UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO APLICADA

JOSIVAN PEREIRA DE SOUZA

UMA ARQUITETURA PARA COMPUTAÇÃO SENSÍVEL AO CONTEXTO APLICADA A PROCESSOS DE SOFTWARE

DISSERTAÇÃO DE MESTRADO

CURITIBA

2012

JOSIVAN PEREIRA DE SOUZA

UMA ARQUITETURA PARA COMPUTAÇÃO SENSÍVEL AO CONTEXTO APLICADA A PROCESSOS DE SOFTWARE

Dissertação apresentada ao Programa de Pós-graduação em Computação Aplicada da Universidade Tecnológica Federal do Paraná como requisito parcial para obtenção do título de "Mestre em Computação Aplicada" – Área de Concentração: Engenharia de Sistemas Computacionais.

Orientador: Prof. Dr. Gustavo Alberto Giménez Lugo

Co-orientador: Prof. Dr. Cesar Augusto Tacla

CURITIBA

2012

Dados Internacionais de Catalogação na Publicação

S729 Souza, Josivan Pereira
Uma arquitetura para computação sensível ao contexto aplicada a processos de software / Josivan Pereira de Souza. — 2012.
101 f. : il. ; 30 cm

Orientador: Gustavo Alberto Giménez Lugo.
Coorientador: Cesar Augusto Tacla.
Dissertação (Mestrado) – Universidade Tecnológica Federal do Paraná. Programa de Pós-graduação em Computação Aplicada, Curitiba, 2012.
Bibliografia: f. 80-84.

1. Arquitetura de computador. 2. Ontologias (Recuperação da informação). 3. Detectores. 4. Padrões de software. 5. Software – Desenvolvimento. 6. .Engenharia de sistemas. 7. Computação – Dissertações. I. Lugo, Gustavo Alberto Giménez, orient. II. Tacla, Cesar Augusto, coorient. III. Universidade Tecnológica Federal do Paraná. Programa de Pós-graduação em Computação Aplicada. IV. Título.

CDD (22. ed.) 004

Biblioteca Central da UTFPR, Campus Curitiba

Título da Dissertação Nº: 03

"Uma Arquitetura para Computação Sensível ao Contexto Aplicada a Processos de Software".

por

Josivan Pereira de Souza

Esta dissertação foi apresentada como requisito parcial à obtenção do grau de **MESTRE COMPUTAÇÃO APLICADA** - Área de Concentração: Engenharia de Sistemas Computacionais, pelo PPGCA - Programa de Pós-Graduação em Computação Aplicada - Mestrado Profissional - da Universidade Tecnológica Federal do Paraná - UTFPR - Câmpus Curitiba, às 14 horas do dia 30 de agosto de 2012. O trabalho foi aprovado pela Banca Examinadora, composta pelos professores:

Prof. **Gustavo Alberto Giménez Lugo, Dr.**
presidente - (UTFPR - CT)

Profª. **Andreia Malucelli, Drª.**
(PUC-PR)

Prof. **Emerson Paraíso, Dr.**
(PUC-PR)

Prof. **Laudelino Cordeiro Bastos, Dr.**
(UTFPR-CT)

Prof. **Adolfo Gustavo Serra Seca Neto, Dr.**
(UTFPR-CT)

Dedico todo meu trabalho para minha amada esposa Aline Coline
e para a princesa dos meus sonhos, Ana Clara de Souza.

AGRADECIMENTOS

Primeiramente, agradeço a Deus, por permitir que a fé e a sabedoria me guiem por toda minha vida. Agradeço às mulheres da minha vida, minha esposa tão amada, Aline Coline, que mesmo sem entender o significado deste trabalho, respeitou e dividiu cada sentimento que passei por esses meses. Agradeço à minha princesa, Ana Clara, que não faz ideia do que é ser mestre, mas saberá que tudo o que o pai dela faz é para que ela sempre se orgulhe.

Agradeço à minha mãe, que sabe mais da minha vida do que eu mesmo e que sempre me mostrou que tudo é possível se acreditarmos de verdade. Agradeço ao meu pai, que é uma prova de que nunca é tarde para aprendermos, e que para sermos melhores do que nós mesmos não devemos nunca desistir de aprender.

Na academia eu agradeço à Professora Doutora Andreia Malucelli, que participou desta batalha comigo desde o início, me recomendou para o programa e seguiu até o final, no dia da minha banca pública. Caso ela não me mostrasse com detalhes o caminho a ser percorrido, talvez eu não teria chegado até aqui. Agradeço ao Professor Doutor Emerson Paraíso, que também me recomendou e me avaliou. Aos dois professores eu posso dizer que o mestre de hoje sente-se melhor que o aluno de uma década atrás. Agradeço ainda ao Professor Doutor Laudelino, que esclareceu o valor da escrita. Ao Professor Doutor Adolfo, que me mostrou que doutores podem codificar, e o fazem isso com muita qualidade. Ao meu co-orientador Professor Doutor Tacla, que com toda a calma e paciência mostrou que o mais importante é aprender e saber pensar. É isso que faz a diferença. E, finalmente, ao meu orientador Professor Doutor Gustavo, que me transmitiu segurança durante todo o tempo, me mostrou que nossa pesquisa é relevante, e que podemos melhorar sempre. Meu orientador é para mim uma prova de que, mesmo com passadas diferentes, podemos caminhar lado a lado quase sempre. Para isso, basta sabermos se devemos e quando descansar.

Agradeço a todos os meus amigos que perceberam que eu me afastei, e entenderam os motivos. Agradeço àqueles que não citei, mas tenham certeza de que eu lembrei de vocês.

RESUMO

A literatura indica que uma forma de reconhecer ações e atividades é por meio do uso de sensores anexados de alguma forma no ambiente do ator. Uma forma de limitar itens a serem considerados em um ambiente é por meio da definição de um contexto que, por sua vez, é uma ferramenta para manipulação do problema de reconhecimento de atividades. Nesta pesquisa, contexto é descrito utilizando ontologias com conceitos associados a ações, eventos e dispositivos. Esta pesquisa apresenta uma arquitetura para identificar as prováveis ações de um ator envolvido em um processo de software em um ambiente monitorado por sensores. A arquitetura está fundamentada na utilização de dados provenientes das ações que atores executam no ambiente e que são coletados por meio dos sensores. O objetivo é identificar as ações que o ator executa com o mínimo de intervenção do mesmo. Os dados coletados dos sensores são descritos por meio de ontologias em graus de abstração variados que permitem descrever dos dados brutos até as atividades e contextos nos quais as ações acontecem. As aplicações desta arquitetura variam do simples monitoramento de atividades até o compartilhamento de informações entre diversos atores em função de índices que representam o conteúdo da atividade e/ou o contexto de realização da mesma. Para a realização deste trabalho, uma das etapas do método consistiu na inspeção dos dados coletados pelos sensores, o que resultou no desenvolvimento de novas ontologias que, após avaliação com arquiteturas propostas anteriormente, demonstram que a arquitetura proposta neste trabalho, por meio do uso de um motor de inferência, consegue inferir em dados coletados por sensores e representados com o uso de ontologias. Uma contribuição desta pesquisa é a implementação de uma arquitetura flexível, que permite que novos sensores sejam adicionados, sem alterar o núcleo da arquitetura proposta.

Palavras-chave: Arquitetura Sensível ao Contexto, Ontologia, Reconhecimento de Atividades, Processo de Software, Sensores

ABSTRACT

The present work undertakes the problem of recognizing activities of an actor engaged in a software process. The main goal is to identify the actions that the actor performs with minimal intervention of him/herself. The literature indicates that a way to recognize actions and activities is through the use of sensors attached to the environment. Still, when targeting a specific scope, a very useful approach is to explicitly define and describe a context, e.g. through specific ontologies, as was the choise of this work.

An architecture integrating ontologies and sensors is defined. The ontologies describe actions, events and devices with different granularities, acting as tools for the activity recognition problem. A specific implementation of the architecture is detailed along with the method used to derive the ontologies. Finally, the results are analyzed and evaluated vis-à-vis existing architectures. The main contribution of this research is a flexible architecture, allowing new sensors to be added in an incremental manner, just extending the ontologies, that are detached from the code.

Keywords: Context Aware Architecture, Ontologies, Activity Recognition, Software Process, Sensors

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

IDE	Integrated Development Environment
HMM	Hidden Markov Model
CE	*Contextual Element*
SeCoM	Semantic Context Model
XPO	Extreme Programming Ontology
DOLCE	Descriptive Ontology for Linguistic and Cognitive Engineering
TOVE	Toronto Virtual Enterprise
RDF	Resource Description Framework
SDSA	Software Development Stream Analysis
TDD	*Test Driven Development*
CoSEEEK	Context-aware Software Engineering Environment Eventdriven frameworK
OWL	Web Ontology Language

SUMÁRIO

1 INTRODUÇÃO

Dispositivos eletrônicos, chamados também de *gadgets*[1], estão cada vez mais presentes na vida das pessoas, seja por meio de computadores, *smartphones*[2] ou qualquer dispositivo com características computacionais. Para Maes (1994), o uso diário destes dispositivos deveria torná-los capazes de entender e interpretar o que os usuários supostamente fazem, além de fazer tal ação de forma automática. Quando computadores são utilizados, são dadas diretivas explícitas para a realização de algo. Maes (1994, 1996) sugere evoluções destes aparelhos e desta interação entre os usuário e os aparelhos para cenários nos quais a máquina tenta adivinhar as intenções do usuário. E, lentamente, os usuários tentarão ser menos diretivos e mais pró-ativos. Neste processo de interação entre o usuário e algum dispositivos, o usuário pode ser chamado de ator.

Uma das formas de entender as intenções do ator durante a interação com outros é por meio do reconhecimento de atividades que, segundo Chen e Nugent (2009), é o processo em que o comportamento de um ator em seu ambiente é monitorado e analisado para inferir sobre atividades executadas.

1.1 MOTIVAÇÃO

Reconhecer atividades executadas por um ator é o primeiro passo para tentar entender suas intenções. Atores, para este trabalho, são pessoas que executam atividades durante o processo de desenvolvimento de software. O processo de desenvolvimento de software, segundo Horstmann (2007), consiste nas fases de análise, projeto e implementação. Nas atividades realizadas durante o processo de desenvolvimento de software, os atores podem assumir vários papéis de acordo com a fase, experiência ou especialidade do ator. Um ator pode assumir, por exemplo, o papel de analista de negócio, desenvolvedor de sistemas,

[1] O dicionário Houaiss define *gadget* como: "pequeno objeto ou aparelho, mecânico ou eletrônico, mais engenhoso do que útil, que constitui uma novidade"

[2] *Smartphone* é um telefone inteligente (numa tradução livre do inglês), que possui funcionalidades avançadas como acesso à internet

administrador de banco de dados, entre outros.

Ao considerar as tarefas executadas durante o processo de desenvolvimento de software, há uma tendência de que algumas sequências sejam repetidas de forma mais explícita que durante a execução de tarefas e atividades cotidianas. Além disso, tarefas cotidianas exigem a atribuição de muitos outros fatores e condições que devem ser analisadas. Por isso, é menos difícil determinar atividades e sequências durante o processo de desenvolvimento de software.

Neste trabalho, atividade é definida como uma sequência temporal de ações. Então, antes do reconhecimento de atividades, existe a necessidade de reconhecimento de ações em uma sequência temporal. Reconhecendo sequências de ações, e posteriormente atividades, é possível recomendar sequências de ações, consideradas ideais, para execução de determinada atividade.

1.2 JUSTIFICATIVA

Um dos problemas computacionais deste trabalho é detectar atividades executadas por um ator em um determinado contexto. Outro é dar significado a uma sequência de ações. Reconhecer atividades de maneira automatizada pode diminuir a quantidade de repetições de uma atividade reconhecida anteriormente, diminuindo, por exemplo, os trabalhos duplicados ou repetitivos em determinado ambiente.

Uma forma de tornar menos difícil a modelagem de um ambiente é através do uso de contexto. Neste trabalho, será utilizada a abstração de contexto aplicada em uma área particular: processo de software. Como ferramenta para representação de contexto e de ações serão utilizadas ontologias. Tanto contexto quanto ontologias são descritas no capítulo 2.

Esta pesquisa é o primeiro passo para o reconhecimento de atividades executadas por um ator por meio do uso de sensores, e que converte os dados automaticamente para ontologias e utilizando ontologias para representação dos dados coletados. O trabalho considera, neste momento, o reconhecimento de ações executadas por um usuário. A partir da descrição de uma sequência temporal de ações, é possível inferir qual atividade um ator executou em determinado momento. O fato de um dispositivo com características computacionais aprender através de algum mecanismo uma sequência que pode ser considerada correta, ou apenas a mais utilizada, tal sequência pode, futuramente, ser recomendada através deste dispositivo. Ainda que a arquitetura proposta nesse trabalho

não faça recomendações; se coerentes, as recomendações podem facilitar ou até mesmo melhorar a qualidade do trabalho no processo de desenvolvedores de software.

A literatura indica que o "Estado da Arte" no reconhecimento de atividades baseia-se no uso de sensores anexados de alguma forma ao ator e ao ambiente que este executa suas ações.

1.3 OBJETIVOS

O presente trabalho tem como objetivos:

1.3.1 Objetivo geral

Definir uma arquitetura que, baseada em dados coletados por sensores convertidos automaticamente para ontologias e utilizando ontologias, possa identificar as ações explícitas executadas por um ator, em um contexto de desenvolvimento de software.

1.3.2 Objetivos Específicos

Para atingir o objetivo geral, são considerados objetivos específicos:

- Explorar dados coletados por sensores em um ambiente de desenvolvimento de software;
- Desenvolver ontologias que representem os dados coletados por sensores em um ambiente de desenvolvimento de software;
- Desenvolver ontologia que permita a integração das ontologias de sensores com as ontologias de contexto e de processo de software;
- Utilizar técnica que permita a incorporação de novos tipos de sensores através da característica de extensibilidade;
- Implementar a arquitetura proposta;
- Comparar a arquitetura proposta;

1.4 CONTRIBUIÇÕES

As contribuições, sumarizadas, desta pesquisa são:

- Uma arquitetura integradora de várias técnicas e enfoques, utilizando ontologias que descrevem contexto e processo de software;
- Um Modelo Conceitual, descrito com o uso de Ontologias, para a extração e a inferência de dados capturados por sensores, com poucas regras definidas no código fonte da implementação;
- Uma fonte de dados abstrata que pode ser representada a partir de dados coletados por sensores que avaliam registros[3] em servidores de aplicações e em servidores de banco de dados ou outra forma qualquer, como dados em arquivos textos posicionais, desde que sejam desenvolvidas ontologias representando os dados facilitando outras implementações;
- A demonstração da possibilidade de representar e dar significado a dados coletados por sensores, através do uso de ontologias sem a intervenção do usuário;
- O passo inicial para o reconhecimento de atividades através de sequências de ações utilizando ontologias para representar dados coletados por sensores;

1.5 ESTRUTURA DA DISSERTAÇÃO

Além desta Introdução, o capítulo 2 apresenta a revisão da literatura nas quais são descritas as pesquisas mais recentes dos assuntos relevantes para este trabalho incluindo o reconhecimento de atividades, o uso de sensores, a definição de contexto, modelagem e computação sensível ao contexto e às formas de obter informações, nível de interação do e com o contexto e tarefas de sistemas sensíveis ao contexto. O capítulo 2 inclui também a definição de ontologia, e descreve as ontologias consideradas relevantes para este trabalho, além das ferramentas necessárias para uma implementação da arquitetura proposta e finaliza com a apresentação de arquiteturas propostas como trabalhos correlatos. O capítulo 3 descreve cada uma das etapas necessárias para atingir os objetivos específicos. O capítulo 4 apresenta a arquitetura proposta, descrevendo com mais detalhes as ontologias existentes relevantes para este trabalho chamadas de Ontologias de Nível Superior. O capítulo apresenta ainda a Ontologia de Integração requerida para que a arquitetura, através de seus componentes, possa inferir sobre as ontologias que representam os dados coletados por sensores. O capítulo 5 descreve os experimentos realizados para avaliação da arquitetura em relação aos trabalhos correlatos, assim como apresenta as ontologias para alguns sensores, desenvolvidas a partir dos dados coletados por sensores. Finalmente, o

[3] As vezes chamados de *logs*.

capítulo 6 apresenta as conclusões finais, e inclui as contribuições, limitações e propostas para trabalhos futuros.

