Igor Magri Vale

UMA ABORDAGEM ORIENTADA A MODELOS PARA DESENVOLVIMENTO DE APLICAÇÕES SENSÍVEIS AO CONTEXTO NO AMBIENTE DE TV DIGITAL

Vitória - ES
15 de abril de 2011

Igor Magri Vale

UMA ABORDAGEM ORIENTADA A MODELOS PARA DESENVOLVIMENTO DE APLICAÇÕES SENSÍVEIS AO CONTEXTO NO AMBIENTE DE TV DIGITAL

Dissertação apresentada para obtenção do grau de Mestre em Informática pela Universidade Federal do Espírito Santo.

Orientadora:
Prof[a]. Dr[a]. Patrícia Dockhorn Costa

Co-Orientador:
Prof. Dr. João Paulo Andrade Almeida

UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO
CENTRO TECNOLÓGICO
DEPARTAMENTO DE INFORMÁTICA

Vitória - ES
15 de abril de 2011

Dissertação de Mestrado sob o título "Uma Abordagem Orientada a Modelos Para Desenvolvimento de Aplicações Sensíveis ao Contexto no Ambiente de TV Digital", defendida por Igor Magri Vale e aprovada em 15 de abril de 2011, em Vitória, Estado do Espírito Santo, pela banca examinadora constituída por:

_____

Profª. Drª. Patrícia Dockhorn Costa
Orientadora

_____

Prof. Dr. João Paulo Andrade Almeida
Co-Orientador

_____

Profª. Drª. Roberta Lima Gomes
Universidade Federal do Espírito Santo

_____

Profª. Drª. Flávia Coimbra Delicato
Universidade Federal do Rio Grande do Norte

DEDICATÓRIA

À minha mãe, por estar ao meu lado sempre!

À minha avó Felicina e ao meu avô Telmo que, juntamente com a minha mãe,

são os responsáveis por cada uma das minhas conquistas!

AGRADECIMENTOS[1]

Inicialmente, gostaria de agradecer à minha família, que sem dúvida me forneceu a base para que eu conseguisse alcançar meus objetivos. Também gostaria de agradecer aos meus amigos, seja pelo apoio ao longo do mestrado, seja pelos momentos de diversão.

Devo agradacer também aos professores e bolsistas que trabalharam junto comigo no projeto CTIC – GingaFrEvo & GingaRAP. As discussões feitas ao longo dos entregáveis do projeto foram fundamentais para o desenvolvimento deste trabalho.

Um agradecimento especial deve ser feito ao Izon. Sua ajuda foi importantíssima para a implementação dos cenários e para o entendimento do ambiente Ginga e suas linguagens NCL e NCLua.

Finalmente, gostaria de agradecer à minha orientadora, Patrícia, e ao meu co-orientador, João Paulo. Dedicados, estiveram sempre dispostos e prontos para ajudar. Aprendi bastante com eles e sem as suas contribuições, ideias e comentários, este trabalho não seria possível.

A todos vocês, meu muito obrigado!

[1]Este projeto é financiado parcialmente pela CAPES – Brasil (Projeto RH-TVD #225/2008)

RESUMO

Com a implantação do Sistema Brasileiro de Televisão Digital, e com a possibilidade de criação de aplicações interativas, o domínio de Televisão Digital (TVD) é um ambiente a ser explorado por aplicações inovadoras, como é o caso das *Aplicações Sensíveis ao Contexto*. Aplicações Sensíveis ao Contexto utilizam informações contextuais do usuário para acionar serviços de acordo com a necessidade ou a situação atual do usuário, e podem, portanto, potencialmente enriquecer a experiência do usuário ao assistir TV. O objetivo desta dissertação é propor uma metodologia orientada a modelos para auxiliar o desenvolvimento dessas aplicações no ambiente de TV Digital, desde a fase de modelagem à realização, tendo como alvo de implementação a plataforma Ginga, que é o *middleware* definido pelo SBTVD para desenvolvimento de aplicações interativas. A metodologia oferece suporte à modelagem do universo de discurso da aplicação através de modelos de contexto e situação, bem como à definição de comportamentos reativos de aplicações usando uma abordagem baseada em regras. Para isso, este trabalho propõe uma linguagem baseada em regras denominada ECA-DL TVD, que permite a especificação de comportamentos reativos através de eventos, condições e ações. Este trabalho também define uma arquitetura conceitual, que visa estruturar o Gerenciador de Contexto, um dos componentes do Ginga. Uma das principais contribuiçoes deste trabalho volta-se para a geração automática de código NCL referente a (parte) dessa arquitetura conceitual. O trabalho propõe a utilização de técnicas da área de desenvolvimento orientado a modelos (MDD) e *frameworks* de transformação para que, a partir de modelos de contexto, situações e regras ECA-DL TVD, seja gerado código NCL. Finalmente, são apresentados dois cenários de aplicação que visam demonstrar a validade da metodologia proposta.

Palavras-chave: Aplicações Sensíveis ao Contexto, TV Digital, Ginga, NCL, MDD.

ABSTRACT

With the definition of the Brazilian System of Digital Television, and the possibility of creating interactive applications, the field of Digital Television (DTV) is an environment to be explored by innovative applications, such as Context-Aware Applications. Context-aware applications use contextual information from the user to trigger services according user's needs or his current situation, and therefore can potentially enrich the user experience when watching TV. The objective of this dissertation is to propose a model-driven methodology to assist the development of these applications in the DTV environment, starting from the modeling phase all the way to the implementation. The target platform is called Ginga, which is the middleware defined by SBTVD in order to support the development of interactive applications. The methodology proposed by this work offers support for (i) modeling the universe of discourse of context-aware applications by means of context models, situation models and (ii) specifying the reactive behaviours of these applications using a rule-based approach. For that, this work proposes a domain-specific language, coined ECA-DL TVD, which allows the specification of reactive behaviours by means of events, conditions and actions. This work also defines a conceptual architecture, which is used to structure the Ginga Component called Context Manager. One of the main contributions of this work lies in the automatic generation of NCL code concerning part of this conceptual architecture, using transformation frameworks and other Model-Driven Development techniques. Finally, two application scenarios are presented in order to demonstrate the viability of such methodology.

LISTA DE SIGLAS/ACRÔNIMOS

EMF – Eclipse Modeling Framework

MDD – Model Driven Development

NCL – Nested Context Language

PIM – Platform-Independent Model

PSM – Platform Specific Model

OCL – Object Constraint Language

SBTVD – Sistema Brasileiro de Televisão Digital

TVD – Televisão Digital

UML – Unified Modeling Language

SUMÁRIO

1 INTRODUÇÃO 13
1.1 MOTIVAÇÃO 13
1.2 OBJETIVOS 15
1.3 ABORDAGEM 16
1.4 ESTRUTURA 18

2 CONCEITOS BÁSICOS 20
2.1 CONTEXTO E APLICAÇÕES SENSÍVEIS AO CONTEXTO 20
2.1.1 Contexto 20
2.1.2 Aplicação Sensível ao Contexto 21
2.1.3 Situações 23
2.1.4 Resumo dos conceitos de contexto 23
2.2 DESENVOLVIMENTO DE APLICAÇÕES SENSÍVEIS AO CONTEXTO 24
2.2.1 Modelagem Contextual 25
2.2.2 Processo de Captura de Contexto 26
2.2.3 Reatividade a mudanças de contexto 26
2.2.4 Padrões Arquiteturais 27
2.3 TELEVISÃO DIGITAL 29
2.3.1 O *Middleware* Ginga 30
2.3.2 Aplicação NCL 31
2.3.3 A Linguagem Declarativa NCL 32
2.3.4 Scripts NCLua 33
2.3.5 Mecanismo Padrão de Tratamento de Eventos do NCLua 34
2.4 DESENVOLVIMENTO ORIENTADO A MODELOS 36
2.4.1 Modelo e metamodelo 36
2.4.2 Abordagem MDD 37
2.5 CONSIDERAÇÕES DO CAPÍTULO 39

3 TRABALHOS RELACIONADOS 41
3.1 MODELAGEM CONTEXTUAL 41
3.1.1 Suporte a computação sensível ao contexto em documentos estruturados para TV interativa 42
3.1.2 Arquitetura de serviço para avaliação de contextos em redes de TV digital 43
3.1.3 Discussão 44

3.2 ARQUITETURAS PARA MANIPULAÇÃO DE INFORMAÇÃO CONTEXTUAL 45
3.2.1 Uma arquitetura para suporte ao desenvolvimento de aplicações sensíveis a contexto em cenário de convergência 46
3.2.2 PersonalTVware: Uma proposta de arquitetura para apoiar a recomendação sensível ao contexto de programas de TV 47
3.2.3 Discussão 48
3.3 DESENVOLVIMENTO ORIENTADO A MODELOS 49
3.3.1 Composição de serviços sensíveis ao contexto usando políticas e modelos 50
3.3.2 Direcionamentos para o desenvolvimento orientado a modelos de aplicações pervasivas sensíveis ao contexto 51
3.3.3 Discussão 52

4 ARQUITETURA CONCEITUAL 53
4.1 PRINCÍPIOS E REQUISITOS 54
4.2 ARQUITETURA PROPOSTA PARA O GERENCIADOR DE CONTEXTO 55
4.2.1 Detalhamento da Arquitetura 56
4.3 DISCUSSÃO 59

5 ECA-DL TVD 62
5.1 MODELAGEM CONTEXTUAL 62
5.1.1 Entidades, Contextos e Situações 63
5.1.2 Exemplos de Modelos 64
5.2 ECA-DL TVD: CONCEITOS BÁSICOS 67
5.2.1 Navegação nos Modelos de Contexto e Situação 68
5.2.2 Eventos 69
5.3 SINTAXE E SEMÂNTICA 73
5.3.1 Elementos Contextuais 73
5.3.2 Cláusula *Upon* 75
5.3.3 Cláusula *Where* 77
5.3.4 Cláusula *Do* 79
5.3.5 Exemplos de regras ECA-DL TVD 81
5.4 METAMODELO 82
5.5 EXECUÇÃO DE UMA REGRA ECA-DL TVD 84
5.6 CONSIDERAÇÕES DO CAPÍTULO 85

6 APLICAÇÕES SENSÍVEIS AO CONTEXTO EM NCL 86
6.1 ARQUITETURA DE UMA APLICAÇÃO SENSÍVEL AO CONTEXTO EM NCL 86

6.2 ELEMENTOS DA LINGUAGEM NCL QUE FORNECEM SUPORTE AO DESENVOLVIMENTO DE APLICAÇÕES SENSÍVEIS AO CONTEXTO 87
6.2.1 Área Funcional *Components* 88
6.2.2 Área Funcional *Interfaces* 88
6.2.3 Área funcional *Connectors* 89
6.2.4 Área funcional *Linking* 93
6.3 METAMODELO SIMPLIFICADO DE NCL 94
6.4 SCRIPTS NCLUA 97
6.5 CONSIDERAÇÕES DO CAPÍTULO 99

7 REALIZAÇÃO ORIENTADA A MODELOS DO MONITOR 100
7.1 ENTIDADES, ATRIBUTOS E CONTEXTOS INTRÍNSECOS 100
7.2 CONTEXTOS RELACIONAIS 102
7.3 SITUAÇÕES CONTEXTUAIS 104
7.4 REGRA ECA-DL TVD 106
7.4.1 Cláusula *Upon* 106
7.4.2 Cláusula *When* 109
7.4.3 Cláusula *Do* 111
7.4.4 Janela de Eventos 116
7.5 CONSIDERAÇÕES DO CAPÍTULO 116

8 ESTUDOS DE CASO 118
8.1 CASA INTELIGENTE 118
8.1.1 Modelagem Contextual 119
8.1.2 Especificação do Comportamento Reativo 121
8.1.3 Estrutura Conceitual da Aplicação no Gerenciador de Contexto 122
8.1.4 Realização na Plataforma Ginga 124
8.2 POLÍTICA DE CONTROLE DE PRIVACIDADE 129
8.2.1 Modelagem Contextual 129
8.2.2 Especificação do Comportamento Reativo 131
8.2.3 Estrutura Conceitual da Aplicação no Gerenciador de Contexto 132
8.2.4 Realização na Plataforma Ginga 134
8.3 CONSIDERAÇÕES DO CAPÍTULO 139

9 CONSIDERAÇÕES FINAIS 141
9.1 CONCLUSÕES 141
9.2 TRABALHOS FUTUROS 143

REFERÊNCIAS **145**

PUBLICAÇÕES DO AUTOR **150**

1 INTRODUÇÃO

1.1 MOTIVAÇÃO

Durante os últimos anos, muitos ramos de pesquisa têm sido combinados para prover avanços na computação ubíqua (frequentemente conhecida como computação pervasiva) (HANSMANN, MERK, *et al.*, 2003). Computação ubíqua suporta a visão na qual a computação é transparentemente integrada no ambiente cotidiano. Nesta visão, há uma total integração da computação nas atividades humanas. Microprocessadores se tornam pequenos e baratos o suficiente para serem embutidos em quase tudo - não somente em dispositivos digitais, carros, eletroeletrônicos, brinquedos, ferramentas, mas também em objetos (lápis, por exemplo) e roupas. Uma característica essencial da computação ubíqua que possibilita esta transparência é a *sensibilidade ao contexto*, que lida com a habilidade das aplicações utilizarem informações sobre o ambiente do usuário (*contexto*) para acionar serviços de acordo com a necessidade ou a situação atual do usuário (DOCKHORN COSTA, 2007).

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”. Portanto, de acordo com essa definição, se uma informação é utilizada para caracterizar um participante da interação usuário-aplicação, esta informação é contexto. Capacitando os computadores com sensibilidade ao contexto dos usuários, a comunicação homem-computador pode ser enriquecida, levando a uma aplicação mais centrada no usuário e que oferece serviços mais adequados.

Outra área proeminente de pesquisa está no campo da Televisão Digital (TVD) (FERNANDES, LEMOS e ELIAS, 2004). A TVD é um tipo inovador de tecnologia para transmissão de áudio e vídeo, que introduz uma melhoria significativa na qualidade de som e imagem se comparado com a tecnologia atual da televisão convencional. A tecnologia de TVD também permite uma série de novos serviços, tais como vídeo sob demanda e interatividade. Interatividade, em particular, permite que os usuários contribuam e participem ativamente da programação televisiva (FERNANDES, LEMOS e ELIAS, 2004). Neste âmbito, verifica-se que a utilização de informações contextuais nas aplicações da TVD pode

potencialmente enriquecer a interação humano-TV, melhorando, portanto, a experiência do usuário ao assistir TV. Por exemplo, se um usuário gosta de esportes, a televisão – utilizando informações de posicionamento (contexto) e sobre o perfil do usuário – pode automaticamente sintonizar um canal sobre esportes quando este usuário se aproxima da televisão. Em um cenário diferente, informações de contexto poderiam ser usadas para detectar a presença de crianças na sala, o que levaria a TV a exibir somente a programação adequada para a faixa etária.

Um dos grandes desafios para a concepção dessas aplicações sensíveis ao contexto no ambiente de TVD é a dificuldade de se modelar adequadamente o contexto, haja vista a sua natureza complexa, heterogênea e distribuída (DOCKHORN COSTA, 2007). A maioria dos trabalhos que tratam da manipulação de informação contextual (THAWANI, GOPALAN e SRIDHAR, 2004) (MOON, KIM, *et al.*, 2007), seja através da definição de modelos de contexto, seja pela definição de elementos arquiteturais não consideram uma abordagem formal e integrada de desenvolvimento de aplicações sensíveis contexto. A modelagem de contexto, ou seja, o processo de identificar condições de contexto relevantes para as aplicações, é uma fase fundamental no desenvolvimento de tais aplicações já que produz os modelos de referência que serão usados para promover o entendimento comum entre os diversos atores em um certo domínio, e que servirão como insumo para as fases subsequentes do processo de desenvolvimento (MYLOPOULOS, 1998).

Outro grande desafio está relacionado à carência de abordagens sistemáticas para lidar com a diversidade de fontes de dados contextuais, combinada com as múltiplas tecnologias de sensoriamento atualmente disponíveis (SALBER, DEY e ABOWD, 1999). Existe, assim, a necessidade da investigação e da captura de problemas e soluções comuns com respeito à modelagem de contexto e à aquisição de dados contextuais provenientes de diversas fontes de naturezas heterogêneas.

Além disso, existem os desafios inerentes à plataforma de TVD brasileira. No Brasil, o Sistema Brasileiro de Televisão Digital (SBTVD) (SBTVD, 2009) define um *middleware* próprio, denominado Ginga (LABORATÓRIO TELEMÍDIA, 2007), para desenvolvimento de aplicações interativas. As aplicações desenvolvidas para esse *middleware* podem ser declarativas, utilizando a linguagem NCL (ABNT NBR 15606-2, 2007), ou procedurais, utilizando Java (SOUZA FILHO, LEITE e BATISTA, 2007). Porém, o suporte a aplicações procedurais não é obrigatório para dispositivos receptores móveis. Portanto, espera-se que grande parte das aplicações seja desenvolvida utilizando NCL, que é uma linguagem recente, necessitando, assim, adaptar as atuais ferramentas de desenvolvimento de aplicações sensíveis ao contexto a essa nova linguagem.

Além da dificuldade imposta pela nova linguagem de desenvolvimento, o problema de aquisição de dados contextuais provenientes de diversas fontes de naturezas heterogêneas também é exarcebado no caso do Ginga. Conforme verificado por (VALE, MIELKE, *et al.*, 2010), o Ginga ainda carece de serviços genéricos que deem suporte à captura e manipulação de contexto.

Uma forma de lidar com todos esses desafios é por meio de uma metodologia de desenvolvimento apropriada, partindo de um alto nível de abstração até a implementação na plataforma devida. Nesse caso, pode-se pensar na utilização de uma abordagem de Desenvolvimento Orientada a Modelos – Moden Driven Development (MDD) (ALMEIDA, PIRES e VAN SINDEREN, 2004) – em que as aplicações são estruturadas em um modelo que não leva em consideração especifidades/particularidades da plataforma, como a tecnologia e a linguagem específica que é utilizada por esta plataforma. Posteriormente, esse modelo é transformado em um modelo específico de plataforma que, por outro lado, considera essas especifidades/particularidades. Além de permitir o desenvolvimento em um nível de abstração alto, essa abordagem favorece o reuso, pois um mesmo modelo independente de plataforma pode ser transformado em diferentes modelos específicos, sendo modificada apenas a transformação.

Dessa forma, este trabalho visa definir uma metodologia orientada a modelos apropriada para o desenvolvimento de aplicações sensíveis ao contexto no ambiente de TVD, o que permitirá a concepção de aplicações interativas mais ricas e centradas no usuário, em diferentes domínios.

1.2 OBJETIVOS

Este trabalho tem como objetivo geral propor uma metodologia para auxiliar o desenvolvimento de aplicações sensíveis ao contexto no ambiente de TVD, desde a fase de modelagem à realização, sendo a plataforma Ginga o alvo de implementação.

Esse objetivo geral pode ser detalhado nos seguintes objetivos específicos:

- **Objetivo 1**: Definição de conceitos de modelagem contextual que sirvam de fundamentação para especificação de aplicações sensíveis ao contexto em diversos domínios. Os conceitos de modelagem propostos devem ser adequados para (i) definir o universo de discurso de uma aplicação; (ii)

especificar contextos e situações, que são estados particulares de interesse da aplicação, (iii) especificar eventos, que são caracterizados por transições entre situações, e (iv) definir regras de reatividade que especifiquem o comportamento reativo de aplicações sensíveis ao contexto.

- **Objetivo 2**: Definição de uma arquitetura conceitual composta de elementos arquiteturais que promovam o desenvolvimento de aplicações sensíveis ao contexto. Estes elementos devem atender às necessidades específicas do domínio de TVD considerando as características da plataforma de implementação (Ginga) e do domínio de aplicações.

- **Objetivo 3**: Realização de elementos da arquitetura conceitual do Objetivo 2 na plataforma Ginga. Estes elementos visam facilitar o desenvolvimento de aplicações sensíveis ao contexto e permitir a execução destas aplicações provendo um conjunto de serviços genéricos a serem utilizados pelas aplicações, em tempo de execução.

1.3 ABORDAGEM

A abordagem para a definição da metodologia de desenvolvimento de aplicações sensíveis ao contexto começa pela definição de técnicas de modelagem de contexto e situação, e pela definição de uma linguagem adequada para a especificação de comportamentos reativos dessas aplicações (Objetivo 1). Como ponto de partida para definição dessa metodologia, propõe-se a utilização de uma adaptação da abordagem de modelagem contextual definida por (DOCKHORN COSTA, 2007), que define abstrações para os conceitos contextuais baseadas em ontologias de fundamentação (GUIZZARDI, 2005). Como parte da modelagem, (DOCKHORN COSTA, 2007) propõe uma linguagem específica de domínio, denominada ECA-DL, para especificação de comportamentos reativos. Por exemplo, pode-se especificar em ECA-DL através de regras declarativas o seguinte comportamento: uma vez que uma pessoa chegue em casa, a TV deve ser ligada. Este trabalho define a linguagem ECA-DL TVD, que especializa ECA-DL de forma a atender os requisitos inerentes à plataforma Ginga.

Como a plataforma Ginga é o alvo de implementação das aplicações, deve-se utilizar mecanismos eficientes para realização dos modelos de contexto, situações e regras ECA-DL TVD nesta plataforma. Conforme mencionado na Seção 1.1, as aplicações podem ser

desenvolvidas no Ginga em linguagem declarativa, através da linguagem NCL, ou em linguagem procedural, utilizando a linguagem Java. Optou-se neste trabalho por focar na linguagem declarativa NCL, com *scripts* Lua (SANT'ANNA, CERQUEIRA e SOARES, 2008) uma vez que a máquina virtual Java não é obrigatória no Ginga para aplicações móveis.

Observou-se em trabalhos preliminares (VALE, MIELKE, *et al.*, 2010) que uma simples regra ECA-DL TVD gera uma quantidade considerável de linhas de código NCL e demanda profundo conhecimento da linguagem e grandes esforços de programação. Portanto, a criação de regras ECA-DL TVD mais complexas em NCL torna-se praticamente inviável. Neste escopo, este trabalho propõe a utilização de técnicas da área de MDD (ALMEIDA, PIRES e VAN SINDEREN, 2004) a fim de permitir a geração automática de código NCL a partir de regras ECA-DL TVD. MDD permite automatizar o mapeamento entre as linguagens supracitadas a partir da especificação dos metamodelos de ECA-DL TVD e NCL e de *frameworks* de transformação de modelos (THE ECLIPSE FOUNDATION, 2010).

A definição da arquitetura conceitual (Objetivo 2) baseia-se nos padrões arquiteturais apresentados em (DOCKHORN COSTA, FERREIRA PIRES e SINDEREN, 2005). Estes padrões arquiteturais permitem identificar os componentes de manipulação de contexto, situações e regras ECA-DL TVD que podem ser generalizados e incluídos como parte de uma *plataforma de gerenciamento de contexto*, que visa oferecer componentes e serviços genéricos para suportar o desenvolvimento de aplicações sensíveis ao contexto, independente do domínio da aplicação. Estes componentes e serviços podem ser combinados e configurados para satisfazer os requisitos de uma aplicação específica. Por exemplo, usando o padrão arquitetural *Event-Control-Action*, identifica-se a necessidade de distinguir as tarefas de captura e processamento de contexto, das tarefas que disparam ações em resposta a mudanças no contexto, sob o controle de uma descrição de um comportamento reativo (regra ECA-DL TVD). Assim, podem ser definidos componentes da plataforma que serão responsáveis por cada uma dessas tarefas identificadas.

Finalmente, com base nas capacidades e limitações da plataforma Ginga, elementos da arquitetura conceitual serão realizados nesta plataforma (Objetivo 3). A viabilidade dos modelos e da plataforma de gerenciamento de contexto é demonstrada através da construção de protótipos de aplicações sensíveis ao contexto para a plataforma Ginga. A técnica empregada inclui, primeiramente, a definição de modelos de contexto, situações e regras ECA-DL TVD. Seguindo a trajetória de desenvolvimento, são definidos e implementados os componentes de manipulação de contexto necessários para capturar e processar informações contextuais. Estes componentes são especializações dos componentes definidos na arquitetura conceitual mencionada anteriormente. As aplicações

especificadas em termos de modelos de contexto, situações e regras ECA-DL TVD são mapeadas automaticamente para NCL através de transformações de modelos ECA-DL TVD em NCL. Para isso, é utilizada a plataforma aberta de software livre Eclipse (THE ECLIPSE FOUNDATION, 2010), que permite desenvolvimento do mapeamento automático via metamodelos.

A Figura 1 resume a metodologia proposta pelo trabalho.

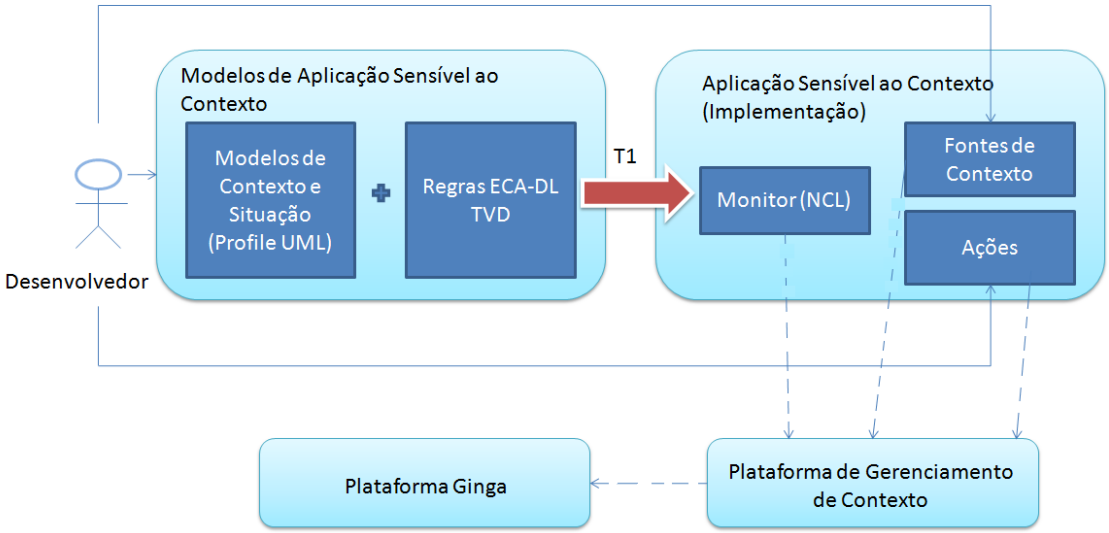


Figura 1. Visão geral da metodologia proposta.

Conforme mostra a Figura 1, o desenvolvedor modela a aplicação sensível ao contexto por meio de modelos de contexto e de situação, e descreve o comportamento reativo utilizando regras ECA-DL TVD. Esse conjunto sofre uma tranformação T1, utilizando técnicas de MDD, e gera uma instância do componente Monitor. O Monitor, as Fontes de Contexto e as Ações são elementos da Plataforma de Gerenciamento de Contexto proposta, que estrutura conceitualmente uma aplicação sensível ao Contexto. Na realização da arquitetura conceitual na Plataforma Ginga, o Monitor é um documento NCL, enquanto que as Fontes de Contexto e Ações são, geralmente, *scripts* NCLua que contêm comportamentos específicos da aplicação e são desenvolvidos à parte pelo desenvolvedor.

1.4 ESTRUTURA

Este trabalho está estruturado como segue:

- O Capítulo 2 apresenta os principais conceitos referentes à área de Sensibilidade ao Contexto, TVD e MDD;

- O Capítulo 3 discute os trabalhos relacionados analisando-os a fim de reforçar a justificativa e a relevância do trabalho que está sendo proposto, além de analisá-los quanto à possibilidade de atender aos objetivos definidos;

- O Capítulo 4 apresenta a arquitetura conceitual proposta para o componente Gerenciador de Contexto da plataforma Ginga. Segundo (SOARES e CASTRO, 2008), o Gerenciador de Contexto é um componente encarregado de coletar informações contextuais do dispositivo receptor, informações sobre o perfil do usuário e sua localização, para alterar a forma como os conteúdos televisivos deverão ser apresentados, conforme determinado pelas aplicações. Considerando diversos cenários de aplicações, essas funcionalidades são muito limitadas e, portanto, propõe-se uma arquitetura arquitetural para este componente a fim de promover o atual suporte oferecido pelo Ginga ao desenvolvimento de aplicações sensíveis ao contexto;

- O Capítulo 5 apresenta ECA-DL TVD, linguagem específica de domínio para especificação de regras que descrevem o comportamento reativo das aplicações sensíveis ao contexto. Também neste capítulo, discute-se a sintaxe e semântica da linguagem e define-se o seu metamodelo. Além disso, ilustra a utilização da abordagem de modelagem contextual definida por (DOCKHORN COSTA, 2007), que fornece os elementos de contexto e situação que são utilizados para definição de regras ECA-DL TVD;

- O Capítulo 6 discute a linguagem declarativa NCL e seus scripts NCLua, identificando os elementos que são úteis para o desenvolvimento de aplicações sensíveis ao contexto. Com base nesses elementos, é definido um metamodelo da linguagem NCL;

- O Capítulo 7 define os mapeamentos entre os elementos dos metamodelos de ECA-DL TVD e NCL;

- O Capítulo 8 demonstra a viabilidade da metodologia proposta com base em cenários de aplicação;

- Finalmente, o Capítulo 8 traz as conclusões e os trabalhos futuros.

2 CONCEITOS BÁSICOS

Este trabalho explora diversas áreas de pesquisa atuais, como Sensibilidade ao Contexto, TVD e MDD. Para uma melhor discussão sobre os trabalhos relacionados e as contribuições deste trabalho, este capítulo realiza uma revisão sobre os conceitos básicos dessas áreas de pesquisa.

O capítulo está organizado da seguinte forma: a Seção 2.1 apresenta definições de contexto e aplicações sensíveis ao contexto; a Seção 2.2 discute o processo de desenvolvimento de aplicações sensíveis ao contexto; a Seção 2.3 apresenta a área de pesquisa de TVD, mais especificamente o SBTVD e seu *middleware* Ginga; a Seção 2.4 discute alguns conceitos da área de MDD; e, finalmente a Seção 2.5 traz as considerações do capítulo.

2.1 CONTEXTO E APLICAÇÕES SENSÍVEIS AO CONTEXTO

Desde o início da década de 90, podem-se observar contribuições para área de sensibilidade ao contexto, particularmente na comunidade de Inteligência Artificial. Atualmente, com o grande desenvolvimento das tecnologias de comunicação e computação móvel e a proliferação dos dispositivos portáteis multifuncionais (ex: *smartphones*), contexto tornou-se um tópico destacado de pesquisas na comunidade de Ciência da Computação, recebendo especial interesse da nova área de pesquisa em Computação Ubíqua e Pervasiva (*"Ubiquitous / Pervasive Computing"*) (WEISER, 1991). Computação ubíqua suporta a visão na qual a computação é transparentemente integrada no ambiente cotidiano, sendo que uma característica essencial da computação ubíqua que possibilita esta transparência é a sensibilidade ao contexto, que lida com a habilidade das aplicações utilizarem informações sobre o ambiente do usuário (contexto) para acionar serviços de acordo com a necessidade ou a situação atual do usuário (DOCKHORN COSTA, 2007).

2.1.1 Contexto

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". Portanto, de acordo com essa definição, se um pedaço de informação é utilizado para caracterizar um participante da interação usuário-aplicação, esta informação é contexto.

Exemplos comuns de contexto são a localização de um usuário, a iluminação de um ambiente, a orientação de um dispositivo móvel, etc. A Figura 2 ilustra um modelo intuitivo de contexto, exibindo uma pessoa e os vários contextos que podem estar associados a esta pessoa (proximidade, localização, acesso a rede através de um dispositivo, etc.).

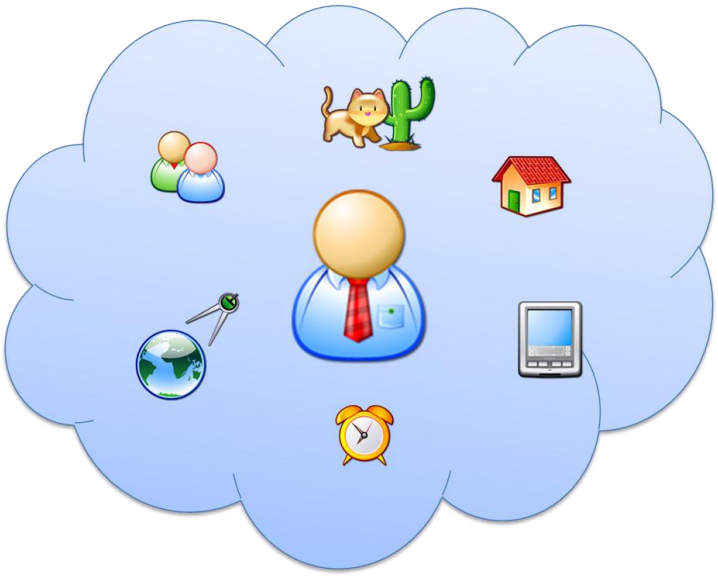


Figura 2. Contexto relacionado a um usuário (MIELKE, 2010).

2.1.2 Aplicação Sensível ao Contexto

De acordo com (DOCKHORN COSTA, 2007), uma aplicação sensível ao contexto pode ser definida como "uma aplicação distribuída, cujo comportamento é afetado pelo contexto do usuário". Em (DEY, 2001), a definição é semelhante, porém, também descreve o que é oferecido por uma aplicação sensível ao contexto: "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".

Portanto, pode-se entender Aplicação Sensível ao Contexto como uma aplicação que, para oferecer informações ou serviços para o usuário, faz uso de contexto. Uma aplicação pode, por exemplo, utilizar a localização do usuário a fim de informá-lo sobre a disponibilidade de serviços próximos, como restaurantes, lojas, etc. A Figura 3 mostra uma

visão intuitiva de um Usuário e seu contexto (Contexto do Usuário), e uma Aplicação Sensível ao Contexto (focando em um usuário apenas).

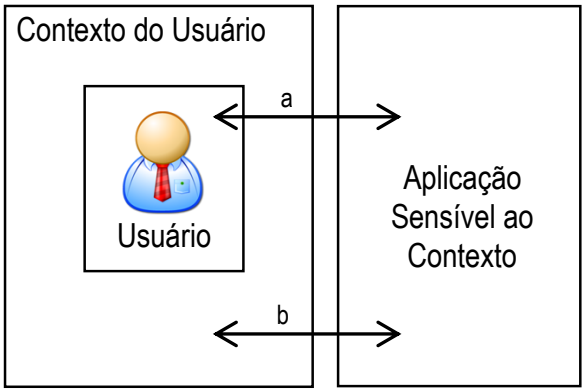


Figura 3. Visão intuitiva de aplicação sensível ao contexto interagindo com o usuário de contexto do usuário (DOCKHORN COSTA, 2007).

Na Figura 3, a seta “a” mostra que o usuário e a aplicação sensível ao contexto interagem. Semelhantemente, a seta “b” mostra que o contexto do usuário e a aplicação sensível ao contexto também interagem. Uma interação representada pela seta “a” pode ser exemplificada por entradas fornecidas pelo usuário à aplicação, ou pelo usuário utilizando serviços fornecidos pela aplicação. As interações do tipo “b” possibilitam a aplicação sensível ao contexto capturar condições contextuais do usuário, ou alterar os valores dessas condições.

Em (DEY, 2001), as aplicações sensíveis ao contexto são categorizadas segundo a finalidade a que elas se propõem:

- Apresentação de informação e serviços para o usuário;
- Execução automática de um serviço para o usuário;
- Adição de contexto à informação para futuras recuperações.

Como exemplos da primeira categoria, podem ser citados (SCHILIT, ADAMS e WANT, 1994) e (SCHMIDT, MICHAEL BEIGL e GELLERSEN, 1999). O primeiro discute sobre uma aplicação que exibe uma série de impressoras, em um dispositivo móvel, enfatizando as que estão mais próximas do usuário. O segundo mostra uma aplicação em que a orientação da exibição da interface com o usuário é de acordo com a própria orientação do dispositivo.

Na segunda categoria, estão aplicações que iniciam ações com base em informações contextuais. Um bom exemplo é encontrado no cenário de Telemedicina (PESSOA, CALVI, *et al.*, 2006), em que acionamento de alarmes e solicitação de serviços de ambulância são

gerados a partir da detecção de situações de risco. As condições de risco são avaliadas com base no monitoramento remoto de pacientes, sendo adquiridas algumas condições contextuais do usuário, como sinais cardíacos, temperatura e localização.

A terceira categoria pode ser exemplificada por uma aplicação que analisa e armazena informações sobre a utilização de dependências de uma empresa. O contexto de uma sala como, por exemplo, em qual horário foi ocupada, por quem e para quê ela foi utilizada, é armazenado e pode ser agregado a outras informações para atender futuras consultas.

2.1.3 Situações

Uma situação representa um *estado de interesse* de uma aplicação sensível ao contexto. Este tipo de aplicação, geralmente, não tem interesse apenas nos valores associados a uma determinada condição contextual, mas no significado que aquele valor representa. Uma aplicação pode, por exemplo, estar interessada em saber se uma pessoa está com febre, ou seja, se o valor da temperatura é maior que um determinado valor, sem estar necessariamente interessada no valor exato da temperatura.

Uma situação representa um contexto de alto nível de abstração, podendo ser formada por um conjunto de estados de interesse e combinada a outras situações. Por exemplo, uma situação que pode significar boas condições para ir à praia pode ser: “é manhã, não há previsão de chuva e não faz frio”.

Situações são identificadas por meio de inferências sobre informações contextuais, geralmente utilizando mecanismos baseados em regras (DOCKHORN COSTA, 2007). Quando existe uma mudança de contexto, é verificado se o novo estado da aplicação representa um estado de interesse. Caso a mudança de contexto seja considerada relevante, a aplicação pode reagir à mudança efetuando alguma ação, como invocar algum serviço.

2.1.4 Resumo dos conceitos de contexto

A Figura 4 mostra um diagrama UML que resume os conceitos de contexto apresentados até aqui.

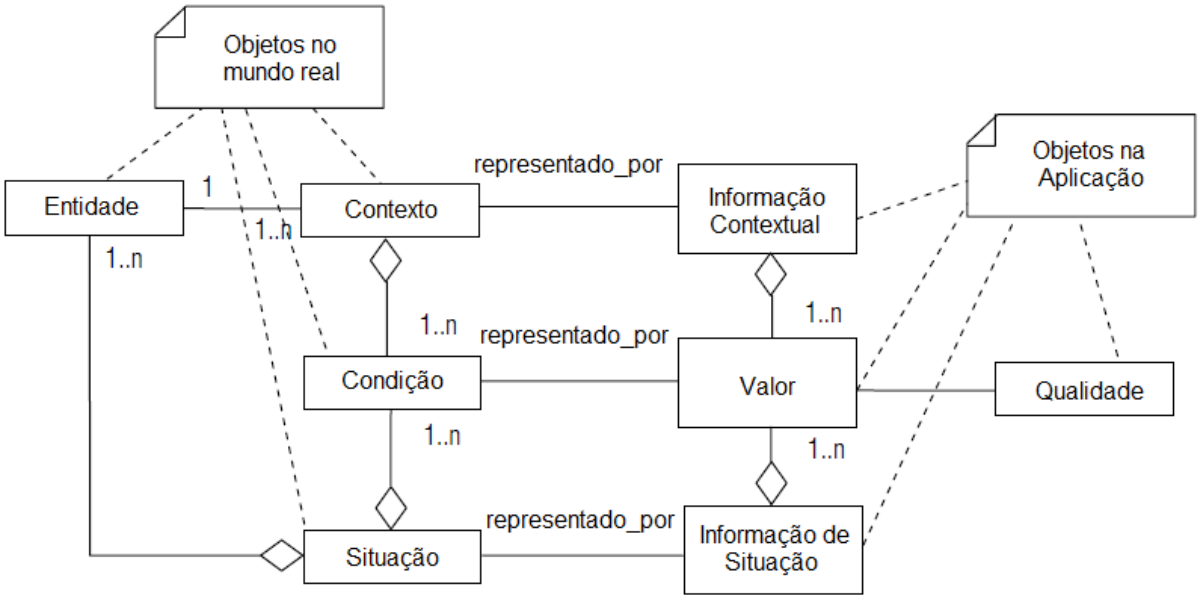


Figura 4. Resumo dos conceitos de contexto.

Este modelo representa o contexto no mundo real (ambiente do usuário) e na aplicação (como informação contextual). Contexto sempre se refere a uma entidade, e pode ser representado por uma coleção de condições (contextuais). Nas aplicações sensíveis ao contexto, contexto é referenciado como uma informação contextual, e suas correspondentes condições são representadas por valores na aplicação. Além disso, esses valores são relacionados a algum parâmetro de qualidade de medida, que determina a qualidade da informação, ou seja, quão precisa, exata, recente é a informação. Determinar a qualidade do contexto é importante, pois as aplicações sensíveis ao contexto são extremamente dependentes da qualidade dos sensores e dos seus mecanismos para captura de contexto. Contudo, não é do escopo deste trabalho a análise da qualidade de contexto.

Figura 4 também representa as situações, que são composições particulares de entidades e condições contextuais. Condições contextuais numa situação podem pertencer a diferentes entidades. Uma situação é representada como informação de situação numa aplicação sensível ao contexto.

2.2 Desenvolvimento de Aplicações Sensíveis ao Contexto

O desenvolvimento de uma aplicação sensível ao contexto é uma tarefa desafiadora, pois, geralmente, esse tipo de aplicação deve ser capaz de: (i) detectar o contexto do ambiente, (ii) observar, coletar e compor as informações contextuais utilizando diversos

meios (frequentemente sensores), (iii) automaticamente detectar mudanças relevantes no contexto, (iv) reagir a estas mudanças, adaptando seu comportamento ou invocando (composição de) serviços, e (v) interoperar com serviços providos por terceiros. Estes desafios exigem que a plataforma de apoio para o desenvolvimento de aplicações sensíveis ao contexto forneça mecanismos capazes de atender aos requisitos relacionados a este tipo de aplicação.

2.2.1 Modelagem Contextual

Uma parte importante do processo de desenvolvimento de uma aplicação sensível ao contexto consiste em realizar uma modelagem contextual. Um modelo contextual identifica e representa as condições contextuais das entidades que são relevantes para a aplicação sensível ao contexto. A localização e a temperatura de uma pessoa são exemplos de condições contextuais de uma entidade. Neste trabalho, entende-se um modelo de contexto como um modelo conceitual de contexto. Modelos conceituais são, de acordo com (GUIZZARDI, 2005), representações abstratas de um determinado domínio, independentemente de um *design* específico ou escolhas tecnológicas. Portanto, em um modelo conceitual de contexto, abstrai-se como esse contexto é detectado, providenciado, aprendido, produzido e/ou usado.

