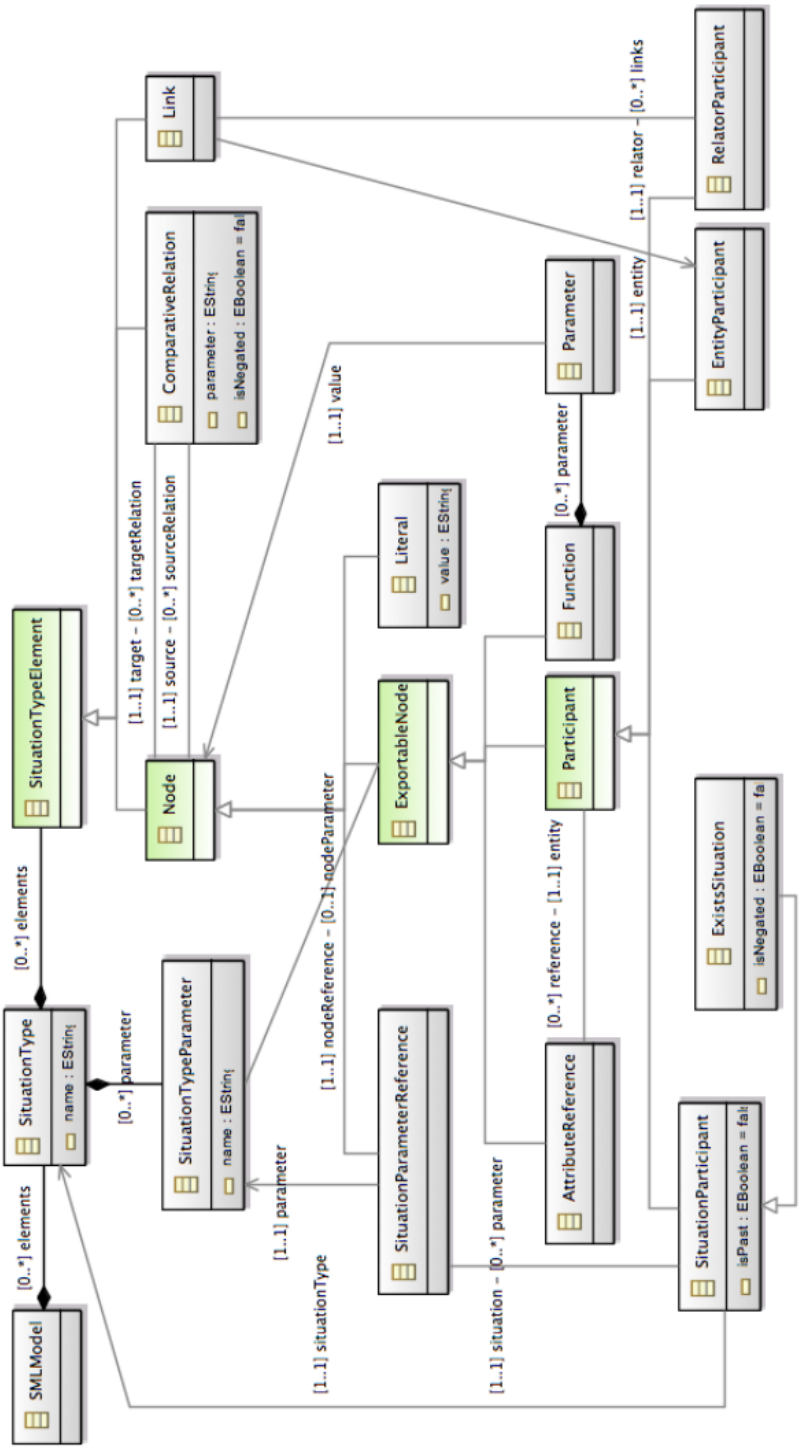


Figura 28: Metamodelo de Situação da SML (MIELKE, 2013)