2 REVISÃO DA LITERATURA

Monitorar o comportamento do ator em conjunto com as mudanças no ambiente é, segundo Chen e Nugent (2009), uma tarefa crítica no reconhecimento de atividades. Este processo de monitoramento é responsável por obter informações contextuais relevantes para o sistema de reconhecimento de atividades inferir a atividade do ator. A primeira parte deste capítulo descreve inicialmente o uso de sensores e o uso destes para reconhecimento de ações e atividades.

Uma forma útil de pensar em domínio de aplicação é trabalhar com a noção de contexto. Ao definir um contexto, limita-se os itens que podem ser considerados em um ambiente. Assim, o contexto é uma "porta de entrada" para chegar aonde deseja. Trata-se do ambiente. Nesta pesquisa, será considerado contexto o "Processo de Software". Dentro deste, será usada uma granularidade menor de contexto, em específico o processo de desenvolvimento. Além disso, contexto é uma ferramenta para manipulação do problema de reconhecimento de atividades. A segunda parte deste capítulo descreve contexto, incluindo a modelagem e a computação sensível ao contexto.

Ainda em relação ao reconhecimento de atividades, Chen e Nugent (2009), Chen e Khalil (2011) afirmam que o uso de ontologias é um esforço recente e que tem despertado algum interesse. Os autores concordam com a necessidade da representação das definições das atividades e que ontologias para atividades não dependem do algoritmo escolhido. Portanto, facilitam a portabilidade, interoperabilidade, reuso e compartilhamento entre tecnologias e sistemas subjacentes. Estudos apontados pelos autores indicam que o reconhecimento de atividade baseado em observação usa, principalmente, ontologias para prover descritores de atividades para definições de atividades. Este capítulo apresenta ainda a descrição de ontologia e o Modelo Semântico de Informações de Contexto proposto por Neto (2006).

O capítulo é finalizado com a descrição da metodologia *Ontology Development 101*, proposta por Noy e McGuinness (2001), e aplicada na modelagem de novas ontologias

implementadas para este trabalho. Para cada item apresentado nesta revisão bibliográfica, são listados alguns trabalhos correlatos e resumos que descrevem a metodologia e os resultados obtidos.

2.1 SENSORES E RECONHECIMENTO DE ATIVIDADES

Reconhecimento de Atividades, segundo Chen e Nugent (2009), é o processo em que o comportamento de um ator em seu ambiente é monitorado e analisado para inferir sobre atividades executadas. Atividades compreendem diversas tarefas diferentes, modelagem de atividades com nomes definidos, monitoramento do comportamento e do ambiente, processamento de dados e reconhecimento de padrões. Para executar o reconhecimento de atividades, segundo Chen e Khalil (2011), é necessário:

1. Criar modelos computacionais de atividades de forma a permitir que agentes/sistemas de software conduzam raciocínios e manipulações;
2. Monitorar e capturar o comportamento do usuário conforme a mudança de estado do ambiente;
3. Processar informações capturadas através da agregação e fusão para gerar uma abstração de alto nível do contexto ou situação;
4. Decidir qual algoritmo de reconhecimento de atividade utilizar;
5. Compreender padrões reconhecidos para determinar a atividade executada;

O objetivo do reconhecimento de atividades, segundo Companjen (2009), é determinar as ações ou o estado de uma ou mais pessoas através da análise dos dados dos sensores no ambiente ou de sensores presos ao corpo das pessoas. No caso desta pesquisa, as ações do ator são importantes, o que faz com que o reconhecimento de ações também seja importante. Alguns enfoques relacionados ao uso de sensores são abordados a seguir.

2.1.1 Uso de Sensores

Chen e Nugent (2009), Chen e Khalil (2011) descrevem duas abordagens principais na maneira de coleta de dados para reconhecimento de atividades, uma baseada em observações (chamada de visão por Chen e Khalil) e outra baseada em sensores. Para este trabalho, interessa o reconhecimento de ações baseado em sensores, especificamente

sensores que capturam ações de desenvolvimento de software que ocorrem em um *IDE (Ambiente Integrado de Desenvolvimento).*

O reconhecimento de atividades baseado em sensores explora as tecnologias emergentes de redes de sensores para monitorar o comportamento do ator em conjunto com o ambiente. Os dados por eles coletados são geralmente analisados usando técnicas de mineração de dados e de aprendizagem de máquina para construir modelos de atividades e executar outros meios de reconhecimento de padrão. Nesta abordagem, Chen e Khalil (2011) descrevem que sensores podem ser anexados ao ator que está sendo observado ou em objetos que compõem o ambiente. Chen e Nugent (2009), Chen e Khalil (2011) definem este cenário como o "estado da arte" do reconhecimento de atividades. Embora não considerem explicitamente o conceito de contexto, Chen e Khalil (2011) coletam informações das atividades em sensores junto ao corpo do ator, ou anexados em objetos em um ambiente em particular.

2.1.2 Algoritmos para Reconhecimento de Atividades

Segundo Chen e Nugent (2009), os algoritmos de reconhecimento de atividades podem, de um modo geral, ser divididos em duas vertentes. A primeira é baseada em técnicas de aprendizagem de máquina, que inclui métodos de aprendizagem supervisionados e não-supervisionados, usando principalmente raciocinadores probabilísticos e estatísticos.

O processo geral que utiliza algoritmos de aprendizagem supervisionado para reconhecimento de atividade é descrito, por Chen e Nugent (2009), Chen e Khalil (2011), como:

1. Adquirir dados de sensores que representam as atividades, incluindo anotações apontadas manualmente pelo ator, descrevendo o que o ator fez e quando o fez;
2. Determinar as características dos dados de entrada e sua representação;
3. Agregar dados a partir de múltiplas origens e transformá-los em características dependentes da aplicação, com o uso da fusão, eliminação de ruídos, redução de dimensões e normalização de dados;
4. Dividir os dados em dados de treinamento e dados de teste;
5. Treinar o algoritmo de reconhecimento com o conjunto de treinamento;

6. Testar a performance da classificação do algoritmo usado no treino com o uso do conjunto de teste;

7. Aplicar o algoritmo no contexto de reconhecimento de atividade.

A segunda vertente dos algoritmos de reconhecimento, ainda segundo Chen e Nugent (2009), é baseada em modelagem lógica e raciocínio. A abordagem é explorar a representação do conhecimento lógico para atividades e modelagem de dados dos sensores, e usar o raciocínio lógico para realizar o reconhecimento de atividades. O procedimento geral da abordagem lógica inclui:

1. Usar o formalismo lógico para definir e descrever explicitamente conjuntos de modelos para todas as atividades possíveis em um determinado domínio;

2. Agregar e transformar dados dos sensores em fórmulas e termos lógicos;

3. Realizar raciocínios lógicos como, por exemplo, dedução, abdução e subsunção, para extrair um conjunto mínimo de modelos de interpretações que cubram a biblioteca de modelos de atividades baseado em um conjunto de ações observadas que podem explicar as observações.

Chen e Nugent (2009), Chen e Khalil (2011) descrevem vários algoritmos empregados, como o Modelo Oculto de Markov *(Hidden Markov Model - HMM)*, *dynamic* e *naive Bayes networks*, árvores de decisão, vizinhos próximos *(nearest neighbour)* e máquinas de vetores de suporte. HMM e redes Bayesianas são os métodos mais comumente usados no reconhecimento de atividades.

Uma atividade pode ser considerada como uma série de ações. Nesta pesquisa, foi explorado o reconhecimento de ações na área de processo de software, com foco especial no ator com papel de desenvolvedor. Segundo Schlesinger e Jekutsch (2006), em Engenharia de Software o termo "processo" é usado de forma prescritiva, onde são definidas a ordem e as atividades do desenvolvimento de software. Desenvolvedores são encorajados a adotar, em seu processo de trabalho, modelos ou linhas gerais para aumentar a produtividade ou a qualidade do produto. Para os autores, o termo "processo" é usado de forma descritiva, a fim de informar não só o que o desenvolvedor está realmente fazendo, mas também o que deve fazer. Portanto, nesta pesquisa, será considerado o termo "processo" de forma descritiva.

Uma forma de limitar itens a serem considerados em um ambiente é por meio da definição de um contexto que, por sua vez, é uma ferramenta para manipulação do problema de

reconhecimento de atividades, embora este trabalho explore o reconhecimento de ações. A próxima seção descreve contexto, detalhando informações adicionais, computação sensível ao contexto e modelagem de contexto.

2.2 CONTEXTO

Nesta pesquisa será considerado como contexto o processo de desenvolvimento de software. No caso, contexto é apenas o cenário onde serão aplicados os experimentos, descritos no capítulo 5. Este contexto pode ser composto de sub-contextos ou micro-contextos como análise, desenvolvimento, reunião com partes interessadas, entre outros. O contexto de processo de software pode ser classificado como amplo, considerando as diversas partes menores que o compõem. Nesta pesquisa decidiu-se por um contexto de granularidade menor, especificamente o desenvolvimento de software. Existem outros contextos no processo de software como: análise, modelagem, reunião com cliente, reunião com equipe de desenvolvimento, testes de integração, entre outros. Contexto é uma forma de expressar quais são as variáveis e condições consideradas em um ambiente. Assim, contexto é uma abstração que pode tornar menos difícil a modelagem de um ambiente. Para Sebesta (2011), de forma breve, abstração significa a habilidade de definir e usar estruturas ou operações complicadas de forma a permitir que muitos dos detalhes sejam ignorados.

Para Dey, Abowd e Salber (2001), aplicações sensíveis ao contexto (do inglês *context-aware*) são aquelas que utilizam informações de contexto para fornecer serviços e informações relevantes a usuários e a outras aplicações na realização de alguma tarefa. Além disso, o contexto contém informações que podem ser utilizadas para caracterizar a situação de uma entidade que é considerada relevante entre o usuário e a aplicação. Para Dey, Abowd e Salber, um sistema é ciente de contexto quando usa o mesmo para fornecer informações relevantes e/ou serviços ao usuário, no qual a relevância depende das tarefas do usuário.

Nos primeiros sistemas sensíveis ao contexto são consideradas, principalmente, as informações de localização, embora existam outras. Schmidt, Beigl e Gellersen (1999) sugerem classificações para características do contexto, incluindo os fatores humanos: o usuário em seu ambiente social, tarefas e o ambiente físico. Neste caso, alguns exemplos são a localização, infra-estrutura e condições físicas.

Abowd e Mynatt (2000) descrevem outra taxonomia sobre contexto na qual são des-

critas cinco dimensões contextuais, conhecidos como *"five W's"*, em função das iniciais das cinco dimensões em inglês: *Who* (Quem), *What* (O quê), *Where* (Onde), *When* (Quando), e *Why* (Por quê) descritas com mais detalhes na seção 2.2.4.

Para Brézillon (1999) contexto pode ser visto como um conjunto de condições e influências relevantes que tornam a situação única e compreensível. Essa situação pode referir-se a uma pessoa, um grupo de pessoas, um objeto físico, uma entidade computacional, entre outros. Em outras palavras, segundo Brézillon (1999), contexto é o conhecimento que está por trás da habilidade de discriminar o que é ou não importante em um dado momento, colaborando com indivíduos para melhorar a qualidade da conversação e a compreender certas situações, ações ou eventos. Para Vieira et al. (2007), o termo elemento contextual (CE, do inglês *Contextual Element*) refere-se a dados, informações ou conhecimento que podem ser utilizados para definir um contexto.

Vieira, Tedesco e Salgado (2009) citam em sua pesquisa que a definição do que é o contexto e o que ele engloba, no caso particular de sistemas computacionais, ainda não é um consenso entre os pesquisadores. Dey, Abowd e Salber (2001), como citado anteriormente, propõe uma definição clássica e bastante referenciada, em que o contexto é qualquer informação que caracteriza a situação de uma entidade, sendo que uma entidade pode ser uma pessoa, um lugar ou um objeto considerado relevante para a interação entre um usuário e uma aplicação, incluindo o próprio usuário e a aplicação. O contexto é tipicamente a localização, a identidade e o estado das pessoas, grupos ou objetos físicos e computacionais.

Zimmermann, Lorenz e Oppermann (2007), em busca de uma definição operacional para contexto, estendem a definição de Dey, Abowd e Salber por meio da separação dos elementos que caracterizam a situação de uma entidade em cinco categorias: individualidade (propriedades e atributos que definem a entidade em si); atividade (todas as tarefas que a entidade pode estar envolvida); localização e tempo (coordenadas espaço-temporais da entidade); e relações (informações sobre qualquer relação que a entidade possa estabelecer com outras entidades), como representado na Figura 1 onde são mantidos os termos originais sem tradução. Na área da Inteligência Artificial, Brézillon (1999) define contexto como "o que restringe a solução de um problema sem, entretanto, interferir nele explicitamente".

Vieira, Tedesco e Salgado (2009) fazem uma distinção clara entre contexto e elemento contextual, como descrito a seguir:

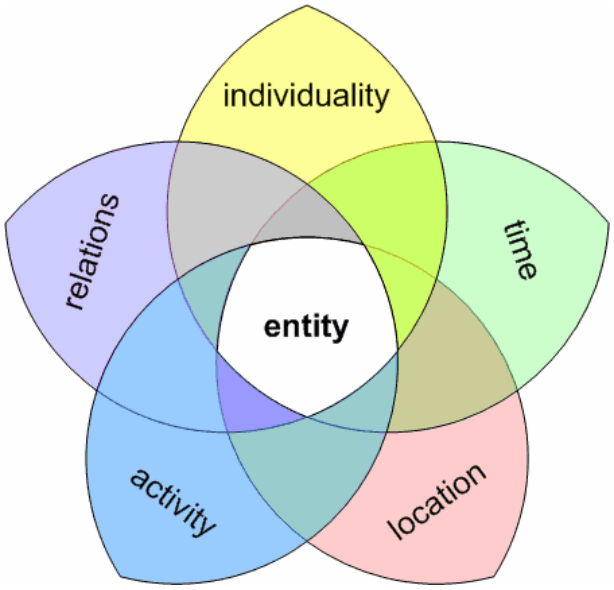

Figura 1: Cinco Categorias Fundamentais para Informação de Contexto
Fonte: Zimmermann, Lorenz e Oppermann (2007)

> "Um elemento contextual (CE, do inglês *Contextual Element*) é qualquer dado, informação ou conhecimento que permite caracterizar uma entidade em um domínio.
>
> O contexto da interação entre um agente e uma aplicação, para executar alguma tarefa, é o conjunto de elementos contextuais instanciados que são necessários para apoiar a tarefa atual."