A importância de uma modelagem conceitual no processo de *design* da aplicação é enfatizada por (GUIZZARDI, 2005): “especificações conceituais são usadas para suportar o entendimento, solução do problema, e comunicação, entre os *stakeholders* sobre um determinado domínio. Uma vez que um entendimento e acordo sobre um determinado domínio é alcançado, a especificação conceitual é usada como um *blueprint* para as subsequentes fases do desenvolvimento do sistema”. Portanto, a qualidade de uma aplicação sensível ao contexto depende da qualidade dos modelos conceituais em que o desenvolvimento se baseia. Segundo (DOCKHORN COSTA, 2007), “a modelagem conceitual de contexto deve anteceder o *design* detalhado das aplicações sensíveis ao contexto, da mesma forma que as análises devem anteceder o *design* detalhado de um sistema de informação”. Um modelo conceitual de contexto abstrai como o contexto é detectado, providenciado, aprendido, produzido e/ou usado.

A modelagem conceitual de contexto pode ser baseada em Ontologias de Fundamentação (GUIZZARDI, 2005), que proveem um conjunto de conceitos para representar conceitualizações que são fiéis à realidade. Modelos conceituais obtidos usando

conceitos bem fundamentados ajudam a caracterizar mais precisamente o domínio que eles representam.

2.2.2 Processo de Captura de Contexto

As informações contextuais são disponibilizadas para as aplicações sensíveis ao contexto por meio das chamadas *fontes de contexto*, que capturam e fornecem informações contextuais de interesse para aplicação, permitindo que estas operem independentemente de como as informações são capturadas. Uma fonte de contexto pode (ou não) ser externo à aplicação e disponibilizar contexto para várias aplicações. Geralmente, essas fontes de contexto são sensores, que fornecem, por exemplo, dados sobre localização, temperatura, etc.

Também é possível que essas fontes de contexto apenas coletem informações contextuais fornecidas manualmente pelo usuário como, por exemplo, sua idade, nome, etc. Mais ainda, essas fontes de contexto podem coletar informações contextuais sobre a própria aplicação, como no caso em que a própria aplicação obtém informações sobre um determinado conteúdo televisivo (classificação, gênero, etc).

De qualquer forma, uma fonte de contexto deve fornecer informações contextuais relevantes e no nível de abstração requerido pela aplicação. Suponha, por exemplo, que uma aplicação esteja interessada em identificar a ocorrência de arritmia cardíaca em um paciente. Esta aplicação pode obter dados brutos de um sensor de eletrocardiograma e processar estes dados a fim de refinar as informações fornecidas por este sensor, permitindo, por exemplo, classificar o estado cardíaco do paciente e monitorar estes valores a fim de identificar possíveis irregularidades cardíacas. Por outro lado, o sensor já pode refinar os dados brutos e informar os estados cardíacos do paciente. Contextos “brutos”, geralmente detectados diretamente de um sensor, são denominados *contextos primitivos*, enquanto que os contextos processados, de mais alto nível, são denominados de *contextos compostos*.

2.2.3 Reatividade a mudanças de contexto

As aplicações sensíveis ao contexto devem reagir a mudanças de contexto, adaptando seu comportamento ou invocando serviços. Este comportamento pode ser atendido por

meio de um sistema baseado em regras (DOCKHORN COSTA, 2007), no qual o contexto é continuamente monitorado a fim de verificar se certas condições (especificação da situação) são satisfeitas. Ao contrário das soluções procedurais, soluções baseadas em regras normalmente oferecem flexibilidade com relação à manutenção das regras, que podem ser modificadas e adicionadas durante a execução da aplicação sem necessidade de recompilação.

De maneira geral, sistemas baseados em regras atendem a um domínio de problemas por meio de uma base de conhecimento expressa em termos de *regras*, que são repetidamente aplicadas a uma coleção de *fatos* (DOCKHORN COSTA, 2007). Estes dois conceitos são essenciais para um sistema baseado em regras:

- *Fatos* representam circunstâncias que descrevem certa situação no mundo real;
- *Regras* representam heurísticas que definem um conjunto de ações para serem executadas numa dada situação.

Uma máquina de inferência é responsável por comparar os fatos de uma base de fatos com as regras de base de regras que determina quais regras são aplicáveis (de acordo com o método de raciocínio adotado), e executar as ações associadas às regras aplicávéis (DOCKHORN COSTA, 2007).

2.2.4 Padrões Arquiteturais

Padrões arquiteturais têm sido propostos em vários domínios como meio de captação de soluções para os problemas recorrentes de *design*, que surgem em situações de projeto específico. Os padrões contêm experiências de desenvolvimento bem comprovadas, permitindo a reutilização dessas experiências por outros profissionais.

No caso de aplicações sensíveis ao contexto, (DOCKHORN COSTA, 2007) apresenta três padrões arquiteturais que apresentam soluções para problemas recorrentes associados ao gerenciamento de informações contextuais de forma a reagir a mudanças de contexto. Estes padrões são discutidos nas seções que seguem.

Padrão Arquitetural *Event-Control-Action*

O padrão arquitetural *Event-Control-Action* provê uma estrutura de alto nível para os sistemas que reagem a mudanças de contexto. Esse padrão foi elaborado a fim de dissociar

aspectos de contexto dos aspectos de reação (comunicação e uso do serviço). A estrutura deste padrão arquitetural pode ser vista na Figura 5.

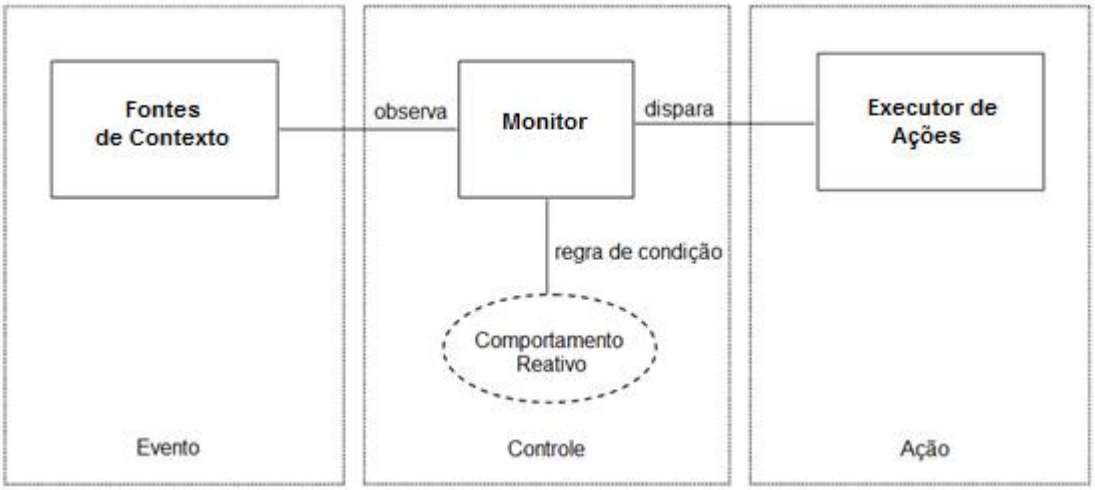


Figura 5. Estrutura do padrão de arquitetura *Event-Control-Action*.

Aspectos contextuais são tratados pelas *Fontes de Contexto*, que gera e observa *Eventos*, os quais, por sua vez, são mudanças de contexto ou situação. O componente *Monitor* exerce o *Controle* do comportamento reativo da aplicação. Ele observa as condições contextuais e aciona uma ou mais ações com base nas mudanças de contexto. O componente *Executor de Ações* trata dos aspectos relacionados às Ações, como a decomposição e ligação entre elas. Uma ação pode ser um serviço web simples de chamada ou uma entrega de SMS, ou pode ser uma composição complexa de serviços.

Padrão Arquitetural de Hierarquia de Fontes e Gerenciadores de Contexto e Padrão Arquitetural de Ações

Além do padrão *Event-Condition-Action*, (DOCKHORN COSTA, 2007) também propõe o padrão arquitetural de *Hierarquia de Fontes e Gerenciadores de Contexto* (C*ontext Sources and Managers Hierarchy Architectural Pattern*) e o padrão arquitetural de ações (*Actions Architectural Pattern*).

O padrão arquitetural de Hierarquia de Fontes e Gerenciadores de Contexto provê uma estrutura hierárquica dos componentes Fontes de Contexto. Este modelo foi concebido de forma a aplicar recursivamente as operações de processamento de informação contextuais de uma hierarquia de componentes fontes de contexto. Já o Padrão Arquitetural de Ações fornece uma estrutura de componentes de apoio à concepção e execução no que diz respeito à ação dentro da plataforma. Este modelo foi proposto a fim de separar os

propósitos das ações de suas respectivas implementações, assim como coordenar a composição das ações.

2.3 TELEVISÃO DIGITAL

A Televisão Digital (TVD) (FERNANDES, LEMOS e ELIAS, 2004) é um tipo inovador de tecnologia para transmissão de áudio e vídeo, que introduz uma melhoria significativa na qualidade de som e imagem se comparado com a tecnologia atual da televisão convencional. A melhoria da qualidade da imagem é facilmente percebida pelo telespectador. Dada a natureza discreta do conteúdo digital, é possível realizar uma recuperação muito precisa a partir do sinal recebido. Sendo assim, quando esse procedimento é feito corretamente, o conteúdo recebido é o mais próximo possível daquele que foi transmitido. De forma similar, a qualidade do áudio também sofre melhoras e mais canais de áudio também podem ser oferecidos. Isso possibilita o uso de tecnologias, como o *surround*, ou a escolha de línguas diferentes durante a exibição de um determinado programa (CRUZ, MORENO e SOARES, 2008).

A tecnologia de TVD também permite uma série de novos serviços, tais como vídeo sob demanda e interatividade. Interatividade, em particular, permite que os usuários contribuam para a geração e distribuição de conteúdos de televisão (FERNANDES, LEMOS e ELIAS, 2004). Neste âmbito, verifica-se que a utilização de informações contextuais nas aplicações da TVD pode potencialmente enriquecer a interação humano-TV, melhorando, portanto, a experiência do usuário ao assistir TV. Por exemplo, se um usuário gosta de esportes, a televisão – utilizando informações de posicionamento (contexto) – pode automaticamente sintonizar um canal sobre esportes quando este usuário se aproxima da televisão. Em um cenário diferente, informações de contexto poderiam ser usadas para detectar a presença de crianças na sala, o que levaria a TV a exibir somente a programação adequada para a faixa etária.

O Sistema Brasileiro de TVD (SBTVD) (SBTVD, 2009) é formado por vários padrões, que especificam a modulação, transmissão e codificação de dados, além de um *middleware*, que é uma camada posicionada entre as aplicações e a infraestrutura de execução que abstrai a complexidade dos protocolos de comunicação, do sistema operacional e do hardware do equipamento. A padronização de uma camada de *middleware* permite a construção de aplicações interativas independentes do hardware e do sistema operacional, facilitando a portabilidade das aplicações.

O SBTVD define de um *middleware* próprio, denominado Ginga (SOARES e CASTRO, 2008), que inova em alguns aspectos antes não considerados por outros *middlewares*, como o uso de múltiplos dispositivos (SOARES, COSTA, *et al.*, 2009), acesso à rede (ABNT NBR15607-1, 2008), edição ao vivo (SOARES, MORENO e MORENO, 2009), etc.

2.3.1 O *Middleware* Ginga

O *middleware* Ginga é dividido em dois ambientes, que permitem o desenvolvimento de aplicações seguindo os paradigmas declarativo e procedural. As aplicações do ambiente declarativo do Ginga, chamado de Ginga-NCL (SOARES, RODRIGUES e MORENO, 2007), são desenvolvidas utilizando a linguagem NCL (TELEMIDIA, 2006). Já as aplicações do ambiente procedural, conhecido como Ginga-J (SOUZA FILHO, LEITE e BATISTA, 2007) são baseadas na linguagem Java. De acordo com (SOARES e CASTRO, 2008), a camada Ginga-J não é obrigatória em todos os dispositivos. Assim, este trabalho foca no ambiente declarativo do Ginga (Ginga-NCL).

A Figura 6 mostra a arquitetura de referência do *middleware* Ginga, que pode ser dividida em três grandes subsistemas: A Máquina de execução Ginga-J (ambiente procedural), a Máquina de apresentação Ginga-NCL (ambiente declarativo) e o Núcleo comum Ginga-CC (*Common Core*).

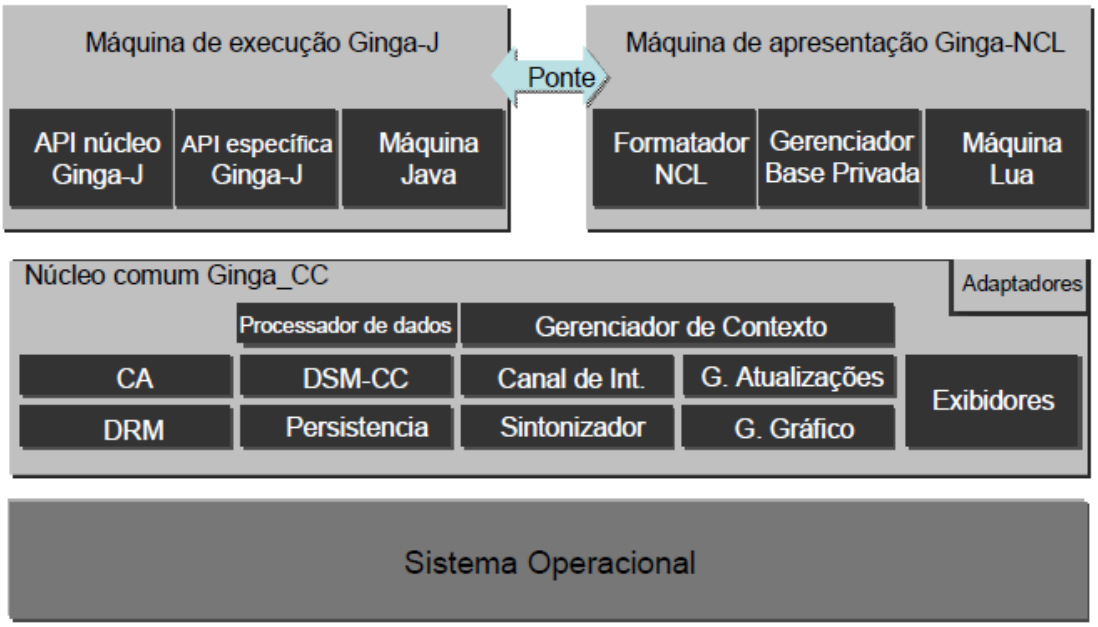


Figura 6. Arquitetura do middleware Ginga(SOARES e CASTRO, 2008).

O Núcleo Comum do Ginga (Ginga *Common Core*), mostrado na Figura 6, é responsável por realizar a comunicação direta com o sistema operacional e hardware. Este subsistema é formado por componentes que disponibilizam serviços comuns para as máquinas de apresentação (Ginga-NCL) e execução (Ginga-J). Na Figura 6 podem ser identificados dois componentes de possível interesse para o desenvolvimento de aplicações sensíveis ao contexto: O *Gerenciador de Contexto* e o *Canal de Interatividade*.

O Gerenciador de Contexto é um componente encarregado de coletar informações do dispositivo receptor, informações sobre o perfil do usuário e sua localização, e torná-las disponíveis ao Ginga-NCL e Ginga-J, para que eles possam efetuar adaptação de conteúdos ou da forma como conteúdos deverão ser apresentados, conforme determinado pelas aplicações (SOARES e CASTRO, 2008). Porém, por tratar contexto como apenas informações sobre o perfil do usuário e sua localização, o número de cenários de aplicações sensíveis ao contexto que poderiam ser desenvolvidos é bastante limitado.

O Canal de Interatividade, por sua vez, é um componente que provê acesso à rede no *middleware* Ginga, o que possibilita a comunicação do Ginga com dispositivos remotos. Essa funcionalidade pode ser explorada para, por exemplo, realizar comunicação com os sensores da aplicação sensível ao contexto.

2.3.2 Aplicação NCL

De acordo com (SOARES e CASTRO, 2008), a camada Ginga-J é opcional em dispositivos móveis, sendo as demais camadas obrigatórias a todos as implementações do Ginga para quaisquer dispositivos. Assim, este trabalho foca no desenvolvimento de aplicações interativas para o Ginga utilizando a linguagem declarativa NCL com elementos procedurais Lua, denominadas de Aplicações NCL. A arquitetura típica de uma aplicação NCL é mostrada na Figura 7.

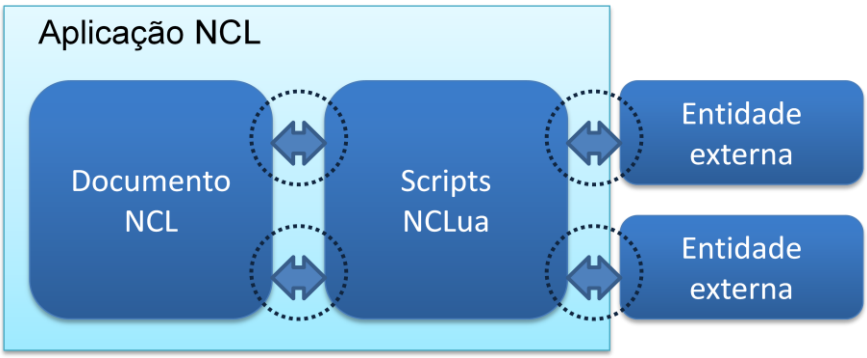


Figura 7. Arquitetura típica de uma Aplicação NCL.

Conforme mostrado na Figura 7, uma aplicação NCL (recebida e armazenada em um dispositivo recepetor de TVD com o *middleware* Ginga) é composta por um documento NCL e uma combinação de *scripts* NCLua. Estes *scripts* podem, por exemplo, realizar a comunicação, via rede, entre uma aplicação NCL e elementos remotas, como sensores, aplicações Web, etc.

2.3.3 A Linguagem Declarativa NCL

Linguagens declarativas são, em geral, voltadas para um determinado domínio e permitem a especificação de aplicações em um nível mais alto de abstração quando comparadas a linguagens imperativas. A *Nested Context Language* (NCL) (TELEMIDIA, 2006) é a linguagem declarativa, baseada em XML, adotada pelo ambiente Ginga-NCL. Esta linguagem permite o desenvolvimento de aplicações multimídia com sincronismo espaço-temporal entre objetos de mídia, como vídeos, áudios, imagens, etc. Com a finalidade de ampliar a gama de aplicações que podem ser desenvolvidas, NCL também suporta objetos de mídia imperativos, escritos na linguagem Lua (Seção 2.3.4). NCL atua como linguagem de cola, relacionando no tempo e espaço diferentes nós de mídia, criando o que se denomina de uma apresentação multimídia. Uma aplicação NCL deve conter detalhes sobre (SANT'ANNA, NETO, *et al.*, 2010):

- **O que tocar**: Definido por objetos de mídia (elementos <media>), também chamados de nós de mídia, que podem ser um áudio, vídeo, imagem, etc.
- **Onde tocar**: Região da tela que será exibido o conteúdo a ser tocado.
- **Como tocar**: Exemplos de como tocar são o volume de um áudio, a transparência de uma imagem, etc.

- **Quando tocar**: Um determinado conteúdo pode ser tocado no inicio da aplicação, após o fim de outro objeto de mídia, junto a outro objeto de mídia, etc.

Um exemplo de sincronismo espaço temporal definido por NCL é: exibir um vídeo inicial em tela cheia; quando este vídeo atingir 10 segundos de exibição, uma imagem com 50% de transparência deve ser mostrada no canto superior da tela; esta imagem deve desaparecer após 20 segundos de exibição.

A Figura 8 mostra a estrutura básica de uma aplicação NCL, que é dividida em um cabeçalho (*tag* <head>) e um corpo (*tag* <body>). A aplicação da Figura 8 exibe um vídeo na tela, em uma determinada região (*tag* <regionBase>).

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<ncl id="exemplo"
xmlns="http://www.ncl.org.br/NCL3.0/EDTVProfile">
  <head>
    <regionBase>
      <region id="rg_video" width="100%" height="100%"/>
    </regionBase>
    <descriptorBase>
      <descriptor id="dp_video" region="rg_video"/>
    </descriptorBase>
  </head>
  <body>
    <port id="inicio" component="video"/>
    <media id="video" src="video.mpg" descriptor="dp_video"/>
  </body>
</ncl>
```

Figura 8. Estrutura básica de um documento NCL (MIELKE, 2010).

Mais detalhes sobre NCL podem ser encontrados em (SOARES e BARBOSA, 2009).

2.3.4 Scripts NCLua

Lua (LUA.ORG, 2011) é uma linguagem de *script* brasileira, conhecida por sua simplicidade, eficiência e portabilidade. Para se adequar ao ambiente de TVD e se integrar à NCL, a linguagem Lua foi estendida com novas funcionalidades que permitem, por exemplo, a comunicação entre os *scripts* Lua e o documento NCL. Neste caso, *scripts* Lua são chamados de NCLua (SANT'ANNA, CERQUEIRA e SOARES, 2008).

Os *scripts* NCLua dão às aplicações declarativas NCL a possibilidade de executar código imperativo, permitindo criar aplicações mais diversificadas. No ambiente declarativo do Ginga, o acesso ao Canal de Interatividade e às informações enviadas pela emissora

(multiplexadas em um fluxo de transporte) pode ser realizado por meio de *scripts* NCLua. Portanto, o uso de *scripts* NCLua é importante ao desenvolvimento de aplicações sensíveis ao contexto neste ambiente, uma vez que o acesso à rede, necessário, por exemplo, para obter informações contextuais de fontes de contexto externa, pode ser realizado através destes.

A comunicação entre *scripts* NCLua e os demais componentes de uma aplicação NCL é feita por meio de troca de eventos.

2.3.5 Mecanismo Padrão de Tratamento de Eventos do NCLua

O mecanismo de difusão e recepção de eventos do ambiente de NCLua pode ser comparado ao padrão *publish/subscribe*, no qual há um elemento, ou um serviço, que gerencia a troca de eventos entre produtores e consumidores, gerando um desacoplamento espaço temporal entre estes (EUGSTER, FELBER, *et al.*, 2003).

A Figura 9 mostra um diagrama que representa os componentes básicos que fazem parte do padrão *publish/subscribe*, dentre eles os produtores de eventos (*Publisher*) que utilizam um serviço de eventos (*Event Service*) para publicar eventos (*Publish*). O serviço de eventos é responsável por distribuir as notificações de eventos recebidas dos produtores, para isso deve fornecer funcionalidades para receber notificações (*Notify*), registrar (*Subscribe*) e desregistrar (*Unsubscribe*) consumidores de eventos (*Subscriber*). Os consumidores devem fornecer um método para recepção de notificações e, para serem notificados sobre eventos dos produtores, devem se registrar no serviço de eventos.

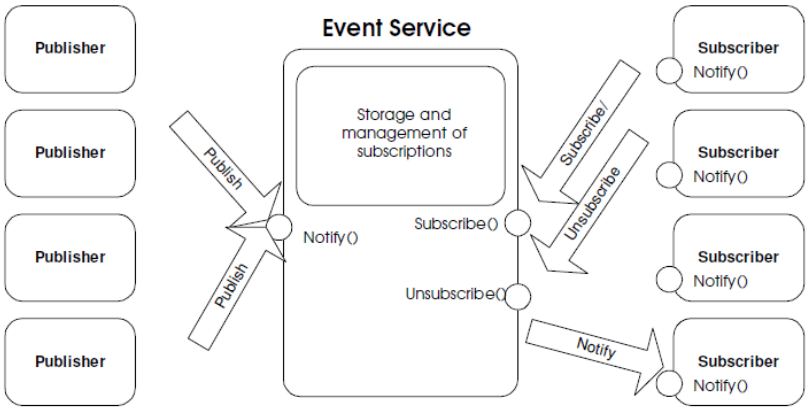


Figura 9. Componentes do padrão *publish/subscribe* (EUGSTER, FELBER, *et al.*, 2003).

Para realizar a troca de eventos, *scripts* NCLua devem fazer uso de um serviço de eventos deste ambiente, disponibilizado através de um módulo chamado *event* (também conhecido como mecanismo de tratamento de eventos). Este módulo fornece funcionalidades de publicação de eventos, através de um método *post* e registro e desregistro de consumidores pelos métodos *register* e *unregister,* respectivamente. Os consumidores de eventos, neste caso chamados de tratadores de eventos (*handlers*), são funções NCLua que recebem como único parâmetro um evento, representados por tabelas de Lua (LUA, 2010) e categorizados por classes (*class*) (SANT'ANNA, NETO, *et al.*, 2010). Cada classe de eventos é responsável por funcionalidades específicas, como por exemplo, a classe “ncl”, responsável pela comunicação com o documento NCL, a classe “tcp”, responsável pela comunicação com o Canal de Interatividade e a classe “key” responsável pelos eventos do controle remoto. Cada classe de eventos possui eventos relacionados, como por exemplo, o tipo de evento que representa a conexão (tipo “connect”) ou dados (tipo “data”) do Canal de Interatividade.

A principal vantagem do mecanismo de comunicação baseado em eventos é o baixo grau de acoplamento entre as entidades de uma aplicação, o que aumenta a coesão e facilita o reuso de funcionalidades.

Comunicação com o Canal de Interatividade

A comunicação com o Canal de Interatividade é realizada por meio de eventos da classe “tcp”. Assim como no uso de *sockets TCP*, a troca de dados só pode ser realizada após estabelecida uma conexão entre cliente e servidor. Para solicitar uma conexão, deve ser lançado um evento do tipo “*connect”*, informando o servidor (*host*) e a porta com a qual se deseja conectar. Como resposta a este evento, é gerado um evento do tipo “*connect”* contendo um identificador da conexão, ou um erro, indicando o motivo pelo qual a conexão não pôde ser estabelecida.

Após efetuada a conexão, dados podem ser enviados e recebidos por meio de eventos do tipo “*data”*. Ao publicar um evento de envio de dados, deve ser informado o identificador da conexão pela qual os dados serão enviados. O identificador de conexão também está presente em eventos de dados recebidos, informando a conexão a qual eles pertencem. Além disso, há ainda o tipo de evento “*disconnect”*, que sinaliza o fim da conexão.

Comunicação com o Documento NCL

A comunicação entre *scripts* NCLua e o documento NCL no qual estão inseridos é realizada por meio de âncoras de propriedade (representadas na Figura 10), que podem ser descritas como atributos compartilhados entre os ambientes NCL e NCLua.