Com esta definição, Vieira, Tedesco e Salgado (2009) consideraram o contexto aplicado à interação entre um agente e uma aplicação. Um agente pode ser um agente humano ou agente de software. Um CE é um tipo de informação que pode ser conhecida, codificada e representada antecipadamente; ela é estável e pode ser definida em tempo de projeto. O contexto é dinâmico, depende da tarefa atual do agente e deve ser construído em tempo de execução, quando uma interação ocorre. Assim, aplicado a este trabalho, ontologias e sensores são considerados um tipo de elemento contextual.

2.2.1 Visões sobre Contexto

Schilit, Adams e Want (1994) empregaram pela primeira vez o termo Computação Sensível ao Contexto *(Context-Aware Computing)* para designar os sistemas capazes de examinar o ambiente computacional e reagir a mudanças nesse ambiente.

Vieira, Tedesco e Salgado (2009) citam em seu estudo que contexto aplica-se, ainda,

a problemas associados ao gerenciamento de conteúdo e recuperação de informações. Os autores citam também que o contexto pode ser empregado no apoio à resolução de conflitos semânticos em sistemas de integração de informação.

Para Vajirkar, Singh e Lee (2003), o uso de contexto pode reduzir espaços de busca e otimizar padrões de identificação em minerações de dados. Bunningen (2004), em seu relatório técnico, diz que contexto pode ser empregado também na otimização do desempenho de consultas. Por fim, Stefanidis, Pitoura e Vassiliadis (2005) consideram a investigação da comunidade de Banco de Dados quanto ao uso de técnicas de representação de informação para a modelagem do contexto.

2.2.2 Informações de Contexto

Para Schilit, Adams e Want (1994), o contexto é adquirido diretamente pela pessoa de forma explícita ou implícita, monitorando-a com base nas atividades executadas no computador.

Henricksen, Indulska e Rakotonirainy (2002) classificaram contexto baseado em suas associações, sendo estas estáticas ou dinâmicas. O contexto dinâmico tem características de persistência relacionadas à forma como as informações de contexto são obtidas. Essas características podem ser classificadas de acordo com a origem: *sensed* (fornecidas por um sensor), *derived* (derivadas de outra origem) ou *profiled* (definidas por uma pessoa ou por um serviço).

Considerando a aquisição e a persistência da informação, Escobedo (2008) realizou as seguintes reclassificações:

- **percebida por sensores:** informação gerada através de sensores com alguma precisão;
- **definida pelo usuário:** o usuário fornece alguns valores à aplicação durante uma configuração;
- **fornecido por um serviço:** a informação é dada por algum fornecedor de serviços, com uma agenda ou cronograma onde há uma lista de atividades da pessoa;
- **derivada:** a qual pode ser aprendida ou inferida;

Níveis de Interação

Chen e Kotz (2000) definem duas formas de uso do contexto: (1) contexto ativo e (2) contexto passivo.

- **ciente de contexto de forma ativa:** A aplicação se adapta de forma automática à descoberta do contexto, mudando o seu comportamento;
- **ciente de contexto de forma passiva:** A aplicação apresenta a informação nova e atualizada ao usuário, e a armazena para um uso posterior.

Não há uma maneira padrão de adquirir e manipular informações de contexto. De maneira geral, informações de contexto são manipuladas de forma *ad hoc*, sem particular preocupação com generalização e reuso. Para a aquisição de informações de contexto, o gerenciamento de eventos é normalmente explorado, ou via consulta (técnica de *polling*), ou via notificação (técnica de *callback*). Embora ambos não sejam necessariamente implementados, é desejável que aplicações possam se beneficiar desses mecanismos por razões de flexibilidade. Informações de contexto devem ser interpretadas antes de serem utilizadas por aplicações. Como não se sabe quando aplicações solicitarão informações de contexto, os componentes devem ser executados continuamente para permitir que aplicações os contatem quando necessário.

Para Neto (2006) um exemplo de componente pode ser algum mecanismo emissor de algum tipo de sinal. Ao se comunicar com dispositivos de captura de informações de contexto, uma aplicação deve saber que tipos de informações o dispositivo pode fornecer, qual a localização deste dispositivo e que protocolos e linguagens devem ser utilizados para se comunicar com o mesmo. Assim que uma determinada aplicação é iniciada, deve-se especificar o tipo de informação de contexto requisitado. Com isso, o mecanismo de descoberta de recursos informa onde encontrar os componentes adequados, bem como descreve os mecanismos para acesso a ele.

2.2.3 Computação Sensível a Contexto

Para Neto (2006), a computação sensível ao contexto investiga o emprego de informações que caracterizam a situação de uma interação usuário-computador no sentido de fornecer serviços adaptados a usuários e aplicações. Essas informações, conhecidas como informações de contexto, podem ser obtidas de duas formas: explícita, quando a informação obtida é expressa intencionalmente pelo usuário, como o reconhecimento de sua voz;

ou implícita, quando a informação é obtida sem a comunicação intencional do usuário, como seu foco de atenção ou sua localização em um ambiente físico.

Tarefas de Sistemas Sensíveis a Contexto

Abowd (1999) generaliza as tarefas de sistemas sensíveis a contexto da seguinte forma:

1. Aquisição de uma grande e diversificada quantidade de informações de um ambiente de interação;
2. Organização dessa gama de informações em uma estrutura de representação eficiente para recuperação e consulta;
3. Análise dessas informações como variáveis independentes, ou por meio da combinação dessas informações com outras registradas no passado ou presente;
4. Transmissão da interpretação das informações envolvidas a aplicações que realizam alguma ação com base nessa análise;
5. Repetição de todo o processo de forma automática, transparente e adaptada de acordo com as mudanças nas informações de contexto do ambiente.

2.2.4 Modelagem de Contexto

Um sistema sensível ao contexto requer que informações contextuais sejam trocadas e utilizadas por diferentes entidades, como agentes humanos e de software, dispositivos e serviços, com uma mesma compreensão semântica. Para isso, um modelo de contexto apropriado deve dar suporte à interoperabilidade semântica e permitir que esquemas comuns sejam compartilhados entre diferentes entidades.

Especificação e Gerenciamento do Contexto

Em complemento ao conteúdo descrito na seção 2.2, diversos outros autores [tais como Morse, Armstrong e Dey (2000), Vieira et al. (2004), Neto (2006), Nunes, Santoro e Borges (2007)] indicam que as informações contextuais podem ser identificadas a partir da análise de seis dimensões básicas, referenciadas como 5W+1H. Essas dimensões visam responder às questões quem (*who*) está fazendo o quê (*what*), em que local (*where*), em que momento (*when*), com qual motivação (*why*) e de que maneira (*how*). Essas questões são consideradas básicas para contextualizar uma situação. *Who* (identificação) indica

informações contextuais relacionadas à identidade das entidades (como, por exemplo, nome, e-mail, impressão digital). *Where* (localização) determina informações contextuais que indicam a localização da entidade (como, por exemplo, longitude, latitude, cidade, país). *What* (atividade) identifica as atividades em que uma entidade (por exemplo, pessoa) está envolvida (por exemplo, caminhando, ministrando uma palestra). *When* (tempo) indica o contexto temporal relacionado a uma interação (por exemplo, data corrente, estação do ano). *Why* (motivação) relata a motivação por trás das ações do usuário ao executar uma tarefa em uma dada interação. *How* (meio) define a forma como os elementos contextuais são adquiridos (por exemplo, sensor, base de conhecimento).

Para Vieira et al. (2006), o processamento do contexto pode ser entendido como o conjunto de métodos e processos que realizam raciocínio, transformação, combinação e resolução de conflitos das informações contextuais adquiridas, de modo a produzir outras informações mais refinadas, relevantes e coerentes com o que é necessário em um determinado momento. Bases de conhecimento permitem o armazenamento do contexto histórico e, juntamente com motores de inferência, apoiam o processamento do contexto. O mecanismo de processamento do contexto deve considerar, ainda, o tratamento de incertezas, ao pressupor que a informação contextual pode conter inconsistências, ser ambígua ou incompleta.

O uso de ontologias, segundo Chen e Nugent (2009), Chen e Khalil (2011), para representar as definições de atividades torna independente a escolha do algoritmo. Dessa forma, facilita-se a portabilidade, interoperabilidade, reuso e compartilhamento entre tecnologias e sistemas subjacentes. A próxima seção descreve ontologia e Modelo Semântico de Informações de Contexto.

2.3 ONTOLOGIAS

Para Noy e Hafner (1997), em Ciência da Computação o termo "Ontologia" se refere a um artefato de engenharia de software constituído de um vocabulário de termos organizados em uma taxonomia, suas definições e um conjunto de axiomas formais usados para criar novas relações e para restringir as suas interpretações segundo um sentido pretendido. No sentido filosófico, o termo "Ontologia" é referido por Smith (2003) como um sistema particular de categorias que versa sobre uma certa visão do mundo e pode ser visto como um sinônimo de metafísica. Seu propósito é classificar as entidades de uma porção da realidade, definindo seu vocabulário e as formulações canônicas de suas teorias. Desta forma, este sistema não depende de uma linguagem particular. Por exemplo,

uma ontologia de Aristóteles é sempre a mesma, independente da linguagem usada para expressá-la, como cita Guizzardi (2000). Apesar da relação entre essas duas definições, Guarino (1998) propõe, com o intuito de resolver este impasse terminológico, que a definição da comunidade de computação seja adotada para o termo "ontologia" e que para a definição filosófica seja dado o nome de conceituação. A comunidade de Inteligência Artificial adota como definição clássica de ontologia aquela proposta por Gruber (1993), na qual "ontologia é uma especificação formal de uma conceituação". Outra definição amplamente aceita na comunidade é a de Fensel (2001) que estende a definição de Gruber ao afirmar que "ontologia é uma especificação explícita e formal de uma conceituação compartilhada".

Guizzardi (2000) adota o termo ontologia como definido por Guarino (1998), ou seja:

> "Ontologias são tratadas como um artefato computacional composto de um vocabulário terminológico (conceitos/classes), suas definições e suas possíveis propriedades, um modelo gráfico mostrando todas as possíveis relações entre os conceitos e um conjunto de axiomas formais que restringem a interpretação dos conceitos e relações, representando de maneira clara e não ambígua o conhecimento do domínio."

Guizzardi (2000) afirma que a conceituação tem uma importância fundamental em qualquer atividade de modelagem do conhecimento, pois é impossível representar o mundo real, ou mesmo uma parte dele, em sua completa riqueza. Para representar um fenômeno ou parte do mundo, chamado de domínio, é necessário manter o foco em um número limitado de conceitos que são relevantes e suficientes para criar uma abstração do fenômeno manualmente. Assim, o aspecto central de qualquer atividade de modelagem consiste do desenvolvimento de uma conceituação: um conjunto de regras informais que restringem a estrutura de uma parte da realidade, que um agente usa para isolar e organizar relações e objetos relevantes.

Uma ontologia, portanto, passa a ter compromisso apenas com a consistência em um determinado domínio, e não com a completude. Ao conjunto de elementos de um domínio que podem ser representados em uma ontologia, Guizzardi (2000) dá o nome de *universo de discurso*.

Nesta pesquisa, será usado ontologia como ferramenta para representação de um contexto, processo de software e dados coletados por sensores. A escolha deu-se em função das características de ontologia, pois usar um algoritmo específico ou inúmeras regras,

declarados em código fonte, pode tornar a arquitetura proposta específica e condicional.

2.3.1 Ontologia e Reconhecimento de Atividades

Chen e Nugent (2009), Chen e Khalil (2011) apontam estudos que usaram ontologias para prover termos comuns para definição de atividades. Boa parte das pesquisas de reconhecimento de atividades baseado em ontologias, segundo Chen e Nugent (2009), Chen e Khalil (2011), são aplicadas para explicitarem termos comuns e compartilharem especificações de termos e relacionamentos, entre todas as entidades relevantes. Alguns exemplos são objetos, elementos e eventos do ambiente, facilidades para interoperabilidade, reusabilidade e portabilidade dos modelos entre diferentes sistemas e domínios de aplicação.

2.3.2 Modelo Semântico de Informações de Contexto

Como citado anteriormente, uma forma de representar tanto o contexto quanto o reconhecimento de atividades é por meio do uso de ontologias. Neto (2006) apresentou em sua tese de doutorado o Modelo Semântico de Informações de Contexto, denominado modelo *SeCoM (Semantic Context Model)*, que é composto de um conjunto modular de ontologias inter-relacionadas baseadas nas dimensões semânticas de identidade *(Actor)*, localização *(Spatial)*, tempo *(Time)*, atividade *(Activity)* e modo de captura e acesso *(Device)*. Neste modelo, o dimensionamento ou aplicação das cinco dimensões de informações de contexto, descritas na seção 2.2, descreve as classes envolvidas em uma interação usuário-computador. A ontologia do modelo instancia os conceitos de identificação, localização, tempo, atividade e modo de captura e acesso para o domínio em questão, com o intuito de atender a vários domínios de aplicação sensível ao contexto. Numa visão geral, o modelo SeCoM define treze ontologias, dentre as quais Neto (2006) classifica seis como ontologias de apoio e sete como principais. Estas ontologias são: (1) *Activity*, (2) *Actor*, (3) *Contact*, (4) *Device*, (5) *Document*, (6) *Knowledge*, (7) *Project*, (8) *Relationship*, (9) *Role*, (10) *Spatial*, (11) *SpatialEvent*, (12) *TemporalEvent* e (13) *Time*. A Figura 2 representa uma visão geral do modelo SeCoM, onde as ontologias em ovais escuras representam ontologias principais, e ovais claras são ontologias de apoio. Serão descritas com mais detalhes, apenas as ontologias consideradas relevantes para este trabalho; sendo elas (1) *Actor*, (2) *Contact*, (3) *Document*, (4) *Project*, (5) *Role*, (6) *TemporalEvent* e (7) *Time*.

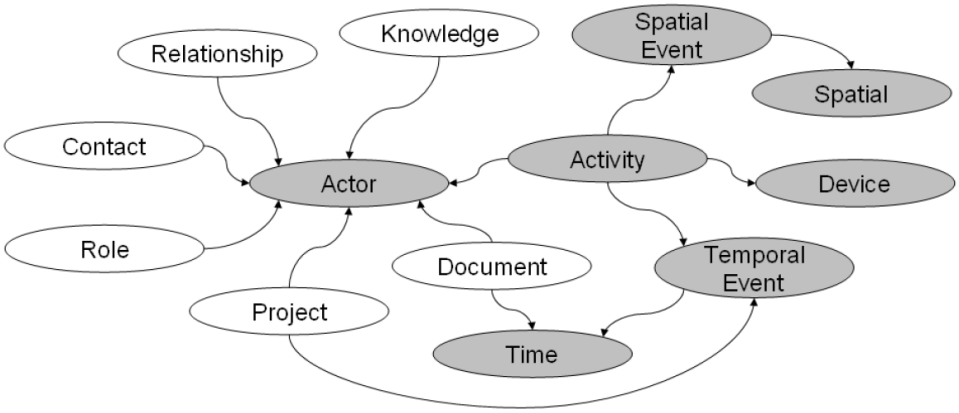

Figura 2: Visão Geral do modelo SeCoM
Fonte: Neto (2006)

Ontologia *Actor*

De acordo com Neto (2006), a ontologia `Actor`, apresentada na Figura 22 do anexo B, modela o conjunto de entidades, chamadas atores, que pode realizar ações em um ambiente de computação sensível a contexto. A principal classe da ontologia é a classe `Actor`, que busca representar de maneira genérica os variados tipos de atores de um ambiente de computação sensível a contexto. A classe `Actor` especifica três tipos de atores, podendo ser estendido de acordo com as necessidades de uma aplicação: pessoas (classe `Person`), grupo (classe `Group`) e organização (classe `Organization`).

Ontologia *Contact*

De acordo com Neto (2006), a ontologia `Contact`, apresentada na Figura 23 do anexo B, descreve informações de contato de um ator por meio da relação `hasContactInformation`, que interliga a classe `act:Actor` à classe `ContactInformation`.

Ontologia *Document*

De acordo com Neto (2006), a ontologia `Document`, apresentada na Figura 24 do anexo B, descreve informações de artefatos produzidos por um ator na forma de documentos físicos ou eletrônicos. Documentos são modelados como entidades que apresentam uma extensão temporal baseada em instantes de tempo. Por isso, esta ontologia importa não apenas à ontologia `Actor`, mas também à ontologia `Time`.

Ontologia *Project*

De acordo com Neto (2006), a ontologia `Project`, apresentada na Figura 25 do anexo B, modela projetos em que atores podem estar envolvidos. Projetos são modelados como eventos com extensão intervalar de tempo, por isso esta ontologia importa não apenas à ontologia `Actor`, mas também à ontologia `TemporalEvent`.

Ontologia *Role*

De acordo com Neto (2006), a ontologia `Role`, embora não aplicada nesta pesquisa, descreve informações sobre o papel social de um ator.

Ontologia *Time*

De acordo com Neto (2006), a ontologia `Time`, apresentada na Figura 27 do anexo B, representa um tipo de informação que envolve conhecimento de senso comum sobre o tempo. Sua principal classe é `TemporalThing` que descreve qualquer tipo de entidade que contenha uma extensão temporal, quer seja uma extensão por instante de tempo (classe `InstantThing`), quer seja uma extensão por intervalo de tempo (classe `IntervalThing`).

Na seção 2.2.4, Especificação e Gerenciamento de Contexto, são descritas seis dimensões que visam responder aos seguintes questionamentos: "**quem** está fazendo **o quê**, em que **local**, em que **momento**, com qual **motivação** e de que **maneira**"? Entre as ontologias do modelo SeCoM descritas anteriormente, algumas possuem semântica que servem como ferramenta para responder à questão anterior. A ontologia *Actor* representa "quem", a ontologia *Time* responde "em que momento". Neste trabalho, o local *(onde)* é conhecido, uma vez que trata-se do ambiente de trabalho do ator com papel de desenvolvedor. E por fim a motivação, que não possui papel importante neste momento. Cada um dos itens citados são considerados elementos contextuais. Para representar o item "o quê", duas ontologias podem ser consideradas. A primeira é a ontologia *Activity* que também faz parte do modelo SeCoM proposto por Neto (2006). A segunda ontologia é a *Extreme Programming Ontology (XPO)*, proposta por Ceravolo et al. (2003), que possui uma classe homônima chamada *Activity* com sub-classes especializadas. Nenhuma das duas serão aplicadas neste trabalho, pois o objetivo do mesmo é reconhecer ações individuais e não atividades.

Outros trabalhos que também exploram a abstração de contexto é o de Escobedo

(2008). Porém, as ontologias desenvolvidas no trabalho são específicas para um ambiente de casa inteligente. Além de Escobedo (2008), existe a ontologia *DOLCE (Descriptive Ontology for Linguistic and Cognitive Engineering)*, proposta por Gangemi et al. (2002), sendo essa mais complexa que as demais.

As ontologias apresentadas até este ponto não são suficientes para contemplar o objetivo geral deste trabalho, pois não permitem representar dados coletados por sensores. Para atingir este objetivo, o desenvolvimento de outras ontologias é necessário. A próxima seção descreve uma metodologia para o desenvolvimento de ontologias.

2.4 DESENVOLVIMENTO DE ONTOLOGIAS

Alguns objetivos específicos deste trabalho, listados na seção 1.3.2, preveem o desenvolvimento de ontologias. Dentre as metodologias consideradas para este trabalho é possível citar a *Methontology*, proposta por FernÁndez, GÓmez-PÉrez e Juristo (); a *TOVE Methodology*, proposta por Gruninger e Fox (1995); a *DILIGENTE Methodology*, proposta por Pinto, Staab e Tempich (2004), e a *Ontology Development 101*, proposta por Noy e McGuinness (2001). Neste trabalho, optou-se por utilizar a metodologia *Ontology Development 101*, proposta por Noy e McGuinness (2001). Os próximos parágrafos descrevem as principais características da *Ontology Development 101*.

Segundo Noy e McGuinness (2001), muitas disciplinas desenvolvem ontologias que profissionais experientes em determinados domínios podem utilizar para compartilhar e anotar informações em suas áreas. Uma ontologia define um vocabulário comum para pesquisadores que precisam compartilhar informações em um domínio, incluindo definições que podem ser interpretadas por máquinas *(machine-interpretable)* de conceitos básicos de um domínio e o relacionamento entre tais conceitos.

Os autores listam algumas razões para se desenvolver ontologias:

- Compartilhar compreensões comuns e da estrutura da informação entre pessoas ou agentes de software, que é o objetivo mais comum no desenvolvimento de ontologias. Os agentes podem usar as informações agregadas para responder perguntas de usuários ou como entrada de dados para outras aplicações.
- Para permitir o reuso do conhecimento de um domínio.
- Para explicitar suposições sobre um domínio. Uma implementação subjacente permite alterações destas suposições facilmente caso os conhecimentos sobre o domínio

mudem.

- Separar conhecimento de domínio de conhecimento operacional, o qual trata-se de um outro uso comum de ontologias.
- Analisar o conhecimento de domínio, o qual é possível uma vez que a especificação formal de termos está disponível.

Para a *Ontology Development 101*, **ontologia** é uma descrição explícita e formal de conceitos sobre um domínio. Inclui as **classes** (às vezes chamadas de **conceitos**), propriedades de cada conceito, os quais descrevem várias características e atributos do conceito (às vezes chamadas de ***slots***, papéis *(**roles**, em inglês)* ou **propriedades**), e restrições em ***slots***. Neste trabalho, ***slots*** são chamados apenas de atributos. Noy e McGuinness (2001) também definem que uma ontologia juntamente com um conjunto de instâncias de uma determinada classe constituem uma **base de conhecimento**. Classes descrevem conceitos de um domínio, podendo ter subclasses que representam conceitos que são mais específicos que a superclasse.

Noy e McGuinness (2001) dizem que o desenvolvimento de ontologias incluem as seguintes etapas:

- Definir classes na ontologia.
- Organizar as classes em uma hierarquia (subclasses e superclasses). Uma subclasse representa um conceito que é um "tipo de" do que a superclasse representa. Os nomes das classes devem representar um conceito em um domínio, e desta forma, sinônimos para um mesmo conceito não devem representar classes diferentes.
- Definir atributos e descrever os valores possíveis destes atributos.
- Preencher os valores desses atributos.

O desenvolvimento de Ontologias é um processo interativo. Os conceitos da ontologia devem estar próximos a objetos físicos ou lógicos e estar relacionados com o domínio explorado. Nas sentenças que descrevem o universo de discurso, os substantivos são os objetos e os verbos são os relacionamentos. Uma ontologia é um modelo da realidade do mundo e os conceitos da ontologia devem refletir esta realidade.

A *Ontology Development 101* define alguns passos para uma metodologia simples da Engenharia do Conhecimento. O primeiro passo consiste em **determinar o domínio e o escopo da ontologia**. Neste passo devem ser respondidas questões básicas como:

- Qual domínio a ontologia deve cobrir?
- Qual o motivo do uso da ontologia?
- Para quais tipos de questões a informação da ontologia devem prover respostas?
- Quem irá usar e manter a ontologia? Se a pessoa que irá manter a ontologia descrever o domínio em uma linguagem diferente da linguagem dos usuários da ontologia, se faz necessário uma mapeamento entre as linguagens.

O segundo passo, considerando de maneira resumida as etapas incluídas na metodologia, consiste em **considerar o reuso de ontologias existentes**. Existem bibliotecas de ontologias para reuso disponíveis na internet e na literatura[1]. O terceiro passo consiste em **enumerar os termos importantes na ontologia**. O quarto passo consiste em **definir as classes e a hierarquia entre elas**, podendo ser usada a abordagem *top-down*, *bottom-up* ou uma **combinação** de ambas. O quinto passo consiste em **definir os atributos das classes**, dado o fato de que as classes isoladas não proveem informações suficientes para responder as perguntas do primeiro passo. O sexto passo consiste em **definir os tipos e valores permitidos dos atributos**. O sétimo e último passo consiste em **criar instâncias das classes** definindo seus atributos, sendo que instâncias individuais representam os conceitos mais específicos na base de conhecimento. Com a criação de instâncias das classes das ontologia, é possível usar a linguagem SPARQL para inferir sobre estas.

2.4.1 SPARQL

SPARQL (2008) é uma linguagem de consulta aplicada sobre dados no formato RDF, sendo que ontologias são instanciadas no formato OWL, o qual trata-se de um formato mais específico de RDF. SPARQL é projetada com a capacidade para expressar restrições sobre os termos de RDF.

2.5 TRABALHOS CORRELATOS

Durante o levantamento bibliográfico, foram analisadas algumas ferramentas de reconhecimento de padrões e sequências, sendo que cada uma delas possui características específicas.

[1] Uma lista de bibliotecas está disponível no apêndice B

Uma das ferramentas, a Análise de Fluxo de Desenvolvimento de Software - (SDSA) *Software Development Stream Analysis* que, segundo Kou (2007) trata-se de uma extensão do *Hackystat* que fornece um arcabouço genérico para organizar os vários tipos de métricas de software capturados pelo primeiro, de maneira apropriada, como entrada para análise de séries temporais baseada em regras. Para isso, o SDSA usa o JESS[2], um sistema baseado em regras. Conjuntos de regras adicionais podem ser definidos para especificar um processo em particular.

O SDSA traduz vários tipos de métricas de produtos e processos de software em um fluxo de desenvolvimento, chamado originalmente de *development stream*, como apresentado na Figura 3. Um fluxo de desenvolvimento é uma coleção de todas as atividades de desenvolvimento ocorridas em uma ordem cronológica. As imagens da Figura 3 qualificadas como *XML Sensor Data* são as métricas efetivamente. As características do *Development Stream* obtidas com o SDSA são relevantes para a pesquisa.

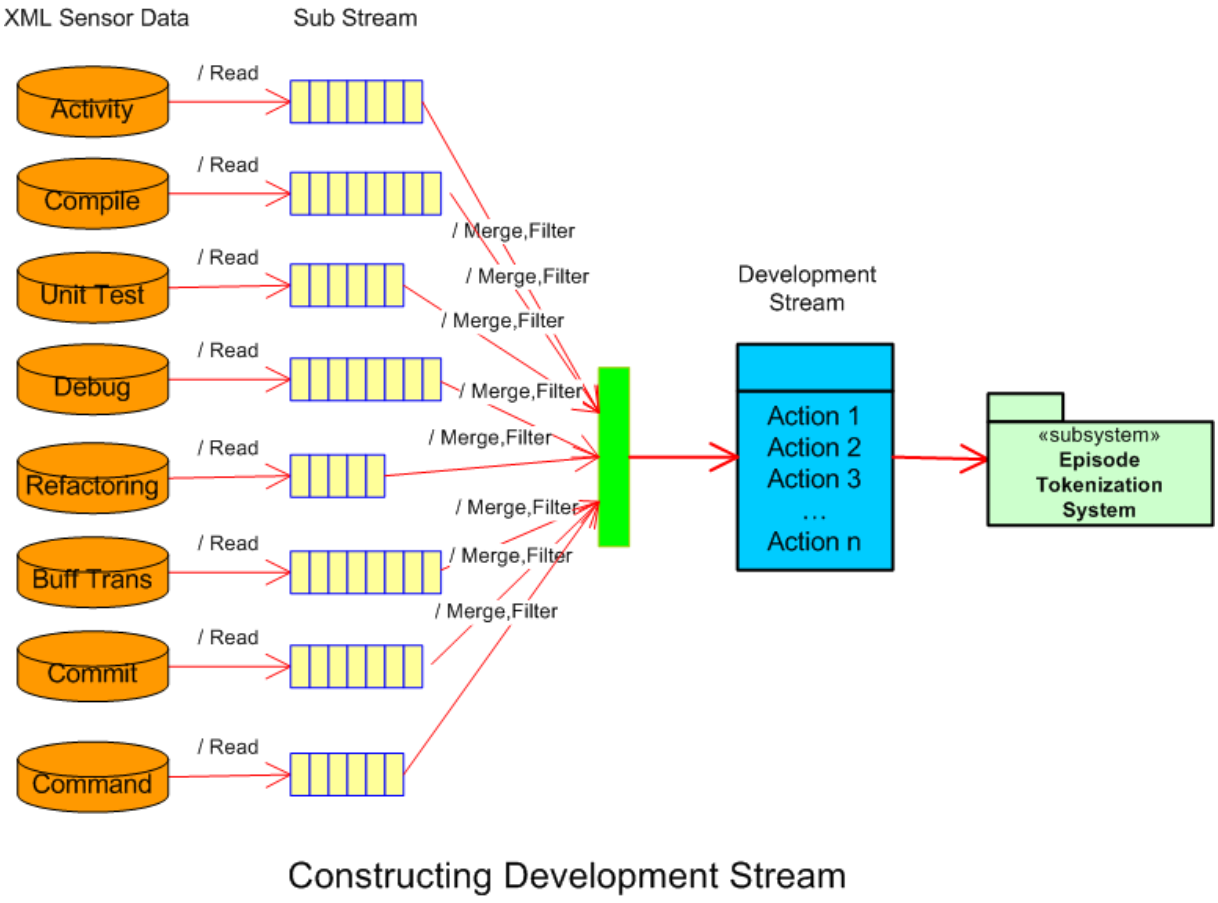

Figura 3: *Hackystat Sensor Data Streaming*

Fonte: Kou (2005)

[2]http://www.jessrules.com/

Zorro, discutido nas publicações de Kou e Johnson (2006), Johnson e Kou (2007), Kou (2007), Kou, Johnson e Erdogmus (2009), é uma ferramenta que, baseada nos dados extraídos com o *Hackystat* e com o auxílio do SDSA e JESS, recupera informações para certificar-se de que um desenvolvedor pratica o TDD TDD durante o processo de desenvolvimento de software. Pedroso (2011) apresenta o Besouro, uma evolução do *Zorro*, que para uma determinada parte da pesquisa solicita consentimento ao desenvolvedor sobre as conclusões em relação ao TDD.

Schlesinger e Jekutsch (2006) apresenta o *ElectroCodeoGram*, uma ferramenta para reconhecimento de padrões, semelhante ao Zorro. Essa implementação tem o conceito de micro-processo, o que difere-se levemente do conceito de "episódio" definido no Zorro. A ferramenta redefine o formato dos dados capturados pelo *Hackystat* para que possam ser consumidos.

O *Esper*[3], citado por Dekkers (2007), Teixeira e Milidiú (2010), Oberhauser (2010) é um motor de código fonte aberto, que combina o processamento de fluxo de eventos e processamento de eventos complexos usando máquina de estado.

Escobedo (2008) descreve em seu trabalho uma casa nomeada "Casa Inteligente" como contexto e ontologias que descrevem semanticamente aquele contexto. Chen, Nugent e Wang (2010) apresentaram um trabalho semelhante ao de Escobedo (2008), mas com sensores adicionais em objetos como, por exemplo, utensílios domésticos.