```
<media id="meu_id" src="arquivo.lua"/>
   <property name="ancora_de_propriedade_a"/>
   <property name="ancora_de_propriedade_b"/>
</media>
```

Figura 10. Elemento media com duas âncoras de propriedade.

Para efetuar mudanças dos valores de âncoras de propriedade de NCL, um *script* NCLua deve lançar eventos da classe "*ncl"* do tipo "*attribution"* informando a propriedade e o valor a ser atribuído. A atribuição é feita em dois passos: inicialmente deve ser lançado um evento notificando o inicio da atribuição; caso a atribuição seja aceita, um evento de fim de atribuição deve ser lançado.

Um documento NCL também pode alterar o valor de uma âncora de propriedade através da ação "*set"* em um conector (conforme descrito na Seção 6.2.3). Esta ação gera um evento de início de atribuição à parte interessada, que deve ser respondido com um evento de fim de atribuição caso esta propriedade possa ser alterada.

2.4 DESENVOLVIMENTO ORIENTADO A MODELOS

A abordagem de Desenvolvimento Orientada a Modelos – Moden Driven Development (MDD) (ALMEIDA, PIRES e VAN SINDEREN, 2004) – permite estruturar as aplicações em um nível alto de abstração e, por meio de transformações entre modelos, obter um modelo de aplicação especifíco de plataforma. Essas transformações são definidas no nível de metamodelo, com a descrição das correspondências entre os elementos dos metamodelos envolvidos.

2.4.1 Modelo e metamodelo

O metamodelo é um modelo que especifica a sintaxe abstrata de um modelo. Por exemplo, o metamodelo de UML descreve os elementos que compõe esta linguagem de modelagem; um diagrama da UML é uma instância do metamodelo de UML. A Figura 11 ilustra os conceitos de modelo e metamodelo para a linguagem UML.

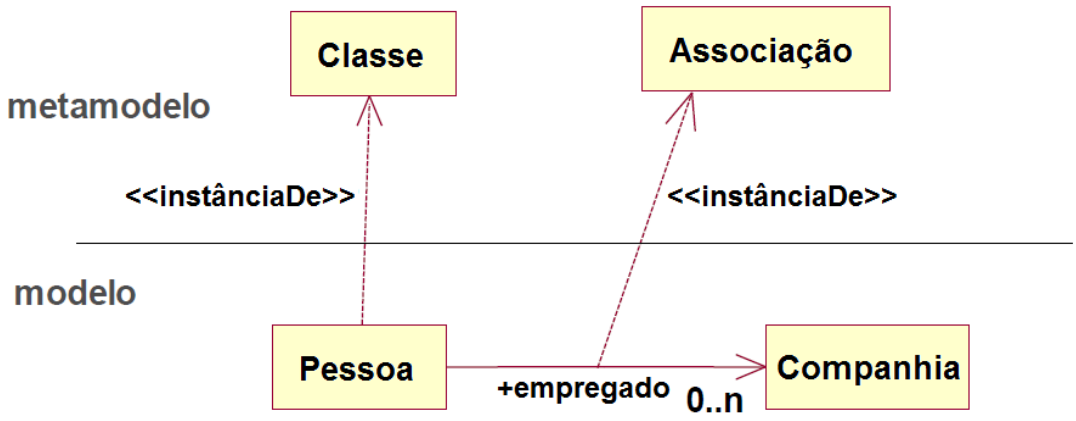


Figura 11. Modelo e Metamodelo.

Como mostrado na Figura 11, no nível de metamodelagem têm-se elementos da linguagem UML, as Classes e as Associações. A partir desses elementos, podem ser criados modelos UML. No caso da Figura 11, o modelo criado é composto de duas instâncias do elemento Classe, denominados de Pessoa e Companhia, e uma instância do elemento Associação, denominado de “empregado”.

2.4.2 Abordagem MDD

Na abordagem MDD, as aplicações são estruturadas com base em um Modelo Independente de Plataforma (*Platform-Independent Models* – PIM). Um PIM permite descrever a aplicação em nível alto de abstração, sem levar em consideração especifidades/particularidas da plataforma, como a tecnologia e a linguagem específica que é utilizada por esta plataforma. Assim, um mesmo PIM pode ser utilizado para implantar a aplicação em diferentes plataformas. Para isso, este modelo independente pode ser transformado para um Modelo de Plataforma Específica (*Platform-Specifc Model* - PSM), que considera especifidades/particularidas da plataforma, como a linguagem e/ou teconologia utilizada pela mesma. Este PSM pode ser transformado para um código de implementação, possivelmente usando transformações automatizadas (DOCKHORN COSTA, 2007).

Como exemplo de MDD, (DOCKHORN COSTA, 2007) utiliza essa abordagem para tranformar regras ECA em regras JESS (SANDIA LABS, 2008), uma tecnologia baseada em regras altamente integrada com Java. A Figura 12 ilustra esse processo.

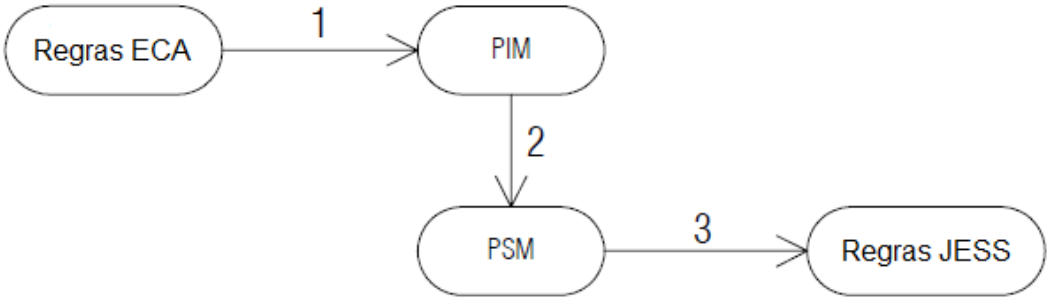


Figura 12. Visão geral da abordagem MDD em (DOCKHORN COSTA, 2007).

Na Figura 12, o comportamento de uma aplicação sensível ao contexto é definido por regras textuais ECA, que fornece um nível de abstração maior, permitindo que o desenvolvedor foque em aspectos específicos da aplicação sem se preocupar com particularidades da plataforma. A partir dessa linguagem é gerado seu modelo PIM (Etapa 1), que é uma instância do metamodelo de ECA. Na Etapa 2, o PIM é transformado no PSM, que é uma instância do metamodelo da linguagem de regras JESS. Finalmente, na Etapa 3, a partir do PSM é gerado o código textual na linguagem JESS.

Durante a Etapa 2, o desenvolvedor da aplicação deriva modelos de menor dependência da plataforma para um de maior dependência da plataforma. A fim de aumentar a eficiência do processo de desenvolvimento da aplicação, o desenvolvedor pode usar transformações automáticas para unir os diferentes modelos. Uma transformação totalmente automática requer que sejam especificadas regras de transformação para todos os possíveis modelos fonte para um modelo alvo apropriado. A Figura 13 mostra em detalhe essa etapa de transformação.

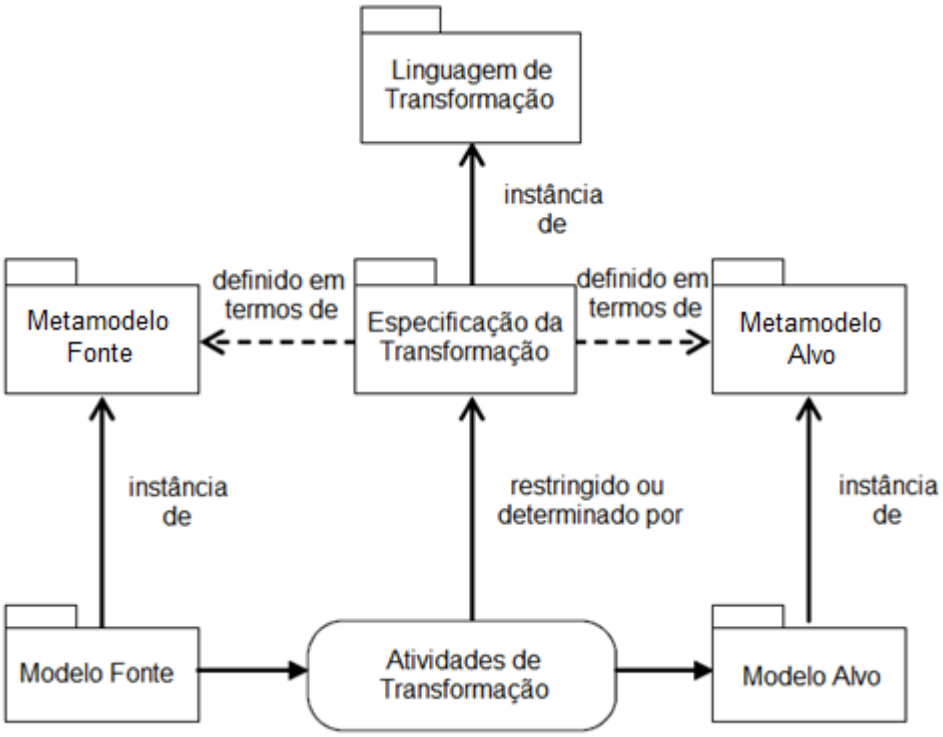


Figura 13. Visão geral da etapa de transformação (ALMEIDA, 2006).

Conforme pode ser visto na Figura 13, o Modelo Fonte, assim como o Modelo Alvo, são instâncias do metamodelo de um Metamodelo Fonte e de um Metamodelo Alvo, respectivamente. As Atividades de Transformação, ou Regras de Transformação, são restringidas pela Especificação da Transformação que, por sua vez, é definida com base nos elementos presentes nos Metamodelos Fonte e Alvo, e codificada em uma linguagem, denominada Linguagem de Transformação.

2.5 CONSIDERAÇÕES DO CAPÍTULO

Neste capítulo foi feito um resumo dos principais conceitos envolvidos neste trabalho. Foram apresentados os conceitos de contexto e aplicações seníveis ao contexto, e discutiu-se sobre o desenvolvimento desse tipo de aplicação, em que a especificação do compotamente reativo é uma etapa importante. Em (DOCKHORN COSTA, 2007), é proposta uma linguagem de regras específica de domínio, a ECA-DL, responsável por descrever o comportamento reativo de aplicações sensíveis ao contexto. A linguagem ECA-DL serviu de base para a ECA-DL TVD, proposta por este trabalho, que tem como objetivo

especificar as regras que descrevem o comportamento de aplicações sensíveis ao contexto no ambiente de TVD. ECA-DL TVD é definida no Capítulo 5.

Além disso, foram apresentados padrões arquiteturais para desenvolvimento de aplicações sensíveis ao contexto. Esses padrões arquiteturais permitem identificar os componentes de manipulação de contexto, situações e regras ECA-DL TVD que podem ser generalizados e incluídos como parte de uma *plataforma de gerenciamento de contexto*, que visa oferecer componentes e serviços genéricos para suportar o desenvolvimento de aplicações sensíveis ao contexto, independente do domínio da aplicação. Mais especificamente, a partir desses padrões arquiteturais, este trabalho propõe uma arquitetura conceitual para o componente Gerenciador de Contexto do *middleware* Ginga, o que é discutido no Capítulo 4.

Também neste capítulo foram discutidos os conceitos do sistema de TV Digital brasileiro, especificamente o *middleware* Ginga e sua linguagem declarativa NCL. No Capítulo 6, é feita uma análise dos elementos dessa linguagem que podem ser utilizados para o desenvolvimento de aplicações sensíveis ao contexto e, com base nesses elementos, o metamodelo de NCL é apresentado. Esse metamodelo de NCL, assim como o metamodelo de ECA-DL TVD (Capítulo 5), são importantes na abordagem MDD (Seção 2.4) proposta neste trabalho para gerar código NCL a partir de regras ECA-DL TVD. Para isso, necessita-se de uma formalização conceitual das correspondências entre elementos dos metamodelos de ECA-DL TVD e NCL, o que é feito no Capítulo 7.

3 TRABALHOS RELACIONADOS

Este capítulo discute propostas e abordagens relacionadas a este trabalho, à luz dos objetivos definidos no Capítulo 1. São apresentados trabalhos na área de TVD que exploram modelagem contextual, propostas de arquiteturas para manipulação de informações contextuais, além de trabalhos que aplicam técnicas de MDD em sistemas baseados em regras.

Este capítulo está estruturado como segue. Na Seção 3.1, discutem-se trabalhos que exploram modelagem contextual em TVD. Na Seção 3.2, são apresentados trabalhos na área de arquiteturas para manipulação de contexto em TVD. Finalmente, a Seção 3.3 analisa trabalhos da área de MDD.

3.1 MODELAGEM CONTEXTUAL

Poucos trabalhos discutem uma modelagem para ambientes de TV digital. Alguns exemplos de trabalhos são (GOULARTE, PIMENTEL e MOREIRA, 2006) e (LEITE, LIMA, *et al.*, 2007), apresentados a seguir.

O escopo da discussão dos trabalhos consite em:

- *Uso de ontologias de fundamentação.* As ontologias de fundamentação proveem um conjunto de conceitos básicos para representação de conceitualizações que são fiéis à realidade. Modelos conceituais obtidos usando conceitos bem fundamentados visam caracterizar o mais preciso possível o domínio da aplicação;

- *Modelagem de situação*. Além de oferecer uma caracterização precisa do universo de discurso da aplicação, técnicas de modelagem contextual devem prover meios de representar estados particulares de interesse da aplicação, que são as siutações contextuais (Seção 2.1.3);

- *Definição de regras para descrição do comportamento reativo da aplicação*. Uma solução baseada em regras aparece naturalmente dada a natureza da detecção de uma situação, na qual o contexto do usuário é continuamente monitorado a fim de verificar se certas condições (especificação da situação) são satisfeitas.

- *Uso em técnicas de MDD*. Visando atender o Objetivo 3 (Seção 1.2), a realização da modelagem contextual, de forma automática, utilizando técnicas de MDD reduz reduz a separação entre os modelos e as aplicações realizadas.

3.1.1 Suporte a computação sensível ao contexto em documentos estruturados para TV interativa

Em (GOULARTE, PIMENTEL e MOREIRA, 2006), é proposto um *framework* que provê: "(a) uma representação para contexto; (b) os meios para definir entidades contextuais classificando-as de acordo com o framework de modelagem de contexto e (c) um conjunto extensível de elementos contextuais pré-definidos - biblioteca de contexto". Nesse trabalho, os modelos contextuais podem ser representados, integrados e instanciados em uma abordagem hierárquica, em que uma situação de uma entidade é caracterizada por (um conjunto de) informações contextuais que podem ser indexadas pelo contexto primário. A representação do contexto primário é dada por uma abstração do tipo *PrimaryContextType* que é composta pelas seguintes abstrações *IdentityType*, *LocationType*, *TimeType* e *ActivityType* – que são, respectivamente, as representações dos contextos primários identidade, localização, tempo e atividade propostos por (DEY e ABOWD, 2000) para uma categorização dos tipos de contexto (Figura 14). Embora não seja mencionado diretamente, essas classes abstratas podem ser vistas como uma espécie de ontologia.

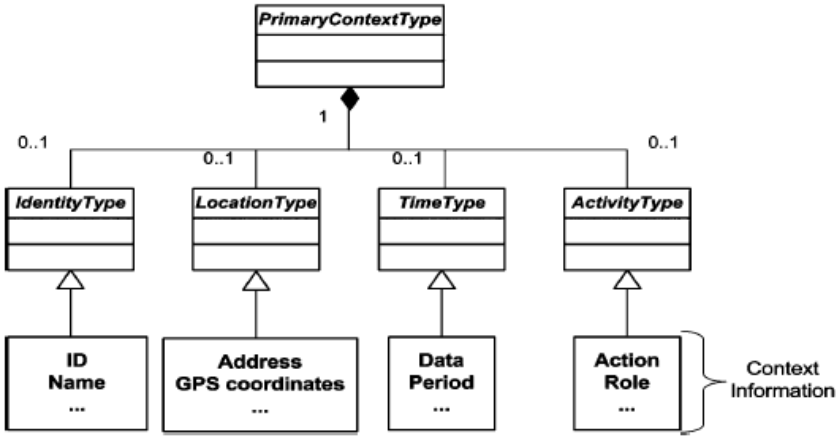


Figura 14. Informações de contexto primário (GOULARTE, PIMENTEL e MOREIRA, 2006).

Na Figura 14, verificam-se alguns tipos de contexto primário, como ID, Nome (*nome*), Endereço (*Address*), Coordenadas GPS (*GPS Coordinates*), Data (*Date*), Period (*Period*), Action (*Action*) e Role (*Papel*). Por exemplo, um usuário poderia ser caracterizado pelo ID e o seu contexto dado por sua localização em coordenadas GPS e ação que ele está executando.

A abordagem proposta por este trabalho, ao contrário, não exige que o desenvolvedor classifique contexto de acordo com uma enumeração de quatro tipos de contexto como proposto por (GOULARTE, PIMENTEL e MOREIRA, 2006). Ao invés, a abordagem deste trabalho permite que o desenvolvedor classifique contexto de acordo com sua natureza, sem enumerar tipos de contexto específicos, baseando-se em ontologias de fundamentação (GUIZZARDI, 2005). O uso de ontologias de fundamentação ajuda a caracterizar mais precisamente o universo de discurso da aplicação.

Também são propostos, como parte da abordagem de modelagem, mecanismos para modelar comportamento reativo das aplicações, através de uma linguagem baseada em regras consistente com o padrão arquitetural *Event-Control-Action* (Seção 2.2.4).. Uma linguagem baseada em regras é uma solução natural dada a característica das aplicações sensíveis ao contexto que devem reagir as mudanças de contexto e invocar ações.

3.1.2 Arquitetura de serviço para avaliação de contextos em redes de TV digital

Uma proposta de ontologia de domínio para TV digital aparece em (LEITE, LIMA, *et al.*, 2007). Nesse trabalho, as classes de ontologia são classificadas em três grupos principais: dos conceitos relacionados aos usuários (os espectadores), relacionados à plataforma e relacionados ao ambiente. Os conceitos relacionados a usuários podem ser usados para identificar quem são os espectadores que estão atualmente utilizando o Dispositivo para Recepção de TV Digital – DRTVD e quais são suas preferências. Os conceitos relacionados com a plataforma podem ser usados para identificar qual o estado atual do DRTVD. Os conceitos relacionados ao ambiente podem ser usados para caracterizar o local em que está inserido o DRTVD, por exemplo, para determinar se está em movimento ou parado.

Na Figura 15, pode ser vista uma pequena parte desses conceitos. Nessa figura, o conceito *User* interage com (*interactWith*) o conceito *Platform*, que por sua vez é dividido em outros conceitos, dentre eles: *AnalogicTelevision* e *DigitalTelevision*. Segundo (LEITE, LIMA, *et al.*, 2007), no escopo dos sistemas de TV digital, o conceito de *AnalogicTelevision* deve estar conectado a um *SetTopBox*, isso porque uma TV analógica não é compatível com o sinal digital. *DigitalTelevision* e *SetTopBox* são conceitos similares por possuírem um grande número de propriedades em comum.

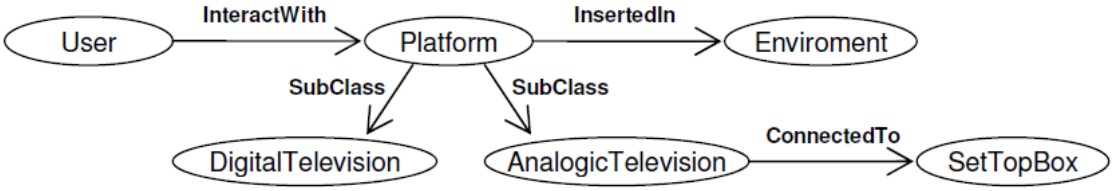


Figura 15. Alguns conceitos presentes na ontologia de domínio para TV digital proposta por (LEITE, LIMA, *et al.*, 2007).

A ontologia definida por (LEITE, LIMA, *et al.*, 2007) é restrita ao ambiente de TVD, enquanto que, por outro lado, este trabalho utiliza os conceitos prospostos por (GUIZZARDI, 2005) e estendidos por (DOCKHORN COSTA, 2007), que permite a caracterização de qualquer cenário de aplicação. Além disso, (LEITE, LIMA, *et al.*, 2007) a linguagem para descrever o comportamento reativo da aplicação é do tipo “Se <condição> então <artefato adequado, ou não>”, ou seja, utilizada apenas para classificar se artefatos televisivos são válidos de serem exibidos. Neste trabalho, por outro lado, é proposta uma linguagem específica de domínio, com um alto nível de abstração, permitindo uma maior praticidade para especificação do comportamento reativo da aplicação em diversos cenários.

3.1.3 Discussão

Conforme mencionado no Capítulo 1, um dos grandes desafios para a concepção de aplicações sensíveis ao contexto no ambiente de TVD é a dificuldade de se modelar adequadamente o contexto, haja vista a sua natureza complexa, heterogênea e distribuída (DOCKHORN COSTA, 2007). A modelagem contextual deve definir conceitos que sirvam de fundamentação para especificação de aplicações sensíveis ao contexto em diversos domínios. Os trabalhos apresentados (GOULARTE, PIMENTEL e MOREIRA, 2006) (LEITE, LIMA, *et al.*, 2007) propõem abstrações para modelar o domínio da aplicação no ambiente de TVD. Contudo, considerando o Objetivo 1 (Seção 1.2), nenhuma das propostas analisadas utiliza uma metodologia sistemática para a representação do conceito de situação como em (DOCKHORN COSTA, 2007). A modelagem contextual proposta por (DOCKHORN COSTA, 2007) é baseada nas teorias de modelagem conceitual e apoiadas pelos desenvolvimentos em ontologias de fundamentação (GUIZZARDI, 2005). Assim, este trabalho toma como ponto de partida a abordagem proposta por (DOCKHORN COSTA, 2007), e a aplica ao ambiente de TVD.

Outro aspecto que deve ser definido na modelagem contextual é o comportamento reativo das aplicações sensíveis ao contexto. Por exemplo, (LEITE, LIMA, *et al.*, 2007) utiliza uma linguagem baseada em regras, mas com poder de expressividade limitado, permitindo avaliar apenas o grau de adequação de um artefato televisivo. Já (DOCKHORN COSTA,

2007) vai além, e define uma linguagem específica de domínio, baseada em regras, denominada ECA-DL (DOCKHORN COSTA, 2007). Por ser específica de domínio, torna-se mais prática para desenvolvedores de aplicações sensíveis ao contexto do que outras linguagens de propósitos gerais. Uma regra ECA-DL é consistente com o padrão arquitetural *Event-Control-Action* – ECA – (Seção 2.2.4) que provê uma estrutura para aplicações que pró-ativamente reagem às mudanças de contexto. Este trabalho, porém, propõe a ECA-DL TVD para a especificação do comportamento reativo da aplicação. ECA-DL TVD é uma especialização de ECA-DL de forma a atender os requisitos inerentes à plataforma Ginga.

Finalmente, tanto (GOULARTE, PIMENTEL e MOREIRA, 2006), quanto (LEITE, LIMA, *et al.*, 2007) não mencionam realização das aplicações à partir da modelagem contextual utilizando técnicas de MDD.

Com base na discussão realizada a Tabela 1 faz uma comparação dos trabalhos, em que X representa que o trabalho não atende o escopo desejado, enquanto que OK representa que o trabalho atende completamente e OK- atende parcialmente.

Tabela 1. Comparação dos trabalhos de modelagem contextual.

Trabalho	Ontologias de Fundamentação	Modelagem de Situações	Uso de Regras	Uso de Técnicas de MDD
(GOULARTE, PIMENTEL e MOREIRA, 2006)	OK-	X	X	X
(LEITE, LIMA, *et al.*, 2007)	OK-	X	OK-	X

Conforme mostrado na Tabela 1, a maioria dos itens do escopo de discussão não é atendida pelos trabalhos analisados. (GOULARTE, PIMENTEL e MOREIRA, 2006) atende parcialmente ao escopo ontologias de fundamentação pois as classes abstratas para representações dos contextos primários identidade, localização, tempo e atividade podem ser vistas como uma espécie de ontologia. Já (LEITE, LIMA, *et al.*, 2007) atende parcialmente ao escopo de ontologias de fundamentação por propor conceitos restritos ao ambiente de TVD, e atende parcialmente ao escopo uso de regras, por utilizar uma linguagem baseada em regras que é limitada e não possui um alto nível de abstração, a fim de permitir uma maior praticidade ao desenvolvedor da aplicação.

3.2 ARQUITETURAS PARA MANIPULAÇÃO DE INFORMAÇÃO CONTEXTUAL

Nesta seção são apresentadas algumas arquiteturas que manipulam informação contextual no ambiente de TV digital. O escopo de discussão consiste em:

- *Flexibilidade e Extensibilidade.* A arquitetura deve fornecer suporte a componentes flexíveis, que são serviços ou componentes genéricos que podem adquirir, como entrada, comportamentos ou procedimentos específicos da aplicação, a fim de acionar mecanismos de inferência de contextos e situações específicos, ou controlar ações, de acordo com uma aplicação específica;

- *Inferência de Contexto e Situações.* A arquitetura deve ser capaz de preencher uma lacuna entre a captura de contexto pelos sensores e as informações contextuais que são de interesse das aplicações;

- *Não realização de mudanças no middleware Ginga.* A arquitetura deve evitar realizar modificações no Ginga, propondo soluções que utilizem os próprios recursos do *middleware*.

3.2.1 Uma arquitetura para suporte ao desenvolvimento de aplicações sensíveis a contexto em cenário de convergência

Em (NETO e FERRAZ, 2006), é apresentada uma arquitetura denominada de ContexTV, que tem o propósito de atender a dispositivos telefônicos móveis. Essa arquitetura é dividida em duas partes, a parte do cliente, que se encontra no dispositivo telefônico do usuário, e a parte do servidor, que se encontra na central da operadora telefônica. O cliente somente receberá instruções e capturará informações, já o servidor será responsável por processar as informações.

Este tipo de arquitetura apresenta algumas vantagens e desvantagens:

- Menor processamento no cliente: Não é exigida uma grande capacidade de processamento do dispositivo do usuário, possibilitando que um número maior de pessoas tenha acesso a essa tecnologia.

- Utilização de outros contextos: Troca de informações de contexto entre os usuários, na qual as informações de programação não ficam restritas somente ao seu contexto, incrementando assim a oferta de sugestões de programas, por exemplo.

- Maior congestionamento da rede: Pode haver congestionamento na rede de comunicação entre o cliente e o servidor, devido à grande quantidade de tráfego. Não ficando este tráfego restrito somente a informações da aplicação, pois, esta solução utiliza a mesma rede de telefonia móvel.

- Privacidade: O trabalho não deixa claro qual o conteúdo das informações enviadas para o servidor, podendo haver problemas quanto à privacidade do usuário.

(NETO e FERRAZ, 2006) é específico para plataformas móveis, e utiliza um componente remoto (chamado Servidor ContexTV) para o processamento de informações contextuais, devido a possíveis limitações de processamento nos dispositivos. Não é especificado o mecanismo de processamento de contexto, assim como não são utilizados os própios recursos do Ginga para realização da arquitetura.

Neste trabalho, ao contrário, é proposta uma arquitetura com componentes e serviços genéricos para o Gerenciador de contexto do Ginga, além de realizá-los utilizando os próprios recursos da plataforma Ginga. Os serviços genéricos favorem o reuso e permitem a utilização da arquitetura em diversos cenários de aplicação, enquanto que a não modificação da plataforma Ginga torna menos dipendioso a implementação da arquitetura.

3.2.2 PersonalTVware: Uma proposta de arquitetura para apoiar a recomendação sensível ao contexto de programas de TV

Em (SILVA, ALVES e BRESSAN, 2009) é proposto que programas de TV sejam sugeridos ao telespectador com base nas informações sobre o programa, enviadas pela emissora através do Guia Eletrônico de Programa, e nas informações coletadas do usuário. Para isso, (SILVA, ALVES e BRESSAN, 2009) apresenta um conjunto de cinco perguntas sobre o contexto do usuário, as quais são fundamentais para a escolha dos programas a serem sugeridos. São elas: **Quem,** indica quem é o usuário que está assistindo TV naquele momento; **Onde**, indica a localização do usuário, se está em casa ou no trabalho; **Como**, indica por meio de qual dispositivo o usuário está assistindo televisão, aparelho fixo ou móvel; **Quando**, indica o momento do dia em que o usuário assiste a um determinado gênero de programa; e **Qual**, indica o conteúdo de interesse do usuário.

Essas informações contextuais servem de referência para a construção de estruturas de metadados em XML *Schemas*. Seguindo a arquitetura da PersonalTVware, o XML com

as informações sobre o usuário e o com as informações sobre os programas serão utilizados como dados de entrada para um componente Interpretador de Contexto.

No Interpretador de Contexto é onde se encontra a máquina de regras do sistema PersonalTVware. A máquina de inferência é baseada na utilização de um conjunto de regras, pois assim, segundo (SILVA, ALVES e BRESSAN, 2009), permite definir de forma flexível estruturas condicionais que refletem as relações entre as dimensões contextuais. Contudo, não define qual é a linguagem utilizada e nem como é a sua realização na plataforma Ginga e, portanto, não é possível analisar a viabilidade prática da proposta.

Este trabalho, ao contrário, propõe uma arquitetura conceitual e uma linguagem baseada em regras de alto nível de abstração e demonstra sua realização utilizando os próprios recursos da plataforma Ginga.

3.2.3 Discussão

Focando no Objetivo 2 (Seção 1.2), que diz respeito à definição de elementos arquiteturais para manipulação de contexto em TVD, (SILVA, ALVES e BRESSAN, 2009) define elementos voltados para suas aplicações específicas, ou seja, não genéricos. Este trabalho, ao contrário, oferece suporte genérico através de componentes e serviços genéricos para o Gerenciador de Contexto Ginga permitindo flexibilidade no desenvolvimento de diferentes aplicações em diversos cenários. (NETO e FERRAZ, 2006) propõe um *middleware* extensível para tratamento de contexto. Este trabalho, ao contrário, utiliza os próprios recursos da plataforma Ginga na realização dos componentes da arquitetura conceitual.

Com base na discussão realizada a Tabela 2 faz uma comparação dos trabalhos, em que X representa que o trabalho não atende o escopo desejado, enquanto que OK representa que o trabalho atende completamente e OK- atende parcialmente.

Tabela 2. Comparação dos trabalhos de arquitetura para manipulação de informação contextual.

Trabalho	Flexibilidade e Extensiblidade	Inferência de Contexto e Situações	Não realização de mudanças no middleware Ginga
(NETO e FERRAZ, 2006)	OK	OK-	X

(SILVA, ALVES e BRESSAN, 2009)	X	OK-	X

A Tabela 2 mostra que apenas (NETO e FERRAZ, 2006) oferece uma solução flexível e extensível para diferentes cenários de aplicação. Também mostra que ambos não propõem soluções que utilizem os próprios recursos da plataforma Ginga. Além disso, não atendem completamente o escopo mecanismo de inferência de contexto e situações porque as informações contextuais não estão disponíveis para as aplicações em diferentes fases do processamento de informação. As informações obtidas são contextos de mais baixo nível. Não existe, por exemplo, uma fonte de situação que detecta e informa à aplicação a ocorrência desta.

Além desses trabalhos, focando especificamente na plataforma Ginga, (BRACKMANN, VENECIAN, *et al.*, 2009) propõe um conjunto de APIs para captura e processamento de informações contextuais através de extensões da arquitetura do Ginga. Apesar de apresentar idéias interessantes no que diz respetio a flexibilidade e extensibilidade da plataforma, o trabalho não descreve protótipos ou aplicações que demonstem a viabilidade de tais ideias.

Assim, verifica-se que a plataforma Ginga carece da definição de elementos arquiteturais genéricos para manipulação de contexto. Além disso, necessita-se que esses elementos estejam alinhados com uma metodologia de modelagem contextual. A arquitetura proposta por este trabalho baseia-se nos padrões arquiteturais apresentados na Seção 2.2.4 e permite identificar os componentes de manipulação de contexto, situações e regras ECA-DL TVD que podem ser generalizados e incluídos como parte de uma plataforma de gerenciamento de contexto. Além disso, não propõe o uso de novo *middleware*, e sim, define uma estrutura conceitual para o Gerenciador de Contexto do próprio Ginga.

3.3 DESENVOLVIMENTO ORIENTADO A MODELOS

Nesta seção são apresentados alguns trabalhos que utilizam o processo de desenvolvimento orientado a modelos no cenário de aplicações sensíveis ao contexto. O escopo da discussão consiste em:

- *A linguagem utilizada no nível independente de plataforma é baseada em regras*. A linguagem fonte deve possuir um alto nível de abstração e deve ser baseada em regras, dado a natureza das aplicações sensíveis ao contexto.

- *A linguagem resultante da abordagem MDD é uma linguagem declarativa*. Como alvo da transformação é a plataforma Ginga e sua linguagem NCL, a linguagem alvo deve ser uma linguagem declarativa.

- *Abordagem MDD aplicada ao ambiente de TVD*. O trabalho deve utilizar a tranformação MDD para o desenvolvimento de aplicações sensíveis ao contexto no ambiente de TVD.

3.3.1 Composição de serviços sensíveis ao contexto usando políticas e modelos

Em (YANG, OU, *et al.*, 2005), é proposta uma metodologia para desenvolvimento, execução e manutenção de serviços sensíveis ao contexto. De acordo com (YANG, OU, *et al.*, 2005), essa metodologia é baseada em técnicas de MDD e uma visão do processo é mostrada na Figura 16.

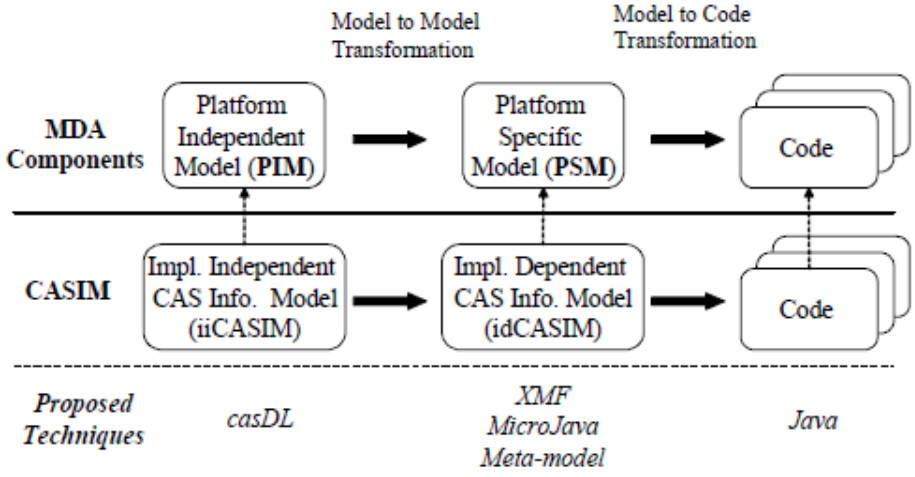


Figura 16. Desenvolvimento MDD proposto por (YANG, OU, *et al.*, 2005).

Conforme mostra a Figura 16, o processo parte de um Modelo Independente de Plataforma (PIM), utlizando uma linguagem denominada de linguagem de descrição de serviços baseadas em políticas (*policy-based context-aware service description language* – casDL), e tem como objetivo gerar código Java. A linguagem casDL é utilizada para descrever um serviço sensível ao contexto e dessa descrição é gerada o PIM. Além disso, utiliza o *XMF toolkit* de *Xactium Ltd*. (XACTIUM ON DEMAND SOLUTIONS, 2011) para

construir os modelos. XMF é um ambiente de meta-programação genérica que visa suportar uma ampla variedade de cenários de desenvolvimento MDD, e que já define um meta-modelo MicroJava como PSM e sua transformação para código Java correspondente. As tranformações do PIM para o PSM também são feitas por XMF.

Apesar de casDL ser baseada em regras, o alvo da transformação é gerar código Java, o que torna a proposta inadequada considerando a plataforma Ginga. Conforme mencionado na Seção 1.1, Java não possui suporte obrigatório nos dispositivos móveis que implementem essa plataforma. Este trabalho, por outro lado, tem objetivo de gerar código NCL, que é uma linguagem declarativa e que possui um comportamento diferenciado em relação às linguagens orientadas a objeto como Java, além de possuir suporte obrigatório em todas as versões do Ginga.

3.3.2 Direcionamentos para o desenvolvimento orientado a modelos de aplicações pervasivas sensíveis ao contexto

Em (SERRAL, VALDERAS e PELECHANO, 2010) também é proposto uma metodologia MDD para o desenvolvimento de sistemas sensíveis ao contexto resultando em código Java. (SERRAL, VALDERAS e PELECHANO, 2010) define uma linguagem de modelagem de sistemas pervasivos específica de domínio (PervML), que é independente de plataforma. Nesta linguagem, os comportamentos reativos das aplicações são definidos usando os diagramas de seqüência UML, que são independentes de plataforma. Condições para invocação das ações são especificadas utilizando OCL.

Além disso, (SERRAL, VALDERAS e PELECHANO, 2010) define a transformação para o código Java, com métodos "para verificar a condição para invocação e executar as ações que são especificadas no diagrama de sequência" (SERRAL, VALDERAS e PELECHANO, 2010).

A utilização de diagramas UML e condições OCL proposta por (SERRAL, VALDERAS e PELECHANO, 2010) para descrever o comportamento reativo da aplicação sensível ao contexto não é um vocabulário comum ao domínio desse tipo de aplicação. Uma solução baseadas em regras é da natureza das aplicações sensíveis ao contexto, uma vez que o contexto do usuário é continuamente monitorado a fim de verificar se certas condições são satisfeitas e, nesse caso, invocar os serviços/ações devidos. Assim, este trabalho, por outro lado, utiliza uma linguagem baseada em regras para descrever as aplicações sensíveis ao

contexto. Além disso, tem como objetivo gerar código NCL, que é uma linguagem declarativa e cujo suporte é obrigatório a todas as implementações da plataforma Ginga.

3.3.3 Discussão

Focando no Objetivo 3 (Seção 1.2), que diz respeito à realização de elementos arquiteturais na plataforma Ginga, verifica-se que (YANG, OU, *et al.*, 2005) e (SERRAL, VALDERAS e PELECHANO, 2010) possuem como objetivo gerar código Java. Por outro lado, este trabalho visa gerar código NCL, que é uma linguagem declarativa utilizada pelo Ginga.

Ao contrário deste trabalho, (SERRAL, VALDERAS e PELECHANO, 2010) utiliza uma linguagem de propósito geral e, portanto, não oferece diretamente suporte a eventos de contexto, decrição de situações e não suporta o padrão arquitetural *Event-Condition-Control* (ECA) (DOCKHORN COSTA, FERREIRA PIRES e SINDEREN, 2005).

Enfim, este trabalho utiliza uma metodologia MDD inovadora e desafiadora, que gera código NCL à partir de uma linguagem baseada em regras. Além disso, a abordagem é aplicada ao ambiente de TVD, o que não é verificado nos trabalhos analisados. A Tabela 3 resume a discussão.

Tabela 3. Comparação dos trabalhos de desenvolvimento orientado a modelos.

Trabalho	Linguagem fonte baseada em regras	Linguagem alvo declarativa	Aplicação ao ambiente de TVD
(YANG, OU, *et al.*, 2005)	OK	X	X
(SERRAL, VALDERAS e PELECHANO, 2010)	X	X	X

A Tabela 3 mostra que nenhum dos trabalhos tem como objetivo gerar código em linguagem declarativa e também não são aplicados ao ambiente de TVD.. Quanto à linguagem fonte, apenas (YANG, OU, *et al.*, 2005) utiliza uma linguagem baseada em regras, enquanto que (SERRAL, VALDERAS e PELECHANO, 2010) utiliza uma linguagem de propósito geral.

4 ARQUITETURA CONCEITUAL[2]

Conforme discutido na Seção 1, atualmente o Ginga carece de serviços genéricos que deem suporte ao desenvolvimento de aplicações sensíveis ao contexto. Estas aplicações seriam amplamente beneficiadas se pudessem compartilhar mecanismos para capturar informações de contexto de uso comum, ou, ainda, reutilizar funções comumente usadas por meios de serviços genéricos oferecidos por componentes da arquitetura (DOCKHORN COSTA, 2007).

O componente do núcleo comum do Ginga responsável por prover tais mecanismos e funções é o *Gerenciador de Contexto*. Contudo, por ser um componente opcional (SOARES e CASTRO, 2008) o Gerenciador de Contexto ainda não foi significantemente explorado nas implementações de referência do Ginga e nem em propostas de arquiteturas conceituais para este componente relatadas na literatura. Em (CRUZ, MORENO e SOARES, 2008) é realizada uma implementação do Ginga para dispositivos portáteis, na qual o Gerenciador de Contexto implementado apenas armazena as informações do perfil do usuário.

Este trabalho apresenta uma proposta de arquitetura conceitual para o componente Gerenciador de Contexto do *middleware* Ginga. O objetivo é prover serviços que suportem aplicações sensíveis ao contexto independentes do domínio da aplicação. Esses serviços podem ser combinados e configurados de forma a atender os requisitos específicos de cada aplicação.

Este capítulo está estruturado como segue. Na Seção 4.1 são inicialmente discutidos os princípios e os requisitos que serviram de base para a definição da arquitetura. A Seção 4.2 propõe a arquitetura para o Gerenciador de Contexto. Finalmente, a Seção 4.3 posiciona o presente trabalho com relação à contribuição para a realização dessa arquitetura conceitual.

[2] A arquitetura conceitual foi proposta no entregável Suporte Arquitetural (*Framework*) para Aplicações Sensíveis ao Contexto (Parte II – Projeto da 1ª versão do *framework*), documento do Projeto GingaFrEvo & GingaRAP - Sub-Projeto: Suporte à modelagem de contexto no ambiente de TV digital interativa.

4.1 PRINCÍPIOS E REQUISITOS

Conforme visto na Seção 3.2, a maioria dos trabalhos que tratam da manipulação de informação contextual, seja através da definição de modelos de contexto, seja pela definição de elementos arquiteturais não consideram uma abordagem formal e integrada de desenvolvimento de aplicações sensíveis contexto. Em particular, no ambiente de TV Digital existe o potencial de desenvolvimento de aplicações em diferentes domínios (LEITE, LIMA, *et al.*, 2007) (THAWANI, GOPALAN e SRIDHAR, 2004) (MOON, KIM, *et al.*, 2007). Neste sentido, uma abordagem de desenvolvimento que permita o reuso de funcionalidades de natureza genérica é desejável.

A arquitetura conceitual proposta está inserida numa abordagem de desenvolvimento centrada nos seguintes princípios:

- **Uso de modelos formais para representação de contextos, situações contextuais e regras para especificação de comportamentos reativos.** A modelagem formal de contexto, situações e regras deve permitir definir e especificar formalmente o domínio da aplicação, assim como o comportamento reativo das aplicações. A utilização de um modelo formal apresenta uma série de vantagens, pois possibilita: (i) a definição de modelos não ambíguos; (ii) a delimitação precisa do escopo dos estados contextuais possíveis da aplicação; e (iii) o entendimento entre os envolvidos no desenvolvimento de aplicações sensíveis ao contexto no ambiente de TVD;

- **Projeto orientado ao reuso.** A arquitetura conceitual para gerenciamento de contexto deve ser orientada para a concepção de elementos arquiteturais genéricos (padrões arquiteturais) que apoiem o desenvolvimento de aplicações de diferentes complexidades e em domínios variados, promovendo o reuso, a extensibilidade e a escalabilidade.

Além desses princípios, a arquitetura proposta deve ainda atender aos seguintes requisitos:

- *Flexibilidade e Extensibilidade.* A arquitetura deve fornecer suporte a componentes flexíveis, que são serviços ou componentes especiais que podem adquirir, como entrada, comportamentos ou procedimentos específicos da aplicação, a fim de acionar mecanismos de inferência de contextos e situações específicos, ou controlar ações, de acordo com uma aplicação específica. Além

disso, deve ser possível estender a funcionalidade da arquitetura sob demanda;

- *Inferência de Contexto e Situações.* A arquitetura deve ser capaz de interpretar (processar) contexto capturado por sensores de forma a inferir contexto de alto nível, que são de interesse das aplicações. Além disso, informações contextuais devem estar disponíveis para as aplicações em diferentes fases do processamento de informação;

- *Distribuição.* A fim de permitir o reuso, escalabilidade e confiabilidade da inferência de contexto, as fases de processamento de contexto mencionadas no item anterior podem estar distribuídas entre vários componentes, sendo possível adicioná-los em tempo de execução;

- *Mobilidade.* As aplicações sensíveis ao contexto, assim como as fontes de contexto, podem ser móveis e, portanto, a arquitetura deve prever a execução dos componentes em dispositivos móveis;

- *Rápido Desenvolvimento e Instalação de Aplicações.* O tempo de desenvolvimento e instalação com o suporte da arquitetura deve ser reduzido em relação ao tempo necessário para desenvolver e instalar aplicações sem o suporte da arquitetura.

Para a definição da arquitetura, foram utilizados os padrões arquiteturais apresentados na Seção 2.2.4. Inicialmente, como base no padrão ECA (Seção 2.2.4), são identificados três módulos principais: (i) *Fonte de Contexto*, que trata dos aspectos de captura e processamento de informações contextuais; (ii) *Monitor*, que exerce o *Controle* do comportamento reativo da aplicação; e (iii) *Executor de Ações*, que trata dos aspectos relacionados às Ações. Em seguida, aplicados os demais pradrões, estes módulos são refinados em componentes internos e externos até a definição final da plataforma de gerenciamento de contexto.

4.2 ARQUITETURA PROPOSTA PARA O GERENCIADOR DE CONTEXTO

A Figura 17 ilustra, ainda sem qualquer detalhamento, o componente *Gerenciador de Contexto* dando apoio a duas aplicações dentro do cenário do padrão de TVD brasileiro. O *Gerenciador de Contexto* oferece serviços genéricos para as aplicações sensíveis ao

contexto, as quais também implementam serviços específicos oferecidos por componentes específicos da aplicação. As formas ovais ilustram os pontos de interação entre usuários, seus contextos, aplicações e o gerenciador de contexto.

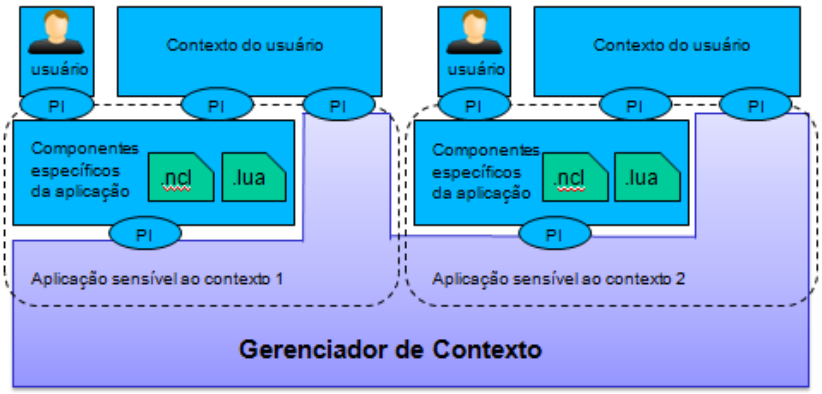


Figura 17. Gerenciador de Contexto dando apoio a duas aplicações sensíveis ao contexto.

Por exemplo, considere uma aplicação sensível ao contexto (Aplicação sensível ao contexto 1), na qual um filme deve ser pausado assim que a pipoca estiver pronta no forno microondas. Os componentes específicos da aplicação, que podem ser escritos com uma combinação de scripts NCL e NCLua, responsáveis por exibir o vídeo e o áudio, e pelo controle de sequência do filme na tela da TV, poderiam fazer uso dos serviços genéricos do gerenciador de contexto para: (i) obter informações sobre o status do microondas em relação ao preparo da pipoca; (ii) obter informações contextuais sobre a localização do usuário; e (iii) delegar o monitoramento do status do forno para que o gerenciador autonomamente pare o filme assim que detectar que a pipoca está pronta.

Outra aplicação sensível ao contexto (Aplicação sensível ao contexto 2) poderia utilizar as informações contextuais de que a pipoca está pronta e o filme está pausado para, por exemplo, entrar em modo de baixo consumo de energia até que o usuário volte e recomece o filme, ou, se ainda houver uma pessoa em frente à TV, propagandas poderiam ser exibidas.

4.2.1 Detalhamento da Arquitetura

Para que se possa melhor elaborar a arquitetura proposta – e também para fins de maior clareza do desenho – a Figura 17 será modificada de forma a ilustrar apenas as interações relevantes para o *Gerenciador de Contexto*, ou seja, suas interações com o

contexto do usuário e com os componentes específicos das aplicações. Essas modificações são ilustradas na Figura 18.

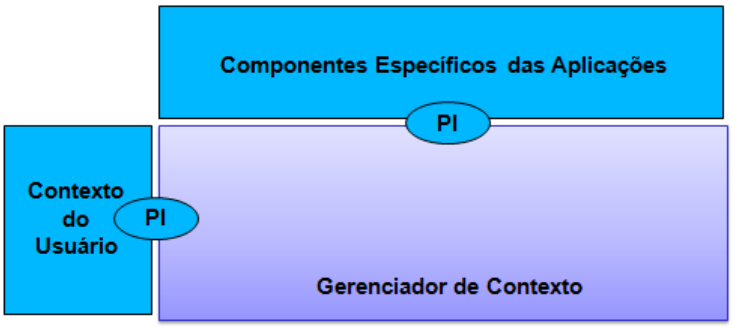


Figura 18. Interações relevantes para o Gerenciador de Contexto.

O padrão arquitetural ECA (Seção 2.2.4) pode ser utilizado para refinar a arquitetura do *Gerenciador de Contexto* de forma a incluir três subcomponentes: *Fonte de Contexto*, *Monitor* e Executor de Ações. A utilização desse padrão é representada na Figura 19.

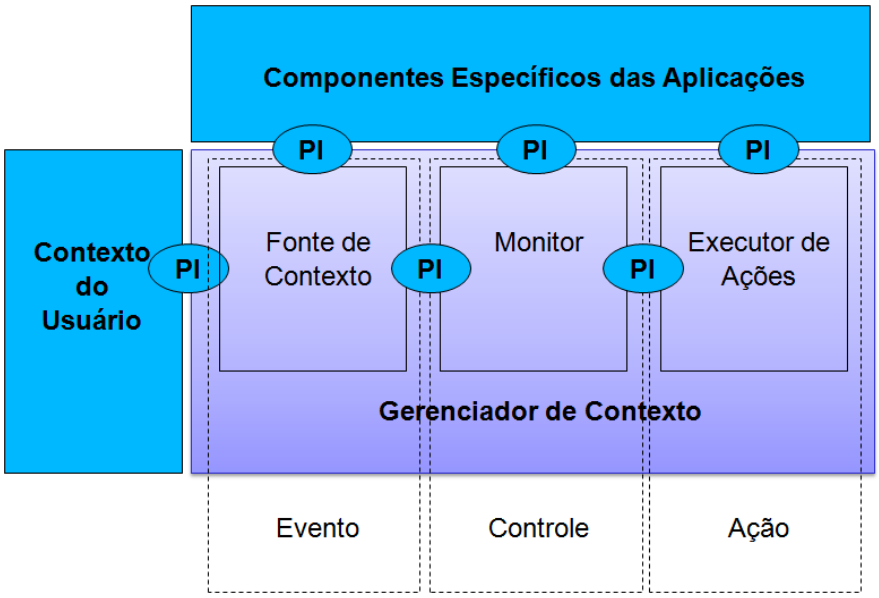


Figura 19. Padrão ECA aplicado ao Gerenciador de Contexto.

O componente Fonte de Contexto é responsável por interagir com o contexto do usuário de forma que, com base nas informações contextuais capturadas, ele gere eventos de contexto, os quais são observados pelo componente Monitor. A Fonte de Contexto também pode interagir com os componentes específicos da aplicação de forma a provê-la de informações sobre o contexto específico da aplicação.

O Monitor recebe instruções dos componentes específicos da aplicação sobre como deve ser o comportamento da aplicação sensível ao contexto, dadas situações de contexto específicas. Quando uma condição definida pelas instruções passadas pelos componentes

específicos é satisfeita, o Monitor aciona uma ou mais ações gerenciadas pelo Executor de Ações. Os componentes específicos da aplicação também podem acionar ações oferecidas pelo Componente de Ação diretamente.

Partindo dessa arquitetura ECA para a definição da arquitetura final, uma série de refinamentos sucessivos são feitos baseados nos outros padrões arquiteturais mencionados na Seção (Seção 2.2.4). O padrão arquitetural de ações (DOCKHORN COSTA, FERREIRA PIRES e SINDEREN, 2005), sugere a divisão do Componente de Ação em dois componentes distintos, denominados *Resolvedor de Ação* e *Provedor de Ação*. O componente Resolvedor de Ação realiza composição de ações, resolvendo conflitos de precendência, caso necessário. O Provedor de Ação é um provedor de serviços, tanto internos, quanto externos. Os componentes Provedores Internos de Ação são responsáveis por realizar ações com os recursos do próprio Ginga, como pausar um filme ou exibir propagandas, por exemplo, enquanto que os componentes Provedores Externos de Ação ativam serviços providos por outros componentes fora da *middleware*, como o envio de SMS ou controle de um eletrodoméstico.

Assim como os Provedores de Ações foram divididos entre Internos e Externos, o componente Fonte de Contexto também pode ser dividido dessa forma. O componente *Fonte de Contexto Interno* é responsável por prover serviços que vão desde a recuperação de informações de perfil de um determinado usuário, ou informações e status de um filme, até o encapsulamento de informações de localização do usuário, como, por exemplo, suas coordenadas geográficas. O componente *Fonte de Contexto Externo* pode ser utilizado para pré-processar informações de contexto e entregar seus resultados para a Fonte de Contexto Interno. Ele poderia, por exemplo, captar e utilizar as coordenadas geográficas do usuário e repassar para a Fonte de Contexto Interno somente o cômodo da casa em que ele se encontra.

Seguindo o padrão arquitetural de Hierarquia de Fontes e Gerenciadores de Contexto (DOCKHORN COSTA, FERREIRA PIRES e SINDEREN, 2005), a Fonte de Contexto ainda pode ser redefinida nos componentes *Fontes de Contexto* e *Fontes de Situações*. Fontes de Contexto são sensores únicos, como um GPS ou medidor de temperatura. Fontes de Situação detectam se uma situação está ou não ocorrendo. Para isso, utiliza as informações capturadas pelas Fontes de Contexto. Por exemplo, para detectar se a situação TV Ligada passou a existir, é necessário que uma Fonte de Contexto esteja capturando o estado atual da TV. Dessa forma, esses componentes podem colaborar entre si de forma a oferecer ao Monitor e aos componentes específicos da aplicação o nível necessário de abstração das informações de contexto.

A Figura 20 ilustra a arquitetura depois de aplicados os refinamentos.

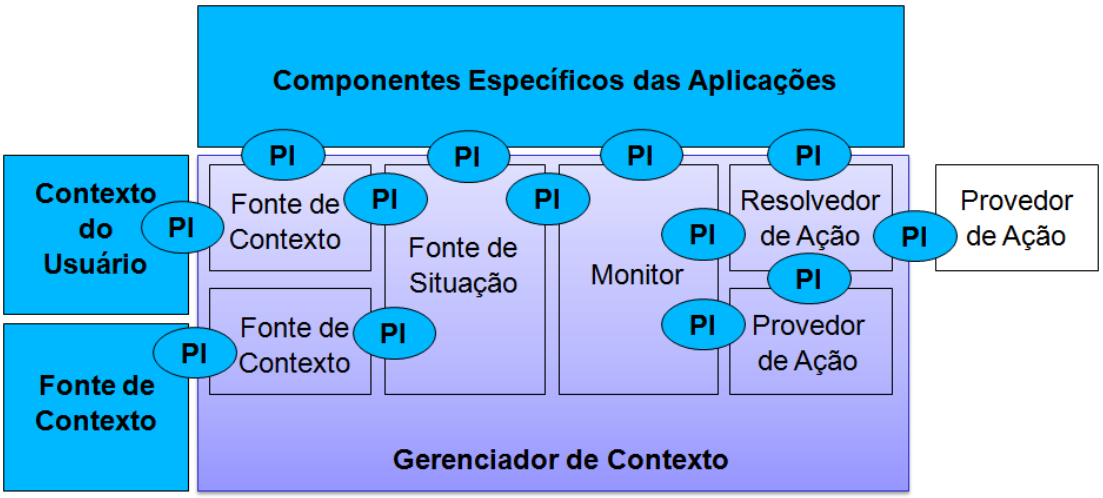


Figura 20. Gerenciador de Contexto após a aplicação dos padrões arquiteturais.

4.3 DISCUSSÃO

Este capítulo apresentou uma proposta de arquitetura conceitual para o Gerenciador de Contexto, componente do núcleo comum do *Ginga*. Esta arquitetura é consistente com padrões arquiteturais (DOCKHORN COSTA, FERREIRA PIRES e SINDEREN, 2005) e atua como uma plataforma de gerenciamento de contexto, fornecendo serviços genéricos através de componentes genéricos.

Além de definir uma arquitetura conceitual para o Gerenciador de Contexto, este trabalho também visa realizar elementos dessa arquitetura na plataforma Ginga. Mais especificamente, este trabalho propõe uma metodologia orientada a modelos para a realização do componente Monitor, que determina como deve ser o comportamento reativo da aplicação sensível ao contexto, ou seja, dada a ocorrência de determinadas situações contextuais, determinadas ações e/ou serviços devem ser invocados. Portanto, não está no escopo deste trabalho realizar de forma orientada a modelos os componentes Fontes de Contexto, de Situação, Resolvedor de Ação e Provedor de Ação, que devem ser especificados pelo desenvolvedor da aplicação. Também se pode identificar a necessidade de componentes que gerenciam cadastro/descadatro de componentes na plataforma, ou que tratam do controle de privacidade das informações. Contudo, tais componentes não fazem parte do escopo deste trabalho.

Na metodologia orientada a modelos proposta, o desenvolvedor de aplicações sensíveis ao contexto fornece modelos de contexto e situação, juntamente com comportamentos reativos das aplicações. A modelagem contextual utiliza as abstrações porpostas por (DOCKHORN COSTA, 2007), que seguem os conceitos de ontologias de fundamentação (GUIZZARDI, 2005), enquanto que a descrição do comportamento reativo é feita por regras ECA-DL TVD, uma linguagem específica de domínio proposta neste trabalho. A modelagem contextual e ECA-DL TVD são discutidos no Capítulo 5.

A partir dos modelos contextuais e as regras ECA-DL TVD, o Monitor é realizado em um documento NCL seguindo um processo de transformação orientado a modelos, conforme ilustrado na Figura 21. De acordo com a Figura 21, um Modelo Fonte, no caso as regras ECA-DL TVD (definidas com base nos modelos contextuais), irá gerar um Modelo Alvo, no caso, um modelo NCL, seguindo uma série de Atividades de Tranformação, que utilizam o *framework* EMF da plataforma Eclipse (THE ECLIPSE FOUNDATION, 2010). O EMF é um *framework* que facilita a modelagem e a geração de código para construção de ferramentas e outras aplicações baseadas em modelos de dados estruturados. O *framework* EMF permite a modelagem do domínio utilizando um metamodelo próprio, denominado *Ecore* (**.ecore*), e com base nesse metamodelo, é feita uma transformação para um metamodelo específico (**.genmodel*) para a geração de código.

Ainda conforme a Figura 21, o Modelo Fonte é uma instância do metamodelo da linguagem ECA-DL TVD, que é definido no Capítulo 5. O Modelo Alvo, por sua vez, é uma instância do metamodelo da linguagem NCL, que é definido Capítulo 6. Finalmente, as atividades de transformação são determinadas pela Especificação da Transformação, definida no Capítulo 7 e foi implementada utilizando Java. Finalmente, após gerar o modelo NCL, com base na sintaxe da linguagem, é obtido o código NCL, ou seja, o documento NCL.

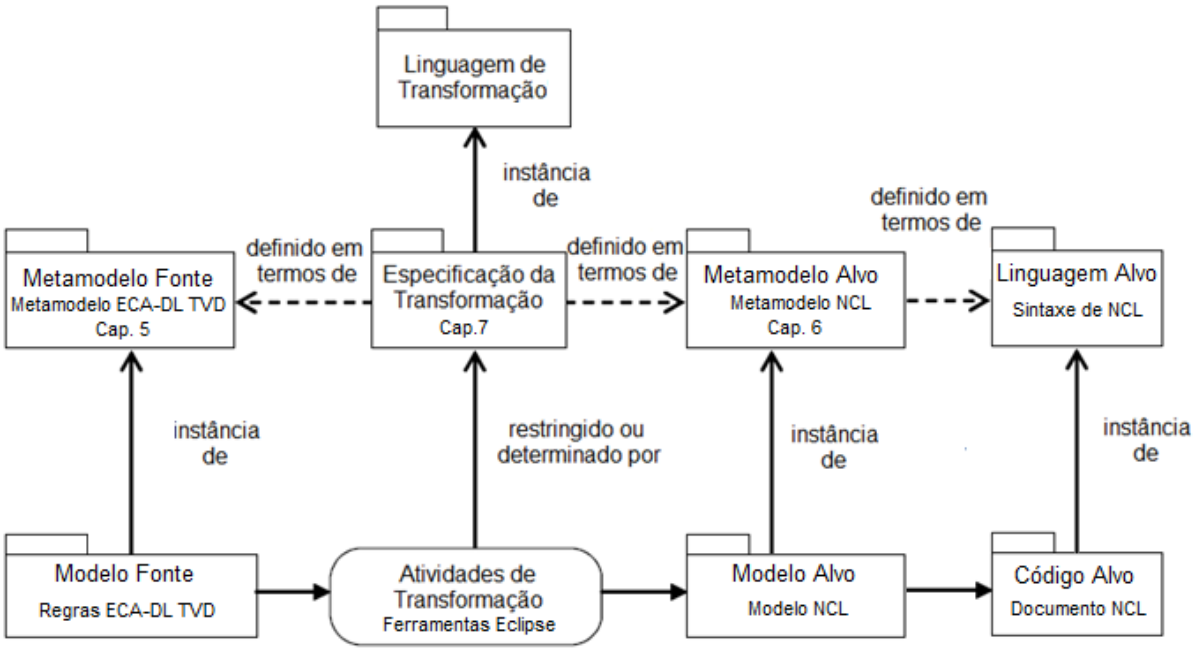


Figura 21. Visão do processo de transformação proposto.

5 ECA-DL TVD

Aplicações sensíveis ao contexto devem reagir a mudanças de contexto, adaptando seu comportamento ou invocando serviços. Adotou-se neste trabalho uma solução baseada em regras para modelar o comportamento reativo das aplicações. (DOCKHORN COSTA, 2007) propõe uma linguagem de especificação de regras específicas para o domínio de aplicações sensíveis ao contexto, a ECA-DL. Por ser específica de domínio, torna-se mais prática para desenvolvedores de aplicações sensíveis ao contexto do que outras linguagens de propósitos gerais. Como este trabalho pretende facilitar o desenvolvimento de aplicações sensíveis ao contexto no domínio de TV Digital, propõe-se uma adaptação de ECA-DL, denominada ECA-DL TVD, que incorpora conceitos do domínio de TVD.

Ambas as linguagens ECA-DL e ECA-DL TVD utilizam os modelos contextuais e de situação como base para especificação das regras. Pode-se dizer que os modelos de contexto e situação constituem o vocabulário para especificação de regras ECA-DL e ECA-DL TVD. Este trabalho utiliza o perfil UML para modelagem de contexto e situações proposto por (DOCKHORN COSTA, 2007) como base para modelar contextos e situações no domínio de TVD.

Este capítulo está estruturado como segue. A Seção 5.1 discute a abordagem utilizada para modelagem de contexto e situação; a Seção 5.2 apresenta os conceitos básicos da linguagem ECA-DL TVD; a Seção 5.3 define a sintaxe e a semântica dessa linguagem; na Seção 5.4 é apresentado o seu metamodelo; a Seção 5.5 discute o ciclo de execução de uma regra ECA-DL TVD; e, finalmente, a Seção 5.6 traz as considerações do capítulo.

5.1 MODELAGEM CONTEXTUAL

Um modelo contextual é a representação das condições contextuais das entidades no domínio da aplicação que são relevantes para a aplicação, ou um conjunto de aplicações. Neste trabalho, utiliza-se um modelo contextual como sendo um modelo conceitual de contexto. No sentido usado por este trabalho, os modelos conceituais são representações abstratas de um “dado domínio independente de design específico ou opções tecnológicas”. Portanto, em um modelo conceitual de contexto, abstrai-se como contexto é percebido, fornecido, aprendido, produzido e/ou utilizado.

A Seção 5.1.1 apresenta as abstrações de modelagem de contexto propostas por (DOCKHORN COSTA, 2007) que servem de base para os modelos de contexto das aplicações sensíveis ao contexto produzidos por este trabalho.

5.1.1 Entidades, Contextos e Situações

(DOCKHORN COSTA, 2007) define os conceitos de fundamentação Entidade (*Entity*) e Contexto (*Context*). Estes conceitos servem de base para que o desenvolvedor classifique os conceitos específicos do domínio da aplicação de maneira fundamentada, de acordo com as teorias propostas por (GUIZZARDI, 2005). Por exemplo, seguindo as diretrizes de modelagem propostas, o conceito Pessoa deve ser caracterizado como *Entity*, enquanto que o conceito Localização deve ser caracterizado como *Context*. Ambos os conceitos Pessoa e Localização são tipos, enquanto que João e sua atual localização são instâncias desses tipos, respectivamente.

A Figura 22 mostra a classe *Entity* representando o conceito de entidade e a classe *Context* representando uma condição contextual particular (por exemplo, localização, temperatura). A relação entre as classes *Context* e *Entity* é caracterizada pela associação *hasContext* e *isContextOf*.

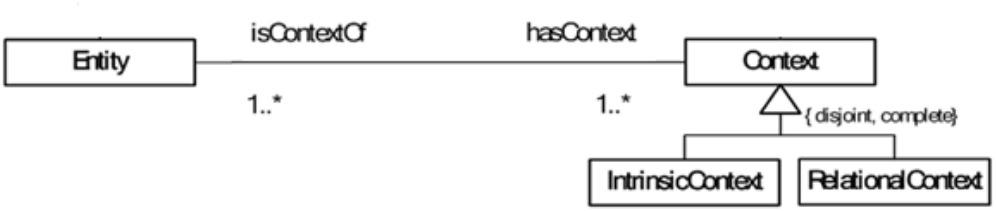


Figura 22. Abstrações para conceitos de entidade e contexto.

Contexto pode ser categorizado como Contexto Intrínseco (*IntrinsicContext*), ou como Contexto Relacional (*RelationalContext*). Contexto intrínseco define um tipo de contexto que pertence à natureza de uma simples entidade e não depende da relação com outras entidades. Um exemplo de um contexto intrínseco é a localização de uma pessoa ou de um objeto. Contexto relacional define um tipo de contexto que depende da relação entre entidades distintas. Um exemplo de contexto relacional é “uma TV exibe uma partida de futebol”, que relaciona uma entidade TV à outra, a partida de futebol.

Situações são elementos compostos de contextos e entidades. O tipo Situação Contextual (*ContextSituation*) ajuda a caracterizar situações com propriedades similares. Por exemplo, o tipo de situação contextual “Uma pessoa está a 5 metros da TV” caracteriza todas as instâncias de situações nas quais a distância entre qualquer pessoa e uma TV é menor que 5 metros. Como mostrado na Figura 23, uma situação contextual é composta de entidades e contextos.

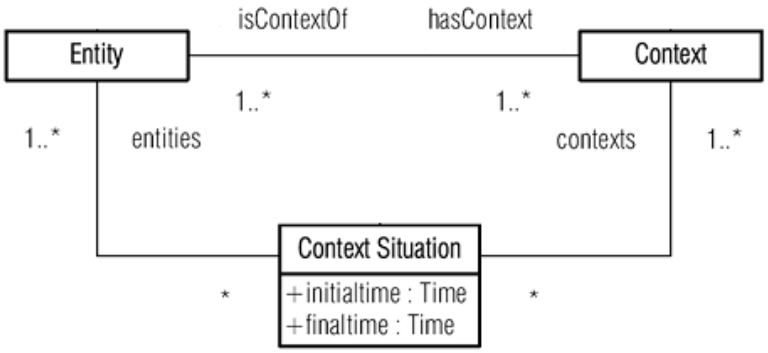


Figura 23. Abstrações para os conceitos de contexto incluindo situações.

Uma situação contextual exibe propriedades temporais, como o intervalo durante o qual a situação existe. Isso é representado pelos atributos *initialtime* (tempo inicial) e *finaltime* (tempo final), representados como atributos da classe *Context Situation*. O atributo *initialtime* captura o momento em que a situação começa, e o atributo *finaltime*, o momento em que ela se encerra. Uma vez que o *finaltime* é capturado, o modelo representa ocorrências de situações passadas. Assim, operações temporais podem ser incluídas para relacionar situações nos seus intervalos de ocorrência, como precedência, sobreposição, e pós-ocorrência.

5.1.2 Exemplos de Modelos

A Figura 24 mostra um modelo contextual contendo exemplos de tipos de entidade e contexto intrínseco. A fim de classificá-los de acordo com os conceitos entidade e contexto intrínseco, os estereótipos <<Entity>> e <<IntrinsicContext>> foram usados, respectivamente. O uso de estereótipos denota a classificação de um conceito específico do domínio com respeito aos conceitos da ontologia de fundamentação (GUIZZARDI, 2005). Por exemplo, usando o estereótipo <<Entity>> para a classe, o desenvolvedor explicita que essa classe possui as características inerentes do conceito Entidade.

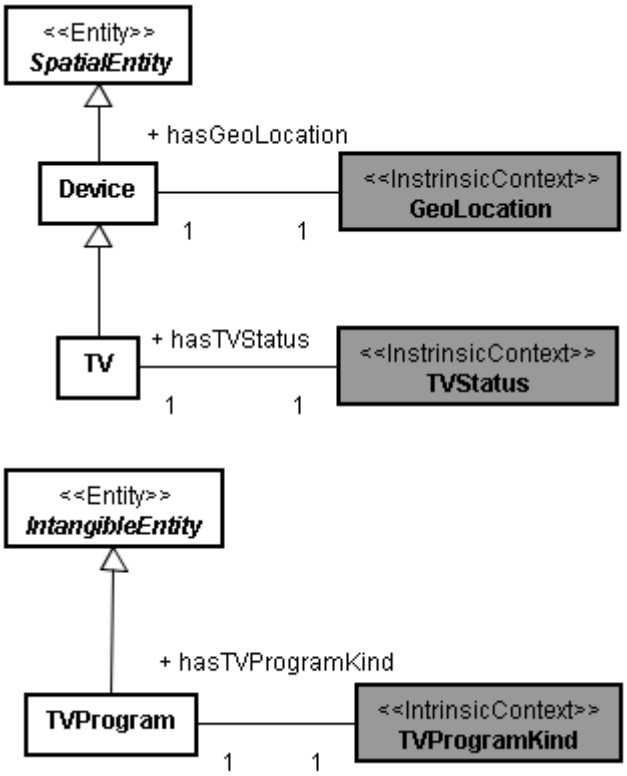


Figura 24. Tipos entidade e contexto intrínseco.

O modelo mostrado na Figura 24 define dois tipos de entidades, entidade espacial (*SpatialEntity*) e entidade intangível (*IntangibleEntity*), ambas estereotipadas como Entidade. As entidades espaciais representam os objetos que podem ser tangíveis, como uma pessoa, um dispositivo, um quarto ou um prédio, enquanto que entidades intangíveis representam objetos que não podem ser tocados, como uma aplicação e um programa de TV.

O tipo contexto intrínseco representa um tipo de contexto que é inerente a uma única entidade. Na Figura 24, os tipos contextos intrínsecos apresentados são: localização geográfica (*GeoLocation*), que é um contexto inerente a todas as entidades espaciais; o estado da TV (*TVStatus*), que é o contexto associado a todas as TVs e diz se ela está ligada ou não; e o programa de TV (*TVProgramKind*), que é inerente a um programa de TV.

Por outro lado, o tipo contexto relacional é inerente a uma variedade de entidades. A fim de especificar que um conceito específico do domínio da aplicação é um contexto relacional, o estereótipo <<RelationalContext>> é utilizado. A Figura 25 mostra um modelo contextual com um exemplo de contexto relacional entre entidades.

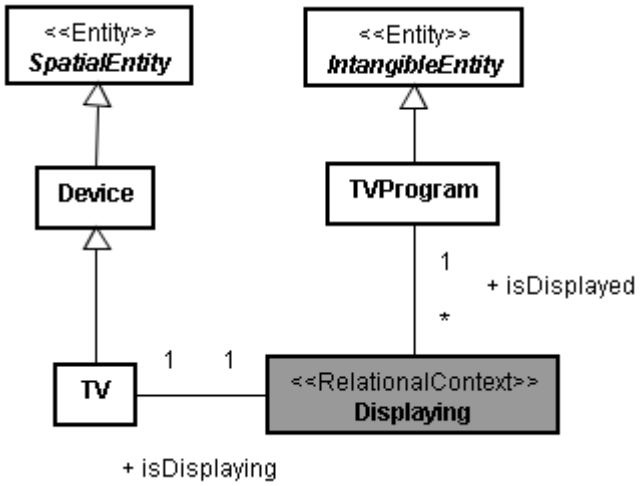


Figura 25. Tipos entidade e contexto relacional.

O contexto relacional Exibindo (*Displaying*) representa a relação entre uma TV e o programa sendo exibido por ela. Este contexto pode, portanto, ser usado para detectar se uma determinada TV está exibindo um programa de TV. De acordo com a cardinalidade especificada, uma TV pode exibir apenas um programa de TV, enquanto que um programa de TV pode ser exibido por várias TVs.

Finalmente, as situações contextuais são definidas em um modelo de situação, que usa o diagrama de classes padrão de UML 2.0 (OMG, 2010), juntamente com regras OCL – *Object Constraint Language* – 2.0 (OBJECT MANAGEMENT GROUP, 2003) para definir as condições para que certo tipo de situação contextual exista. Por exemplo, a Figura 26 mostra a representação da uma situação intrínseca que verifica se uma TV está ligada (seu status é igual a “On”).

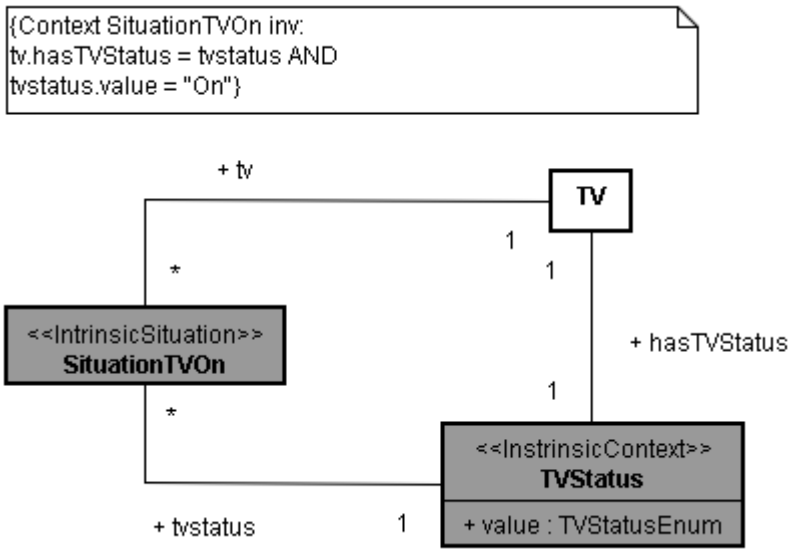


Figura 26. Situação intrínseca TV ligada.

O tipo situação *SituationTVOn* é composto de um tipo entidade *TV* e seu estado (*TVStatus*), que é do tipo contexto intrínseco. A invariante OCL define um predicado que deve permanecer verdadeiro para todas as instâncias de *SituationTVOn*. Por exemplo, o predicado "*tv.hasTVStatus = tvstatus AND tvstatus.value = "On"*" define que todas as instâncias da classe *SituationTVOn* devem estar associadas a TV's cujos contextos *TVStatus* devem ter valores "On". A cardinalidade também restringe as instâncias da situação, por exemplo, instâncias de *SituationTVOn* devem ser associadas a uma TV e a um estado da TV. A invariante OCL posteriormente restringe as instâncias de *SituationTVOn* por definir que o estado da TV dever ser "ligada" ("*On*"). A palavra "*context*" na invariante OCL é uma primitiva reservada de OCL que define a classe para qual a restrição deve ser aplicada. Portanto, a restrição OCL da Figura 11 deve ser aplicada à classe *SituationTVOn*.

5.2 ECA-DL TVD: CONCEITOS BÁSICOS

ECA-DL TVD é uma linguagem específica de domínio, que visa especificar comportamentos reativos de aplicações sensíveis ao contexto por meio de regras, conhecidas como regras Evento-Condição-Ação (*Event-Condition-Action*), ou apenas regras ECA. ECA-DL TVD foi desenvolvida considerando os seguintes requisitos:

- Poder de expressividade, a fim de permitir especificação de combinação de eventos. Para tal, ECA-DL TVD permite o uso de operadores relacionais (<, >, =), e o uso de conectivos lógicos (AND, OR, NOT) para combinar eventos e condições;

- Uso conveniente, a fim de ser fácil de ser utilizada por desenvolvedores de aplicações sensíveis ao contexto. ECA-DL TVD provê construtores de alto nível para facilitar composição de eventos, condições e ações;

- Suporte a elementos de TVD, a fim de permitir a definição de eventos e ações característicos do domínio de TVD.

Uma regra ECA-DL TVD é consistente com o padrão arquitetural *Event-Control-Action* – ECA (Seção 2.2.4), que provê uma estrutura de alto nível para sistemas que pró-ativamente reagem a mudanças no contexto. Uma regra consistente com esse padrão arquitetural deve conter os seguintes elementos:

- Um ou mais eventos, que modelam a ocorrência de mudanças relevantes no contexto do usuário. A ocorrência desses eventos “dispara” a avaliação da regra;

- Nenhuma ou mais condições, que representam a situação sob a qual as ações das regras são iniciadas, dado que os eventos ocorreram. Uma condição é tipicamente expressada como uma (simples ou complexa) expressão booleana;

- Uma ou mais ações, que representam as operações quando a regra é aplicável.

Eventos, condições e ações podem ter uma estrutura interna. Por exemplo, uma condição pode consistir de múltiplas subcláusulas, ou uma ação pode ser implementada como um procedimento que invoca uma série de sub-procedimentos.

Nesse sentido, a linguagem ECA-DL TVD permite a definição de regras contendo os seguintes elementos:

- Eventos (*Events*), que permitem definir mudanças relevantes de situações por meio de combinações simples ou complexas de eventos;

- Condição (*Condition*), parte opcional que permite definir uma expressão lógica que deve permanecer verdadeira simultaneamente à ocorrência de eventos especificados e anteceder a execução de uma ação;

- Ação (*Action*), que permite a definição de invocações de operações para serem executadas, devido à ocorrência de eventos e ao preenchimento das condições associadas a esses eventos.

5.2.1 Navegação nos Modelos de Contexto e Situação

A navegação visa alcançar os valores ou objetos de interesse em modelos contextuais e de situação (Seção 5.1). Assim como em OCL, em ECA-DL TVD, são usados *pontos* para navegar dos objetos para seus atributos. O elemento alvo de uma expressão de navegação é sempre um tipo de dados primitivo, ou seja, um valor booleano ou uma string. Geralmente, a navegação em ECA-DL TVD começa com uma entidade, tal como *EntityType.entityID*, em que *EntityType* se refere ao tipo da entidade e *entityID* ao seu identificador único. ECA-DL TVD assume que entidades possuem identificadores únicos, que são compartilhados pelas

plataformas e aplicações. Os elementos seguindo os identificadores únicos das entidades visam navegar através dos atributos das entidades até que os valores desejados dos atributos das entidades sejam alcançados. A Figura 27 mostra um fragmento de um modelo contextual. A fim de navegar até a localização geográfica de um dispositivo, deve-se definir a seguinte expressão em ECA-DL TVD: *Device.id1.hasGeoLocation.value*, em que *id1* é o identificador único de uma entidade do tipo *Device*. De forma semelhante, a fim de navegar até o estado da TV, deve-se utilizar a expressão *TV.tv1.hasTVStatus.value*.

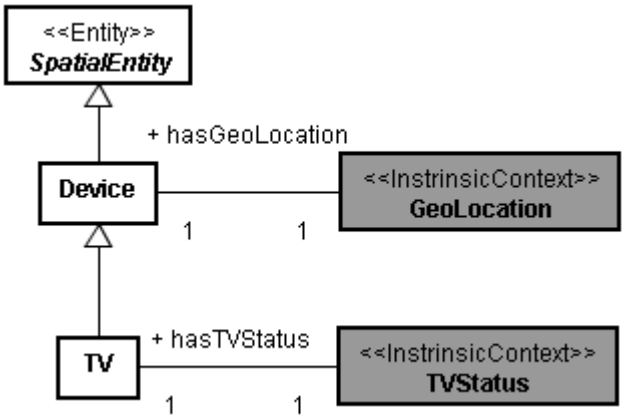


Figura 27. Fragmento de um modelo contextual.

5.2.2 Eventos

Um evento representa algum acontecimento de interesse, que tipicamente representa uma mudança no contexto. Dois tipos de eventos são suportados em ECA-DL: eventos de situação (*situation events*) e eventos primitivos (*primitive events*).

Eventos de Situação

Eventos de situação são definidos em termos de transições de situação. Dois tipos de transições de situação são suportados: *EnterTrue* e *EnterFalse*. *EntreTrue* (*S1*) representa quando a situação S1 passa a existir, e *EntreFalse* (*S1*) quando S1 deixa de existir.

A Figura 26 (Seção 5.1.2) mostra um exemplo simples de uma especificação de uma situação denominada *SituationTVOn*, que define a situação em que uma TV está ligada. Para se referir ao evento em que a TV de João é ligada, a seguinte transição deve ser definida: *EnterTrue (SituationTVOn (TV.TVJoao))*. Semelhantemente, para se referir ao evento que ocorre quando a TV de João é desligada, a seguinte expressão deve ser definida: *EnterFalse (SituationTVOn (TV.TVJoao))*.

Em geral, a fim de verificar se uma entidade específica está atualmente em uma determinada situação, pode-se passar o identificador único da entidade como parâmetro da situação como, por exemplo, *SituationType (EntityType.entityID)*. Se a especificação da situação envolve mais de uma entidade, estas entidades podem ser fornecidas como parâmetros complementares, ou seja, *SituationType (EntityType.entityID1, EntityType.entityID2)*.Por exemplo, seja a situação *SituationContained (container, contained)*, que especifica uma relação de "conter" entre duas entidades, a entidade recipiente (*container*) e a entidade contida (*contained*). Para saber quando João entra em sua casa, deve-se utilizar a seguinte expressão: *EnterTrue (SituationContained (Building.CasaJoao, Person.Joao))*.

Além disso, considerando expressões de filtragem, é possível não definir alguns parâmetros. Por exemplo, para saber quando alguém entra na casa de João, deve-se usar a expressão *EnterTrue (SituationContained (Person.Joao.house,))*. Semelhantemente, para saber quando João deixa qualquer entidade recipiente, deve-se usar a expressão *EnterFalse (SituationContained (, Person.Joao))*.

Eventos Primitivos

Em ECA-DL TVD, eventos primitivos representam os eventos que são típicos do ambiente de TVD e não são detectados por meio de situações. Ao contrário dos eventos de situação, eventos primitivos são gerados por componentes que não são necessariamente implementados por mecanismos de detecção de situação. Normalmente, esses eventos têm como parâmetro o conteúdo televisivo a qual eles estão relacionados. Um exemplo de evento primitivo pode ser *onBegin (Movie.Movie1)*, que representa o momento em um vídeo, cujo *id* é *Movie1*, é iniciado. Contudo, também é possível que estes eventos tenham como parâmetros entidades, contextos, atributos e literais. Seções 5.3.2, 5.3.3, 5.3.4 mostram exemplos de regras ECA-DL TVD que utilizam alguns destes eventos.

A Tabela 4 mostra os tipos de eventos primitivos suportados por ECA-DL TVD.

Tabela 4. Eventos TVD.

Evento TVD	Descrição
onBegin	Quando o parâmetro passado para esse evento for iniciado.
onEnd	Quando o parâmetro passado para esse evento for terminado.
onAbort	Quando o parâmetro passado para esse evento for abortado.

onPause	Quando o parâmetro passado para esse evento for pausado.
onResume	Quando o parâmetro passado para esse evento retomar a atividade após ser pausado.
onSelection	Quando o parâmetro passado para esse evento for selecionado.
onBeginAttribution	Logo que um valor (a ser especificado) seja atribuído a uma propriedade do parâmetro passado para esse evento.
onEndAttribution	Logo que um valor (a ser especificado) tiver sido atribuído a uma propriedade do parâmetro passado para esse evento.

Composição de Eventos

Expressões de composição de eventos podem combinar eventos primitivos usando os operadores de composição de eventos da Tabela 5. O resultado de uma composição de eventos é chamado de evento composto.

Tabela 5. Operadores de composição de eventos.

Operador	Evento Composto
e1 and e2	Ocorre quando ambos os eventos e1 e e2 ocorrem independentemente da ordem.
e1 or e2	Ocorre quando pelo menos um dos eventos e1 e e2 ocorrem.

Como um exemplo de um evento composto, considere o operador "*and*" para especificar o evento "quando o vídeo de um comercial (*Video1*) começar e quando a imagem do produto (*Image1*) começar a ser exibida". A especificação da composição de eventos deve ser a seguinte: *OnBegin (Video.Video1) and OnBegin(Image.Image1)*. Semelhantemente, se o evento fosse "quando o filme (*Movie1*) acabar ou quando ele for pausado", a especificação da composição de eventos deve ser a seguinte: *OnEnd (Movie.Movie1) or OnPause(Movie.Movie1)*.

Janela de Eventos

Notificações de eventos primitivos que fazem parte de uma composição de eventos não devem ser levadas em consideração indefinidamente para detectar um evento particular. Por exemplo, considere o evento "*onResume(Movie.Movie1) and onEnd(Image.Image1)*", que especifica o evento no qual o filme *Movie1* retorna de uma

pausa e a imagem *Image1* deixou de ser exibida. Considere que a imagem começou a ser exibida e que, no intervalo de uma hora, o filme retormou de três pausas. De acordo com ECA-DL TVD, quando a imagem finalmente deixar de ser exibida, apenas a última ocorrência do evento *onResume(Movie.Movie1)* deve ser considerada da composição. Isso evita um armazenamento de histórico de eventos, ou seja, exige menos espaço em memória e menor processamento, o que favorece o uso de ECA-DL TVD para o ambiente de TVD, o qual possui recursos computacionais limitados.

Além disso, a fim de evitar que o mesmo evento primitivo seja considerado pela mesma composição, ECA-DL TVD define que as notificações de eventos elementares devem ser consideradas apenas uma vez por composição. Ou seja, quando a composição do exemplo acima foi detectada, os eventos elementares *onResume(Movie.Movie1)* e *onEnd(Image.Image1)* foram "consumidos" pela detecção da composição e, para uma nova detecção, apenas novos eventos devem ser considerados. A Figura 28 ilustra um exemplo de composição de eventos.

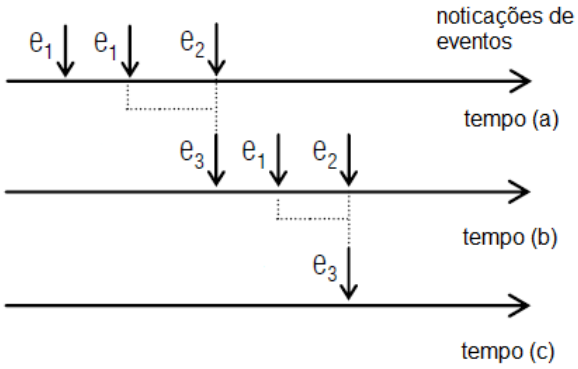


Figura 28. Detecção de composição de eventos.

Na Figura 28, o evento *e3* representa o evento composto "*e1 and e2"*. Considere que a seguinte sequencia de notificação de eventos primitivos é recebida: *e1*, *e1*, *e2*, *e1*, *e2*. O evento composto *e3* é detectado duas vezes. Na primeira vez (linha do tempo (b)), as últimas notificações de *e1* e *e2* são consumidas. Na segunda (linha do tempo (c)), as últimas ocorrências de *e1* e *e2* são consideradas para a detecção do evento composto.

Para a composição de eventos, não são considerados os eventos de situação. Apesar de não serem considerados na composição, os eventos de situação são consumidos isoladamente e, portanto, também não devem ser considerados indefinidamente. Para solucionar este problema, ECA-DL TVD define um conceito de *Janela de Detecção*. A janela de detecção estabelece um tempo para o qual as notificações de eventos, tanto de eventos primitivos, quanto de situação, devem ser consideradas. Ocorrências fora desta janela não

devem ser mais consideradas para a detecção. A versão atual de ECA-DL TVD considera uma mesma janela para todas as regras definidas. A Figura 29 ilustra a utilização da janela de detecção de eventos.

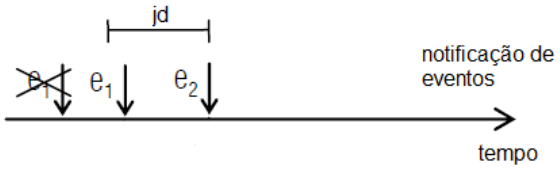


Figura 29. Janela de Detecção.

O evento *e1* representa o evento *EnterTrue* de uma situação *Situation1*, enquanto que *e2* representa o evento *EnterFalse* de uma situação *Situation2*. De acordo com a janela de detecção (*jd*), apenas a segunda ocorrência de *e1* e o evento *e2* serão consideradas na execução de uma regra ECA-DL TVD que faz referência aos eventos *e1* e *e2*.

5.3 SINTAXE E SEMÂNTICA

Foram mostrados na Seção 5.2 os conceitos básicos de ECA-DL TVD que resultaram nas cláusulas *Upon*, *When* e *Do*. Eventos são definidos na cláusula *Upon*, enquanto que as condições são especificadas na cláusula *When* e, finalmente, ações são especificadas na cláusula *Do*. A cláusula *When* pode ser omitida se não há condições a serem especificadas. Dessa forma, uma regra ECA-DL TVD tem a seguinte estrutura:

```
Upon    <uponExpression>
When    <conditionExpression>
Do      <actionExpression>
```

Cada uma dessas cláusulas é especificada com base nos elementos dos modelos contextuais e de situação descritos na Seção 5.1. Assim, a sintaxe de ECA-DL TVD faz referência aos elementos dos modelos contextuais e de situação.

5.3.1 Elementos Contextuais

Conforme mencionado na Seção 5.1, os modelos de contexto e situação definem os elementos que serão utilizados na especificação das regras. Esses elementos são

representados em ECA-DL TVD por Entidades (*Entity*), Contextos (*Context*) e Situações Contextuais (*ContextSituation*), os quais são mostrados no fragmento de metamodelo na Figura 30.

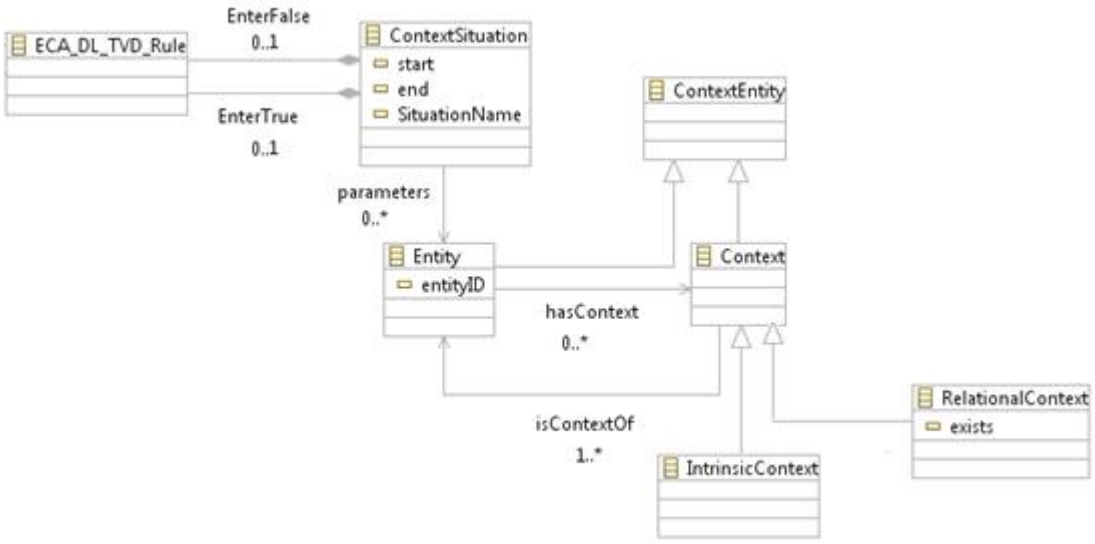


Figura 30. Elementos Contextuais presentes em regras ECA-DL TVD.

Como pode ser visto na Figura 30, uma entidade tem como atributo o seu identificador único (*entityID*) e pode conter contextos associados a ele. Um contexto pode ser um Contexto Intrínseco (*IntrinsicContext*) ou um Contexto Relacional (*RelationalContext*). Cada contexto se refere a pelo menos uma entidade. O contexto relacional possui um atributo indicando se ele existe naquele momento (*exists*). No metamodelo de ECA-DL TVD, Entidades e Contextos são definidos como Entidades Contextuais (*ContextEntity*). São exemplos de entidades e contextos aqueles apresentados na Figura 24 e na Figura 25.

Também na Figura 30, nota-se que uma situação contextual (*ContexSituation*) possui dois atributos, início (*start*) e fim (*end*), indicando se a situação já iniciou e se já deixou de existir, respectivamente. Uma situação pode se referir a várias entidades e seus respectivos contextos (através da associação *parameters*).

Para atualizar os atributos *start* e *end*, uma situação pode referenciar duas regras ECA-DL TVD: uma que verifica quando a situação passa a existir (associação *EnterTrue*) e outra que verifica quando a situação deixa de existir (associação *EnterFalse*). Um exemplo de situação é mostrado na Figura 26.

5.3.2 Cláusula *Upon*

Uma expressão <*uponExpression*> pode ser definida por um evento de situação ou (uma combinação de) eventos primitivos (discutidos na Seção 5.2.2). A ocorrência do evento (ou da composição de eventos) especificado na cláusula *Upon* dispara a avaliação da cláusula *When*.

Assim, os possíveis elementos de uma cláusula *Upon* são: um Evento de Situação (*SituationEvent*) ou um Evento Primitivo (*PrimitiveEvent*). A Figura 31 mostra o fragmento do metamodelo de ECA-DL TVD referente aos elementos da cláusula Upon. Um evento de situação (*SituationEvent*) possui como atributo o tipo da transição (*EnterTrue* ou *EnterFalse*) e referencia uma situação (*ContexSituation*). Já o evento primitivo (*PrimitiveEvent*) pode ser um Evento de TVD (*TVDEvent*) ou uma Composição de Eventos de TVD (*CompositeTVDEvent*).

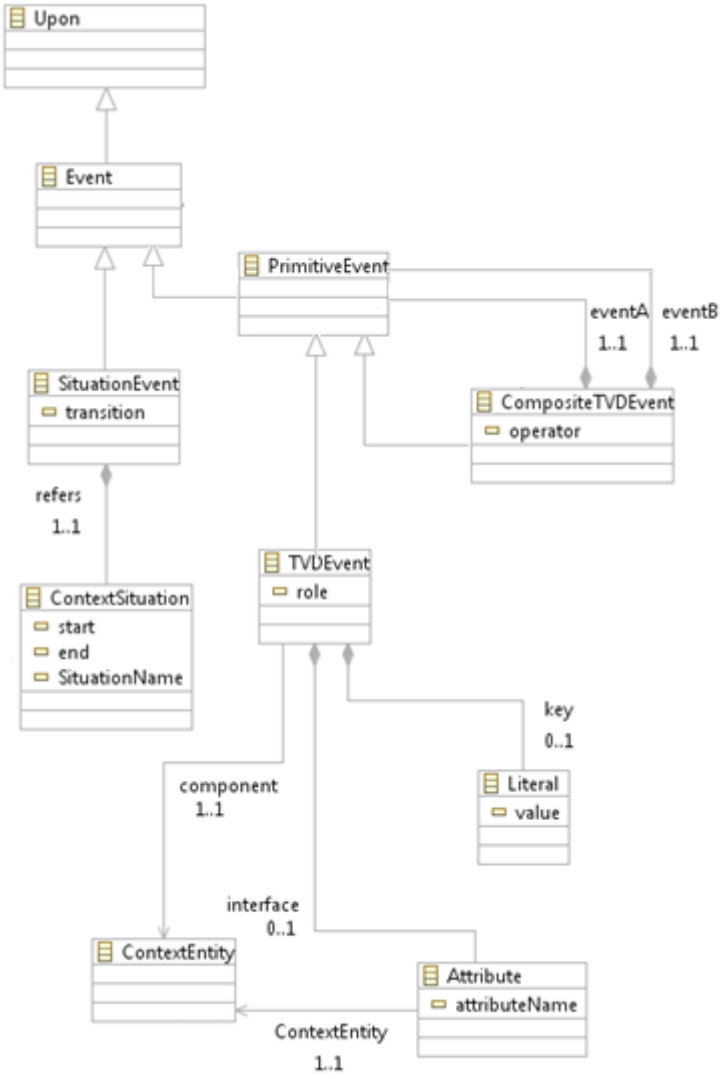


Figura 31. Elementos da cláusula *Upon*.

Também pode ser visto no modelo da Figura 31 que um Evento de TVD (*TVDEvent*) possui como atributo o tipo do evento (*role*), que pode ser um dos descritos na Tabela 4, e é composto dos possíveis parâmetros: uma Entidade Contextual (*ContexEntity*), Atributos (*Attribute*) dessas entidades contextuais, e literais (*Literal*). Um atributo necessariamente se refere a uma entidade definida nos modelos contextuais.

Além disso, um Evento de TVD composto (*CompositeTVDEvent*) consiste de um evento composto, em que a composição é especificada em termos de um dos operadores binários *and* e *or*. Cada evento composto possui dois operandos, representados na Figura 31 pelas composições denominadas *operandA* e operandB, e que são também eventos primitivos, ou seja, podem ser um evento de TVD ou novamente um evento de TVD composto.

Exemplos de possíveis cláusulas *Upon* são:

- *Upon onBegin (Movie.Movie1)* , que representa um evento de TVD de início de exibição de um filme (*Movie1*);
- *Upon EnterFalse (SituationTVOn (TV.TVJoao))*, representando o evento de situação que indica o momento em que a TV de João foi desligada;
- *Upon onEnd(Movie.Movie1) or onPause(Movie.Movie1)*, ilustrando um evento de TVD composto que indica que ou o filme terminou ou o filme foi pausado.

5.3.3 Cláusula *Where*

Uma expressão <*conditionExpression*> consiste de uma expressão booleana que pode ser composta de outras expressões booleanas (recursivamente) por meio de operadores booleanos unários e binários. As expressões booleanas da cláusula *When* podem ser:

- Uma verificação se uma situação existe ou não;
- Uma comparação binária usando os operadores > (maior que), >= (maior igual), < (menor que), <= (menor igual), = (igual) e <> (diferente). Um dos operandos deve ser um atributo de uma entidade contextual e outro um valor a ser comparado (literal).

Os elementos da cláusula *When* são mostrados no fragmento de metamodelo na Figura 32.

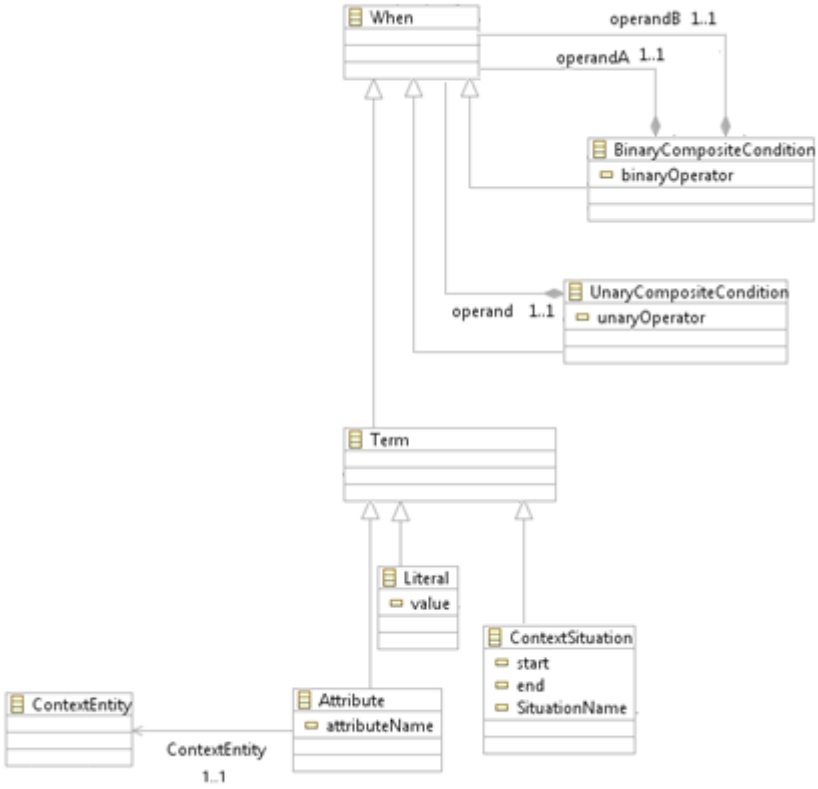


Figura 32. Elementos da cláusula *When*.

Conforme mostrado no metamodelo da Figura 32 uma cláusula *When* pode ser um dos seguintes elementos:

- Uma condição composta unária (*UnaryCompositeCondition*), que possui como atributo o operador unário (*unaryOperator*) e contém um único operando (*operandA*). O operador unário suportado em ECA-DL TVD é o operador *not*. O *operandA* é novamente um elemento *When*, permitindo, por exemplo, descrever que uma situação não está ocorrendo.

- Uma condição composta binária (*BinaryCompositeCondition*), que possui como atributo um dos seguintes operadores binários (*binaryOperator*): *and*, *or*, >, >=, <, <=, =, <>, e contem dois operandos (*operandA* e *operandB*). Os operandos podem ser novamente qualquer elemento *When*.

- Um termo *(Term*), que pode ser uma situação contextual *(ContextSituation*), um atributo de uma entidade contextual *(Attribute*) e um literal *(Literal*).

Conforme pode ser visto na Figura 32, um atributo de uma entidade contextual (*Attribute*) se refere a um valor de atributo de uma entidade ou de um contexto (associação *ContextEntity*). Portanto, uma expressão de atributo é composta de uma referência a uma entidade contextual seguida no nome do atributo, tal como *EntityType.entityID.attribute*, ou *EntityType.entityID.Context.attribute*. Por exemplo, considerando o modelo contextual da Figura 24, a seguinte expressão é válida: *TV.tv1.hasTVStatus.value*.

São exemplos de cláusulas *When*:

- *When not (SituationContained (Building.CasaJoao, Person.Joao))*, que representa a condição em que João não esteja na sua casa;
- *When Person.Joao.idade < 18 and SituationTurnOn(TV.TVJoao)*, definindo a condição em que João tenha menos de 18 e que sua TV esteja ligada.

5.3.4 Cláusula *Do*

Uma expressão <*actionExpression*> consiste de uma lista de ações a serem invocadas. Os elementos da cláusula *Do* são mostrados no fragmento de metamodelo na Figura 33.

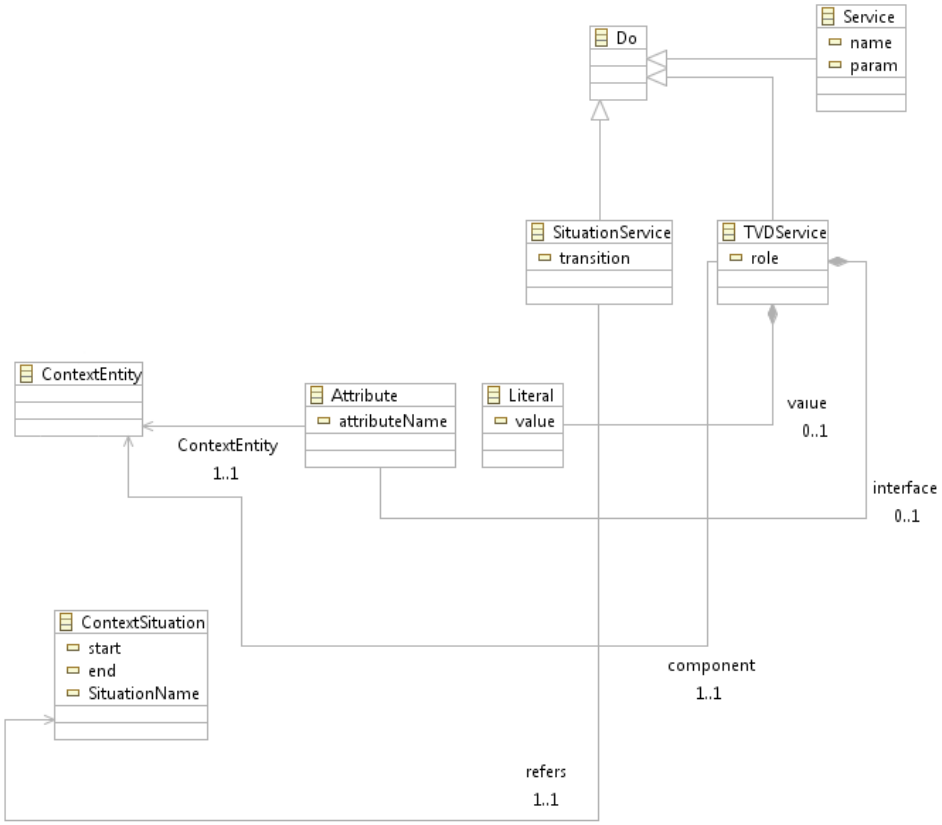


Figura 33. Elementos da cláusula *Do*.

ECA-DL TVD define três tipos de ações:

- Serviços gerais (*Service*), que são funções de ação que são externos à plataforma sensível ao contexto. Exemplos são *sendSMS (Person.Joao)*, que requisita a entrega de um SMS ao Joao, ou *TelephoneCall (Person.Joao, Person.Maria)*, que inicia uma conversa telefônica entre duas pessoas;

- Serviços de Situação (*SituationService*), que são utilizados pelas regras ECA-DL TVD responsáveis por indicar que uma situação teve início ou foi finalizada. Portanto, um serviço de situação se refere a uma situação contextual e tem como atributo *EnterTrue* ou *EnterFalse*, que define se a situação deve ser ativada ou desativada, respectivamente.

- Serviços de TVD (TVDService), que são ações características do ambiente de TVD. Um exemplo de serviço de TVD pode ser *pause (Movie.Movie1)*, que pausa um filme, ou *stop (Movie.Movie1)* que para a exibição do filme. A Tabela 6 mostra os tipos de serviços de TVD primitivos suportados pela versão atual de ECA-DL TVD.

Tabela 6. Serviços de TVD.

Serviço de TVD	Descrição
Start	Inicia o parâmetro passado para esse serviço.
Stop	Termina o parâmetro passado esse serviço.
Abort	Aborta o parâmetro passado para esse serviço.
Pause	Pausa o parâmetro passado para esse serviço.
Resume	Retorna de uma pausa o parâmetro passado para esse serviço.
Set	Estabelece um valor (a ser especificado) à propriedade de do parâmetro passado para esse serviço.

São exemplos de cláusulas *Do*:

- *Do sendSMS (Person.Joao, "Hello")*, que envia uma mensagem para o João;
- *Do EnterTrue (SituationTurnOn (TV.TVJoao))*, que sinaliza o início da situação *SituationTurnOn*;
- *Do stop (Movie.Movie1)*, que encerra a apresentação do filme *Movie1*.

5.3.5 Exemplos de regras ECA-DL TVD

A seguir são mostrados alguns exemplos de regras ECA-DL TVD para diferentes domínios. A seguinte regra especifica que quando João chegar em casa, a sua TVD deve ser ligada:

```
Upon    EnterTrue (SituationContained (Building.CasaJoao, Person.Joao))
Do      TurnOn (TV.TVJoao)
```

A seguinte regra define que quando o filho de João chegar em casa, e João estiver vendo um filme inapropriado para a idade de seu filho, o filme deve ser parado.

```
Upon    EnterTrue        (SituationContained        (Building.CasaJoao,
Person.JoaoSon))
When    SituationWatchingMovie(Person.Joao) and
        Movie.Movie1. classification > 14
Do      stop (Movie.Movie1)
```

Finalmente, a seguinte regra ECA-DL especifica que sempre que houver um evento de alarme de ataque epiléptico para o usuário João, e ele estiver dirigindo, ele deve receber um alerta via SMS (DOCKHORN COSTA, 2007).

```
Upon    EnterTrue (SituationEpilepticAlarm (Patient.Joao))

When    SituationDriving (Patient.Joao)

Do      SendSMS (Patient.Joao, "João, você vai ter um ataque
epiléptico, por favor, pare o carro")
```

5.4 METAMODELO

Com base nos elementos sintáticos descritos na Seção 5.3, a Figura 34 mostra o metamodelo completo da linguagem ECA-DL TVD.

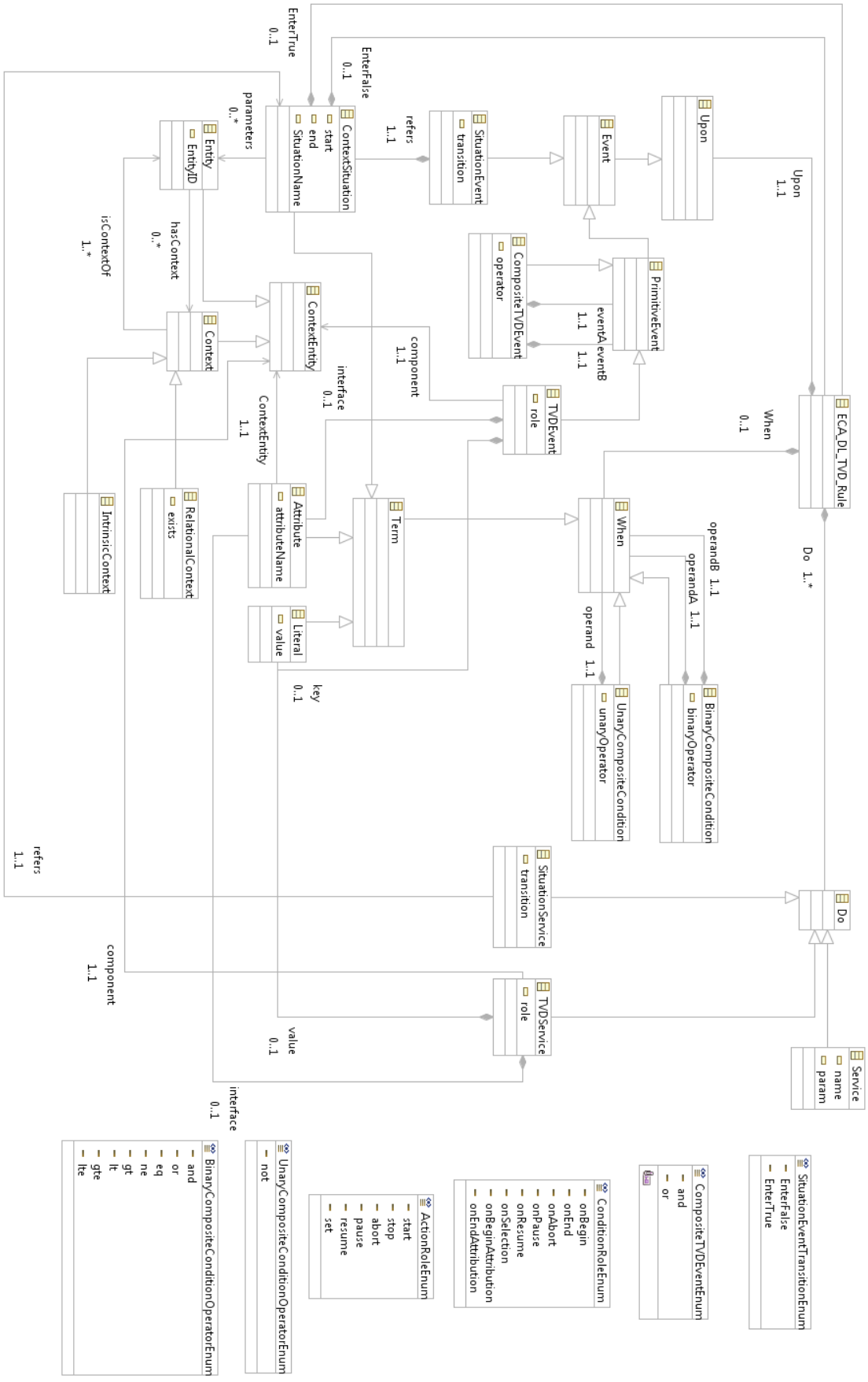


Figura 34. Metamodelo de ECA-DL TVD.

Na Figura 34, podem ser vistos alguns elementos do tipo enumeração, que são usados, por exemplo, para descrever os tipos de eventos de TVD, os serviços de TVD, os tipos de transição de situação, comparadores e operadores. Isso visa limitar as possibilidades dos valores assumidos por um atributo, permitindo a criação de regras válidas.

O metamodelo da Figura 34 tem um papel importante nos mapeamentos apresentados no Capítulo 7, que são feitos no nível de metamodelo. Com base nesses mapeamentos, instâncias dos elementos de ECA-DL TVD podem ser mapeadas para instâncias dos elementos do metamodelo da linguagem NCL, que é apresentado no Capítulo 6.

5.5 EXECUÇÃO DE UMA REGRA ECA-DL TVD

Em geral, o ciclo de execução de uma regra ECA-DL TVD consiste em detectar eventos, verificar as condições e invocar ações. Uma vez estabelecida a regra ECA-DL em um Monitor (discutido no Capítulo 4), este passa continuamente a verificar se os eventos definidos numa cláusula *Upon* ocorreram ou não. A Figura 35 ilustra o ciclo de execução de uma regra ECA-DL TVD.

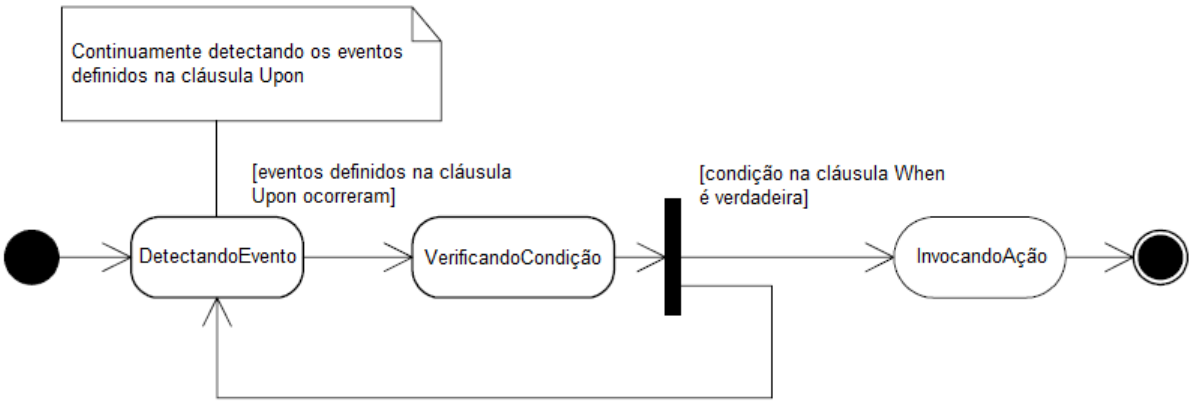


Figura 35. Ciclo de execução de uma regra ECA-DL TVD.

A detecção contínua de eventos inicia no estado *DetectandoEvento*. Quando o evento ocorre (ou uma combinação de eventos), o Monitor verifica se a condição definida na cláusula *When* é verdadeira ou falsa. A verificação da condição é feita no estado *VerificandoCondição*. Se a condição é avaliada verdadeira, as ações são invocadas, as quais têm início no estado *InvocandoAção*. Independente se a condição foi avaliada

verdadeira ou falsa, depois de deixar o estado *VerificandoCondição*, o monitor volta a detectar os eventos.

5.6 CONSIDERAÇÕES DO CAPÍTULO

Neste capítulo foi definida a linguagem específica de domínio ECA-DL TVD, que é usada para especificar o comportamento reativo de aplicações sensíveis ao contexto no ambiente de TVD. Com relação à arquitetura conceitual para o Gerenciador de Contexto do Ginga (apresentada no Capítulo 4), ECA-DL TVD pode ser usada para descrever uma instância do componente Monitor, que é o componente responsável por acionar os serviços devidos com base nas mudanças de contexto.

Vale mencionar que, em certos casos, ECA-DL TVD pode não ser adequada para descrever o comportamento reativo, como por exemplo, os cenários que requerem o uso de conjuntos de entidades ou expressões matemáticas complexas. Caso haja a necessidade de se utilizar caracterizar um conjunto de entidades, pode-se contornar utilizando uma entidade com identificador genérico, para especificar que qualquer instância daquela entidade pode assumir aquele papel na regra. Um exemplo é definir uma instância “person” que identifica qualquer pessoa que esteja envolvida na aplicação. No caso das expressões matemáticas, pode-se pensar em um trabalho futuro para incorporar elementos matemáticos nas regras e permitir dizer que um atributo é duas vezes maior que outro.

Finalmente, além de definir ECA-DL TVD, este capítulo mostrou o metamodelo dessa linguagem. Com base nesse metamodelo e no metamodelo de NCL (a ser definido no Capítulo 6), este trabalho propõe a realização do componente Monitor em NCL de forma automática, usando técnicas de MDD. A realização do componente Monitor é discutida no Capítulo 7.

6 APLICAÇÕES SENSÍVEIS AO CONTEXTO EM NCL

Conforme discutido no Capítulo 1, as aplicações desenvolvidas no Ginga podem ser declarativas, utilizando a linguagem NCL, ou procedurais, utilizando Java. Este trabalho visa facilitar o desenvolvimento de aplicações sensíveis ao contexto para TVD tendo NCL como linguagem alvo. Na Seção 2.3.3, foi visto que NCL é uma linguagem baseada em XML e permite o desenvolvimento de aplicações multimídia com sincronismo espaço-temporal entre objetos de mídia, como vídeos, áudios, imagens, etc. NCL também suporta objetos de mídia imperativos, escritos na linguagem Lua (discutida na Seção 2.3.4). NCL atua como linguagem de cola, relacionando no tempo e espaço diferentes nós de mídia, criando o que se denomina de uma apresentação multimídia.

Em trabalhos preliminares (VALE, MIELKE, *et al.*, 2010) (MIELKE, 2010) buscou-se desenvolver aplicações sensíveis ao contexto utilizando NCL, visando analisar a linguagem NCL quanto ao seu suporte ao desenvolvimento de tais aplicações. Isso permitiu evidenciar elementos da linguagem NCL que são repetidamente utilizados no desenvolvimento de aplicações sensíveis ao contexto, além de observar uma estrutura padrão dessas aplicações desenvolvidas em NCL. Neste capítulo são discutos esses elementos de NCL e o metamodelo correspondente da linguagem, que é utilizando no processo de realização orientada a modelos do monitor (Capítulo 7).

Este capítulo está estruturado como segue. A Seção 6.1 mostra como a arquitetura conceitual do Capítulo 4 pode ser realizada na plataforma Ginga; a Seção 6.2 discute os elementos de NCL que fornecem o suporte ao desenvolvimento de aplicações sensíveis ao contexto; a Seção 6.3 apresenta o metamodelo simplificado de NCL; a Seção 6.4 discute como os *scripts* NCLua podem ajudar no desenvolvimento das aplicações sensíveis ao contexto; e, finalmente, a Seção 6.5 traz as considerações do capítulo.

6.1 ARQUITETURA DE UMA APLICAÇÃO SENSÍVEL AO CONTEXTO EM NCL

Com base no que foi visto ao longo dos trabalhos preliminares (VALE, MIELKE, *et al.*, 2010) (MIELKE, 2010), pode-se fazer um esboço da arquitetura conceitual da aplicação sensível ao contexto (Capítulo 4) realizada na plataforma Ginga. A Figura 36 mostra como os elementos conceituais são representados por um documento NCL e *scripts* NCLua.

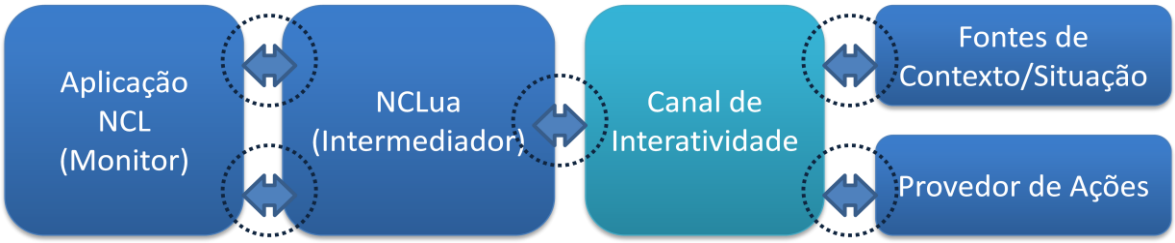


Figura 36. Aplicação sensível ao contexto na plataforma Ginga.

Na Figura 36, pode ser visto que a aplicação NCL (ou documento NCL) representa o Monitor, sendo o comportamento reativo da aplicação sensível ao contexto definido por meio de conectores e *links*. Além dos conectores, também são definidos no documento NCL as mídias referentes às entidades do universo de discurso de aplicação. Para atualizar os valores de suas condições contextuais, o *script* NCLua atua como um intermediador (ou Ponto de Interação) com o canal de interatividade para realização a comunicação entre aplicação NCL a as Fontes de Contexto e de Situação. Além disso, os *scripts* NCLua também servem de intermediadores para a comunicação os provedores de ações externos à plataforma Ginga. Caso as ações sejam próprias do ambiente de TVD, elas podem ser representadas no documento NCL.

6.2 Elementos da Linguagem NCL que fornecem suporte ao desenvolvimento de aplicações sensíveis ao contexto

Os módulos da linguagem NCL são agrupados em Áreas Funcionais, que definem os elementos da linguagem e suas funcionalidades. Segundo (ABNT NBR 15606-2, 2007) as áreas funcionais da linguagem NCL são: *Structure*, *Layout*, *Components*, *Interfaces*, *Presentation Specification*, *Linking*, *Connectors*, *Presentation Control*, *Timing*, *Reuse*, *Navigational Key*, *Animation*, *Transition Effects*, *Meta-Information*.

Conforme observado em trabalhos preliminares (VALE, MIELKE, *et al.*, 2010) (MIELKE, 2010), dentre estas áreas funcionais, quatro merecem destaque, pois apresentaram funcionalidades úteis a aplicações sensíveis ao contexto: *Components, Interfaces*, *Connectors* e *Linking*. As seções que seguem discutem cada um destes componentes.

6.2.1 Área Funcional *Components*

Um dos elementos especificados pela área funcional *Components* é o elemento mídia, que é a base de um documento NCL. Isto se deve ao fato de NCL atuar como linguagem de cola, relacionando no tempo e espaço diferentes objetos (*nós*) de mídia, criando o que se denomina de uma apresentação multimídia. Cada objeto de mídia geralmente possui, além de seu identificador, os atributos “src”, que define a URI do conteúdo do objeto (isto é, onde o objeto se encontra), “type”, que define o tipo do objeto (vídeo, áudio, texto, *script* NCLua, etc.) e um “descritor” que referencia o descritor (área funcional *Presentation Specification*) que define como a mídia será apresentada. O objeto (*nó*) de mídia ou de conteúdo é representado pelo elemento <media>. A Figura 37 mostra um exemplo de nó de mídia, cujo *id* é “video1”, que está localizada no caminho “media/video1.mpg” e que faz referencia ao descritor “dVideo1”. Conforme pode ser observado, o tipo da mídia é opcional, uma vez que alguns tipos formatos de arquivos já são pré-definidos em NCL. Dentre esses tipos estão os *scripts* NCLua (formato .lua).

```
<media id="video1" src="media/video1.mpg" descriptor="dVideo1" />
```

Figura 37. Exemplo de um objeto (nó) de mídia.

Em (VALE, MIELKE, *et al.*, 2010) e (MIELKE, 2010), pode ser observado que os elementos mídia têm papel importante no desenvolvimento de aplicações sensíveis ao contexto. Em geral, as entidades definidas na modelagem contextual da aplicação são representadas no código NCL como mídias do tipo NCLua e seus atributos e contextos intrínsecos são representados por âncoras de propriedade (Seção 6.2.2) dessas mídias. Desta forma, os *scripts* NCLua, são utilizados para atualizar os valores dessas âncoras de propriedade com as informações contextuais capturadas. As âncoras de propriedade são discutidas na Seção 6.2.2, a qual também mostra um exemplo de uma entidade representada em uma mídia, e seu contexto intrínseco por uma âncora de propriedade (Figura 39).

6.2.2 Área Funcional *Interfaces*

A área funcional *Interfaces* permite a definição de interfaces em *nós* de mídia que serão utilizadas em relacionamentos com outras interfaces de *nós* (ABNT NBR 15606-2, 2007). Como exemplo de interfaces, têm-se as *âncoras de propriedade*.

Uma *âncora de propriedade* define uma propriedade ou grupos de propriedades de um nó como interfaces que podem ser manipuladas pelos *links* (Seção 6.2.4). Por exemplo, um vídeo pode ter uma propriedade que é sua classificação, e o valor desta propriedade pode ser utilizado para verificar se será possível exibi-lo para um determinado usuário. Uma âncora de conteúdo é definida como um elemento <property> dentro do elemento <media>. A Figura 38 ilustra a âncora de propriedade de um vídeo, com uma propriedade "classificacao", cujo valor é igual "18".

```
<media id="video1" src="media/video1.mpg" descriptor="dVideo1">
  <property name="classificacao" value="18"/>
</media>
```

Figura 38. Exemplo de âncora de propriedade.

No caso de aplicações sensíveis ao contexto, (VALE, MIELKE, *et al.*, 2010) observou que a âncora de propriedade tem um papel importante na intermediação entre o dado obtido via um *script* NCLua (que captura, por exemplo, informações de sensores remotos) e seu uso pelo código NCL. Suponha que uma mídia NCLua fosse utilizada para representar a TV de João no documento NCL, conforme mostra a Figura 39. Nesta Figura, uma âncora de propriedade "TVStatus" é atualizada pelo *script* NCLua e seu valor pode ser utilizado na ativação de algum *link*.

```
<media id="TVJoao" src="TVJoao.lua" descriptor ="dTVJoao">
   <property name="TVSatus" value="off"/>
</media>
```

Figura 39. Âncoras de propriedade utilizadas em aplicações sensíveis ao contexto.

6.2.3 Área funcional *Connectors*

Na área funcional *Connectors* são encontrados os elementos responsáveis por estabelecer relações causais (chamados de conectores causais). Uma relação causal é definida por uma expressão de cola (*glue expression*), que define uma expressão de condição e uma de ação. Quando a expressão de condição é satisfeita, a expressão de ação deve ser obrigatoriamente executada.

Conectores não definem os *nós* que participam no processo, apenas são definidos papéis a serem instanciados por mídias específicas no *link*, favorecendo o reuso dos mesmos. A Figura 40 ilustra um conector que define o seguinte comportamento: "Ao iniciar, pare". Uma condição simples (*simpleCondition*) especifica a condição para que a ação simples (*simpleAction*) seja realizada. Cada condição ou ação é associada a um papel

(*role*). No elemento *link* são definidas as mídias que assumem esse papel, podendo existir diferentes *links* que fazem referencia a um mesmo conector.

```
<connectorBase>
 <causalConnector id="onBeginStop">
  <simpleCondition role="onBegin" />
  <simpleAction role="stop" />
 </causalConnector>
<connectorBase>
```

Figura 40. Exemplo de definição de um conector.

A área funcional *Connectors* define vários módulos associados à estrutura de um conector causal. A Figura 41 mostra os módulos (modelados como pacotes UML) especificados por esta área funcional e a estrutura associada a um conector causal (*causalConenctor*), que associa uma condição a uma ação, podendo conter parâmetros associados a esta associação (*connectorParam*).

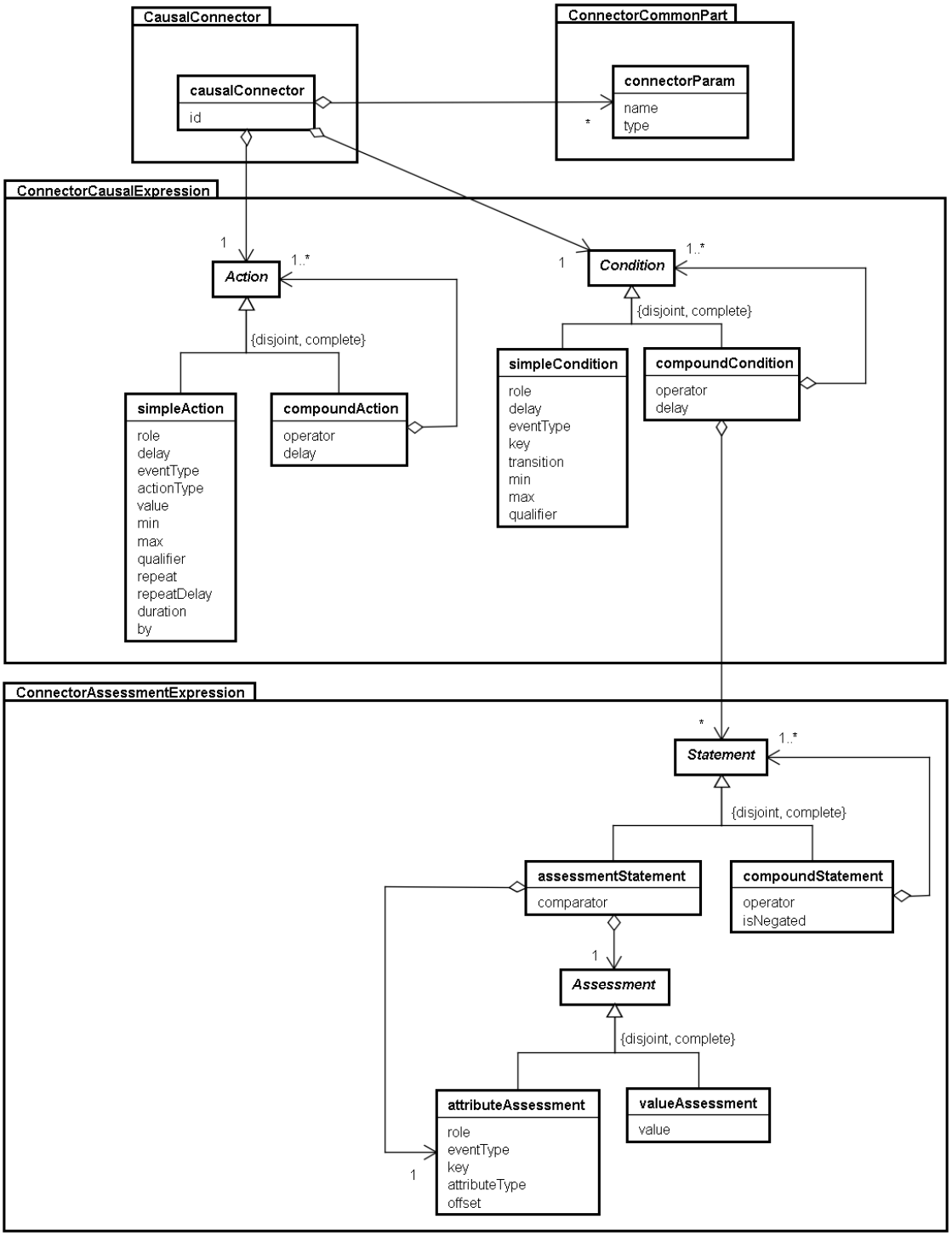


Figura 41. Estrutura associada a um conector causal (MIELKE, 2010).

A Figura 41 mostra que uma ação pode ser uma ação simples (*simpleAction*) ou uma ação composta (*compoundAction*), que agrupa ações simples por meio de um operador que especifica se as ações ocorrem sequencialmente ou em paralelo. Assim como as ações, as condições podem ser simples (*simpleCondition*) ou compostas (*compoundCondition*), associando condições simples por operadores lógicos. NCL possui um conjunto de nomes reservados para os papéis (*role*) associados às condições e ações. Os valores associados

aos papéis de condições e a ações são mostrados nas Tabela 7 e Tabela 8 respectivamente.

Tabela 7. Papéis associados a condições de um conector causal.

Papel (role)	Descrição
onBegin	Ativado no início da exibição de um conteúdo
onEnd	Ativado ao final da exibição de um conteúdo
onAbort	Ativado ao fim forçado da exibição de um conteúdo
onPause	Ativado ao pausar a exibição de um conteúdo
onResume	Ativado na volta da exibição de um conteúdo pausado
onSelection	Ativado quando alguma tecla é pressionada
onBeginAttribution	Ativado antes da atribuição de valores a uma âncora de propriedade
onEndAttribution	Ativado após a atribuição de valores a uma âncora de propriedade

Tabela 8. Papéis associados a ações de um conector causal.

Papel (role)	Descrição
Start	Inicia a exibição de um conteúdo
Stop	Finaliza a exibição de um conteúdo
Abort	Aborta a exibição de um conteúdo
Pause	Pausa a exibição de um conteúdo
Resume	Retorna a exibição de um conteúdo pausado
Set	Atribui um valor a uma âncora de propriedade

Uma condição composta de um conector causal pode conter uma expressão avaliadora simples (*assessmentStatement* da Figura 41) ou composta (*compountStatement*), utilizada para comparar uma variável associada a um papel de um conector a uma outra variável ou valor. Uma expressão avaliadora permite, por exemplo, acessar o valor de uma âncora de propriedade de um nó de mídia, ou verificar o estado associado uma âncora de conteúdo (ocorrendo, não ocorrendo, pausada), etc.

No caso de aplicações sensíveis ao contexto, o recurso de conectores causais pode atuar como mecanismo de reatividade, que responde à ocorrência de uma situação (ou situações) acionando alguma ação. Por exemplo, o conector da Figura 42 é utilizado para definir o seguinte comportamento: *“uma vez que* a TV estiver ligada, *faça* iniciar o filme”.

```
<causalConnector id=”onEndAttributionPropertyStart”>
   <compoundCondition operator="and">
```

```
            <simpleCondition role="onEndAttribution"/>
            <assessmentStatement comparator="eq">
               <attributeAssessment role="pTest"
eventType="attribution" attributeType="nodeProperty"/>
               <valueAssessment value="on"/>
            </assessmentStatement>
         </compoundCondition>
         <simpleAction role="start"/>
      </causalConnector>
```

Figura 42. Uso de um conectore para especificar o comportamento "uma vez que a TV estiver ligada, faça iniciar o filme".

A Figura 42 mostra um conector que define uma condição (*simpleCondition*) que verifica se o valor atribuído ao papél (*role*) pTest é igual a "on". Caso isso seja verdadeiro, a ação simples (*simpleAction*), cujo papél é "start", é acionada. As mídias que assumiram esses papéis são definidas no elemento *link*, como mostrado na Figura 43. Por exemplo, este elemento *link* define que a âncora de propriedade "TVStatus" da mídia da Figura 39 assume o papel (*role*) "pTest" especificado no conector da Figura 42 .

```
<link xconnector="onEndAttributionPropertyStart>
  <bind component="TVJoao" interface="TVStatus"
role="onEndAttribution"/>
  <bind component="TVJoao" interface="TVStatus" role="pTest" />
  <bind component="Movie1" role="start"/>
</link>
```

Figura 43. Uso de links em aplicações sensíveis ao contexto.

6.2.4 Área funcional *Linking*

Esta área funcional define possíveis relações entre nós de mídia em NCL. É com auxilio destes elementos que podem ser definidos comportamentos como: "Ao selecionar o botão verde do controle remoto, pause o vídeo principal e exiba uma imagem com instruções de ajuda"

O relacionamento entre nós é feito por meio de elos (elemento <link>) que associa uma relação causal (comportamento de causa e efeito) a um conjunto de nós. Uma relação causal é definida por um conector (Seção 6.2.3), que associa condições a serem avaliadas e ações realizadas caso as condições forem satisfeitas.

A Figura 44 mostra um exemplo de utilização de *links*. Neste exemplo, são associados papéis (*role*) de uma relação causal "ao iniciar, pare" (conector "onBeginStop" definido na Figura 40) a nós de conteúdo "vídeo1" e "video2". O elemento *link* da Figura 44 associa a

condição de início do nó de mídia “vídeo2” (condição “onBegin”) a ação de finalizar a exibição do nó de mídia “vídeo1” (ação “stop”).