Chen e Nugent (2009) propõem uma arquitetura para reconhecimento de atividades, usando ontologias para o contexto de sua casa inteligente, descrita no mesmo artigo. Na arquitetura proposta, apresentada na Figura 19 do anexo A, diversos componentes estão conectados ao componente principal que executa o reconhecimento de atividades. Embora seja aplicado o uso de ontologias para representação de uma casa inteligente, o componente principal usa um raciocinador baseado em regras. Chen e Nugent (2009) definiram ainda uma série de ontologias de sensores específicas para o domínio da casa inteligente.

Senin (2010) sugere aplicar técnicas de descoberta de conhecimento e mineração de dados no domínio de engenharia de software para avaliar padrões recorrentes de comportamentos em artefatos coletados em processos de software. O arcabouço *"Software Trajectory"* proposto por Senin (2010), exibido em uma visão de alto nível na Figura 21 do anexo A, funciona de maneira que dados coletados pelo *Hackystat* são convertidos em um formato simbólico intermediário e indexados para uso futuro com mineração de dados.

[3]http://esper.codehaus.org/

Oberhauser (2010) propõe o *CoSEEEK (Context-aware Software Engineering Environment Eventdriven frameworK)*, que é um arcabouço que considera um paradigma baseado em eventos e em espaço/localização. O arcabouço usa ontologias, agentes, extração e processamento de eventos, gerenciamento de contextos, processos e regras, além da ferramenta *Hackystat* e *Esper*. Na época desta escrita, o arcabouço ainda não estava disponível publicamente (OBERHAUSER, 2011). Em uma visão conceitual do CoSEEEK, apresentada na Figura 20 do anexo A, percebe-se que artefatos e ferramentas de engenharia de software estão conectados a diversos componentes. O Componente de extração de eventos *(Event Extraction)* é responsável pela coleta de dados e pelos sensores de eventos em ferramentas de engenharia de software.

Como citado anteriormente, Kou e Johnson (2006), Kou (2007) apresentaram o *Zorro*, que monitora automaticamente o comportamento do desenvolvedor e produz análises que descrevem certas sequências de comportamentos que estão ou não de acordo com o *TDD (Test Driven Development).* Johnson e Kou (2007), por meio do uso de vídeos para registrar o ambiente de desenvolvimento, avaliaram os resultados obtidos com o *Zorro* e compararam com as imagens gravadas. Para avaliação, Johnson e Kou (2007) criaram uma planilha com os resultados das avaliações das imagens geradas e com anotações manuais das atividades executadas pelos desenvolvedores durante a gravação das imagens. Com isso, os autores consideraram que dos 92 episódios sob investigação, 82 foram validados e corretamente classificados, com a precisão de 89%.

2.6 CONSIDERAÇÕES FINAIS

Neste capítulo foram abordados temas como o reconhecimento de ações executadas pelo ator e o uso de sensores para coletar dados das atividades. Uma forma de representar os dados coletados pelos sensores e contexto é através de uso de ontologias. Desta forma, o capítulo descreveu, com mais detalhes, a definição de contexto, incluindo as visões, métodos para extrair as informações de contexto, o comportamento da Computação Sensível ao Contexto e Modelagem. Para dar base a esta pesquisa, os estudos realizados durante uma década foram resumidos e retratados. O capítulo é finalizado com a descrição de ontologias, citando a influência originadas na filosofia, inteligência artificial e engenharia de software. Foi apresentado também o Modelo Semântico de Informações de Contexto e suas ontologias, descrevendo dentre suas principais classes as que são consideradas para a definição da arquitetura proposta. Apresentou-se também a metodologia *Ontology Development 101* para o desenvolvimento de ontologias e, por fim apresentou-se

outros trabalhos correlatos e arquiteturas propostas anteriormente. No próximo capítulo são descritas as etapas do método de pesquisa para o desenvolvimento da arquitetura proposta.

3 ESTRUTURAÇÃO DA PESQUISA

Este capítulo apresenta as etapas do método de pesquisa para o desenvolvimento da arquitetura proposta neste trabalho para que o objetivo geral seja atingido.

Para a construção da arquitetura proposta neste trabalho, foram utilizadas palavras como: *Architecture, Context, context-aware, ontology, sensor*. Foram consideradas pesquisas nas bases *ACM Digital Library, IEEEXplore Digital Library* e *CiteSeerX* realizadas entre os anos de 2006 e 2010, sendo esta pesquisa realizada entre o segundo semestre de 2010 e o primeiro trimestre de 2012. O processo de busca resultou em pouco mais de 300 artigos, sendo que, após uma seleção, de acordo com avaliação do autor deste trabalho, dezenas foram considerados como referência para este trabalho.

A outra parte deste trabalho consiste nos estudos e análises comparativas do conteúdo e enfoques existentes apresentados no capítulo 2. Os conceitos, trabalhos correlatos e informações contidas no capítulo 2 constituem o conteúdo fundamental para a idealização da arquitetura proposta. Por meio do estudo inicial, uma arquitetura foi estabelecida conceitualmente, e realizada por meio do método descrito neste capítulo.

A Figura 4 exibe as etapas do método, sendo que as etapas contidas na região destacada ocorreram mais de uma vez e de maneira não-sequencial para a concepção da arquitetura idealizada durante a revisão da literatura. O método é composto por oito etapas que serão descritas a seguir:

3.1 CONFIGURAÇÃO DE SERVIDOR DE COLETA DE DADOS DE SENSORES

A literatura aponta o uso de sensores como um item essencial no reconhecimento de atividades e como ferramenta para auxiliar na coleta de dados em um contexto definido. A ferramenta *Hackystat* (HACKYSTAT, 2010) apresentada por Johnson et al. (2003) possui diversos sensores anexados a diversas ferramentas do ambiente de desenvolvimento de software como, por exemplo, o IDE, além de ferramentas de construção como o Apache

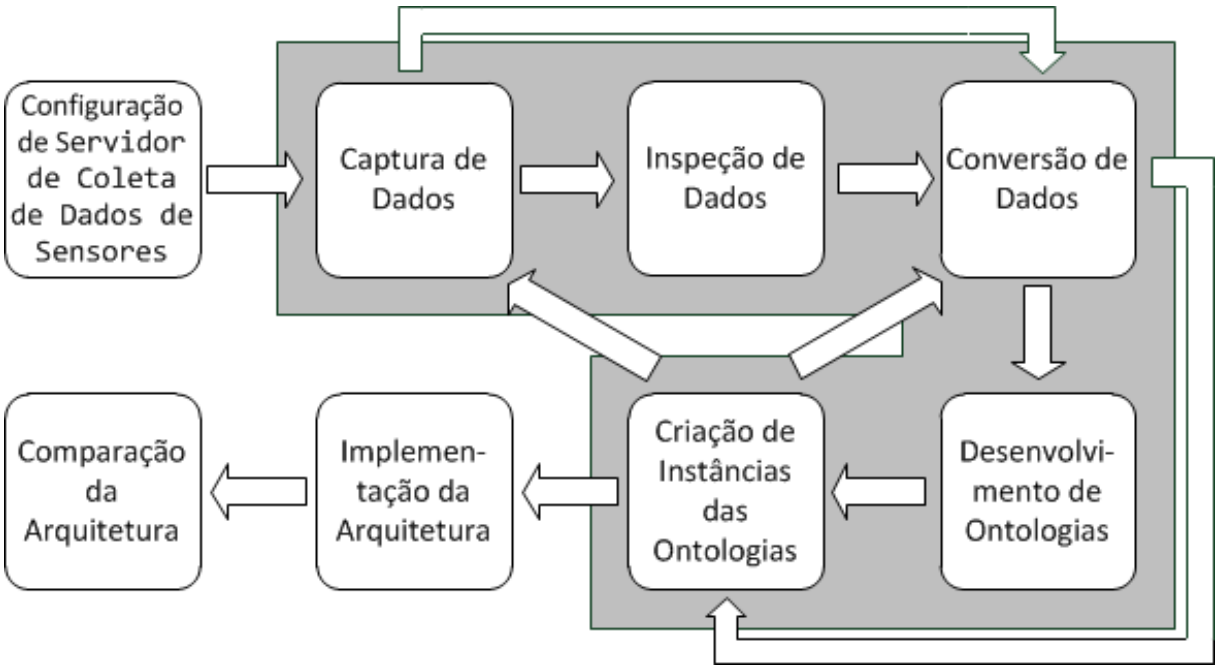

Figura 4: Etapas do Método

Fonte: O Autor

Ant^{TM}, entre outros, que foram usados para coleta de dados das ações do ator. Um repositório central foi configurado[1] com o objetivo de manter os dados coletados por sensores no contexto de desenvolvimento de software.

3.2 CAPTURA DE DADOS

O objetivo desta etapa foi produzir ações para que sensores pudessem coletar dados e enviar estes para o repositório central da ferramenta *Hackystat*. Com o servidor instalado e disponível para uso, um sensor[2] foi então adicionado no ambiente de desenvolvimento para coleta de dados das ações executadas na ferramenta Eclipse (ECLIPSE, 2010).

Com o sensor configurado e disponível para uso, foram desenvolvidos códigos de programas de computador de vários tipos, por exemplo, programas com algoritmos para mineração de dados, programas para persistência de dados, entre outros. Assim como novos programas foram desenvolvidos, as manutenções em programas anteriores também foi executada. Nas duas condições, foi utilizada a linguagem de programação Java[3].

[1]De acordo com as orientações disponíveis no sítio oficial do *Hackystat* (https://code.google.com/p/hackystat/wiki/InstallationGuide)

[2]A instalação do sensor (SENSOR, 2010) foi executada seguindo as orientações disponíveis no sítio oficial do sensor (https://code.google.com/p/hackystat-sensor-eclipse/wiki/InstallationGuide)

[3]http://www.oracle.com/technetwork/java/index.html

3.3 INSPEÇÃO DE DADOS

O objetivo desta etapa foi compreender o formato e o conteúdo dos dados gerados pelos sensores durante a etapa "Captura de Dados". Após o processo de desenvolvimento para a captura de dados, foi feita uma análise humana dos dados coletados pelos sensores.

A análise foi executado através de consultas executadas no banco de dados do servidor central, configurado na etapa "Configuração de Servidor de Coleta de Dados de Sensores". Em um primeiro momento foram avaliadas as tabelas e suas estruturas e, em seguida, os dados contidos em cada tabela. Por meio desta análise, descobriu-se que a ferramenta *Hackystat* concentra os dados fundamentais para a arquitetura proposta em duas tabelas. A primeira contém dados de atores e projetos aos quais os atores estão associados, e a segunda contém os dados emitidos pelos sensores.

Os dados, na forma apresentada pelo *Hackystat*, estavam em um formato que não era facilmente interpretável, pois tratam-se de dados abstratos limitados a várias ocorrências de pares chave e valor. Na análise destes pares de chave e valor, concluiu-se que muitas informações poderiam ser obtidas. Por isso, houve a necessidade de converter estes dados, como será descrito na próxima seção.

3.4 CONVERSÃO DE DADOS

O objetivo desta etapa foi transformar os dados do formato original para um formato que permitisse a leitura e compreensão de maneira mais simples por meio da análise humana. Além disso, buscou-se também facilitar o processamento posterior. Com este objetivo definido, os dados foram convertidos para um formato intermediário que permitiu a partição mais clara e a classificação dos dados de uma ação.

Foram extraídos os seguintes dados da ação: o momento em que ocorreu a ação, o ator que executou a ação, o recurso (artefato de software) associado que está envolvido diretamente na ação, as chaves e seus respectivos valores que, em conjunto, possuem algum significado relevante na ação. O uso de chaves nos dados coletados denotam termos associados a atributos dentro do código do sensor. Durante a etapa de extração, os dados também foram classificados, o que resultou em quatro tabelas em um repositório dedicado à arquitetura, contendo dados do recurso, das chaves geradas pelos sensores, os valores para cada chave gerada e o momento em que ocorreu a ação.

Decorrente da atividade de conversão e classificação, observou-se a necessidade de

ontologias para expressar a semântica dos dados coletados.

3.5 DESENVOLVIMENTO DE ONTOLOGIAS

Nesta etapa foram desenvolvidas ontologias com o objetivo de dar significado para os dados coletados pelos sensores. Além disso, foi desenvolvida uma ontologia com o objetivo de integrar ontologias de sensores com ontologias que representam contexto e processo de software. Foi utilizada a metodologia *Ontology Development 101*, proposta por Noy e McGuinness (2001), descrita na seção 2.4.

A metodologia *Ontology Development 101* descreve alguns passos que foram considerados nesta etapa. O **domínio** da ontologia é constituído pelos dados de sensores, e a **motivação** é dar significado para os dados coletados. O passo **enumerar os termos importantes na ontologia** foi aplicado ao observar os dados gerados pelos sensores, nos quais alguns termos utilizados como **chaves** de algumas ações resultaram em **conceitos importantes**. O passo **definir atributos das classes** foi aplicado observando chaves geradas pelos sensores. Não foram utilizadas todas as chaves porque os sensores geraram dezenas delas com dezenas de combinações de valores; apenas as mais significativas para a interpretação do analista/modelador foram consideradas. O passo **definir os tipos e valores permitidos dos atributos** não foi inicialmente considerado, pois nem todas as chaves eram conhecidas e seus respectivos valores não apresentavam um formato específico. O passo **criar instâncias das classes** foi aplicado na etapa seguinte nomeada "Criar Instâncias das Ontologias".

3.6 CRIAÇÃO DE INSTÂNCIAS DAS ONTOLOGIAS

O objetivo desta etapa foi dar significado aos dados coletados pelos sensores através do uso de ontologias. Com base nas ontologias desenvolvidas na etapa anterior foram lidos os dados gerados pelos sensores que já haviam sido convertidos para o formato intermediário. Para cada dado que denotou uma ação compatível com uma ontologia desenvolvida, foi criada uma instância da classe e persistida no repositório dedicado à arquitetura. Ao final desta etapa, os dados coletados por sensores passaram a ser associados a um significado através do uso de ontologias, e puderam ser inferidos através do uso de um motor de inferência em conjunto com esta nova representação.

3.7 IMPLEMENTAÇÃO DA ARQUITETURA

Nesta etapa, o objetivo foi implementar uma arquitetura, considerando um modelo conceitual abstrato, que permitisse a inclusão de novos sensores sem modificar o núcleo da arquitetura proposta. A arquitetura deve reconhecer atores e ações executadas pelos atores em um contexto definido.

Com base nos trabalhos pesquisados e nas etapas anteriores, identificou-se que além das ontologias desenvolvidas na etapa "Desenvolvimento de Ontologias" havia a necessidade de outros componentes na arquitetura proposta. Estes componentes são como ontologias de domínio que representam as informações de atores, projetos, momento em que ocorreu uma ação, além da representação de contexto e de processos de software e um "motor de inferência" para realizar consultas sobre estas ontologias.

Com os componentes e informações foi possível a identificação e representação do ator que executou determinada ação, em que momento a ação ocorreu e sob qual contexto. O contexto, como definido anteriormente, está limitado ao "Processo de Software". A ação executada é dependente do sensor e varia conforme o objetivo, dados coletados e características particulares do sensor.

Para que novos sensores pudessem ser adicionados, sem alterar o núcleo da arquitetura, concluiu-se que uma ontologia integradora seria a forma ideal de conectar novas ontologias de sensores. A ontologia integradora seria o caminho utilizado pela arquitetura para fazer inferências sobre as ontologias dos sensores. As informações acerca do ator, do projeto e do momento em que ocorreu a ação ficam definidas na ontologia integradora. A ação, que é particular de cada sensor, deve ser definida na ontologia do sensor.

A última etapa do método foi avaliar a arquitetura, que consistia na avaliação de características requeridas consideradas fundamentais para a arquitetura proposta.

3.8 COMPARAÇÃO DA ARQUITETURA

Durante esta etapa foram realizados experimentos, descritos no capítulo 5, com o objetivo de avaliar se a arquitetura proposta capturava os dados coletados pelos sensores através do uso de ontologias. Nesta etapa, a arquitetura proposta também foi comparada com as arquiteturas já existentes, descritas na seção 2.5. Bolchini et al. (2007) consideraram uma série de itens para a comparação de arquitetura para a computação sensível ao contexto, sendo alguns descritos na seção 5.8. Além dos itens indicados por Bolchini

et al. (2007), a arquitetura é comparada considerando características como:

- A arquitetura é extensível? Extensível no sentido de que novos sensores podem ser adicionados sem alteração no núcleo da arquitetura proposta.
- A arquitetura consegue identificar eventos que venham de sensores variados? Sensores variados no sentido de que a arquitetura não é dependente de um tipo ou tipos particulares de sensores.
- Regras de inferência são definidas no código fonte da implementação?
- Identificação das ações executadas pelo desenvolvedor.

3.9 CONSIDERAÇÕES FINAIS

Este capítulo descreveu as etapas do método utilizado na concepção da arquitetura, considerando o conteúdo avaliado na revisão da literatura. O capítulo descreveu também os métodos que foram aplicados no desenvolvimento de novas ontologias, incluindo a etapa na qual foram representados os dados coletados por sensores através do uso de ontologias. As principais etapas foram a captura, inspeção e conversão de dados, bem como o desenvolvimento e criação de instâncias de ontologias. A primeira de todas as etapas do método consistiu na revisão e avaliação da literatura, apresentada no capítulo 2.

No próximo capítulo é apresentada uma das possíveis implementações da arquitetura proposta com detalhes mais aprofundados.

4 ARQUITETURA PROPOSTA

Embora existam propostas para algumas arquiteturas, conforme apresentado no item 2.5, algumas estão concebidas apenas em modelos formais e não implementadas. Outras estão apenas concebidas teoricamente através de um modelo conceitual e não estão disponíveis. Isto motivou a elaboração de uma nova arquitetura que é apresentada neste capítulo.

Atribuiu-se o nome **eXTadEi** *(lê-se équistádi)*, que é um acrônimo para três conceitos principais, em inglês, usados pela arquitetura, são eles: *Extractor*, *Task*, *Development*, o (i) é de implementação. Além disso, a pronúncia fica próxima do termo *Hackystat*, ferramenta usada pela arquitetura. A partir desse ponto, ao ler **eXTadEi**, trata-se da arquitetura proposta.

4.1 IMPLEMENTAÇÃO EXTADEI

A Figura 5 apresenta um visão de alto nível da arquitetura proposta neste trabalho. A arquitetura é composta por três conjuntos de ontologias divididos em camadas. O primeiro conjunto, chamado neste de trabalho de Ontologia de Nível Superior, está no nível mais alto. O segundo conjunto, chamado neste trabalho de Ontologia de Integração, está localizado entre as ontologias de nível mais alto e as ontologias de sensores. O terceiro e último conjunto de ontologias, chamado neste trabalho de ontologia de sensores, está localizado no nível mais inferior, logo abaixo da Ontologia de Integração. Além dos conjuntos de ontologias, a arquitetura proposta possui um Motor de Inferência. Os conjuntos de ontologias e o motor de inferência são detalhados a seguir.

4.1.1 Ontologia de Nível Superior

No nível mais alto da arquitetura está a ontologia nomeada “Ontologia de Nível Superior”. A ontologia é composta por classes que representam Contexto e Processos de Software.

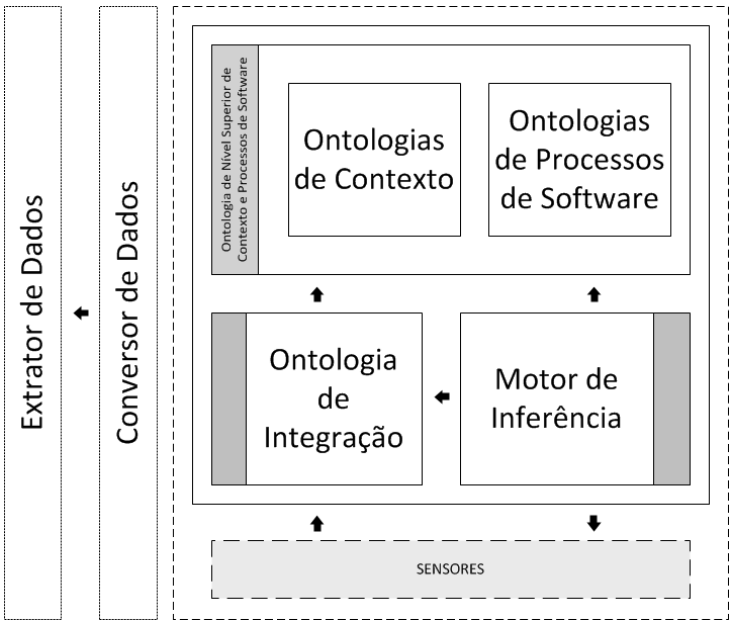

Figura 5: Arquitetura eXTadEi

Fonte: O Autor

As "Ontologias de Contexto" descrevem informações do ator, do projeto ao qual o ator está associado, do papel do ator, informações temporais e de documentos que representam artefatos de software. Todas as ontologias que compõem a ontologia chamada neste trabalho de "Ontologias de Contexto" fazem parte do Modelo SeCoM, descrito na seção 2.3.2. A escolha das classes que fariam parte da "Ontologia de Nível Superior" deu-se durante a etapa "Inspeção de Dados" do método, descrita na seção 3.3. A investigação dos dados resultou no conteúdo da Listagem 4.1.

A interpretação humana do conteúdo da Listagem 4.1 descreve um projeto criado pelo ator "josivanps@gmail.com" cuja descrição é "The default Project", no qual o projeto tem duração de pouco mais de quinze anos, pois o momento inicial ocorreu em 01/01/2000 e o momento final deverá ocorrer em 03/05/2015. Concluiu-se também que estes dados possuem o mesmo significado de algumas ontologias do Modelo SeCoM, definido por Neto (2006), descrito na seção 2.3.2. É importante ressaltar que o prazo definido para este projeto trata-se de um cenário inicialmente fictício.

Listagem 4.1: Dados do Projeto

```
1 <Project
2   LastMod="2010-05-04T00:02:00.991-03:00"
3   Name="Default">
4   <Description>The default Project</Description>
5   <StartTime>2000-01-01T00:02:00.945-03:00</StartTime>
```

```
6   <EndTime>2015-05-03T00:02:00.945-03:00</EndTime>
7   <Owner>josivanps@gmail.com</Owner>
8 </Project>
```

As classes do Modelo SeCoM consideradas na "Ontologia de Nível Superior" fazem parte das seguintes ontologias: **Ontologia *Actor***, **Ontologia *Contact***, **Ontologia *Role***, **Ontologia *Project***, **Ontologia *Time*** e **Ontologia *Document***. Estas ontologias foram descritas na seção 2.3.2.

As ontologias do Modelo SeCoM possuem semântica que permitem o alinhamento com o conteúdo da Listagem 4.1 da seguinte maneira: a *tag*[1] `<Project>` (na linha 1) descreve um indivíduo do tipo *Project*. As *tags* `<StartTime>` (na linha 5) e `<EndTime>` (na linha 6) descrevem indivíduos do tipo *TimeInstant*. A *tag* `<Owner>` (na linha 7) pode ser representada como um indivíduo do tipo *Actor* em conjunto com uma *Role* específica. O relacionamento entre o dado coletado pelo sensor e a ontologia é demonstrado na Figura 6.

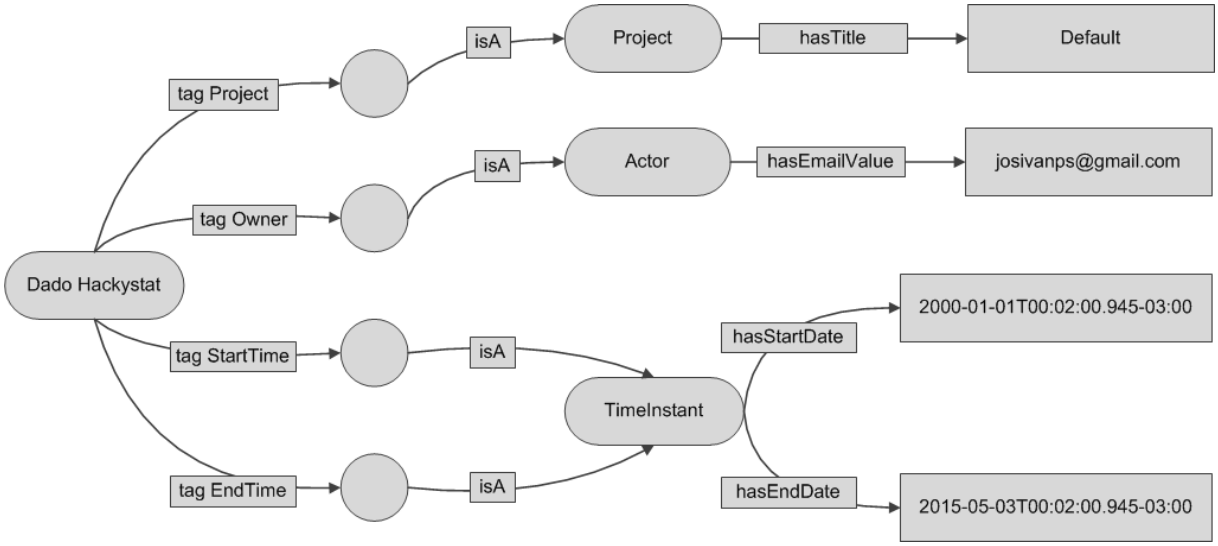

Figura 6: Dados coletados como instâncias da ontologia

Fonte: O Autor

As ontologias de Processo de Software, propostas por Falbo (1998), Bertollo (2006), foram usadas de maneira restrita na implementação da arquitetura apresentada neste trabalho. Estas ontologias são consideradas parte integrante da "Ontologia de Nível Superior" para que outras implementações tenham acesso às classes definidas pelas ontologias, uma vez que a arquitetura proposta neste trabalho foi concebida para além dos experimentos

[1] Também pode ser chamada de marcação ou etiqueta.

descritos no capítulo 5.

Como explicado previamente na seção 3.7 "Implementação da Arquitetura", para que novos sensores pudessem ser adicionados, sem alterar o núcleo da arquitetura, concluiu-se que uma ontologia integradora seria a forma ideal de conectar novas ontologias de sensores. Com isso foi desenvolvida uma ontologia nomeada "Ontologia de Integração".

4.1.2 Ontologia de Integração

A "Ontologia de Integração", desenvolvida durante a execução deste trabalho, tem como objetivo tornar a arquitetura proposta extensível, no sentido de permitir que novos sensores sejam adicionados sem alterar o núcleo da arquitetura.

A Figura 7 apresenta um diagrama com as classes da "Ontologia de Integração", e a Figura 12 do apêndice A exibe outra representação destas classes. As classes na cor cinza são classes que já existem, e a arquitetura proposta neste trabalho fez o reuso das mesmas. As classes na cor branca são classes novas criadas durante a implementação da arquitetura proposta neste trabalho.

A ontologia é composta de duas classes: *BasicEvent* e *BasicAction*. A primeira descreve, de maneira básica, um evento. Indivíduos desta classe possuem propriedades que definem um ator *(hasActor)*; o instante do evento *(hasInstant)*; o projeto ao qual este evento está associado *(hasAssignedProject)* e a ação que ocorreu *(hasAction)*. Na propriedade *hasActor* é utilizado um indivíduo do tipo *Actor*. Na propriedade *hasInstant* é utilizado um indivíduo do tipo *TimeInstant*. Na propriedade *hasAssignedProject* é utilizado um indivíduo do tipo *Project*. Estas classes foram descritas nas seção 2.3.2. Na propriedade *hasAction* é utilizado um indivíduo do tipo *BasicAction* descrito a seguir.

A classe *BasicAction* representa uma ação executada pelo ator no processo de software [por exemplo, construção *(Build)* de um artefato]. Esta deve ser a classe mais abstrata para todas as possíveis classes que representem ações que a arquitetura deve reconhecer.

Na Figura 7 também são apresentadas classes das "Ontologias de Contexto" consideradas na "Ontologia de Nível Superior". Embora a "Ontologia de Integração" não use explicitamente as classes *Text* e *Role*, exibidas na Figura 7, a ontologia *Hackystat* descrita na seção 5.7.1, resultante deste trabalho, utiliza a classe *Text* e a classe *Role*. A classe *Text* faz parte da Ontologia *Document* e a classe *Role* faz parte da Ontologia *Role*, ambas descritas na seção 2.3.2.

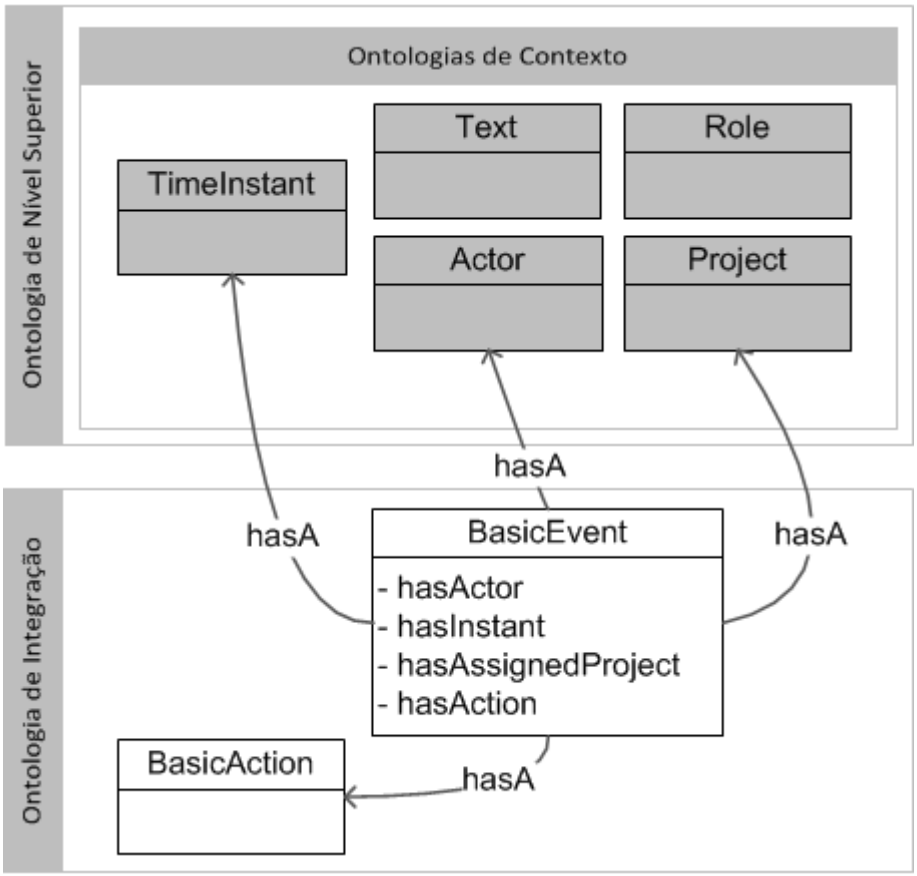

Figura 7: Ontologia de Integração

Fonte: O Autor

4.1.3 Motor de Inferência

Shaw (1987) descreve, em sua pesquisa, que o papel do motor de inferência é aplicar o conhecimento existente na base de conhecimento para formular decisões bem como simples inferências dedutivas onde o sistema tem de ser capaz de trabalhar com dados incompletos e a incerteza introduzida pela degradação da condição de execução. É o motor de inferência que faz o raciocínio e que deve processar regras armazenadas na base de conhecimento para a sua conclusão final, bem como através de listas de combinações para encontrar condições que tornam certas regras dadas como verdadeiras. Na arquitetura proposta, o motor de inferência tem essencialmente o mesmo papel, ou seja, de verificar respostas através do uso de consultas SPARQL[2].