```
<link xconnector="onBeginStop">
 <bind component="video2" role="onBegin"/>
 <bind component="video1" role="stop"/>
</link>
```

Figura 44. Exemplo de utilização do elemento *link*.

Em (VALE, MIELKE, *et al.*, 2010) e (MIELKE, 2010), foi verificado que o elemento *Link* em conjunto com os conectores (Seção 6.2.3) podem ser usados para implementar o comportamento reativo da aplicação sensível ao contexto. Nesse caso, os *links* associam mídias ou âncoras às relações causais que são definidas por um conector.

6.3 METAMODELO SIMPLIFICADO DE NCL[3]

Conforme mencionado, os módulos de NCL apresentados na Seção 6.2 são que se destacam por fornecerem suporte ao desenvolvimento de aplicações sensíveis ao contexto nessa linguagem. Existem, porém, outros módulos, agrupados ao longo das 14 áreas funcionais de NCL. Dessa forma, definiu-se um metamodelo parcial de NCL (Figura 45), que contém os módulos discutidos na Seção 6.2, além de alguns módulos de papel obrigatório na estrutura do documento NCL. Vale ressaltar que este metamodelo não considera os módulos de NCL que estão fora do escopo deste trabalho.

Conforme pode ser visto no metamodelo, um documento NCL é composto de um elemento *Head* e um *Body*. No elemento *Head*, podem ser definidas bases de regiões (*RegionBase*), de descritores (*DescriptorBase*) e conectores (*ConnectorBase*). Cada uma dessas bases podem conter a definição de várias regiões (*Region*), descritores (*Descriptor*) e conectores (*CausalConnector*), respectivamente. Os conectores e seus subelementos foram apresentados na Seção 6.2.3. As regiões são elementos que definem onde uma mídia vai ser apresentada na TV, enquanto que os descritores definem como essa mídia será apresentada nessa região. Conforme mencionado na Seção 6.2.1, uma mídia deve fazer uma referência a um descritor que, por sua vez, faz referência a alguma região.

[3] O metamodelo simplificado contém apenas os elementos utilizados pela transformação proposta por este trabalho. Em (LIMA, SOUSA e LOPES, 2007) pode ser encontrado um metamodelo mais completo de NCL.

No elemento *Body*, são definidas as mídias (*Media*), os *links* (*Link*) e as Portas (*Port*). As mídias e suas âncoras de propriedade (*PropertyAnchor*) são apresentadas nas Seções 6.2.1 e 6.2.2, respectivamente. Os *links* e seus relacionamentos com os conectores e mídias são apresentados na Seção 6.2.4. As portas sãos elementos que definem as mídias, ou âncoras de uma mídia, que devem ser ativadas assim que a aplicação NCL for iniciada.

Na etapa de realização do componente Monitor, elementos das regras ECA-DL TVD são transformados automaticamente em instâncias dos elementos do metamodelo de NCL. Portanto, cada um dos elementos definidos no metamodelo da Figura 45 tem papel importante na abordagem de deste trabalho para o desenvolvimento de aplicações sensíveis ao contexto na plataforma Ginga.

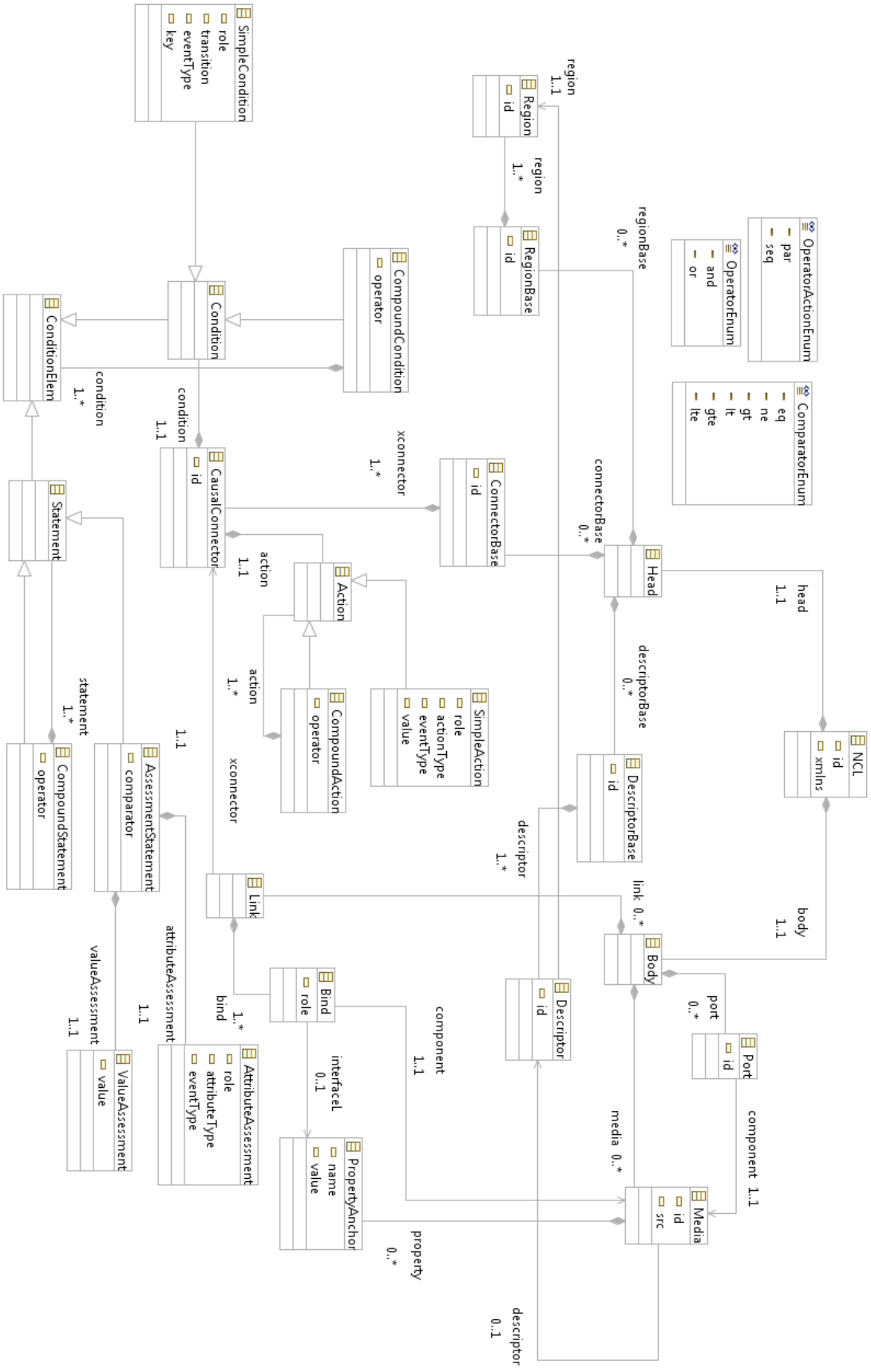


Figura 45. Metamodelo simplificado de NCL.

6.4 SCRIPTS NCLUA

Na Figura 36 foi visto como uma aplicação sensível ao contexto pode ser realizada na plataforma Ginga, considerando a arquitetura conceitual proposta no Capítulo 4. Conforme mencionado, os *scripts* NCLua podem atuar como um intermediador com o Canal de Interatividade para realização da comunicação entre a aplicação NCL a as Fontes de Contexto e de Situação. Além disso, os *scripts* NCLua também servem de intermediadores para a comunicação com os provedores de ações externos à plataforma Ginga. Portanto, os *scripts* geralmente são responsáveis por efetuar a troca de dados, processar informações e realizar comunicação com o documento NCL.

Conforme discutido na Seção 2.3.4, a comunicação entre os *scripts* NCLua e os demais componentes de uma aplicação NCL é feita por meio de troca de eventos. O mecanismo de difusão e recepção de eventos do ambiente de NCLua pode ser comparado ao padrão *publish/subscribe*, no qual há um elemento, ou um serviço, que gerencia a troca de eventos entre produtores e consumidores, gerando um desacoplamento espaço temporal entre estes (EUGSTER, FELBER, *et al.*, 2003).

Essa comunição entre os *scripts* NCLua e os demais componentes não faz distinção entre funcionalidades específicas de cada componente, fornecendo apenas a possibilidade de envio e recebimento de eventos. Esta estratégia de comunicação nem sempre facilita o trabalho do desenvolvedor, que geralmente deve construir mecanismos para gerenciar a comunicação, identificando os interessados em certos eventos.

Dessa forma, com o objetivo de auxiliar na distribuição e no tratamento de eventos na plataforma Ginga, em (MIELKE, 2010), são propostas duas bibliotecas NCLua, denominadas “TCPEventHandler” e “Properties”, que facilitam o acesso ao Canal de Interatividade e a comunicação entre *scripts* NCLua e documentos NCL, respectivamente. As bibliotecas propostas são genéricas e podem ser reutilizadas em diversos cenários de aplicação. A Figura 46 ilustra a arquitetura de uma aplicação NCL utilizando as bibliotecas.

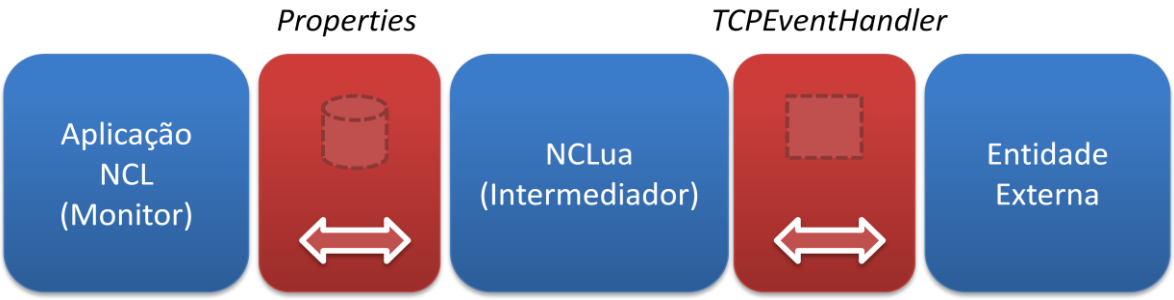


Figura 46. Aplicação sensível ao contexto na plataforma Ginga utilizando as bibliotecas NCLua.

A biblioteca “Properties” facilita a comunicação entre o documento NCL (Monitor) e os *scripts* NCLua. No mecanismo de tratamento de eventos padrão do Lua (Seção 2.3.5) não há uma correspondência direta entre as âncoras de propriedade representadas em um documento NCL e variáveis no *script* NCLua. A biblioteca “Properties” faz o controle das variáveis atribuídas, disponibilizando métodos para atribuir valores a âncoras de propriedades no documento NCL, obter o valor das propriedades armazenadas e responder à atribuição de variáveis, por meio de funções tratadoras.

Já a biblioteca “TCPEventHandler” torna transparente a comunicação entre o *script* e uma entidade externa (Fontes de Contexto/Situação e Provedores de Ação Externo), abstraindo as trocas de eventos com o Canal de Interatividade. A biblioteca adota conceitos de orientação a objetos para controlar as conexões e seus dados, visto que a mecanismo de tratamento de enventos padrão do Lua (discutido na Seção 2.3.5) não fornece esta facilidade. As funcionalidades de envio de eventos são disponibilizadas como métodos em um objeto que representa uma conexão (*Connection*) com um dispositivo remoto via Canal de Interatividade. Este objeto utiliza um tratador de eventos (*handler*) para responder à chegada de um evento de conexão, de desconexão, de dados e de erros. Para utilizar esta biblioteca, basta instanciar um objeto da classe *Connection*, passando como argumentos o *host* e porta que será efetuada a conexão e um tratador de eventos (*handler*) específico para esta conexão.

Neste trabalho, essas bibliotecas são usadas nos *scripts* NCLua implementados para os cenários de aplicação discutidos no Capítulo 8.

6.5 CONSIDERAÇÕES DO CAPÍTULO

Neste Capítulo foram discutidos os elementos de NCL que dão suporte ao desenvolvimento de aplicações sensíveis ao contexto no Ginga. As mídias NCL e suas âncoras de propriedade podem ser utilizadas para representar entidades e contextos, enquanto que o comportamento reativo da aplicação pode ser definido por meio de conectores e *links*.

Além disso, foi proposto um metamodelo parcial de NCL, contendo os elementos da linguagem que dão suporte ao desenvolvimento de aplicações sensíveis ao contexto. Esse metamodelo tem papel importante na abordagem MDD para geração automática de código NCL a partir de regras ECA-DL TVD, discutido no Capítulo 7.

Finalmente, também foi discutida neste capítudo a realização da arquitetura conceitual da aplicação sensível ao contexto na plataforma Ginga. A aplicação sensível ao contexto em NCL é composta por um documento NCL e *scripts* NCLua. O documento NCL é uma instância do componente Monitor, que se comunica com as entidades externas (Fontes de Contexto, de Situação, Provedores de Ação) através de *scripts* NCLua, utilizando o Canal de Interatividade. A comunicação entre *scripts* NCLua e documento NCL, e entre *scripts* e Canal de Interatividade, é realizada através das bibliotecas NCLua propostas por (MIELKE, 2010).

7 REALIZAÇÃO ORIENTADA A MODELOS DO MONITOR

Conforme discutido na Seção 4.2.1, o componente Monitor determina como deve ser o comportamento reativo da aplicação sensível ao contexto, i.e., o monitor tem a função de observar a ocorrência de determinadas situações contextuais, e invocar as ações e/ou serviços que forem definidos pelo desenvolvedor (através de regras ECA-DL TVD). Este capítulo descreve como o componente Monitor é realizado automaticamente a partir de regras ECA-DL TVD.

Conforme visto na Seção 4.3, a partir dos modelos conceituais e as regras ECA-DL TVD, o Monitor é realizado em um documento NCL seguindo um processo de transformação orientado a modelos. Um Modelo Fonte, no caso as regras ECA-DL TVD, irá gerar um Modelo Alvo, no caso, um modelo NCL, seguindo uma série de Atividades de Tranformação. O Modelo Fonte é uma instância do metamodelo da linguagem ECA-DL TVD, que é definido no Capítulo 5. O Modelo Alvo, por sua vez, é uma instância do metamodelo da linguagem NCL, que é definido Capítulo 6.

As Atividades de Transformação são determinadas pela Especificação da Transformação, que são definidas neste capítulo. Portanto, neste capítulo são identificadas as correspondências entre os elementos dos metamodelos de ECA-DL TVD e NCL. Nos exemplos ao longo do capítulo, já são mostrados os códigos NCL resultantes do processo de transformação, uma vez que, após obter o modelo NCL, o documento NCL (Monitor) é gerado com base na sintaxe da linguagem.

Esse capítulo está estruturado como segue. A Seção 7.1 discute a transformação das entidades e contextos intrínsecos. A Seção 7.2 mostra a transformação dos contextos relacionais. Na Seção 7.3 está a descrição da transformação das situações contextuais. A Seção 7.4 discute a transformação das cláusulas *Upon*, *When* e *Do* de uma regra ECA-DL TVD e, finalmente, Seção 7.5 traz as considerações do capítulo.

7.1 ENTIDADES, ATRIBUTOS E CONTEXTOS INTRÍNSECOS

Conforme visto na Seção 6.2.1, as entidades definidas pelo desenvolvedor na modelagem contextual da aplicação são representadas no código NCL como mídias do tipo NCLua e seus atributos e contextos intrínsecos são representados por âncoras de

propriedade (Seção 6.2.2) dessas mídias. Os *scripts* NCLua, por sua vez, são utilizados para atualizar os valores dessas âncoras de propriedade.

Assim, definiu-se que a mídia de NCL gerada para representar uma entidade e seus contextos intrínsecos tem como identificador único (*id*) o mesmo *id* da entidade e os nomes das âncoras de propriedade têm os nomes dos atributos ou dos contextos intrínsecos. Além disso, essa mídia se refere a um documento NCLua, que tem por finalidade atualizar os valores das âncoras de propriedade com, por exemplo, as informações contextuais capturadas por Fontes de Contexto.

Por exemplo, considere uma entidade TV, cujo *id* seja *TV1* e cujo contexto intrínseco seja seu estado atual (*TVstatus*) (entidades e contextos definidos no modelo da Figura 24, da Seção 5.1.2). O código NCL gerado para essas instâncias é mostrado na Figura 47. Nota-se também nesta figura que o *id* da mídia foi utilizado para compor os identificadores da região, do descritor e da porta.

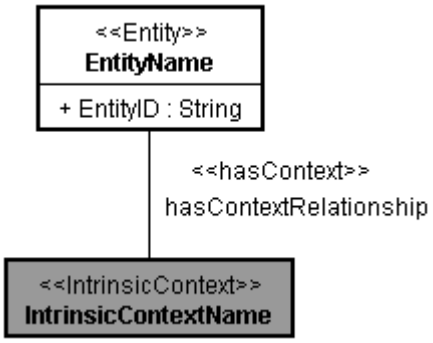
```
<?xml version="1.0" encoding="ISO-8859-1"?>
<ncl id="contextawareapp"
xmlns="http://www.ncl.org.br/NCL3.0/EDTVProfile">
<head>
<regionBase>
…
<region id="TV1Reg" />
…
</regionBase>
<descriptorBase>
…
<descriptor id="TV1Desc" region="TV1Reg" />
</descriptorBase>
…
</head>
<body>
…
<port id="p_TV1" component="TV1" />
…
<media id="TV1" src="scripts/TV1.lua" descriptor="TV1Desc">
       <property name="TVStatus" value=" " />
</media>
…
</body>
</ncl>
```

Mídia com identificador único “TV1”, que faz referencia ao script lua localizado em “scripts/TV1.lua”. Essa mídia define a propriedade “TVStatus”, cujo conteúdo é atualizado pelo *script* TV1.lua.

Figura 47. Mapeamento de Entidades e Contextos Intrínsecos para NCL.

A Tabela 9 resume a transformação de entidades, atributos e contextos intrínsecos. Na Tabela 9, *EntityID* se refere ao identificador único da entidade e *IntrinsicContextName* ao nome de um dos seus contextos intrínsecos. Contextos intrínsecos e atributos são mapeados da mesma forma.

Tabela 9. Transformação de Entidades e Contextos Intrínsecos.

Elemento(s) ECA-DL TVD	Elemento(s) NCL
	<media id="*EntityID*" src="scripts/ *EntityID*.lua descriptor="*EntityID*Desc" > <property name="*IntrinsicContexName*" value=" " /> </media>
	<descriptor id="*EntityID*Desc" region="*EntityID*Reg" />
	<region id="*EntityID*Reg" />
	<port id="p_*EntityID*" component="*EntityID*" />

7.2 CONTEXTOS RELACIONAIS

Para representar um contexto relacional, observou-se que é possível utilizar a mídia NCL. Para isso, o *id* do contexto relacional pode ser utilizado para compor o nome do contexto relacional, combinado como os identificadores das instâncias das entidades envolvidas, permitindo que existam outros contextos relacionais do mesmo tipo, porém, relacionando entidades diferentes. Além disso, essa mídia deve conter como âncoras de propriedades: (i) os nomes das classes das entidades envolvidas, cujos valores serão os identificadores das instâncias das respectivas classes; e (ii) uma propriedade *exists* que pode ser *true* ou *false*, indicando se aquele contexto relacional entre tais entidades existe. A mídia criada deve ser um *script* NCLua, que é responsável pela atualização da propriedade *exists*.

Por exemplo, considere um contexto relacional *Displaying* entre duas entidades (especificado no modelo da Figura 25, Seção 5.1.2), *TV* e *TVProgram*, cujas instâncias sejam identificadas por *TV1* e *TVProgram1*, respectivamente. O código NCL referente a este contexto relacional é mostrado na Figura 48.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<ncl id="contextawareapp"
xmlns="http://www.ncl.org.br/NCL3.0/EDTVProfile">
<head>
<regionBase>
…
<region id="Displaying_TV1_TVProgram1_Reg" />
…
</regionBase>
<descriptorBase>
…
```

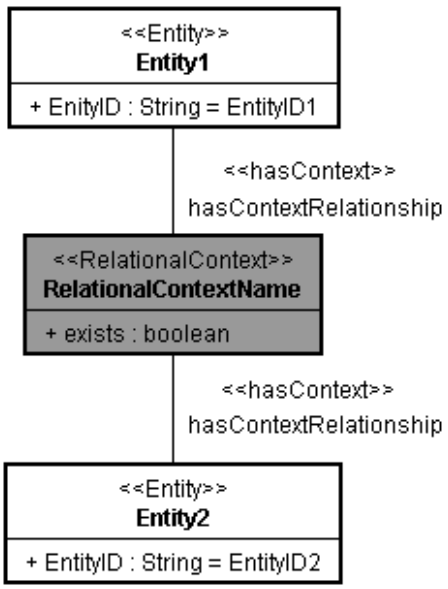
```
<descriptor id="Displaying_TV1_TVProgram1_Desc"
region="Displaying_TV1_TVProgram1_Reg" />
…
</descriptorBase>
</head>
<body>
…
<port id="p_ Displaying_TV1_TVProgram1" component="Displaying_TV1_TVProgram1" />
…
<media id="Displaying_TV1_TVProgram1"
src="scripts/Displaying_TV1_TVProgram1.lua"
descriptor="Displaying_TV1_TVProgram1_Desc">
      <property name="TV" value="TV1" />
      <property name="TVProgram" value="TVProgram1" />
      <property name="exists" value="false" />
</media>
…
</body>
</ncl>
```

Mídia com identificador único “Displaying_TV1_TVProgram1”, que faz referencia ao script lua localizado em “scripts/ Displaying_TV1_TVProgram1.lua”. Além das propriedades “TV” e “TVProgram”, essa mídia define a propriedade “exists”, com valor inicial “false”, cujo conteúdo é atualizado pelo *script*.

Figura 48. Mapeamento de Contexto Relacional para NCL.

A Tabela 10 generaliza o mapeamento de contextos relacionais: *RelationalContextName* se refere ao nome de um contexto relacional; *Entity1* e *Entity2* se referem aos tipos de entidades envolvidas no contexto relacional; *Exists* se refere à propriedade *exists* do contexto relacional e, finalmente, *EntityID1* e *EntityID2* são os identificadores das instâncias das entidades *Entity1* e *Entity2*, respectivamente.

Tabela 10. Transformação de Contextos Relacionais.

Elemento(s) ECA-DL TVD	Elemento(s) NCL
	<media id=”*RelationalContextName_Entity1_Entity2*” src=”scripts/*RelationalContextName_Entity1_Entity2*.lua descriptor=”*RelationalContextName_Entity1_Entity2*_Desc” > <property name=”*Entity1*” value=”EntityID1“ /> <property name=”*Entity2*” value=”EntityID2“ /> <property name=”*exists*” value=”false“ > </media>
	<descriptor id=”*RelationalContextName_Entity1_Entity2*_Desc” region=” *RelationalContextName_Entity1_Entity2*_Reg” />
	<region id=”*RelationalContextName_Entity1_Entity2*_Reg” />
	<port id=”p_ *RelationalContextName_Entity1_Entity2*” component=”*RelationalContextName_Entity1_Entity2*” />

7.3 SITUAÇÕES CONTEXTUAIS

A geração do código NCL para as situações contextuais é semelhante a dos contextos relacionais 7.2. Os identificadores das instâncias envolvidas na situação contextual são utilizados para compor o *id* da mídia NCL referente à situação contextual. As âncoras de propriedade também são semelhantes, com apenas uma diferença: ao invés de uma âncora *exists,* são definidas duas âncoras de propriedade *start* e *end*, que possuirão os valores *true* ou *false* a fim de identificar se existe um início e um fim para a dada situação. Por exemplo, considere uma situação, denominada *SituationTVOn* (especificada no modelo da Figura 26, Capítulo 5), envolvendo a entidade forno TV, cujo *id* é *TV1*. O código NCL referente à situação *SituationTVOn* é mostrado na Figura 49.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<ncl id="contextawareapp"
xmlns="http://www.ncl.org.br/NCL3.0/EDTVProfile">
<head>
<regionBase>
…
<region id="SituationTVOn_TV1_Reg" />
…
</regionBase>
<descriptorBase>
…
<descriptor id="SituationTVOn_TV1_Desc" region="SituationTVOn_TV1_Reg" />
…
</descriptorBase>
</head>
<body>
…
<port id="p_SituationTVOn_TV1" component="SituationTVOn_TV1" />
…
<media id="SituationTVOn_TV1" src="scripts/SituationTVOn_TV1.lua" descriptor="
SituationTVOn_TV1_Desc">
      <property name="start" value="false" />
      <property name="end" value="false" />
      <property name="TV" value="TV1" />
</media>
…
</body>
</ncl>
```

Mídia com identificador único “Situation_TVOn_TV1”. Essa mídia define as propriedades “start” e “end”, que indicam se a situação está ocorrendo ou não.

Figura 49. Mapeamento de Situações Contextuais para NCL.

A atualização das âncoras de propriedade *start* e *end* pode ser feita de duas maneiras: através de scripts NCLua associados à mídia NCL ou através de regras ECA-DL TVD. A detecção de situações através de scripts NCLua está fora do escopo das transformações uma vez que estes scripts devem ser fornecidos pelo desenvolvedor.

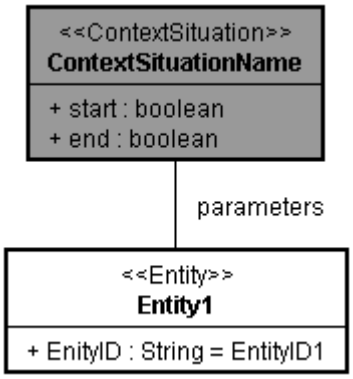
Usando regras ECA-DL TVD, as propriedades “start” e “end” são atualizadas quando as transições *EnterTrue* e *EnterFalse* são detectadas, respectivamente. Inicialmente, são

atribuídos valores iniciais *false* às propriedades *start* e *end*. Assim que a situação for detectada (i.e., ela passa a existir), a propriedade *start* é atualizada para “true” e *end* é atualizada como “false*”*. Da mesma forma, quando for detectado que a situação deixou de existir, é atribuído “true” à propriedade *end*. Desta maneira, pode-se verificar se uma determida situação não está ocorrendo e nunca ocorreu (propriedades *start* e *end* iguais à "false”), se é corrente (propriedade *start* igual à “true” e *end* igual à “false”), ou se é uma situação passada (propriedades *start* e *end* iguais à “true”).

No momento que o desenvolvedor estiver especificando as regras ECA-DL TVD que descrevem o comportamento reativo da aplicação, ele tem a possibilidade de definir regras ECA-DL TVD para descrever a detecção das transições das situações contextuais. Assim, como é descrito na Seção 7.4, serão gerados descritores e *links* responsáveis por atualizar as variáveis *start* e *end* da situação contextual.

A Tabela 11 generaliza a transformação de situações contextuais para elementos NCL.

Tabela 11. Transformação de Situações Contextuais.

Elemento(s) ECA-DL TVD	Elemento(s) NCL
	<media id=”*ContextSituationName_Entity1*” src=”scripts/*ContextSituationName_Entity1*.lua descriptor=”*ContextSituationName_Entity1*_Desc” > <property name=”*start*” value=”false“ /> <property name=”*end*” value=”false“ /> <property name=”*Entity1*” value=”EntityID1“ /> </media>
	<descriptor id=”*ContextSituationName_Entity1*_Desc” region=”*ContextSituationName_Entity1*_Reg” />
	<region id=”*ContextSituationName_Entity1*_Reg” />
	<port id=”p_*ContextSituationName_Entity1*” component=”*ContextSituationName_Entity1*” />

7.4 REGRA ECA-DL TVD

O comportamento reativo de aplicações sensíveis ao contexto, descrito por regras ECA-DL TVD, pode ser representado em um documento NCL por meio de conectores e links. Mais especificamente, cada regra ECA-DL TVD gera um conector e um link. As próximas seções detalham os mapeamentos de cada uma das cláusulas de uma regra ECA-DL TVD (Upon-When-Do).

7.4.1 Cláusula *Upon*

Conforme mostrado na Figura 31, a cláusula *Upon* define uma combinação de um ou mais eventos primitivos ou um evento de situação. Um evento primitivo é um evento específico da área de TVD. Já um evento de situação pode se referir ao momento em que uma situação contextual passa a existir (*EnterTrue*) ou ao momento quando deixa de existir (*EnterFalse*).

Evento Primitivo

Na Seção 5.3.2 apresenta os eventos primitivos de TVD atualmente suportados por ECA-DL TVD. Cada um deles possui uma correspondência com uma condição simples (*simpleCondition*) do conector causal (Seção 6.2.3) criado para a regra ECA-DL TVD. Na Tabela 12 são mostradas essas correspondências. O “X” em “uponX” representa um contador de cláusulas *Upon* ao longo das regras ECA-DL TVD presentes na aplicação sensível ao contexto. Isto gera um identificador único para os papéis (*role*) das condições simples criadas. Cada elemento *bind* do *link* gerado para a regra ECA-DL TVD deve fazer referência a um desses papéis, atribuindo as mídias aos devidos papéis. Assim, se tiverem duas condições simples ao longo do documento NCL gerado, o papel em uma será “upon1”, e na outra será “upon2”, permitindo que os links possam fazer as correspondências corretas.

Tabela 12. Correspondência Eventos de TVD e Elementos NCL.

Evento de TVD	Elemento NCL
onBegin	<simpleCondition role=”uponX” transition=”starts” eventType=”presentation” />
onEnd	<simpleCondition role=”uponX” transition=”stops” eventType=”presentation” />

onAbort	<simpleCondition	role=”uponX”	transition=”aborts” eventType=”presentation” />
onPause	<simpleCondition	role=uponX”	transition=”pauses” eventType=”presentation” />
onResume	<simpleCondition	role=”uponX”	transition=”resumes” eventType=”presentation” />
onSelection	<simpleCondition	role=”uponX”	transition=”starts” eventType=”selection” />
onBeginAttribution	<simpleCondition	role=”uponX”	transition=”starts” eventType=”attribution” />
onEndAttribution	<simpleCondition	role=”uponX”	transition=”stops” eventType=”attribution” />

Além do papel, o elemento *bind* deve indicar a mídia que desempenhará esse papel. Isso é obtido da relação *component* entre o evento de TVD e uma entidade contextual (entidades e contextos), conforme mostra a Figura 31. Em NCL, *component* será o ID da mídia gerada para essa entidade contextual, de acordo com as definições das Seções 7.1 e 7.2.

A Figura 50 mostra o código NCL gerado para seguinte cláusula Upon: *Upon onStart Movie1*.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<ncl id="contextawareapp" xmlns="http://www.ncl.org.br/NCL3.0/EDTVProfile">
<head>
…
<connectorBase>
      <causalConnector id="eca_dl_tvd_rule1">
            <simpleCondition role="upon1" transition="start"
eventType="presentation" />
      </causalConnector>
</connectorBase>
…
</head>
<body>
…
<link xconnector="eca_dl_tvd_rule1">
      <bind component="Movie1" role="upon1"/>
</link>
…
</body>
</ncl>
```

O elemento link instancia o conector causal (condição simples) e atribui a mídia “Movie1” ao papel “upon1”. Desta maneira, a condição do conector será verificada como verdadeira quando a mídia Movie1 começar a tocar.

Figura 50. Mapeamento de Eventos de TVD de uma regra ECA-DL TVD para NCL.

No caso de composição de eventos primitivos, a diferença é que as condições simples geradas para cada evento serão combinadas em um elemento do conector denominado condição combinada (*compoundCondition*). Por exemplo, a Figura 51 mostra o código NCL gerado para seguinte cláusula Upon: *Upon onStart movie1 and onEnd image1*.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<ncl id="contextawareapp" xmlns="http://www.ncl.org.br/NCL3.0/EDTVProfile">
<head>
…
<connectorBase>
      <causalConnector id="eca_dl_tvd_rule1">
            <compoundCondition operator="and"
                  <simpleCondition role="upon1" transition="start"
eventType="presentation" />
                  <simpleCondition role="upon2" transition="stops"
eventType="presentation" />
            </compoundCondition>
      </causalConnector>
</connectorBase>
…
</head>
<body>
…
<link xconnector="eca_dl_tvd_rule1">
      <bind component="movie1" role="upon1"/>
      <bind component="image1" role="upon2"/>
</link>
…
</body>
</ncl>
```

O operador “and” de NCL tem a mesma semântica do operador o mesmo utilizado na especificação da composição de eventos da cláusula Upon.

Figura 51. Mapeamento de Composição de Eventos de TVD de uma regra ECA-DL TVD para NCL.

Eventos de Situação

Conforme definido na Seção 7.3, uma situação contextual presente em uma regra ECA-DL TVD é transformada em uma mídia NCL, que contém, dentre outros elementos, duas âncoras de propriedade: *start* e *end*. Essas âncoras indicam se uma situação passou a existir e se deixou de existir, respectivamente. Uma situação passar a existir significa que ocorreu uma atribuição do valor *true* à sua âncora de propriedade *start*. Por outro lado, uma situação deixa de existir no momento em que sua âncora de propriedade *end* é atribuída do valor *true*. Por exemplo, a Figura 52 mostra o código NCL referente à seguinte cláusula Upon: *Upon EnterFalse (SituationTVOn (TV.TV1)).*

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<ncl id="contextawareapp"
xmlns="http://www.ncl.org.br/NCL3.0/EDTVProfile">
<head>
…
<connectorBase>
      <causalConnector id="eca_dl_tvd_rule1">
            <compoundCondition operator="and">
                  <simpleCondition role="upon1" transition="stops"
eventType="attribution" />
                  <assessmentStatement comparator="eq">
                        <attributeAssessment role="when1"
attributeType="nodeProperty" eventType="attribution"/>
                        <valueAssessment value="true"/>
                  </assessmentStatement>
            </compoundCondition>
```

Verifica se ocorreu um evento de atribuição e se valor “true” foi atribuído a uma âncora de propriedade associada ao papel “upon1”.

```
      </causalConnector>
</connectorBase>
…
</head>
<body>
…
<link xconnector="eca_dl_tvd_rule1">
      <bind component="SituationTVOn_TV1" interface="end" role="upon1"/>
      <bind component="SituationTVOn_TV1" interface="end" role="when1"/>
</link>
…
</body>
</ncl>
```

Figura 52. Mapeamento de um Evento de Situação de uma regra ECA-DL TVD para NCL.

Pode-se notar na Figura 52 que para o evento de situação é criado uma condição composta (*compoundCondition*). Essa condição composta contém uma condição simples (*simpleCondition*), que define o papel “upon1”, que verifica o fim de uma atribuição, e um *assessmentStatement* (Seção 6.2.3), que define o papel “when1” e verifica se uma propriedade é igual a *true*. A associação da situação e sua âncora de propriedade só ocorre no elemento *link*. Nesse caso, são atribuídos tanto ao papel “upon1”, quanto ao papel “when1” a mídia correspondente à situação contextual e sua âncora de propriedade “end” (uma vez que se trata de um evento *EnterTrue*).

No caso do evento de situação *EnterTrue*, a transformação é semelhante. O mesmo conector é utilizado, e a única alteração necessária é a atualização das *interfaces* do elemento *link* para a âncora de propriedade *start* da situação.

7.4.2 Cláusula *When*

Um tipo de cláusula *when* define uma operação condicional sobre algum atributo (ou contexto intrínseco) de uma entidade contextual, como definido no metamodelo da Figura 32. A avaliação de um atributo em NCL é feita pelo elemento *assessmentStatement* (Seção 6.2.3). Além disso, a cláusula *When* deve ser avaliada simultâneamente à ocorrência dos eventos da cláusula *Upon*. Portanto, necessita-se que os elementos *assessmentStatement* referentes à cláusula *When* sejam combinados com as condições simples (ou compostas) geradas pela cláusula *Upon* dentro de uma condição composta, cujo operador é “and”. Para o caso de condições aninhadas na cláusula *When*, basta aninhá-las também em NCL por meio do elemento *coumpoundStatement* (Seção 6.2.3).

Existem outros dois casos para uma cláusula *When* que se referem à especificação de condições que avaliam a ocorrência de situações contextuais. Pode-se, por exemplo,

especificar condições que avaliam se uma situação está ocorrendo ou não. Para verificar se uma determinada situação está ocorrendo, usa-se em NCL um *coumpoundStatement* com dois elementos *assessmentStatement*, um para verificar se a propriedade *start* é igual a *true* e outro para verificar se a propriedade *end* é igual a *false*. Para verificar se uma situação não está ocorrendo, devem ser utilizados: (i) um *coumpoundStatement* com dois elementos *assessmentStatement* para verificar se ambas as propriedades *start* e *end* são *true* (situação ocorreu no passado); (ii) um *coumpoundStatement* com outros dois *assessmentStatement* para avaliar se as propriedades *start* e *end* são ambos *false* (situação nunca ocorreu); e (iii) um *coumpoundStatement* para verificar se (i) ou (ii) são verdadeiros.

A Figura 53 mostra o código NCL correspondente à seguinte cláusula *when*: *When SituationTVOn (TV.TV1)*.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<ncl id="contextawareapp"
xmlns="http://www.ncl.org.br/NCL3.0/EDTVProfile">
<head>
…
<connectorBase>
      <causalConnector id="eca_dl_tvd_rule1">
            <compoundCondition operator="and">
                  …
                  <compoundStatement operator="and">
                        <assessmentStatement comparator="eq">
                              <attributeAssessment role="when1"
attributeType="nodeProperty" eventType="attribution"/>
                              <valueAssessment value="true"/>
                        </assessmentStatement>
                        <assessmentStatement comparator="eq">
                              <attributeAssessment role="when2"
attributeType="nodeProperty" eventType="attribution"/>
                              <valueAssessment value="false"/>
                        </assessmentStatement>
                  </compoundStatement>
                  …
            </compoundCondition>
      </causalConnector>
</connectorBase>
…
</head>
<body>
…
<link xconnector="eca_dl_tvd_rule1">
<bind component="SituationTVOn_TV1" interface="start" role="when1" />
<bind component="SituationTVOn_TV1" interface="end" role="when2" />
</link>
…
</body>
</ncl>
```

Esse elemento obriga que as condições referentes à cláusula *Upon* sejam verdadeiras simultaneamente às condições referentes à cláusula *When.*

Elemento compoundStatement que para avaliar se as âncoras de propriedade “start” e “end” possuem os valores “true” e “false”, respectivamente.

Figura 53. Mapeando de uma cláusula When que verifica se uma situação existe.

Na Figura 53 pode ser visto que o elemento *link* deve fazer a associação entre papéis do conector e as âncoras de propriedade da mídia referente à situação contextual.

7.4.3 Cláusula *Do*

Conforme definido na Seção 5.3.4, na cláusula *Do* podem ser especificados três tipos de serviços: serviços gerais, serviços de TVD e serviços de situação. Para cada um desses tipos de serviço, existe uma transformação definida, como descrito nas próximas seções.

Serviços Gerais

A transformação de um serviço geral para NCL consiste em gerar um elemento Ação Simples (*simpleAction*) para o conector da regra ECA-DL TVD e o elemento *bind* do *link* correspondente. A transformação deve verificar um repositório de ações, que define os serviços que podem ser traduzidos em um serviço de TVD (serviço interno). Caso não conste no repositório de ações, trata-se de um serviço externo.

No caso de serviços externos, uma mídia contendo um *script* NCLua é gerado, e a implementação é de responsabilidade do desenvolvedor. As bilbiotecas propostas por (MIELKE, 2010) e discutidas na Seção 6.4 podem ser utilizadas nessa implementação. A ação simples gerada pelo serviço externo tem como papel “actionX”, em que “X” é um contador de ações e o componente que será associado a esse papel será a mídia NCLua, sendo que o devido *script* possui o nome do serviço. Essa mídia tem como âncoras de propriedade os parâmetros do serviço definidos na regra ECA-DL TVD. A Figura 54 mostra o código NCL referente à seguinte cláusula Do: *Notify (TV.TV1, “A TV está ligada”)*.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<ncl id="contextawareapp"
xmlns="http://www.ncl.org.br/NCL3.0/EDTVProfile">
<head>
…
<regionBase>
…
<region id="NotifyReg" />
…
</regionBase>
…
<descriptorBase>
…
<descriptor id="NotifyDesc" region="NotifyReg" />
…
</descriptorBase>
…
<connectorBase>
      <causalConnector id="eca_dl_tvd_rule1">
```

```
…
            <simpleAction role="action1" actionType="start"
eventType="presentation"" />
…
      </causalConnector>
</connectorBase>
…
</head>
<body>
…
<media id="Notify" src="scripts/Notify.lua" descriptor="NotifyDesc">
      <property name="param1" value="TV1" />
      <property name="param2" value="A_TV esta_ligada" />
</media>
…
<link xconnector="eca_dl_tvd_rule1">
…
<bind component="Notify" role="action1" />
…
</link>
…
</body>
</ncl>
```

Ação de iniciar a mídia associada ao papel “action1”, no caso a mídia referente à ação Notify.

Figura 54. Mapeamento de uma cláusula Do com serviços gerais.

Serviços de TVD

A Seção 5.3.4 apresenta os serviços de TVD atualmente suportados por ECA-DL TVD. Cada um deles possui uma correspondência com uma ação simples (*simpleAction*) do conector causal (Seção 6.2.3) criado para a regra ECA-DL TVD. Na Tabela 13 são mostradas essas correspondências.

Tabela 13. Correspondência Serviços TVD e Elementos NCL.

Serviço de TVD	Elemento NCL
Start	<simpleAction role=”actionX” actionType=”start” eventType=”presentation” />
Stop	<simpleAction role=”actionX” actionType=”stop” eventType=”presentation” />
Abort	<simpleAction role=”actionX” actionType=”abort” eventType=”presentation” />
Pause	<simpleAction role=”actionX” actionType=”pause” eventType=”presentation” />
Resume	<simpleAction role=”actionX” actionType=”resume” eventType=”presentation” />
Set	<simpleAction role=”actionX” actionType=”start” eventType=”attribution” />

O “X” em “actionX” representa um contador de serviços na cláusula *Do* ao longo das regras ECA-DL TVD presentes no domumento NCL. Isto gera um identificador único para os papéis (*role*) das ações simples criadas. Cada elemento *bind* do *link* gerado para a regra ECA-DL TVD associa mídias a cada um desses papéis.

Além do papel, o elemento *bind* deve indicar a mídia que desempenhará esse papel, o que é feito pelo elemento *component*. No caso, *component* indica o ID da mídia gerada para a entidade contextual que é o parâmetro do serviço de TVD (metamodelo da Figura 34).

A Figura 55 mostra o código NCL gerado para seguinte cláusula Do: *Do Pause Movie1*.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<ncl id="contextawareapp"
xmlns="http://www.ncl.org.br/NCL3.0/EDTVProfile">
<head>
…
<regionBase>
…
<region id="Movie1Reg" />
…
</regionBase>
…
<descriptorBase>
…
<descriptor id="Movie1Desc" region="Movie1Reg" />
…
</descriptorBase>
…
<connectorBase>
      <causalConnector id="eca_dl_tvd_rule1">
…
            <simpleAction role="action1" actionType="pause"
eventType="presentation"" />
…
      </causalConnector>
</connectorBase>
…
</head>
<body>
…
<media id="Movie1" src="scripts/Movie1.mpg" descriptor="Movie1Desc" />
…
<link xconnector="eca_dl_tvd_rule1">
…
<bind component="Movie1" role="action1" />
…
</link>
…
</body>
</ncl>
```

Ação primitiva “pause” que se associa a uma mídia pelo papel “action1”.

Figura 55. Mapeamento de Serviços TVD de uma regra ECA-DL TVD para NCL.

No caso de composição de serviços ao longo da cláusula *Do*, as ações simples geradas para cada serviço são combinadas em um elemento do conector denominado ação

combinada (*compoundAction*). Por exemplo, a Figura 56 mostra o código NCL gerado para seguinte cláusula *Do*: *Do Pause (Movie1) and Start (Image1)*.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml version="1.0" encoding="ISO-8859-1"?>
<ncl id="contextawareapp"
xmlns="http://www.ncl.org.br/NCL3.0/EDTVProfile">
<head>
…
<regionBase>
…
<region id="Movie1Reg" />
<region id="Image1Reg" />
…
</regionBase>
…
<descriptorBase>
…
<descriptor id="Movie1Desc" region="Movie1Reg" />
<descriptor id="Image1Desc" region="Image1Reg" />
…
</descriptorBase>
…
<connectorBase>
      <causalConnector id="eca_dl_tvd_rule1">
…
            <simpleAction role="action1" actionType="pause"
eventType="presentation"" />
            <simpleAction role="action2" actionType="start"
eventType="presentation"" />
…
      </causalConnector>
</connectorBase>
…
</head>
<body>
…
<media id="Movie1" src="scripts/Movie1.mpg" descriptor="Movie1Desc" />
<media id="Image1" src="scripts/Image1.jpg" descriptor="Image1Desc" />
…
<link xconnector="eca_dl_tvd_rule1">
…
<bind component="Movie1" role="action1" />
<bind component="Image1" role="action2" />
…
</link>
…
</body>
</ncl>
```

Composição de ações primitivas “pause” e “start”, cujos respectivos papéis são “action1” e “action2”.

Figura 56. Mapeamento de vários serviços de uma regra ECA-DL TVD para NCL.

Na Figura 56 pode-se notar que o operador de composição pelo elemento *compoundAction* é “par” (paralelo), a fim de que as ações possam ser iniciadas simultaneamente.

Serviços de Situação

Conforme descrito na Seção 5.3.4, serviços de situação são utilizados para indicar que uma situação teve início ou que foi finalizada. Um serviço de situação se refere a uma situação contextual e tem como atributo *EnterTrue* ou *EnterFalse*, que define se a situação deve ser ativada ou desativada, respectivamente.

Em NCL, um serviço de situação é uma ação combinada composta de duas ações simples, as quais são responsáveis por atualizar as âncoras de propriedade de *start* e *end* da mídia referente a uma situação contextual. Por exemplo, a seguinte regra ECA-DL TVD define quando a situação *SituationTVOn* (Figura 26) terá início:

```
Upon OnEndAttribution (TV1.hasStatus)
When TV1.hasStatus = "On"
Do EnterTrue SituationTVOn
```

O código NCL referente a essa regra é mostrado na Figura 57.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<ncl id="contextawareapp" xmlns="http://www.ncl.org.br/NCL3.0/EDTVProfile">
<head>
…
<connectorBase>
            <causalConnector id="SituationTVOn_EnterTrue">
                  <compoundCondition operator="and">
                        <simpleCondition role="upon1" transition="stops"
eventType="attribution" />
                        <assessmentStatement comparator="eq">
                              <attributeAssessment role="when1"
attributeType="nodeProperty" eventType="attribution" />
                              <valueAssessment value="On"/>
                        </assessmentStatement>
                  </compoundCondition>
                  <simpleAction role="action1" actionType="start"
eventType="attribution" value="true" />
                  <simpleAction role="action2" actionType="start"
eventType="attribution" value="false" />
            </causalConnector></connectorBase>
…
</head>
<body>
…
<link xconnector="SituationTVOn_EnterTrue">
            <bind component="TV1" interface="TVStatus" role="upon1" />
            <bind component="TV1" interface="TVStatus" role="when1" />
            <bind component="SituationTVOn_TV1" interface="start"
role="action1" />
            <bind component="SituationTVOn_TV1" interface="end"
role="action2" />
</link>
…
</body>
</ncl>
```

Atribui os valores "true" e "false" às âncoras de propriedades associadas aos papéis "action1" e "action2", respectivamente.

Figura 57. Mapeamento de Serviços de Situação de uma regra ECA-DL TVD para NCL.

Na Figura 57 podem ser vistas as ações simples e os respectivos elementos *bind* que atualizam as âncoras de propriedade *start* e *end* com os valores *true* e *false*, respectivamente. Em geral, as regras ECA-DL TVD para detecção de situação verificam se uma determinada (ou conjunto de) âncora de propriedade é atualizada e, caso esse valor seja o desejado, aciona o serviço de situação.

7.4.4 Janela de Eventos

Na Seção 5.2.2 foi discutido o de avaliação de eventos dentro de uma regra ECA-DL TVD, no qual apenas a última ocorrência de um evento é considerada e após a avaliação de uma regra, o evento utilizado é “consumido”. Além disso, deve-se definir uma Janela de Detecção que determina o tempo máximo que um evento pode ficar disponível.

Como as transformações têm como alvo a linguagem NCL, o mecanismo de detecção de eventos é determinado pelo mecanismo de avaliação dos eventos das condições (Simples e Compostas) de NCL. Após o desenolvimento de exemplos de aplicações, observou-se que NCL também considera apenas a última ocorrência do evento, que são “consumidos” da mesma forma que em ECA-DL TVD.

Além disso, verificou-se que NCL não suporta a definição de uma janela de descarte de eventos e, portanto, quando o evento ocorre ele é considerado até que seja “consumido”, ou até que outro evento idêntico o substitua. Contudo, pode-se pensar, por exemplo, em definir como parâmetro do processo de transformação um tamanho de janela, que por sua vez é realizada utilizando conectores e links que têm por finalidade reiniciar os eventos de situação detectados. Atualmente, essa atividade não consta no processo de transformação definido neste trabalho.

7.5 CONSIDERAÇÕES DO CAPÍTULO

Neste capítulo foi vista a etapa de realização do elemento Monitor (apresentado arquitetura conceitual do Capítulo 4) na plataforma Ginga. Para essa etapa de transformação é necessário que o desenvolvedor já tenha produzido os modelos de contexto e situação, bem como especificado as regras ECA-DL TVD que definem o comportamento reativo da aplicação. As regras ECA-DL TVD e os modelos de contexto são então transformados em código NCL por uma série de regras de mapeamento, como

apresentado neste capítulo. Este processo de transformação foi automatizado utilizando técnicas de MDD e *framework* da plataforma Eclipse (THE ECLIPSE FOUNDATION, 2010).

Também foi visto neste capítulo que o código NCL gerado a partir da transformação faz uso de *scripts* NCLua. Não está no escopo deste trabalho a criação desses *scripts*, que representam a realização das fontes de contexto, fontes de situação e ações. Em (MIELKE, 2010), são especificadas bibliotecas que auxiliam o desenvolvimento desses *scripts*, as quais foram utilizadas nos cenários de aplicação implementados no estudo de caso dester trabalho (Capítulo 8).

8 ESTUDOS DE CASO

Este capítulo demonstra a viabilidade das propostas deste trabalho discutidas ao longo dos Capítulos 4 a 7 por meio de dois exemplos. Os cenários considerados para demonstração são os de uma *casa inteligente* e o de *política de controle de privacidade*.

Para cada um dos cenários, é apresentada uma descrição de alto nível da aplicação e em seguida são detalhadas as etapas do desenvolvimento de uma aplicação sensível ao contexto para a plataforma Ginga. As fases de análise e projeto incluem as seguintes atividades: (i) modelagem conceitual de contexto e situações, (ii) especificação do comportamento reativo da aplicação (em ECA-DL TVD) e, (iii) *design* estrutural da aplicação (utilizando a arquitetura conceitual proposta). Em seguida, na fase de implementação da aplicação, são gerados os códigos referentes à aplicação NCL através das transformações automáticas e do desenvolvimento dos *scripts* NCLua complementares.

Este capítulo está estruturado como segue. A Seção 8.1 mostra o cenário da casa inteligente, enquanto que a Seção 8.2 traz o cenário da política de controle de privacidade. Finalmente, a Seção 8.3 discute os cenários analisados.

8.1 CASA INTELIGENTE

O cenário da casa inteligente foi explorado em trabalhos publicados (VALE, GUAITOLINI e DOCKHORN COSTA, 2009) (VALE, MIELKE, *et al.*, 2010) e serviu como base para o desenvolvimento deste trabalho, pois evidenciou a necessidade de um mecanismo eficiente de realização das regras ECA-DL TVD em NCL. Agora, pretende-se desenvolvê-lo seguindo a metodologia proposta. O cenário da casa inteligente tem a seguinte história:

"*João gosta de assistir filmes na sua TV comendo pipoca. Geralmente, no começo do filme, ele deixa uma pipoca sendo preparada no seu forno microondas. Entretanto, João é muito distraído, e quando ele está vendo um bom filme, ele se esquece de pegar a pipoca assim que ela fica pronta. Quando ele se lembra, ela já está fria. Seria interessante que, quando estivesse vendo filme, João fosse notificado na sua TV que sua pipoca está pronta.*"

Para auxiliar João, uma aplicação sensível ao contexto é proposta. Esta aplicação monitora quando a pipoca está pronta no microondas e verifica se existe alguma TV exibindo um filme. Quando ambas as condições são verdadeiras, João é notificado em sua TV, por exemplo, por meio de uma mensagem exibida na tela, e o filme é pausado. Esta aplicação assume que o forno microondas e os televisores estão localizados na mesma casa, o que torna desnecessário modelar as relações que restringem os dispositivos a estarem na mesma casa.

8.1.1 Modelagem Contextual

Seguindo o processo de desenvolvimento, inicialmente é realizada a modelagem contextual do cenário proposto. A Figura 58 mostra o modelo de contexto que representa as entidades e contextos relevantes para a aplicação, que foram identificados analisando o cenário proposto.

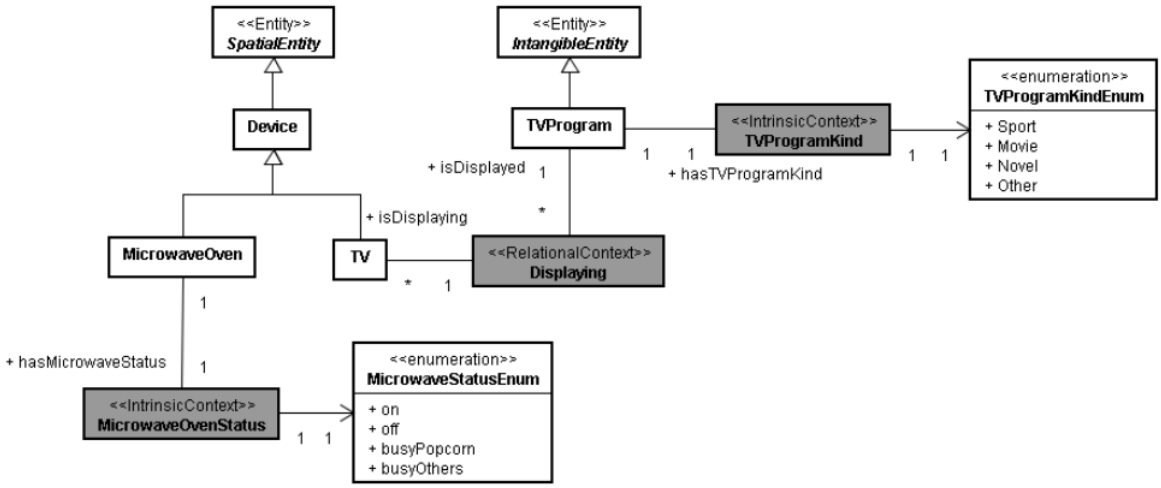


Figura 58. Modelo de Contexto do cenário Casa Inteligente.

Na Figura 58, foram modelados os tipos entidades espaciais (*SpatialEntity*) e entidades intangíveis (*IntangibleEntity*), que foram estereotipadas como Entidade (*Entity*) definida nas abstrações propostas por (DOCKHORN COSTA, 2007), as quais foram discutidas na Seção 5.1. As entidades espaciais do cenário analisado são os dispositivos: televisores e fornos microondas. Já as entidades intangíveis são objetos intangíveis como um programa de TV.

Como mencionado na Seção 5.1, contexto intrínseco define um tipo de contexto que é inerente a uma única entidade. Na Figura 58, os tipos de contexto intrínseco presentes são o *MicrowaveOvenStatus* e *TVProgramKind*, que são inerentes ao forno microondas e ao

programa de TV, respectivamente. Um forno microondas pode ter seu estado como: (i) "*on"*, que especifica que ele está disponível para preparar algo, (ii) "*off"*, que especifica que ele está desligado, (iii) "*busyPopcorn"*, que especifica que ele está preparando uma pipoca, e (iv) "*busyOthers"*, que especifica que ele está preparando algo diferente de pipoca. De forma semelhante, um programa de TV pode ser classificado como (i) um programa de esporte ("*Sport"*), (ii) um filme ("*Movie"*), (iii) uma novela ("*Novel"*), ou (iv) outro ("*Other*"), que é um programa diferente dos mencionados anteriormente. A Figura 58 também mostra o contexto relacional *Displaying*, que relaciona o programa de TV a uma coleção de televisores que estão exibindo este programa.

Continuando o processo de modelagem, determinadas situações de interesse para a aplicação são modeladas por meio dos modelos de situação. Analisando o cenário de aplicação, identificam-se duas situações que são de interesse para esse cenário: *SituationMakingPopcorn*, *SituationDisplayingMovie* **.**

O tipo situação *SituationMakingPopcorn* especifica a condição sob a qual um forno microondas está ocupado preparando pipoca. A Figura 59 mostra a especificação dessa situação usando a abordagem de modelagem apresentada na Seção 5.1.

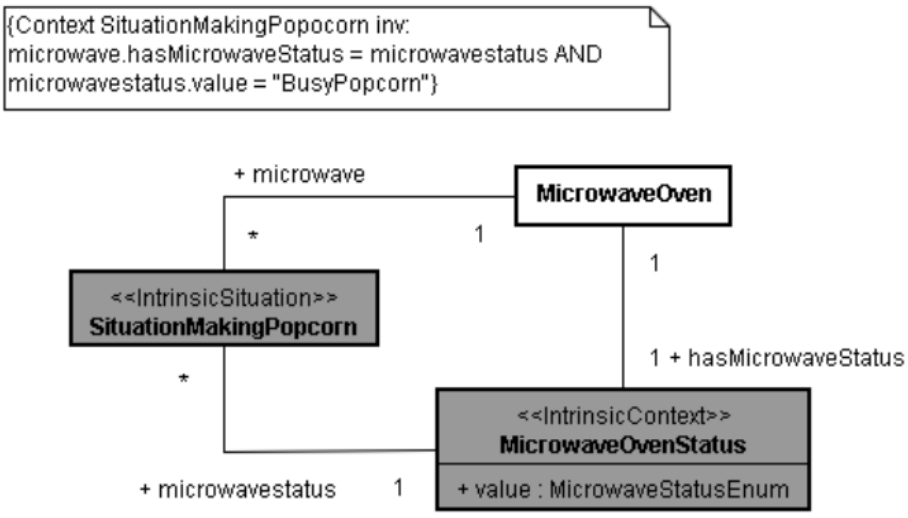


Figura 59. Situação *SituationMakingPopcorn*.

A situação *SituationMakingPopcorn* é composta de um objeto do tipo entidade *MicrowaveOven* e um objeto do tipo *MicrowaveOvenStatus*, que é o tipo contexto intrínseco associado a um forno microondas. Conforme visto na Seção 5.1, a invariante OCL define um predicado que deve permanecer verdadeiro para todas as instâncias de *SituationMakingPopcorn*. A cardinalidade também restringe as instâncias da situação, por exemplo, instâncias dessa situação devem estar associadas com uma instância de *MicrowaveOven* e uma instância de *MicrowaveOvenStatus*. A invariante OCL também

restringe que instâncias dessa situação devem ter o status do forno microondas como “busyPopcorn”. A palavra “context” na invariante OCL é uma primitiva reservada de OCL que define a classe para qual as restrições devem ser aplicadas.

A Figura 60 mostra a especificação da situação*SituationDisplayingMovie*, que detecta quando uma TV está exibindo um filme como seu programa de TV.

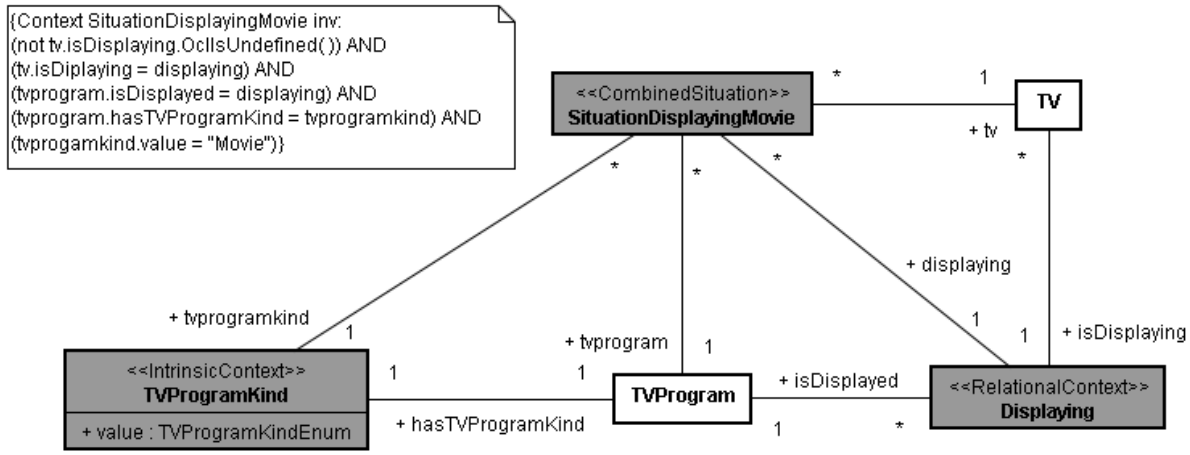


Figura 60. Situação *SituationDisplayingMovie*.

A situação *SituationDisplayingMovie* combina objetos dos tipos contexto intrínseco e relacional. Uma TV está exibindo um filme quando (i) exibe um programa de TV, que é representado pela existência do contexto relacional entre uma TV e um programa de TV, e (ii) este programa de TV é do tipo filme (“*Movie*”), que é detectado pelo contexto intrínseco *TVProgramKind* deste programa de TV. A verificação se a TV está exibindo um programa é feita pela operação OCL *oclIsUndefined()*, que é parte da biblioteca OCL padrão e testa se o valor de uma expressão é indefinido. Desta forma, a restrição “*not tv.isDisplaying.oclIsUndefined()*” detecta se uma instância do contexto relacional *Displaying* existe.

8.1.2 Especificação do Comportamento Reativo

O comportamento reativo da aplicação sensível ao contexto do cenário analisado pode ser descrito por uma regra ECA-DL TVD, que é mostrada na Figura 61.

```
Upon  EnterFalse (SituationMakingPopcorn(MicrowaveOven.MicrowaveOven1))
When  SituationDisplayingMovie (TV.TV1)
Do    Notify (TV.TV1, “A Pipoca está pronta!”), Pause (TV.TV1))
```

Figura 61. Regra ECA-DL TVD para o cenário Casa Inteligente.

Na Figura 61, o evento de interesse definido na cláusula *Upon* ocorre quando a situação *SituationMakingPopcorn* deixa de ocorrer, ou seja, quando a pipoca que estava sendo preparada no forno microondas de João estiver pronta. A condição da cláusula *When* especifica uma condição na qual a TV do João esteja exibindo um filme, ou seja, que a situação *SituationDisplayingMovie* esteja ocorrendo. Finalmente, os serviços que devem ser invocados quando a regra for válida são: notificar o usuário *(Notify (Person.John, "A Pipoca está pronta!"))* e pausar o filme *(PauseMovie (TV.TVJohn))*.

8.1.3 Estrutura Conceitual da Aplicação no Gerenciador de Contexto

Seguindo a metodologia para desenvolvimento da aplicação sensível ao contexto, o próximo passo consiste em estruturar a aplicação de acordo com os elementos conceituais definidos para o Gerenciador de Contexto, os quais foram definidos no Capítulo 4.

Fontes de Contexto e de Situação

De acordo com a modelagem contextual mostrada na Figura 58, pode-se identificar a necessidade de dois componentes do tipo *Fonte de Contexto*, uma externa para monitorar o estado do forno microondas (FC_1), e outra interna para monitorar o tipo de programa que está sendo exibido na TV (FC_2).

As informações sobre o estado do forno microondas e o tipo de programa exibido na TV se apresentam como condições contextuais primitivas, ou seja, podem ser capturadas (quase que) diretamente dos provedores de contexto. Contudo, a aplicação requer contextos mais complexos: a aplicação precisa saber se uma pipoca ficou pronta no forno microondas e se está sendo exibido filme na TV. Para inferir contextos mais complexos a partir das fontes contextuais apresentadas, são necessários dois componentes *Fontes de Situação*:

- FS_1, que utiliza as informações coletadas por FC_1 para determinar o término da situação modelada na Figura 59, ou seja, gerar um evento de que uma pipoca ficou pronta no forno microondas;

- FS_2, que utiliza as informações coletadas por FC_2 para determinar que a situação modelada na Figura 60 está ocorrendo, ou seja, existe um filme sendo exibido na TV.

Monitores

O papel do monitor é controlar o comportamento da aplicação sensível ao contexto, que, neste exemplo, foi definido pela regra ECA-DL TVD, mostrada na Figura 61. Como apenas uma regra ECA-DL foi criada, necessita-se apenas de um componente *Monitor (M_1)*. De acordo com a regra definida, M_1 precisa aguardar a ocorrência do evento gerado pela Fonte de Situação FS_1. Uma vez que o evento ocorreu, o monitor deve verificar se a situação monitorada pela Fonte de Situação FS_2 está ocorrendo. Se o evento ocorreu, e a situação está ocorrendo, M_1 deve acionar duas ações: pausar o filme e exibir a mensagem na tela da TV.

Ações

De acordo com a regra especificada para a aplicação sensível ao contexto, são invocadas duas ações quando a pipoca fica pronta enquanto um filme é exibido na TV. Como essas ações são voltadas para o próprio aparelho que contém o *middleware* Ginga, serão necessários dois *Provedores de Ação Interno*:

- PA_1, que oferece o serviço de pausar o filme na TV;
- PA_2, que oferece o serviço de exibir a mensagem na tela.

Como não existe a necessidade de composição de ações (as ação anteriores são independentes), não há necessidade de um elemento *Resolvedor de Ação*.

A Figura 62 ilustra no Gerenciador de Contexto os componentes identificados para o cenário de aplicação.

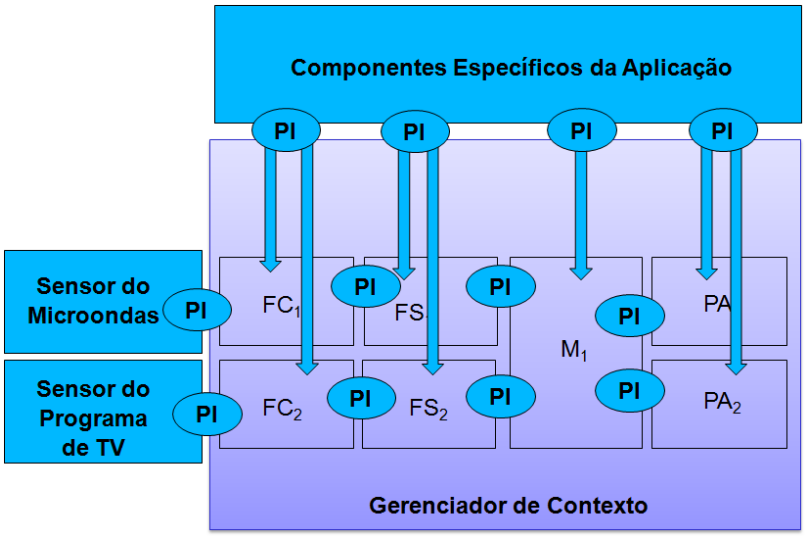


Figura 62. Arquitetura do cenário Casa Inteligente utilizando a arquitetura proposta para o Gerenciador de Contexto.

8.1.4 Realização na Plataforma Ginga

A etapa final da abordagem de desenvolvimento consiste em realizar os elementos da arquitetura conceitual na plataforma Ginga. Como discutido na Seção 0, o monitor pode ser implementado como um documento NCL. *Scripts* NCLua fazem uma interface entre o documento NCL e as fontes de contexto. A ação de pausar o filme é invocada por uma ação simples de NCL. A ação de exibir a mensagem na tela é realizada por um *script* NCLua. No caso das fontes de situação, existe a possibilidade de tratá-la de forma semelhante às fontes de contexto e provedores de ação, ou integrá-las ao monitor, como parte do documento NCL.

As regras OCL que definem as situações da Figura 59 e da Figura 60 não apresentam operações matemáticas mais complexas, sendo realizadas apenas por comparações simples entre um atributo e um valor. Portanto, é possível criar regras ECA-DL TVD de ativação e desativação das respectivas situações (dentro do próprio Monitor), não necessitando assim de componentes externos que realizem a detecção dessas situações.

As regras ECA-DL TVD mostradas na Figura 63, detectam a ocorrência, ou não, da situação *SituationMicrowaveOven*, que verifica se um microondas possui seu estado igual a ocupado com pipoca (“busyPopcorn”), e da situação *SituationDisplayingMovie*, que verifica se a TV está exibindo algum programa e esse programa é um filme.

```
Upon  onEndAttribution (MicrowaveOven1.hasMicrowaveStatus.value)
When  MicrowaveOven1.hasMicrowaveStatus.value = busyPopcorn
Do    EnterTrue SituationMakingPopcorn (MicrowaveOven1)

Upon  onEndAttribution (MicrowaveOven1.hasMicrowaveStatus.value)
When  MicrowaveOven1.hasMicrowaveStatus.value <> busyPopcorn
Do    EnterFalse SituationMakingPopcorn (MicrowaveOven1)

Upon  onEndAttribution (TV1.isDisplaying.exists) ||
      onEndAttribution (TVProgram1.hasTVProgramKind.value)
When  TV1.isDisplaying.exists = true and
      TVProgram1.hasTVProgramKind.value = movie
Do    EnterTrue SituationDisplayingMovie (TV1, TVProgram1)

Upon  onEndAttribution (TV1.isDisplaying.exists) ||
      onEndAttribution (TVProgram1.hasTVProgramKind.value)
When  TV1.isDisplaying.exists <> true or
```

```
      TVProgram1.hasTVProgramKind.value <> movie
Do    EnterFalse SituationDisplayingMovie (TV1, TVProgram1)
```

Figura 63. Regras ECA-DL TVD para ativação e desativação das situações *SituationMakingPopcorn* e *SituationDisplayingMovie*.

O passo seguinte no processo de realização dos modelos e das regras na plataforma Ginga é criar um modelo Ecore da aplicação sensível ao contexto, que é mostrado na Figura 64.

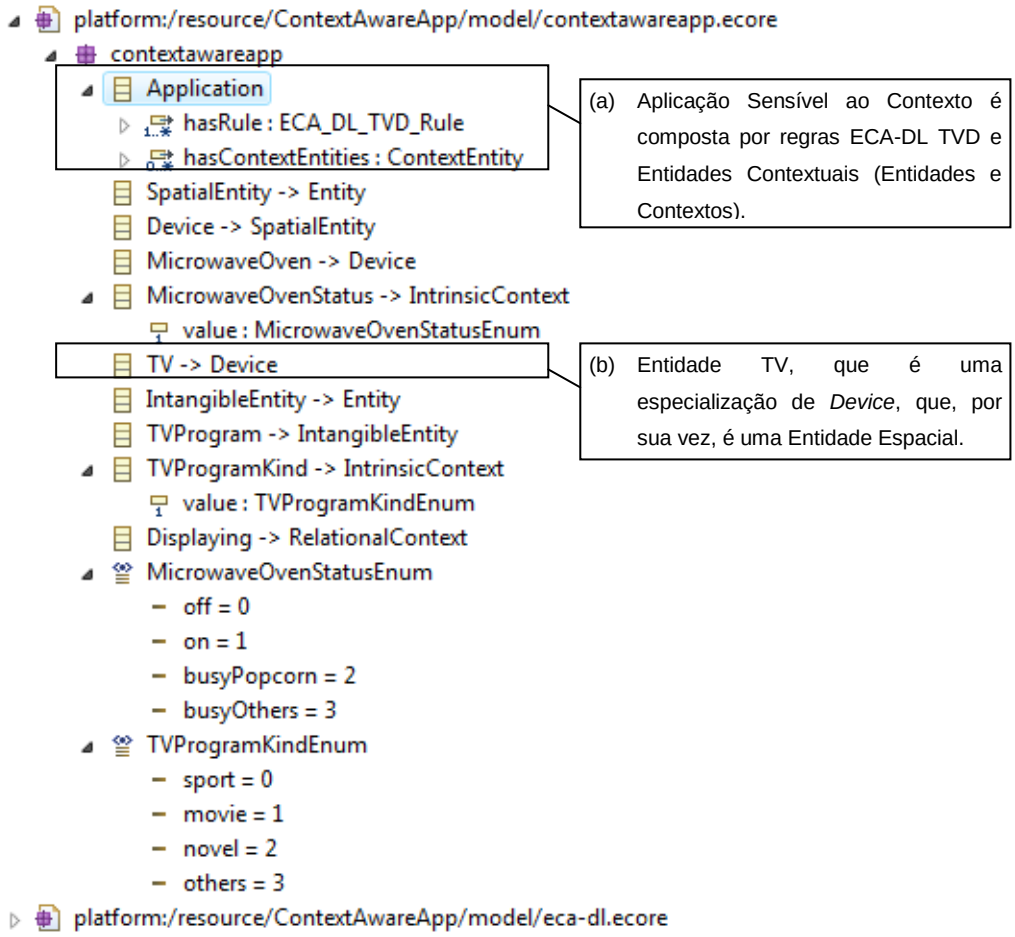


Figura 64. Modelo Ecore da aplicação sensível ao contexto do Cenário Casa Inteligente.

O modelo da Figura 64 define que uma aplicação sensível ao contexto contém regras ECA-DL TVD e Entidades Contextuais (Figura 64, parte (a)). Em seguida, são definidos os tipos de entidades e contextos aceitos por essa aplicação. Por exemplo, na Figura 64, parte (b), é destacada a entidade TV. As entidades e contextos aceitos por esse modelo são as mesmas definidas no modelo de contexto da Figura 58.

Uma vez criado o modelo Ecore da aplicação, para realizar a transformação deve-se criar um modelo da aplicação que reflete as características do cenário analisado. Nesse modelo devem constar as instâncias das entidades e contextos e, principalmente, a regra

ECA-DL TVD que descreve o comportamento reativo da aplicação. O modelo referente ao cenário analisado é mostrado na Figura 65.

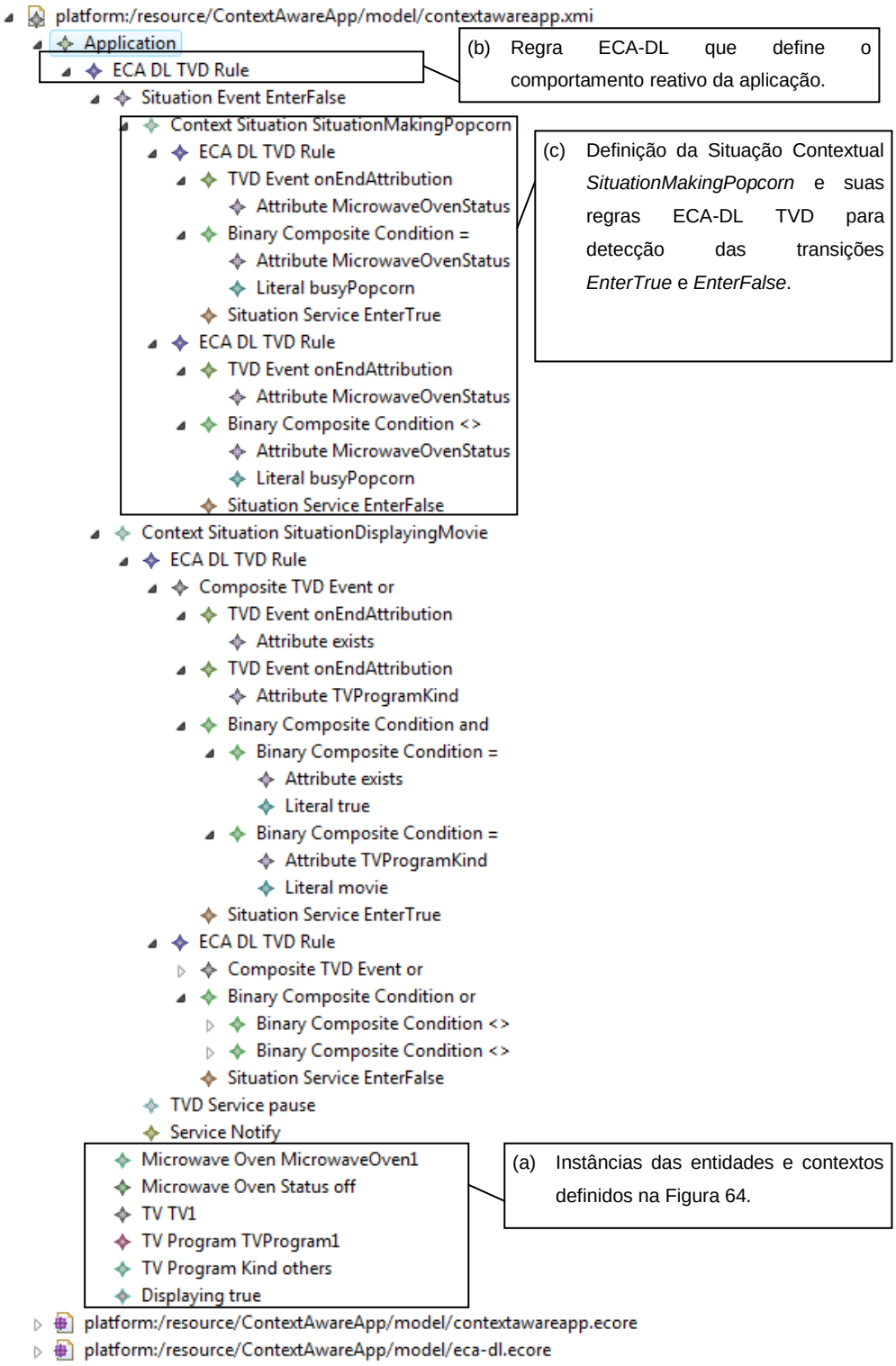


Figura 65. Modelo referente à aplicação sensível ao contexto do cenário Casa Inteligente.

Como pode ser visto na Figura 65, parte (a), foram criadas as instância das entidades *MicrowaveOven*, *TV* e *TVProgram*, cujos identificadores são *MicrowaveOven1*, *TV1* e

TVProgram1, respectivamente. Também na parte (a), foram criados instâncias dos contextos intrínsecos *MicrowaveOvenStatus* e *TVProgram,* e do contexto relacional *Displaying*, os quais se relacionam com as entidades criadas. Finalmente, foi criada a regra ECA-DL TVD (Figura 65, parte (b)) descrita na Figura 58, que utiliza as instâncias das entidades e contextos, e define as situações contextuais de interesse: *SituationMakingPopcorn* (Figura 65, parte (c)) e *SituationDisplayingMovie*. Essas situações possuem também as regras ECA-DL TVD responsáveis por suas ativações e desativações.

Uma vez definido o modelo da aplicação sensível ao contexto desejada, basta utilizá-lo como entrada para o processo de transformação. Seguem alguns trechos do código gerado: a mídia referente à entidade *MicrowaveOven1* é mostrada na Figura 66; os conectores e *links* para desativação da situação *SituationMakingPopcorn* são mostrados na Figura 67; e conector e o *link* referente à regra ECA-DL TVD são mostrados na Figura 68.

```
…
<media id="MicrowaveOven1" src="scripts/MicrowaveOven1.lua"
descriptor="MicrowaveOven1Desc">
      <property name="MicrowaveOvenStatus" value=" " />
</media>
…
```

Figura 66. Mídia NCL referente à entidade *MicrowaveOven1*.

```
…
<causalConnector id="SituationMakingPopcorn_MicrowaveOven1_EnterFalse">
      <compoundCondition operator="and">
            <simpleCondition role="upon4" transition="stops"
eventType="attribution" />
            <assessmentStatement comparator="ne">
                  <attributeAssessment role="when2"
attributeType="nodeProperty" eventType="attribution" />
                        <valueAssessment value="busyPopcorn"/>
            </assessmentStatement>
      </compoundCondition>
      <simpleAction role="action3" actionType="set"
eventType="attribution" value="true" />
</causalConnector>
…
<link xconnector="SituationMakingPopcorn_MicrowaveOven1_EnterFalse">
      <bind component="MicrowaveOven1" interface="microwaveovenstatus"
role="upon4" />
      <bind component="MicrowaveOven1" interface="microwaveovenstatus"
role="when2" />
      <bind component="SituationMakingPopcorn_MicrowaveOven1"
interface="end" role="action3" />
</link>
…
```

Figura 67. *SituationMakingPopcorn – EnterFalse*

```
…
<causalConnector id="eca_dl_tvd_rule1">
      <compoundCondition operator="and">
```

```
            <compoundCondition operator="and">
                  <simpleCondition role="upon1" />
                  <assessmentStatement comparator="eq">
                        <attributeAssessment role="upon2"
attributeType="nodeProperty" eventType="attribution" />
                        <valueAssessment value="true"/>
                  </assessmentStatement>
            </compoundCondition>
            <compoundStatement operator="and">
                  <assessmentStatement comparator="eq">
                        <attributeAssessment role="when7"
attributeType="nodeProperty" eventType="attribution" />
                        <valueAssessment value="true"/>
                  </assessmentStatement>
                  <assessmentStatement comparator="eq">
                        <attributeAssessment role="when8"
attributeType="nodeProperty" eventType="attribution" />
                        <valueAssessment value="false"/>
                  </assessmentStatement>
            </compoundStatement>
      </compoundCondition>
      <compoundAction operator="seq">
            <simpleAction role="action7" actionType="pause"
eventType="presentation" />
            <simpleAction role="action8" actionType="start"
eventType="presentation" />
      </compoundAction>
</causalConnector>
…
<link xconnector="eca_dl_tvd_rule1">
      <bind component="SituationMakingPopcorn_MicrowaveOven1"
interface="end" role="upon3" />
      <bind component="SituationMakingPopcorn_MicrowaveOven1"
interface="end" role="upon4" />
      <bind component="SituationDisplayingMovie_TV1_TVProgram1"
interface="start" role="when7" />
      <bind component="SituationDisplayingMovie_TV1_TVProgram1"
interface="end" role="when8" />
      <bind component="TV1" role="action7" />
      <bind component="Notify_TV1_A pipoca esta pronta!" role="action8" />
</link>
…
```

Figura 68. Conector e Link referente à regra ECA-DL TVD do cenário Casa Inteligente.

Atualmente, o processo de transformação não gera os *scripts* NCLua necessários automaticamente. Para a aplicação criada são necessários os scripts NCLua *MicrowaveOven1.lua*, *TV1.lua*, *TVProgram1.lua*, *Displaying_TV1_TVProgram1.lua*, *SituationMakingPopcorn_MicrowaveOven1.lua* e *SituationDisplayingMovie_TV1_TVProgram1.lua*, que foram criados à parte utilizando as bibliotecas propostas por (MIELKE, 2010). Por exemplo, o *script* NCLua MicrowaveOven1.lua que atualiza o estado do forno microondas é mostrado na Figura 69.

```
require 'microwave'
require 'Properties'
```

```
-- tratador de eventos de estado do microondas
handler = {}

-- quando receber novo estado: repassa os dados ao documento NCL
function handler:new_status(data)
    Properties.set('MicrowaveOvenStatus', data)
end

-- inicia o sensor
audience.MicrowaveSensor('192.168.102.1', 50000, handler)
```

Figura 69. *Script MicrowaveOven1.lua*.

8.2 POLÍTICA DE CONTROLE DE PRIVACIDADE

O segundo cenário considera uma aplicação de *home banking*, na qual os usuários podem verificar suas informações bancárias usando uma aplicação de TV interativa. O foco da aplicação é ocultar informações bancárias enquanto outras pessoas estiverem presentes Para isso, é proposta uma aplicação sensível ao contexto que detecta presença e automaticamente omite informações privadas quando mais de uma pessoa estiver na frente da TV. O cenário tem a seguinte história:

"João utiliza uma aplicação bancária web através da TV. Para verificar sua conta bancária na TV, ele usa o controle remoto para efetuar o *login* e realizar as tarefas bancárias habituais. Quando alguém entra na sala, a aplicação bancária oculta automaticamente informações privadas, tais como o seu saldo. Tudo volta ao normal quando João estiver sozinho na sala novamente."

8.2.1 Modelagem Contextual

Seguindo o processo de desenvolvimento, inicialmente é realizada a modelagem contextual do cenário proposto. A Figura 70 mostra a modelagem contextual que representa as entidades e contextos relevantes para a aplicação, que foram identificados analisando o cenário proposto.

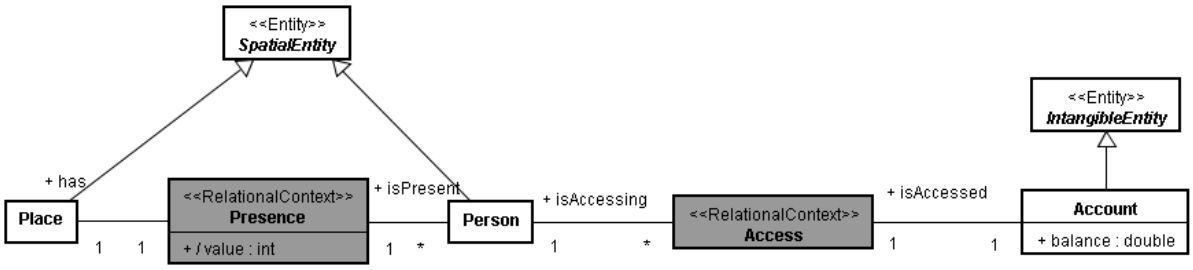


Figura 70. Tipos entidade e contexto relevantes do cenário Poítica de Controle de Privacidade.

As entidades espaciais são Lugar (*Place*) e Pessoa (*Person*), enquanto que Conta (*Account*) é uma entidade intangível. Além disso, também são mostrados os contextos relacionais Presença (*Presence*), que relaciona as pessoas presentes em um lugar, e Acesso (*Access*), que relaciona pessoas às contas que estão acessando.

Continuando o processo de modelagem, determinadas situações de interesse para a aplicação são especificadas por meio dos modelos de situação. Duas situações de interesse para esse cenário são: *SituationOnePersonInRoom*, *SituationNoPersonInRoom, SituationMoreThanOnePersonInRoom e SituationAccessing*.

O tipo situação *SituationOnePersonInRoom* especifica a condição na qual existe apenas uma pessoa na sala. A Figura 71 mostra a especificação dessa situação usando a abordagem de modelagem apresentada na Seção 5.1.

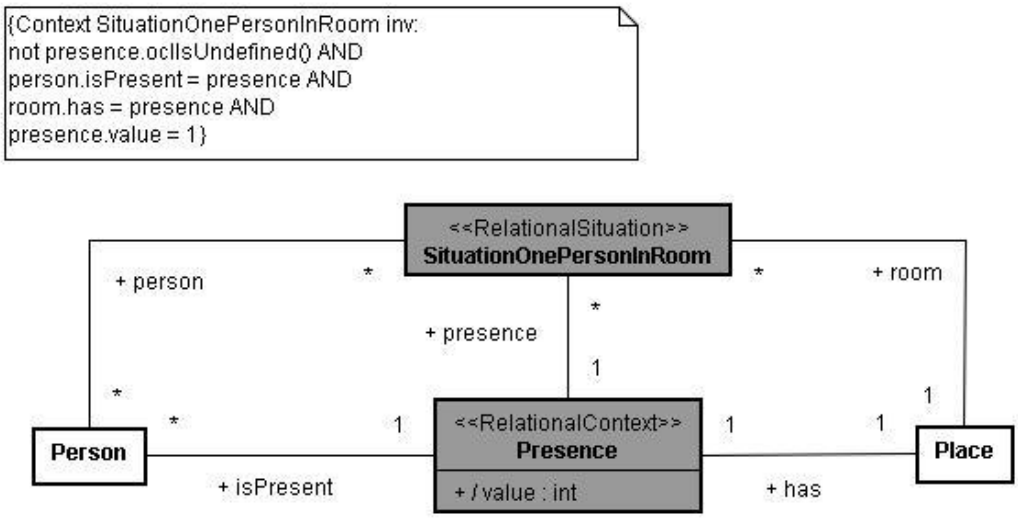


Figura 71. Situação *SituationOnePersonInRoom*.

A invariante OCL da Figura 71 define que lugar (Place) possui uma pessoa (Person) na sala quando (i) existe o contexto relacional *Presence* entre pessoa e lugar, e (ii) o número de pessoas nesse contexto relacional é exatamente “1”. As situações *SituationNoPersonInRoom* e *SituationMoreThanOnePersonInRoom* são definidas de forma

semelhante, sendo que a primeira possui o número de pessoas presentes igual a “0”, enquanto que a última possui o número de pessoas maior que “1”.

Finalmente, a situação *SituationAccessing* é mostrada na Figura 72, que indica que uma pessoa está acessando a conta. De acordo com a invariante OCL, esta situação ocorre quando os existe o contexto relacional *Access* entre os tipos *Person* e *Account*.

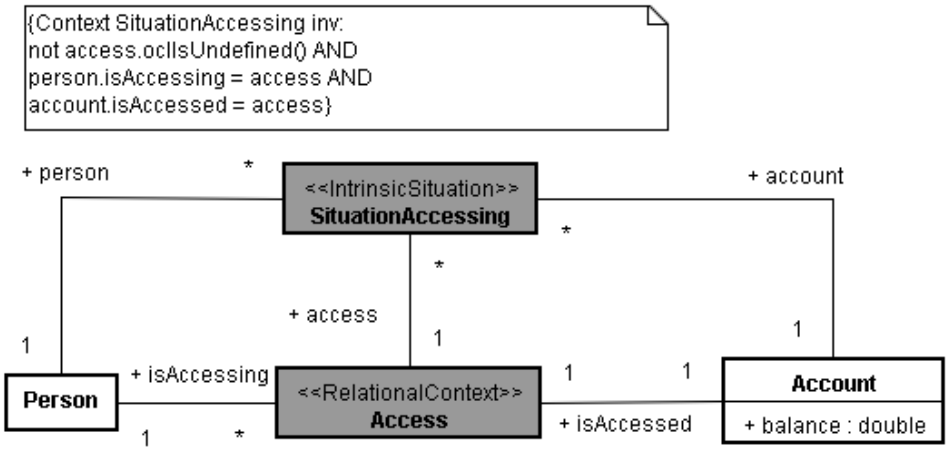


Figura 72. Situação *SituationAccessing*.

8.2.2 Especificação do Comportamento Reativo

O comportamento reativo da aplicação sensível ao contexto do cenário analisado pode ser descrito por cinco regras ECA-DL TVD, que são mostradas na Figura 73.