Neste trabalho foi utilizado o Apache Jena (2012), tornando desnecessária qualquer implementação para o motor de inferência, que foi utilizado para inferir sobre as classes da "Ontologia de Nível Superior" e da "Ontologia de Integração". O motor de inferên-

[2]Uma linguagem para realização de consultas em um repositório de dados no formato OWL.

cia Jena foi descrito anteriormente por McBride (2002), e tem como objetivo facilitar o desenvolvimento de aplicações que usem modelos e linguagens da web semântica. Jena suporta a criação, manipulação e consultas em grafos RDF.

4.1.4 Extrator e Conversor de Dados

Embora não façam partes da arquitetura proposta neste trabalho, os componentes de extração e coleta de dados foram aplicados na implementação atual. O extrator tem a função de extrair os dados coletados pelos sensores para um formato intermediário que permitisse a manipulação dos dados sem danificar os dados originais. O conversor, por sua vez, tem a função de converter os dados extraídos anteriormente para ontologias que estejam de acordo com a ontologia de integração descrita anteriormente.

4.2 CONSIDERAÇÕES FINAIS

Este capítulo apresentou a arquitetura **eXTadEi**, que é uma arquitetura abstrata que permite adicionar novos sensores e suas respectivas ontologias sem alterar o núcleo da arquitetura. Cada componente do núcleo, que são as "Ontologias de Nível Superior", "Ontologia de Integração" e o "Motor de Inferência" foi detalhado. No próximo capítulo são apresentados os experimentos e a avaliação.

5 EXPERIMENTOS E COMPARAÇÃO

Este capítulo apresenta os experimentos realizados para avaliar se a arquitetura reconhece ações executadas pelo ator através do uso de ontologias e compará-la com outras arquiteturas. Os experimentos descritos neste capítulo podem ser tratados como testes para validação de dados coletados por sensores.

Nesta pesquisa, a intenção é reconhecer a ação executada por um ator em determinado momento. Para esta pesquisa, o contexto foi limitado ao processo de desenvolvimento de software, embora possa ser muito mais genérico do que o domínio aplicado. A literatura indica que muitas atividades são executadas pelo ator durante o processo de desenvolvimento de software. Dentre elas, para este trabalho, foi considerada a atividade de refatoração. Murphy-Hill, Parnin e Black (2009) citam que atores geralmente fazem refatoração, geralmente sem o auxílio de assistentes do IDE. A mesma pesquisa aponta que, de maneira geral, o assistente de refatoração mais usado no IDE Eclipse é Renomear *(Rename, em inglês)*. Este assistente representa 74,8% do universo de dados coletados na pesquisa de Li et al. (2010). Fowler (2004) descreve dois significados para a palavra Refatoração, nas quais a primeira trata da forma substantiva e a segunda trata da forma verbal. Na forma substantiva, refatoração é definida como:

> "Uma alteração feita na estrutura interna do software para torná-lo mais fácil de ser entendido e menos custoso de ser modificado sem alterar seu comportamento observável."

Na forma verbal, refatoração é definida como:

> "Reestruturar software aplicando uma série de refatorações sem alterar seu comportamento observável."

Entre diversas refatorações que envolvem a ação renomear, Fowler (2004) catalogou apenas a refatoração **Renomear Método** com a seguinte motivação:

> "Métodos devem ser nomeados de uma maneira que comuniquem sua intenção. Uma boa maneira de fazer isso é pensar em como seria o comentário do método e transformar esse comentário no nome do método."

Com esta motivação decidiu-se executar experimentos envolvendo a refatoração de código, principalmente a refatoração "Renomear Método". Foram realizados seis experimentos, sendo que o objeto dos experimentos foi interpretar o conteúdo gerado pelos sensores e, com a interpretação humana destes resultados, desenvolver ontologias para representar cada dado capturado. O processo de interpretação humana pode ser descrito como a ação de interpretar manualmente e, na condição de especialista, os dados capturados. Todos os experimentos foram realizados utilizando o ambiente de desenvolvimento Eclipse (ECLIPSE, 2010) e a linguagem de programação Java (JAVA, 2010).

5.1 EXPERIMENTO 1 – CRIAR CLASSE

O primeiro experimento tinha como objetivo avaliar o conteúdo gerado pelo sensor durante a ação criar classe. Para executar o experimento, foi usado o assistente do ambiente de desenvolvimento Eclipse para a criação de uma nova classe nomeada *ClassToRefactor*. O primeiro experimento pode ser considerado uma preparação para os demais.

A Listagem 5.1 apresenta o conteúdo resumido dos dados gerados pelo sensor para a ação do Experimento 1 (criar classe *ClassToRefactor*).

Listagem 5.1: Dado extraído por sensor durante o experimento Criar classe

```
<Properties>
  <Property>
    <Key>Subtype</Key>
    <Value>ProgramUnit</Value>
  </Property>
  <Property>
    <Key>Type</Key>
    <Value>Edit</Value>
  </Property>
  <Property>
    <Key>Language</Key>
    <Value>java</Value>
  </Property>
  <Property>
    <Key>Unit-Type</Key>
```

```
16      <Value>Class</Value>
17    </Property>
18    <Property>
19      <Key>Unit-Name</Key>
20      <Value>ClassToRefactor.java</Value>
21    </Property>
22    <Property>
23      <Key>Subsubtype</Key>
24      <Value>Add</Value>
25    </Property>
26  </Properties>
```

Após interpretação humana da Listagem 5.1 conclui-se que esta ação refere-se à edição de uma nova unidade de programa, do tipo classe, usando a linguagem java, nomeada `ClassToRefactor.java`. Estas informações estão explícitas no trecho de código da Listagem 5.1. Considerando os pares chave e valor, tendo como chave o conteúdo das *tags* `<Key>` e valor o conteúdo das *tags* `<Value>` entre as linhas 3 e 24, concluiu-se que houve a **edição** *(Type → Edit)* em uma **unidade de programa** *(Subtype → ProgramUnit)* com a **adição** *(Subsubtype → Add)* de uma **classe** *(Unit-Type → Class)* **nomeada** *(Unit-Name → `ClassToRefactor.java`)* usando a **linguagem java** *(Language → java)*.

5.2 EXPERIMENTO 2 – ADICIONAR MÉTODO

O segundo experimento tinha como objetivo avaliar o conteúdo gerado pelo sensor durante a ação adicionar método na classe criada no Experimento 1. Para executar o experimento, a declaração do método, nomeado *methodToRefactor*, foi implementada sem o auxílio de assistentes do ambiente de desenvolvimento Eclipse. Ou seja, foi digitado diretamente no arquivo com o código fonte.

A Listagem 5.2 apresenta o conteúdo resumido dos dados gerados pelo sensor para a ação do Experimento 2 (adicionar método *methodToRefactor*).

Listagem 5.2: Dado extraído por sensor durante o experimento Adicionar método

```
1 <Properties>
2   <Property>
3     <Key>Subtype</Key>
4     <Value>ProgramUnit</Value>
5   </Property>
6   <Property>
```

```
7       <Key>Type</Key>
8       <Value>Edit</Value>
9     </Property>
10    <Property>
11      <Key>Language</Key>
12      <Value>java</Value>
13    </Property>
14    <Property>
15      <Key>Unit-Type</Key>
16      <Value>Method</Value>
17    </Property>
18    <Property>
19      <Key>To-Unit-Name</Key>
20      <Value>void methodToRefactor()</Value>
21    </Property>
22    <Property>
23      <Key>From-Unit-Name</Key>
24      <Value>methodToRefactor</Value>
25    </Property>
26    <Property>
27      <Key>Subsubtype</Key>
28      <Value>Rename</Value>
29    </Property>
30  </Properties>
```

Após interpretação humana da Listagem 5.2, concluiu-se que esta ação refere-se à edição de unidade de programa, do tipo método, usando a linguagem java, renomeando-o de `methodToRefactor` para `void methodToRefactor()`. Estas informações estão explícitas no trecho de código da Listagem 5.2. Considerando os pares chave e valor, tendo como chave o conteúdo das *tags* `<Key>` e valor o conteúdo das *tags* `<Value>` entre as linhas 3 e 28, concluiu-se que houve a **edição** *(Type → Edit)* em uma **unidade de programa** *(Subtype → ProgramUnit)* com a **renomeação** *(Subsubtype → Rename)* de um **método** *(Unit-Type → Method)* **renomeando de** *(From-Unit-Name →* `methodToRefactor`*)* **para** *(To-Unit-Name →* `void methodToRefactor()`*)* usando a **linguagem java** *(Language → java)*.

Como o Experimento 2 foi implementando sem o uso de assistentes, o sensor disparou várias vezes conteúdos semelhantes ao da Listagem 5.2, e foi alterado somente o valor (conteúdo da *tag* `<Value>`) das chaves (conteúdo da *tag* `<Key>`) *From-Unit-Name*, *To-Unit-Name* e *Unit-Type*. O sensor reconheceu o elemento como um método somente

quando a abertura e fechamento de parênteses ocorreu. Antes disso, para o sensor, tratava-se de um campo que foi renomeado várias vezes.

5.3 EXPERIMENTO 3 – ADICIONAR ATRIBUTO

O terceiro experimento tinha como objetivo avaliar o conteúdo gerado pelo sensor durante a ação adicionar atributo na classe criada no Experimento 1. O experimento foi divido em duas partes. Na primeira parte do experimento, a declaração do atributo, nomeado *fieldToRefactor*, foi implementada sem o auxílio de assistentes do ambiente de desenvolvimento Eclipse. Ou seja, foi digitado diretamente no arquivo com o código fonte. Na segunda parte do experimento foi utilizado o assistente do ambiente de desenvolvimento Eclipse para gerar dois métodos relacionados ao atributo adicionado, sendo os métodos *setFieldToRefactor* e *getFieldToRefactor*.

A Listagem 5.3 apresenta o conteúdo resumido dos dados gerados pelo sensor para a ação da primeira parte do Experimento 3 (adicionar atributo *fieldToRefactor*).

Listagem 5.3: Dado extraído por sensor durante o experimento Adicionar atributo

```
<Properties>
  <Property>
    <Key>Subtype</Key>
    <Value>ProgramUnit</Value>
  </Property>
  <Property>
    <Key>Type</Key>
    <Value>Edit</Value>
  </Property>
  <Property>
    <Key>Language</Key>
    <Value>java</Value>
  </Property>
  <Property>
    <Key>Unit-Type</Key>
    <Value>Field</Value>
  </Property>
  <Property>
    <Key>Unit-Name</Key>
    <Value>String fieldToRefactor</Value>
  </Property>
  <Property>
```

```
23      <Key>Subsubtype</Key>
24      <Value>Add</Value>
25    </Property>
26 </Properties>
```

Após interpretação humana da Listagem 5.3 conclui-se que esta ação refere-se à edição de uma nova unidade de programa, do tipo campo, usando a linguagem java, nomeada `String fieldToRefactor`. Estas informações estão explícitas no trecho de código da Listagem 5.3. Considerando os pares chave e valor, tendo como chave o conteúdo das *tags* `<Key>` e valor o conteúdo das *tags* `<Value>` entre as linhas 3 e 24, concluiu-se que houve a **edição** *(Type → Edit)* em uma **unidade de programa** *(Subtype → ProgramUnit)* com a **adição** *(Subsubtype → Add)* de um **campo** *(Unit-Type → Field)* **nomeado** *(Unit-Name → `String fieldToRefactor`)* usando a **linguagem java** *(Language → java)*.

5.4 EXPERIMENTO 4 – RENOMEAR ATRIBUTO

O quarto experimento tinha como objetivo avaliar o conteúdo gerado pelo sensor durante a ação renomear o atributo adicionado no Experimento 3. Para executar o experimento, o atributo adicionado no Experimento 3 foi renomeado de *fieldToRefactor* para *refactoredField*, utilizando assistentes do ambiente de desenvolvimento Eclipse.

A Listagem 5.4 apresenta o conteúdo resumido dos dados gerados pelo sensor para a ação do Experimento 4 (renomear atributo).

Listagem 5.4: Dado extraído por sensor durante o experimento Renomear atributo

```
1 <Properties>
2   <Property>
3     <Key>Subtype</Key>
4     <Value>ProgramUnit</Value>
5   </Property>
6   <Property>
7     <Key>Type</Key>
8     <Value>Edit</Value>
9   </Property>
10  <Property>
11    <Key>Language</Key>
12    <Value>java</Value>
13  </Property>
14  <Property>
15    <Key>Unit-Type</Key>
```

```
16      <Value>Field</Value>
17    </Property>
18    <Property>
19      <Key>To-Unit-Name</Key>
20      <Value>String refactoredField</Value>
21    </Property>
22    <Property>
23      <Key>From-Unit-Name</Key>
24      <Value>fieldToRefactor</Value>
25    </Property>
26    <Property>
27      <Key>Subsubtype</Key>
28      <Value>Rename</Value>
29    </Property>
30  </Properties>
```

Após interpretação humana da Listagem 5.4 conclui-se que esta ação refere-se à edição de unidade de programa, do tipo campo, usando a linguagem java, renomeando-o de `fieldToRefactor` para `String refactoredField`. Estas informações estão explícitas no trecho de código da Listagem 5.4. Considerando os pares chave e valor, tendo como chave o conteúdo das *tags* `<Key>` e valor o conteúdo das *tags* `<Value>` entre as linhas 3 e 28, conclui-se que houve a **edição** *(Type → Edit)* em uma **unidade de programa** *(Subtype → ProgramUnit)* com a **renomeação** *(Subsubtype → Rename)* de um **campo** *(Unit-Type → Field)* **renomeando de** *(From-Unit-Name →* `fieldToRefactor`*)* **para** *(To-Unit-Name →* `String refactoredField`*)* usando a **linguagem java** *(Language → java)*.

5.5 EXPERIMENTO 5 – RENOMEAR MÉTODO

O quinto experimento tem como objetivo avaliar o conteúdo gerado pelo sensor durante a ação renomear o método adicionado no Experimento 2. Para executar o experimento, o método adicionado no Experimento 2 foi renomeado de *methodToRefactor* para *refactoredMethod*. Este experimento foi implementado sem o auxílio de assistentes do ambiente de desenvolvimento Eclipse, ou seja, foi digitado diretamente no arquivo com o código fonte.

A Listagem 5.5 apresenta o conteúdo dos dados gerados pelo sensor para a ação do Experimento 5 (renomear método).