```
Upon EnterTrue SituationAccessing (person1)
When SituationOnePersonInRoom (room1)
Do startWithBalance(account1)

Upon EnterTrue SituationAccessing (person1)
When SituationMoreThanOnePersonInRoom (room1)
Do startWithoutBalance(account1)

Upon EnterTrue SituationOnePersonInRoom (room1)
Do showBalance(account1)

Upon EnterTrue SituationMoreThanOnePersonInRoom (room1)
Do hideBalance(account1)
```

```
Upon EnterTrue SituationNoPersonInRoom (room1)
Do logout(account1)
```

Figura 73. Regras ECA-DL TVD para o cenário Poítica de Controle de Privacidade.

A primeira regra é acionada quando o usuário efetua o *login* em sua conta bancária e só há uma pessoa neste momento na sala (provavelmente o próprio usuário). Esta regra resulta em mostrar as informações do saldo (comportamento "normal" da aplicação). Por outro lado, a segunda regra é acionada sempre que o usuário efetua o *login* em sua conta bancária e há mais de uma pessoa na sala. Esta regra resulta em esconder o saldo da conta (comportamento “adaptado” da aplicação). As outras duas regras garantem que o saldo da conta continue a ser mostrado ou oculto refletindo o número de pessoas na sala. Finalmente, quando não há mais pessoas na sala, qualquer atividade com a conta deve ser encerrada, o que é representado pela quinta regra.

8.2.3 Estrutura Conceitual da Aplicação no Gerenciador de Contexto

Seguindo a metodologia para desenvolvimento da aplicação sensível ao contexto, o próximo passo consiste em estruturar a aplicação de acordo com os elementos conceituais definidos para o Gerenciador de Contexto, os quais foram definidos no Capítulo 4.

Fontes de Contexto e de Situação

De acordo com a modelagem contextual mostrada na Figura 70, pode-se identificar a necessidade de dois componentes do tipo *Fonte de Contexto*, um para monitorar a presença na sala (FC_1), e outro para monitorar o se o usuário está acessando sua conta bancária (FC_2).

Com base nas informações capturadas pelas fontes de contexto a aplicação precisa saber existe uma, ou mais de uma, ou nenhuma pessoa na sala, e se a pessoas está acessando a conta bancária. Para inferir esses contextos mais complexos a partir das fontes contextuais apresentadas, são necessários dois componentes *Fontes de Situação*:

- FS_1, que utiliza as informações coletadas por FC_1 para determinar o que a situação modelada na Figura 71 está ocorrendo, ou seja, gerar um evento de que existe apenas uma pessoa na sala;

- FS_2, que utiliza as informações coletadas por FC_1 para gerar um evento de que existe nenhuma pessoa na sala;

- FS_3, que utiliza as informações coletadas por FC_1 gerar um evento que existe mais de uma pessoa na sala;

- FS_4, que utiliza as informações coletadas por FC_2 para determinar que a situação modelada na Figura 72 está ocorrendo, ou seja, uma pessoa está acessando os dados bancários.

Monitores

O papel dox monitores é controlar o comportamento da aplicação sensível ao contexto. Este comportamento foi definido pelas regras ECA-DL TVD, mostrada na Figura 73.

Como cinco regras ECA-DL foram criadas, necessita-se de cinco componentes *Monitores*:

- M_1: aguarda a ocorrência do evento gerado pela Fonte de Situação FS_4. Uma vez que o evento ocorreu, o monitor deve verificar se a situação monitorada pela Fonte de Situação FS_1 está ocorrendo. Se o evento ocorreu, e a situação está ocorrendo, M_1 deve acionar a ação: iniciar a tela mostrando o saldo na tela da TV.

- $M_{2:}$, aguarda a ocorrência do evento gerado pela Fonte de Situação FS_4. Uma vez que o evento ocorreu, o monitor deve verificar se a situação monitorada pela Fonte de Situação FS_3 está ocorrendo. Se o evento ocorreu, e a situação está ocorrendo, M_1 deve acionar a ação: iniciar a tela escondendo saldo na tela da TV.

- M_3: aguarda a ocorrência do evento gerado pela Fonte de Situação FS_1. Se o evento ocorreu, M_3 deve acionar a ação: mostrar saldo na tela da TV.

- $M_{4:}$ aguarda a ocorrência do evento gerado pela Fonte de Situação FS_3. Se o evento ocorreu, M_4 deve acionar a ação: esconder saldo na tela da TV.

- $M_{5:}$ aguarda a ocorrência do evento gerado pela Fonte de Situação FS_2. Se o evento ocorreu, M_5 deve acionar a ação: sair da aplicação.

Ações

São invocadas cinco ações diferentes e, portanto, são necessários três *Provedores de Ação Internos*:

- PA_1, que oferece o serviço de iniciar a tela;
- PA_2, que oferece o serviço de mostrar saldo;
- PA_3, que oferece o serviço de esconder o saldo;
- PA_4, que oferece o serviço de sair aplicação;

Como não existe a necessidade de composição de ações (as ação anteriores são independentes), não há necessidade de um elemento *Resolvedor de Ação*.

A Figura 74 ilustra no Gerenciador de Contexto alguns dos componentes identificados para o segundo cenário de aplicação. Por questões de simplicidade, são mostrados apenas os elementos necessários para os comportamentos pela primeira e pela terceira regra ECA-DL TVD da Figura 73.

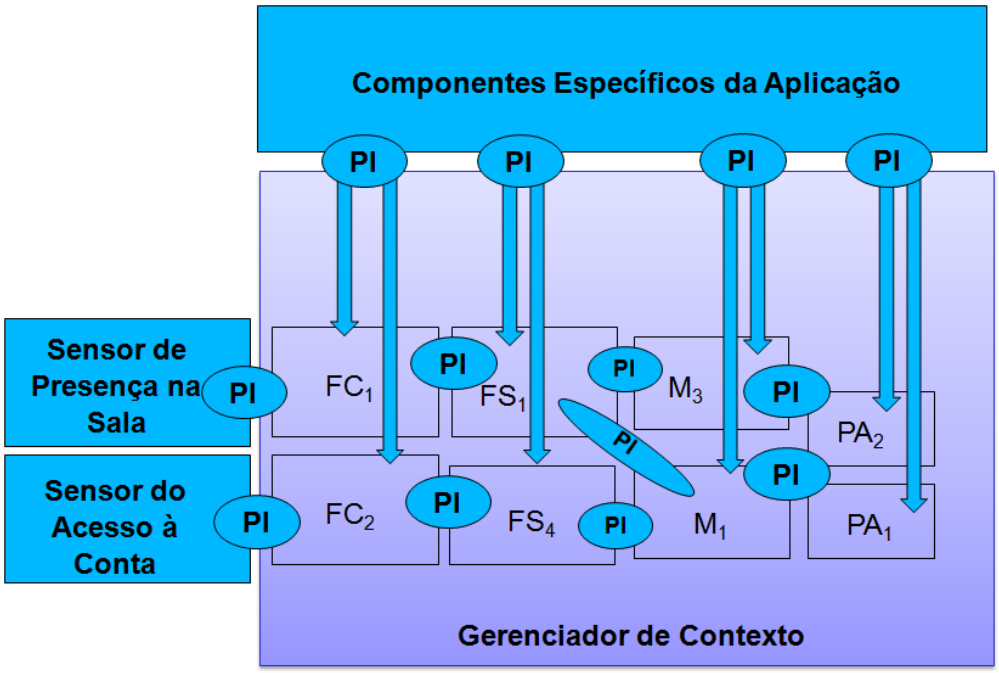


Figura 74. Arquitetura do cenário Poítica de Controle de Privacidade utilizando a arquitetura proposta para o Gerenciador de Contexto.

8.2.4 Realização na Plataforma Ginga

Novamente, como no cenário da Seção 8.1, os monitores são implementados como um único documento NCL. *Scripts* NCLua fazem a interface entre o documento NCL e as fontes de contexto. As ações previstas são realizadas como ações simples no documento

NCL. No caso das fontes de situação, existe a possibilidade de tratá-la de forma semelhante às fontes de contexto e provedores de ação, ou integrá-las ao monitor, como parte do documento NCL.

Assim como no cenário Casa Inteligente, as fontes de situação foram geradas a partir de regras ECA-DL TVD de ativação e desativação das respectivas situações. Essas regras são mostradas na Figura 75 e servem para detectara ocorrência, ou não, das situações *SituationOnePersonInRoom (*verifica se existe exatamente uma pessoa numa sala) e *SituationAccessing (*verifica se uma pessoa está acessando uma conta). As outras duas situações, *SituationNoPersonInRoom* e *SituationMoreThanOnePersonInRoom*, são definidas de forma semelhante à SituationOnePersonInRoom.

```
Upon onEndAttribution (room1.has.value)
When room1.has.value = 1
Do EnterTrue SituationOnePersonInRoom (room1)

Upon onEndAttribution (room1.has.value)
When room1.has.value <> 1
Do EnterFalse SituationOnePersonInRoom (room1)

Upon onEndAttribution (person1.isAccessing.exists)
When person1.isAccessing.exists = true
Do EnterTrue SituationAccessing (person1)

Upon onEndAttribution (person1.isAccessing.exists)
When person1.isAccessing.exists <> true
Do EnterFalse SituationAccessing (person1)
```

Figura 75. Regras ECA-DL TVD para ativação e desativação das situações *SituationOnePersonInRoom* e *SituationAccessing*.

O passo seguinte no processo de realização na plataforma Ginga é criar um modelo Ecore da aplicação sensível ao contexto, que é mostrado na Figura 76.

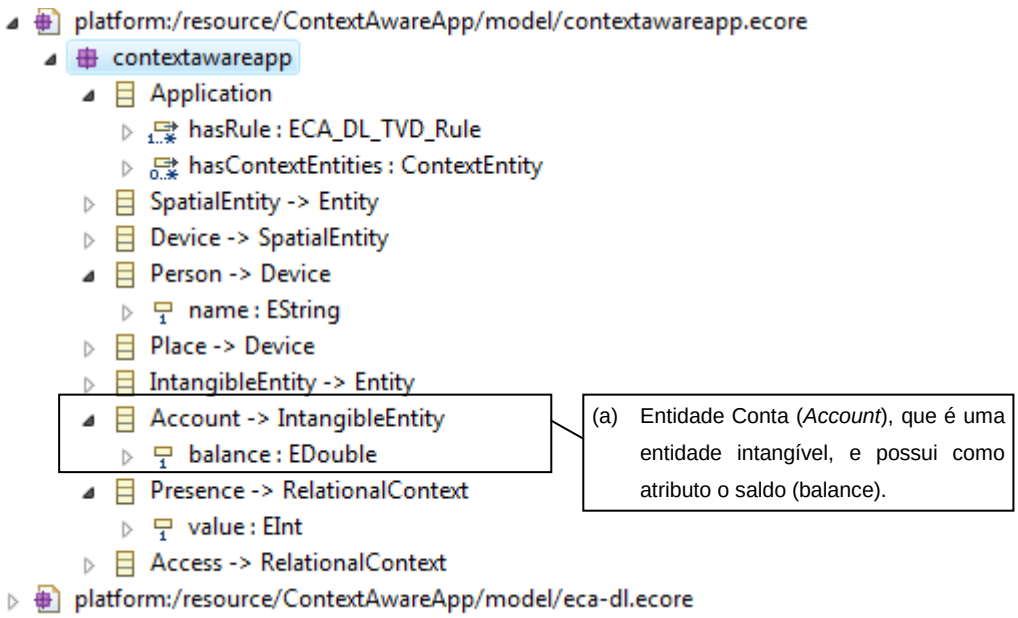


Figura 76. Modelo Ecore da aplicação sensível ao contexto do cenário Política de Controle de Privacidade.

O modelo da Figura 76 define que uma aplicação sensível ao contexto contém regras ECA-DL TVD e Entidades Contextuais. Em seguida, são definidas as entidades e contextos aceitos por essa aplicação, como uma forma de especialização ao metamodelo de ECA-DL TVD. Por exemplo, na Figura 76, parte (a), é destacada a entidade Conta (*Account*) e seu atributo saldo (*balance*). As entidades e contextos aceitos por esse modelo são as mesmas definidas no modelo de contexto da Figura 70.

Uma vez criado o modelo Ecore da aplicação, para realizar a transformação deve-se criar um modelo da aplicação que reflete as características do cenário analisado. Nesse modelo devem constar as instâncias das entidades e contextos e, principalmente, a regra ECA-DL TVD que descreve o comportamento reativo da aplicação. O modelo referente ao cenário analisado é mostrado na Figura 77. Por questões de simplicidade é destacada apenas a terceira regra ECA-DL TVD da Figura 73.

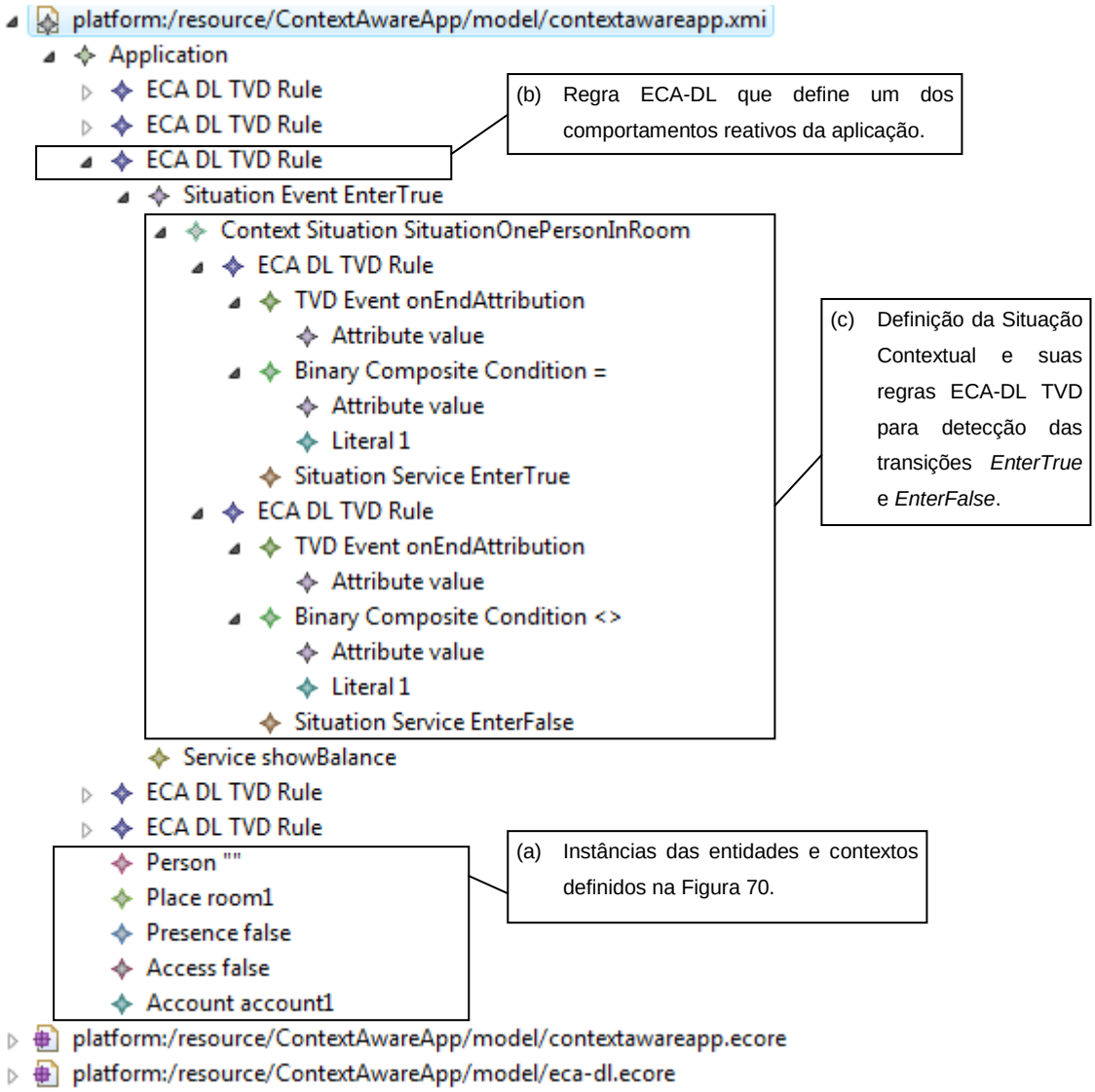


Figura 77. Modelo referente à aplicação sensível ao contexto do cenário Política de Controle de Privacidade.

Como pode ser visto na Figura 77, parte (a), foram criadas as instâncias das entidades *Person*, *Place* e *Account*, cujos identificadores são *person1*, *room1* e *account1*, respectivamente. Também na parte (a), foram criados instâncias dos contextos relacionais *Presence* e *Access*. Finalmente, foram criadas as cinco regras ECA-DL TVD descritas na Figura 73, que utilizam as instâncias das entidades e contextos. Na Figura 77, parte (b), é destacado o modelo referente à terceira regra da Figura 73, que traz também a definição da situação contextual *SituationMoreThanOnePersonInRoom* (Figura 77, parte (c)). Essa situação possui também as regras ECA-DL TVD responsáveis por sua ativação e desativação.

Uma vez definido o modelo da aplicação sensível ao contexto desejada, basta utilizá-lo como entrada para o processo de transformação. Seguem alguns trechos do código

gerado: a mídia referente ao contexto relacional *Presence* é mostrada na Figura 78; os conectores e *links* para desativação da situação *SituationMoreThanOnePerson* são mostrados na Figura 79; e conector e o *link* referente à terceira regra ECA-DL TVD da Figura 73 são mostrados na Figura 80.

```
…
<media id="Presence_person1_room1" src="scripts/Presence_person1_room1.lua"
descriptor="Presence_person1_room1_Desc">
      <property name="person" value="person1" />
      <property name="place" value="room1" />
      <property name="value" value=" " />
      <property name="exists" value="false" />
</media>
…
```

Figura 78. Mídia NCL referente ao contexto relacional *Presence*.

```
…
<causalConnector id="SituationMoreThanOnePersonInRoom_room1_EnterFalse">
      <compoundCondition operator="and">
            <simpleCondition role="upon10" transition="stops"
eventType="attribution" />
            <assessmentStatement comparator="lte">
                  <attributeAssessment role="when8"
attributeType="nodeProperty" eventType="attribution" />
                        <valueAssessment value="1"/>
                  </assessmentStatement>
      </compoundCondition>
      <simpleAction role="action11" actionType="set"
eventType="attribution" value="true" />
</causalConnector>
…
<link xconnector="SituationMoreThanOnePersonInRoom_room1_EnterFalse">
      <bind component="Presence_person1_room1" interface="value"
role="upon10" />
      <bind component="Presence_person1_room1" interface="value"
role="when8" />
      <bind component="SituationMoreThanOnePersonInRoom_room1"
interface="end" role="action11" />
</link>…
```

Figura 79. *SituationMoreThanOnePerson – EnterFalse*.

```
…
<causalConnector id="eca_dl_tvd_rule3">
      <compoundCondition operator="and">
            <simpleCondition role="upon11" />
            <assessmentStatement comparator="eq">
                  <attributeAssessment role="upon12"
attributeType="nodeProperty" eventType="attribution" />
                  <valueAssessment value="true"/>
            </assessmentStatement>
      </compoundCondition>
      <simpleAction role="action14" actionType="set"
eventType="attribution" value="true" />
</causalConnector>
…
<link xconnector="eca_dl_tvd_rule3">
      <bind component="SituationOnePersonInRoom_room1" interface="start"
```

```
role="upon11" />
      <bind component="SituationOnePersonInRoom_room1" interface="start"
role="upon12" />
      <bind component="account1" interface="showBalance" role="action14"/>
</link>
…
```

Figura 80. Conector e Link referente à terceira regra ECA-DL TVD do cenário Política de Controle de Privacidade.

Atualmente, o processo de transformação não gera os *scripts* NCLua necessários automaticamente. Por exemplo, o *script* NCLua Presence_person1_room1.lua, que faz a comunicação com o sensor de detecção de pessoas na sala e atualiza o valor corrente, é mostrado na Figura 81.

```
require 'audience'
require 'Properties'

-- tratador de eventos de audiencia
handler = {}

-- quando receber nova audiencia: repassa os dados ao documento NCL
function handler:new_audience(data)
    Properties.set('value', data)
end

-- inicia o sensor
audience.AudienceSensor('192.168.102.1', 50000, handler)
```

Figura 81. *Script* Presence_person1_room1.lua.

8.3 CONSIDERAÇÕES DO CAPÍTULO

Neste capítulo foram discutidos dois cenários de aplicação, os quais foram desenvolvidos com base na metodologia proposta por este trabalho. O desenvolvimento foi realizado desde a sua fase de modelagem contextual, seguido pela definição do comportamento reativo descrito pelas regras ECA-DL TVD, estruturação da aplicação na arquitetura conceitual do Gerenciador de Contexto do Ginga e, finalmente, a realização de componentes da arquitetura em NCL, utilizando para isso as regras de transformação definidas no Capítulo 7.

Foi verificado que a modelagem contextual foi capaz de descrever o universo de discurso da aplicação, ou seja, as entidades e contextos relevantes para a aplicação. Também observou-se que a linguagem ECA-DL TVD foi capaz de definir todos os comportamentos reativos das aplicações, além das transições de início e fim das situações contextuais necessárias para a aplicação. Além disso, com os cenários modelados e os

comportamentos reativos das aplicações descritos, a arquitetura conceitual para o Gerenciador de Contexto permitiu estruturar as aplicações no Ginga, identificando os componentes necessários à implementação das aplicações.

Finalmente, como parte final do processo de desenvolvimento, foi possível realizar elementos da arquitetura conceitual na plataforma Ginga, utilizando a abordagem orientada a modelos descrita no Capítulo 7. Foram gerados automaticamente os elementos Monitor, Fontes de Situação e Provedores de Ação no próprio documento NCL. As Fontes Contexto e alguns Provedores de Ação foram criadas à parte, por meio de scripts NCLua utilizando as bibliotecas propostas em (MIELKE, 2010). Assim, a metodologia proposta por esse trabalho mostrou-se viável para o desenvolvimento de aplicações sensíveis ao contexto no Ginga.

9 CONSIDERAÇÕES FINAIS

9.1 CONCLUSÕES

Este trabalho apresentou uma metodologia orientada a modelos para o desenvolvimento de aplicações sensíveis ao contexto na plataforma Ginga. A metodologia proposta define as etapas de desenvolvimento de uma aplicação sensível ao contexto desde a fase de análise à implementação na plataforma Ginga.

Para a fase de análise, na qual o desenvolvedor modela o universo de discurso da aplicação, a metodologia de desenvolvimento propõe a definição de modelos conceituais de contexto e situação que utilizam as abstrações de contexto proposta por (DOCKHORN COSTA, 2007) que são baseadas em ontologias de fundamentação (GUIZZARDI, 2005). Como base nos modelos de contexto e situação, o desenvolvedor deve especificar o comportamento reativo da aplicação sensível ao contexto a fim de definir as ações que a aplicação deve executar quando ocorrem mudanças de contexto. Para isso, este trabalho propôs a linguagem ECA-DL TVD, que é específica de domínio, e visa especificar comportamentos reativos de aplicações sensíveis ao contexto por meio de regras, conhecidas como regras ECA (Evento-Condição-Ação). ECA-DL TVD permite definir o comportamento reativo das aplicações através da especificação de eventos, condições e ações. Eventos modelam a ocorrência de mudanças relevantes no contexto do usuário. Condições representam a situação sob a qual as ações das regras são iniciadas, dado que os eventos ocorreram. Ações indicam as operações a serem invocadas quando a regra é aplicável. Além da definição dos elementos sintáticos e semânticos de ECA-DL TVD, este trabalho definiu o metamodelo da linguagem. Esse metamodelo permite descrever conceitualmente a linguagem de uma forma independente de plataforma e realizar modelos de regras ECA-DL TVD em uma linguagem alvo específica de plataforma por meio de regras de transformação de modelos.

Conforme visto, a linguagem ECA-DL TVD se mostrou adequada para descrever diversos cenários, em diferentes domínios como, por exemplo, o de integração de dispositivos numa casa inteligente e o cenário de controle de privacidade. Entretanto, existem casos nos quais ECA-DL TVD pode não ser adequada, como por exemplo, os cenários que requerem o uso de conjuntos de entidades ou expressões matemáticas complexas. Essas limitações (e possíveis soluções) foram discutidas na Seção 5.6.

Após a modelagem do universo de discurso da aplicação e da especificação do seu comportamento reativo, a metodologia sugere que o desenvolvedor estruture a arquitetura da aplicação com o suportede uma arquitetura genérica que foi proposta para o componente Gerenciador de Contexto do Ginga. Essa arquitetura provê serviços genéricos que apoiam as aplicações sensíveis ao contexto independentes do domínio da aplicação. Esses serviços podem ser combinados e configurados de forma a atender os requisitos específicos das aplicações.

Um componente de suma importância nessa arquitetura é o Monitor, que representa o comportameno reativo da aplicação sensível ao contexto. Dadas as mudanças de contexto detectadas pelas Fontes de Contexto, quando uma condição definida para comportamento do Monitor é satisfeita, são invocadas uma ou mais ações. Esse comportamento é definido por meio das regras ECA-DL TVD.

Na fase de implementação da aplicação, os elementos definidos na arquitetura conceitual, os modelos de contexto e as regras ECA-DL TVD devem ser realizados na plataforma Ginga. Este trabalho propôs um processo de transformação que realiza em NCL o componente Monitor, especificado através de regra ECA-DL TVD. Este processo foi implementado utilizando técnicas de MDD e *frameworks* de transformação da plataforma Eclipse (THE ECLIPSE FOUNDATION, 2010), sendo necessário especificar os metamodelos de ECA- TVD e NCL, além de uma série de regras de transformação, como discutido no Capítulo 7.

Considerando os requisitos levantados para arquitetura, após a sua realização nos estudos de caso, pode-se avaliar:

- *Flexibilidade e Extensibilidade.* A arquitetura mostrou-se adequada ao prover suporte necessário para criação e configuração de novas Fontes de Contexto e Situação, Monitores, Provedores de Ação de acordo com as regras ECA-DL TVD definidas. Além disso, esses componentes adquirem o comportanto ou procedimentos específicos de acordo com a aplicação. As Fontes de Contexto e Situação e os provedores de Ação são implementadas pelo desenvolvedor, enquanto o Monitor obtido diretamente a partir das regras ECA-DL TVD;

- *Inferência de Contexto e Situações.* Essas funcionalidades são atendidas pelos componentes Fontes de Contexto e de Situação, que se comunicam com o Monitor (documento NCL) via *scripts* NCLua;

- *Distribuição e Mobilidade.* Conforme visto, a arquitetura provê suporte a Fontes de Contexto Internas e Externas, assim como provedores de Ação Interno e Externo. Os componentes externos podem estar distribuídos e se movendo pelo ambiente, e se comunicam com o documento NCL (Monitor) via *scripts* NCLua (Ponto de Interação) através do Canal de Interatividade;

- *Rápido Desenvolvimento e Instalação de Aplicações.* Apesar de não ter sido feita uma avaliação formal com relação a esse requisito, pôde-se observar através dos estudos de caso que os esforços para desenvolvimento e manutenção de aplicações usando a abordagem proposta ficam reduzidos, se comparados aos esforços despendidos para se desenvolver e manter aplicações sem o suporte apropriado. Além disso, a implementação da aplicação que era derivada manualmente passa a ser derivada automaticamente a partir de um modelo em alto nível de abstração.

Enfim, essa metodologia de desenvolvimento de aplicação sensível ao contexto mostrou-se viável e prática. Essa abordagem fornece abstrações de alto nível para os desenvolvedores expressarem convenientemente o comportamento da aplicação em termos de eventos e ações. Pode-se afirmar que o principal desafio dessa abordagem reside em realizá-la na plataforma Ginga, uma vez que se deve transformar uma especificação baseada em regras, que é independente da plataforma, em uma implementação dependente de plataforma que não é orientada a regras, mas preservando a conformidade com os padrões da plataforma.

9.2 TRABALHOS FUTUROS

Como trabalhos futuros para estender a atual proposta de metodologia para desenvolvimento de aplicações sensíveis ao contexto na plataforma Ginga, podem ser citados:

- Implementar um serviço de repositório de fontes de contexto e ações utilizando o *framework* OSGi. O OSGi fornece funcionalidades de registro e procura de serviços. Essas características são importantes para garantir a possibilidade de aquisição de informações contextuais e/ou execução de ações por dispositivos heterogêneos independente da tecnologia de comunicação e de forma transparente para o usuário;

- Uma vez criado o repositório de serviços do OSGI, pode-se pensar em gerar automaticamente no processo de transformação de ECA-DL TVD para NCL também os *scripts* NCLua. A atividade de criação desses *scripts* ainda exige um conhecimento mais profundo da linguagem por parte do desenvolvedor, e a automatização da geração desses *scripts*, em que são invocados os serviços do OSGi, tornará o desenvolvimento ainda menos custoso;

- Aumentar o poder de expressividade de ECA-DL TVD, possibilitando a descrição de coleções de entidades e funções matemáticas. Como consequência, outro trabalho seria definir a transformação desses novos elementos para NCL;

- Incorporar aspectos de qualidade de contexto durante a aquisição de informações contextuais. As informações contextuais são relacionadas a parâmetros de qualidade de medida, que determina a qualidade da informação, ou seja, quão precisa, exata, recente é a informação. Determinar a qualidade do contexto é importante, pois as aplicações sensíveis ao contexto são extremamente dependentes da qualidade dos sensores e dos seus mecanismos para captura de contexto;

- Definir elementos da arquitetura conceitual que tratem de gerenciamento de componentes. Esses elementos gerenciadores podem, por exemplo, aprimorar o escalonamento de processamento entre diferentes componentes da aquitetura.

- Definir componentes da arquitetura conceitual que lida com o controle de privacidade. Componentes para controle de privacidade devem garantir a segurança da informação, uma atividade importante em se trantando de um aparelho de uso coletivo como a TV.

REFERÊNCIAS

ABNT NBR 15606-2. Televisão digital terrestre – Codificação de dados e especificações de transmissão para radiodifusão digital – Parte 2: Ginga-NCL para receptores fixos e móveis – Linguagem de aplicação XML para codificação de aplicações., Novembro 2007.

ABNT NBR15607-1. Televisão digital terrestre – Canal de interatividade Parte 1: Protocolos, interfaces físicas e interfaces de software, Maio 2008.

ALMEIDA, J. P. A. Model-Driven Design of Distributed Applications. **Centre for Telematics and Information Technology, University of Twente**, Enschede, The Netherlands, 2006.

ALMEIDA, J. P. A.; PIRES, L. F.; VAN SINDEREN, M. Costs and Benefits of Multiple Levels of Models in MDA Development. **2nd European Workshop on Model-Driven Architecture with Emphasis on Methodologies and Transformations**, Canterbury, UK, p. 12-20, 2004.

BRACKMANN, C. P. et al. GingaSC: Uma Proposta de Sensibilidade ao Contexto para TV Digital Brasileira. **Seminfo 2009**, Torres, 2009.

CRUZ, V. M.; MORENO, M. F.; SOARES, L. F. G. Ginga-NCL: Implementação de Referência para Dispositivos Portáteis. **Webmedia 2008**, p. 67-74, 2008.

CRUZ, V. M.; MORENO, M. F.; SOARES, L. F. G. **TV Digital Para Dispositivos Portáteis - Middlewares**. PUC. Rio de Janeiro. 2008. Projeto de Graduação.

DEY, A. K. Understanding and Using Context. **Personal and Ubiquitous Computing**, p. 4-7, 2001.

DEY, A. K.; ABOWD, G. D. Towards a Better Understanding of Context and Context-Awareness. **Proceedings of CHI 2000 - Workshop on The What, Who, Where, When, and How of Context-Awareness**, The Hague, Netherlands, 2000.

DOCKHORN COSTA, P. **Architectural Support for Context-Aware Applications - From Context Models to Services Platforms**. University of Twente. Enschede, The Netherlands. 2007. PhD thesis.

DOCKHORN COSTA, P.; FERREIRA PIRES, L.; SINDEREN, M. V. Architectural Patterns for Context-Aware Services Platforms. **Second International Workshop on Ubiquitous Computing**, Miami, p. 3-18, 2005.

EUGSTER, P. T. et al. The many faces of publish/subscribe. **ACM Computing Surveys (CSUR)**, v. 35, n. 2, p. 114-131, 2003.

FERNANDES, J.; LEMOS, G.; ELIAS, G. Introdução à Televisão Digital Interativa: Arquitetura, Protocolos, Padrões e Práticas. **Jornada de Atualização em Informática do Congresso da Sociedade Brasileira de Computação - JAI-SBC**, Salvador, 2004.

GOULARTE, R.; PIMENTEL, M. D. G. C.; MOREIRA, E. D. S. Context-aware support in structured documents for interactive-TV, v. 11, n. 4, p. 367-382, 2006.

GUIZZARDI, G. **Ontological Foundations for Structural Conceptual Models**. Centre for Telematics and Information Technology, University of Twente. Enschede, The Netherlands. 2005. PhD thesis.

HANSMANN, U. et al. **Pervasive Computing**. 2ª. ed. New York: Springer, 2003.

LABORATÓRIO TELEMÍDIA. Middleware Ginga TV – Interativa se faz com Ginga!, 2007. Disponivel em: <http://www.ginga.org.br>.

LEITE, L. E. C. et al. Uma Arquitetura de Serviço para Avaliação de Contextos em Redes de TV Digital. **25º Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC 2007)**, Belém, PA, p. 1015-1028, 2007. In Anais do 25º Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos.

LIMA, B.; SOUSA, J. G.; LOPES, D. Using MDA to Support Hypermedia Document Sharing. **Proceedings of the International Conference on Software Engineering Advances**, Cap Esterel, French Riviera, France, 2007.

LUA. Lua 5.1 Reference Manual, 2010. Disponivel em: <http://www.lua.org/manual/5.1/manual.html>. Acesso em: Junho 2010.

LUA.ORG. A Linguagem de Programação Lua, 2011. Disponivel em: <http://www.lua.org/portugues.html>.

MIELKE, I. T. **Desenvolvimento de Aplicações Sensíveis ao Contexto no Ambiente Declarativo do Sistema Brasileiro de TV Digital**. Universidade Federal do Espírito Santo. Vitória. 2010. Projeto de Graduação.

MOON, A. et al. Context-Aware Active Services in Ubiquitous Computing Environments, v. 29(2), p. 169-178, 2007.

MYLOPOULOS, J. Information Modeling in the Time of the Revolution. **Information Systems**, v. 23, n. 3-4, 1998.

NETO, F. D. C. A.; FERRAZ, C. A. G. Uma arquitetura para suporte ao desenvolvimento de aplicações sensíveis a contexto em cenário de convergência. **XXIV Simpósio Brasileiro de Redes de Computadores (SBRC 2006)**, Curitiba, Paraná, p. 39-49, 2006. In Anais do 24º Simpósio Brasileiro de Redes de Computadores.

OBJECT MANAGEMENT GROUP. Unified Modelling Language: Object Constraint, 2003.

OMG. UML® Resource Page. **Object Management Group**, 2010. Disponivel em: <http://www.uml.org/>. Acesso em: Junho 2010.

PESSOA, R. M. et al. Aplicação de um Middleware Sensível ao Contexto em um Sistema de Telemonitoramento de Pacientes Cardíacos. **XXXIII Seminário Integrado de Software e Hardware**, Campo Grande, MS, 2006. 32-46.

SALBER, D.; DEY, A. K.; ABOWD, G. D. The Context Toolkit: Aiding the Development of Context-Enabled Applications. **Conference on Human Factors in Computing Systems (CHI '99)**, Pittsburgh, PA, p. 434-441, 1999.

SANDIA LABS. Jess: the Rule Engine for the Java Platform, Jess website, 2008. Disponivel em: <http://herzberg.ca.sandia.gov/jess/>.

SANT'ANNA, F. et al. Desenvolvimento de Aplicações Declarativas para TV Digital no Middleware Ginga com Objetos Imperativos NCLua, 2010. Disponivel em: <http://www.ncl.org.br/documentos/MCNCLua.pdf>. Acesso em: Junho 2010.

SANT'ANNA, F.; CERQUEIRA, R.; SOARES, L. F. G. NCLua - Objetos Imperativos Lua na Linguagem Declarativa NCL. **Simpósio Brasileiro de Sistemas Multimídia e Hipermídia (Webmedia 2008)**, Vila Velha, v. 1, p. 83-90, 2008. In Anais do XIV Simpósio Brasileiro de Sistemas Multimídia e Hipermídia.

SBTVD. Sistema Brasileiro de TV Digital, 2009. Disponivel em: <http://sbtvd.cpqd.com.br/>.

SCHILIT, B. N.; ADAMS, N.; WANT, R. **Context-Aware Computing Applications**. WMCSA '94: Proceedings of the 1994 First Workshop on Mobile Computing Systems and Applications. Santa Cruz, California, USA: IEEE Computer Society. 1994. p. 85-90.

SCHMIDT, A.; MICHAEL BEIGL, B.; GELLERSEN, H.-W. There is more to Context than Location. **Computers and Graphics**, v. 23, n. 6, p. 893–901, 1999.

SERRAL, E.; VALDERAS, P.; PELECHANO, V. Towards the Model Driven Development of context-aware pervasive systems. **Pervasive and Mobile Computing**, v. 6, p. 254-280, 2010.

SILVA, F. S. D.; ALVES, L. G. P.; BRESSAN, G. PersonalTVware: A Proposal of Architecture to Support the Context-aware Personalized Recommendation of TV Programs. **Simpósio Brasileiro de Computação Ubíqua e Pervasiva**, 2009.

SOARES, L. F. G. et al. **Multiple exhibition devices in DTV systems**. International Multimedia Conference. Pequim, China: ACM. 2009. p. 281-290.

SOARES, L. F. G.; BARBOSA, S. D. J. **Programando em NCL 3.0 - Desenvolvimento de Aplicações para o Middleware Ginga**. 1ª Edição. ed. Rio de Janeiro: Elsevier, 2009.

SOARES, L. F. G.; CASTRO, P. H. As Múltiplas Possibilidades do Middleware Ginga. **Produção Profissional: Revista de Comunicação e Técnica Audiovisual**, São Paulo, p. 76-83, Junho 2008.

SOARES, L. F. G.; MORENO, M. F.; MORENO, M. F. **Transmissão de Aplicações e Comandos de Edição ao Vivo em Sistemas de TV Digital**. PUC-Rio. Rio de Janeiro. 2009. Monografia.

SOARES, L. F. G.; RODRIGUES, R. F.; MORENO, M. F. Ginga-NCL: The Declarative Environment of the Brazilian Digital TV System. **Journal of the Brazilian Computer Society**, v. 13, n. 4, p. 7–46, 2007.

SOUZA FILHO, G. L.; LEITE, L. E. C.; BATISTA, C. E. C. F. Ginga-J: The Procedural Middleware for the Brazilian Digital TV System. **Journal of the Brazilian Computer Society**, v. 13, n. 4, p. 47-56, 2007.

TELEMIDIA. NCL - Nested Context Language, 2006. Disponivel em: <http://www.ncl.org.br/>. Acesso em: Junho 2010.

THAWANI, A.; GOPALAN, S.; SRIDHAR, V. Context Aware Personalized Ad Insertion in an Interactive TV Environment. **Proceedings of Workshop on Personalization in Future TV**, 2004.

THE ECLIPSE FOUNDATION. Eclipse Modeling - EMF, 2010. Disponivel em: <http://www.eclipse.org/modeling/emf/>.

THE ECLIPSE FOUNDATION. Eclipse.org, 2010. Disponivel em: <http://www.eclipse.org/>.

VALE, I. M. et al. Regras Contextuais para Aplicações Sensíveis ao Contexto: Modelagem e Realização na Plataforma Ginga. **|| Simpósio Brasileiro de Computação Ubíqua e Pervasiva**, 2010.

VALE, I. M. et al. Regras Contextuais para Aplicações Sensíveis ao Contexto: Modelagem e Realização na Plataforma Ginga. **|| Simpósio Brasileiro de Computação Ubíqua e Pervasiva**, 2010.

VALE, I. M.; GUAITOLINI, F. B.; DOCKHORN COSTA, P. Modelagem Contextual de um Cenário para TV Digital. **Simpósio Internacional de Televisão Digital**, p. 1041-1060, 2009.

WEISER, M. The Computer for the Twenty-First Century. **Scientific American**, p. 94-104, 1991.

XACTIUM ON DEMAND SOLUTIONS, 2011. Disponivel em: <http://www.xactium.com>.

YANG, K. et al. Composition of context-aware services using policies and models. **Global Telecommunications Conference, 2005. GLOBECOM '05 - IEEE**, 2005.

PUBLICAÇÕES DO AUTOR

1. SALVIATO, T. P.; DOCKHORN COSTA, P.; FILHO, J. G. P.; VALE, I. M. Framework for Context-aware Applications on the Brazilian Digital TV. **4th International Conference on Ubi-media Computing (U-Media 2011)**, São Paulo, 2011. (Aceito para publicação).

2. MIELKE, I. T.; DOCKHORN COSTA, P.; VALE, I. M. Supporting Context Events in an Interactive TV Platform. **III Simpósio Brasileiro de Computação Ubíqua e Pervasiva**, Natal, 2011. In Anais do XXXI Congresso da Sociedade Brasileira de Computação. (Aceito para publicação).

3. DOCKHORN COSTA, P.; ALMEIDA, J. P. A.; VALE, I. M.; MIELKE, I. T. A Model-Driven Approach for Incorporating Reactive Rules in Declarative Interactive TV Applications. **POLICY 2011 – IEEE International Symposium on Policies for Distributed Systems and Networks**, Pisa, 2011. (Aceito para publicação)

4. VALE, I. M.; MIELKE, I. T.; GUAITOLINI, F. B.; DOCKHORN COSTA, P. Regras Contextuais para Aplicações Sensíveis ao Contexto: Modelagem e Realização na Plataforma Ginga. **II Simpósio Brasileiro de Computação Ubíqua e Pervasiva**, Belo Horizonte, 2010. In Anais do XXX Congresso da Sociedade Brasileira de Computação.

5. VALE, I. M.; GUAITOLINI, F. B.; DOCKHORN COSTA, P. Modelagem Contextual de um Cenário para TV Digital. **Simpósio Internacional de Televisão Digital**, p. 1041-1060, 2009.

6. VALE, I. M., DALFIOR, J. S., VASSALLO, R. F. Sistema de Visão Estéreo Híbrido com Recuperação da Posição Relativa entre as Câmeras. **IX Simpósio Brasileiro de Automação Inteligente**, Brasília, 2009. In Anais do IX Simpósio Brasileiro de Automação Inteligente.