Listagem 5.5: Dado extraído por sensor durante o experimento Renomear método

```
1  <Properties>
2    <Property>
3      <Key>Subtype</Key>
4      <Value>ProgramUnit</Value>
5    </Property>
6    <Property>
7      <Key>Type</Key>
8      <Value>Edit</Value>
9    </Property>
10   <Property>
11     <Key>Language</Key>
12     <Value>java</Value>
13   </Property>
14   <Property>
15     <Key>Unit-Type</Key>
16     <Value>Method</Value>
17   </Property>
18   <Property>
19     <Key>To-Unit-Name</Key>
20     <Value>void refactoredMethod()</Value>
21   </Property>
22   <Property>
23     <Key>From-Unit-Name</Key>
24     <Value>methodToRefactor()</Value>
25   </Property>
26   <Property>
27     <Key>Subsubtype</Key>
28     <Value>Rename</Value>
29   </Property>
30 </Properties>
```

Após interpretação humana da Listagem 5.5 conclui-se que esta ação refere-se à edição de unidade de programa, do tipo método, usando a linguagem java, renomeando-o de `fieldToRefactor` para `String refactoredField`. Estas informações estão explícitas no trecho de código da Listagem 5.5. Considerando os pares chave e valor, tendo como chave o conteúdo das *tags* `<Key>` e valor o conteúdo das *tags* `<Value>` entre as linhas 3 e 28, conclui-se que houve a **edição** *(Type → Edit)* em uma **unidade de programa** *(Subtype → ProgramUnit)* com a **renomeação** *(Subsubtype → Rename)* de um **método** *(Unit-Type → Method)* **renomeando de** *(From-Unit-Name*

→ `methodToRefactor()`) **para** (*To-Unit-Name* → `void refactoredMethod()`) usando a **linguagem java** (*Language* → *java*).

5.6 EXPERIMENTO 6 – RENOMEAR CLASSE

O sexto e último experimento tem como objetivo avaliar o conteúdo gerado pelo sensor durante a ação renomear a classe criada no Experimento 1. Para executar o experimento, a classe criada no Experimento 1 foi renomeada de *ClassToRefactor* para *RefactoredClass*, utilizando assistentes do ambiente de desenvolvimento Eclipse.

A Listagem 5.6 apresenta o conteúdo dos dados gerados pelo sensor para a ação do Experimento 6 (renomear classe).

Listagem 5.6: Dado extraído por sensor durante o experimento Renomear classe

```
<Properties>
  <Property>
    <Key>Subtype</Key>
    <Value>ProgramUnit</Value>
  </Property>
  <Property>
    <Key>Type</Key>
    <Value>Edit</Value>
  </Property>
  <Property>
    <Key>Language</Key>
    <Value>java</Value>
  </Property>
  <Property>
    <Key>Unit-Type</Key>
    <Value>Class</Value>
  </Property>
  <Property>
    <Key>Unit-Name</Key>
    <Value>ClassToRefactor.java</Value>
  </Property>
  <Property>
    <Key>Subsubtype</Key>
    <Value>Remove</Value>
  </Property>
</Properties>
```

Após a interpretação humana da Listagem 5.6 conclui-se que esta ação refere-se à edição de unidade de programa, do tipo classe, usando a linguagem java, removendo-a. Estas informações estão explícitas no trecho de código da Listagem 5.6. Considerando os pares chave e valor, tendo como chave o conteúdo das *tags* `<Key>` e valor o conteúdo das *tags* `<Value>` entre as linhas 3 e 24, conclui-se que houve a **edição** *(Type → Edit)* em uma **unidade de programa** *(Subtype → ProgramUnit)* com a **remoção** *(Subsubtype → Remove)* de uma **classe** *(Unit-Type → Class)* **nomeada** *(Unit-Name →* `ClassToRefactor.java`*)* usando a **linguagem java** *(Language → java)*.

Os sensores não detectaram a ação "Renomear classe" da mesma forma que foi detectado para a ação "Renomear Atributo" e "Renomear Método". Fez-se uma busca nas ações registradas pelo sensor, considerando a hipótese da detecção da inclusão de uma nova classe, como aconteceu no Experimento 1. Porém, tal registro não ocorreu.

Durante a execução dos experimentos de 1 a 6, percebeu-se que alguns dados coletados pelo sensor eram bastante semelhantes. A Listagem 5.7 apresenta o formato do conteúdo comum coletado pelo sensor. Com interpretação humana é possível extrair informações sobre:

- O tipo do evento (de desenvolvimento) baseado no conteúdo da *tag* `SensorDataType` na linha 4, onde `DevEvent` é um acrônimo para `Development Event`, segundo Kou (2007).
- O recurso ou artefato de software associado à esse evento, baseado no conteúdo da *tag* `Resource` na linha 5;
- O ator que gerou o evento, baseado no conteúdo da *tag* `Owner` na linha 6;

Listagem 5.7: Dado comuns extraído por sensor durante e execução dos experimentos

```
1 <Timestamp>2012-02-11T14:33:12.010-02:00</Timestamp>
2 <Runtime>2012-02-11T14:33:12.010-02:00</Runtime>
3 <Tool>Eclipse</Tool>
4 <SensorDataType>DevEvent</SensorDataType>
5 <Resource>file:/D:/work/Bravo/servidor/br/ufpr/inf/bravo/server/
    ClassToRefactor.java</Resource>
6 <Owner>josivanps@gmail.com</Owner>
```

Alguns experimentos resultaram na implementação de ontologias, que dão significado para as ações executadas durante o experimento, descritas na próxima seção.

5.7 ONTOLOGIAS DE SENSORES

Os experimentos 2, 4 e 5 resultaram no desenvolvimento de uma ontologia com semântica de refatoração. Para o desenvolvimento destas ontologias, também foi utilizada a metodologia *Ontology Development 101*, descrita na seção 2.4, em que um dos passos consiste na **definição de classes** e **hierarquia entre elas**. Nos experimentos desta pesquisa, foram utilizados sensores da ferramenta *Hackystat* e, desta forma, foi desenvolvida uma ontologia abstrata para sensores da ferramenta.

5.7.1 Ontologia *Hackystat*

A Figura 8 apresenta a ontologia nomeada **Ontologia *Hackystat***. As classes na cor cinza são classes que já existem, e a arquitetura proposta neste trabalho fez o reuso das mesmas. As classes na cor branca são classes novas criadas durante a implementação da arquitetura proposta neste trabalho. A ontologia é composta por três classes: *BasicResourceAction*, *SensorDataAction* e *SourceCode*. A ontologia usa uma classe que faz parte da "Ontologia de Nível Superior", descrita na seção 4.1.1, e também a "Ontologia de Integração", descrita na seção 4.1.2. Da Ontologia de Nível Superior, é utilizada a classe *Text*, da Ontologia de Integração, é utilizada a classe *BasicAction*, o que atende assim o requisito da arquitetura **eXTadEi**, descrito na seção 4.1.2.

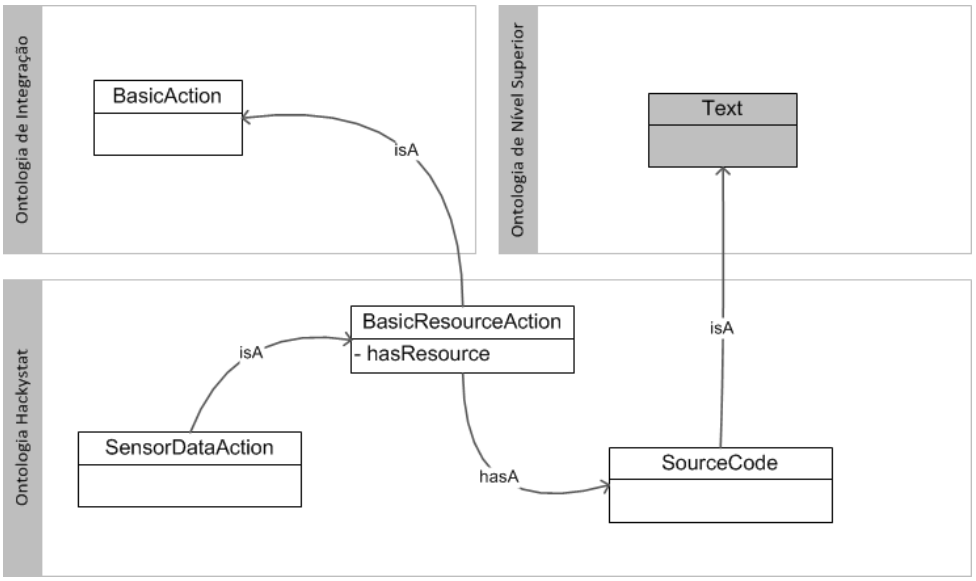

Figura 8: Ontologia *Hackystat*

Fonte: O Autor

A classe *BasicResourceAction* representa uma ação básica que possui um recurso as-

sociado. O recurso associado é um indivíduo do tipo *SourceCode* que é um tipo *Text* da Ontologia *Document*, descrita na seção 2.3.2. O recurso está associado com *BasicResourceAction* através da propriedade funcional *hasResource*. *SensorDataAction* é a classe mais abstrata para ser usada na representação de todos os dados capturados pelos sensores da ferramenta *Hackystat*. Outras visões das classes da Ontologia *Hackystat* são apresentadas nas Figuras 13 e 14 do Apêndice A.

Para representar os dados apresentados nas listagens dos Experimentos 2, 4 e 5 foi desenvolvida uma nova ontologia, descrita a seguir, que usa a Ontologia *Hackystat*.

5.7.2 Ontologia Refatoração

A Figura 9 apresenta a ontologia nomeada **Ontologia Refatoração**. Todas as classes desta ontologia são novas e foram criadas durante a implementação da arquitetura proposta neste trabalho. Todos os experimentos deste trabalho utilizaram sensores da ferramenta *Hackystat*, o que permite o uso da **Ontologia *Hackystat*** descrita previamente.

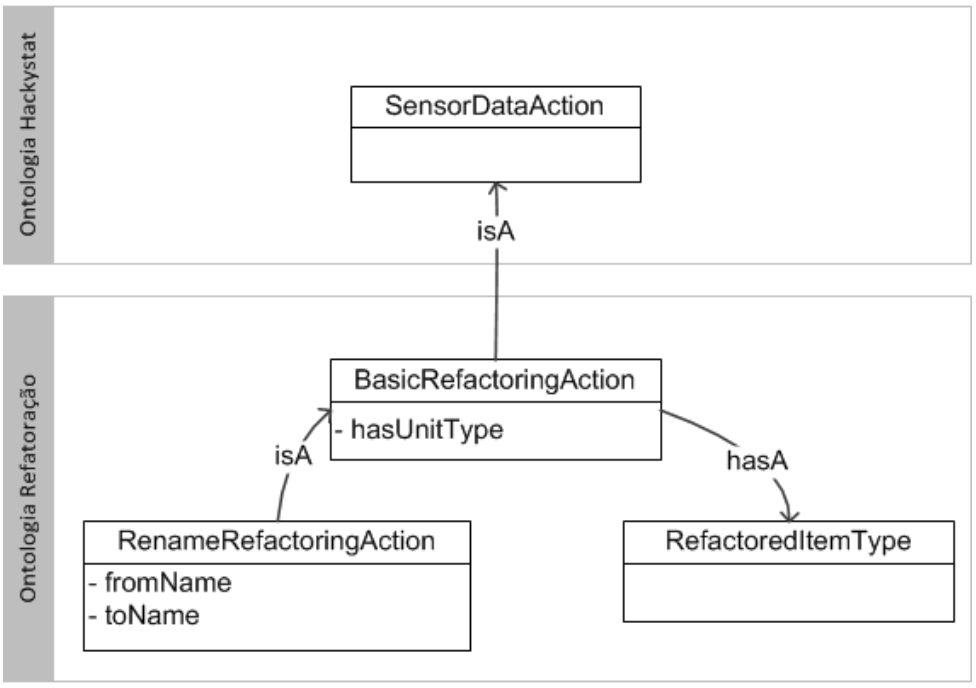

Figura 9: Ontologia Refatoração

Fonte: O Autor

A ontologia é composta por três classes e usa a Ontologia *Hackystat*, que por sua vez usa a Ontologia de Integração, requerida pela arquitetura proposta. A classe *BasicRefactoringAction* é uma classe que representa ações de refatoração, a classe *RefactoredI-*

temType, que associa um recurso com um tipo e, por fim, a classe *RenameRefactoringAction*.

A classe *RenameRefactoringAction* representa a **Refatoração Renomear** tanto de métodos, quanto de atributos ou classes. Indivíduos dessa classe possuem propriedades funcionais que descrevem o nome original *(fromName)*, o novo nome atribuído *(toName)* e o tipo de item que foi refatorado *(hasUnitType)*, sendo este último herança da classe *BasicRefactoringAction*. A propriedade funcional *hasUnitType* usa um padrão de projeto nomeado *Value Partition* que, de acordo com Horridge et al. (2004), permite refinar uma classe. *Value partitions* limitam o intervalo de valores possíveis de acordo com uma lista pré-definida. No caso da propriedade *hasUnitType* os valores são limitados a três indivíduos da classe *RefactoredItemType*: *Field*, *Method* e *Class*, que representam, respectivamente, um atributo ou campo, um método e uma classe. Estas propriedades foram definidas de acordo com as chaves (conteúdo das *tags* `<Key>`) geradas pelo sensor. Outras visões das classes da Ontologia Refatoração são apresentadas nas Figuras 15 e 16 do Apêndice A.

Durante a avaliação dos dados gerados pelos sensores, foi detectada a ocorrência de ações emitidas pelos sensores em intervalos irregulares. Estas ações resultaram na criação de outras ontologias descritas a seguir.

5.7.3 Ontologia Alteração de Estado

Com a avaliação das listagens exibidas previamente, resultadas dos experimentos, percebe-se que todas as chaves (conteúdo da *tag* `<key>`) *Type* possuem o valor (conteúdo da *tag* `<Value>`) *Edit* e que todas as chaves *Subtype* possuem o valor *ProgramUnit*. Percebe-se também que a chave *Subsubtype* apresentou valores *Add* ou *Rename* ou *Remove*. Porém, outros valores foram detectados para as chaves *Subtype*.

Em algumas ações disparadas pelos sensores, em intervalos irregulares, houve variação nos valores da chave *Subtype*. A Tabela 1 apresenta os valores que ocorreram para chave *Subtype* e a interpretação humana para tais valores.

As ações detectadas pelo sensor, cuja chave `Subtype` possuem o valor `StateChange`, tornam explícitas informações sobre o artefato ou recurso associado. A Listagem 5.8 é um exemplo de dado coletado pelo sensor e que mostra, por meio do valor (conteúdo da *tag* `<Value>`), o número de métodos (baseado no conteúdo da *tag* `<Key>`) `(Current-Methods)`; o número de instruções `(Current-Statements)`, além do

Tabela 1: Subtipos dos Sensores

Subtype	Interpretação
`Close`	Indica a ação "Fechar" um recurso ou artefato de software.
`Open`	Indica a ação "Abrir" um recurso ou artefato de software.
`ProgramUnit`	Indica que alguma ação relacionada ao processo de desenvolvimento de software ocorreu em uma unidade de programa.
`Save`	Indica que houve a ação "Salvar", no sistema de arquivos, em um recurso ou artefato de software.
`StateChange`	Indica que um recurso ou artefato de software teve seu estado alterado.

tamanho atual (`Current-Size`) em bytes, de determinada classe (`Class-Name`). Todas as ocorrências da ação `StateChange` possuem esta estrutura. Estes dados permitem concluir, através de interpretação humana, as inúmeras mudanças de estados de um artefato, que também pode representar uma refatoração.

Listagem 5.8: Ações StateChange detectadas por sensores

```
<Properties>
  <Property>
    <Key>Subtype</Key>
    <Value>StateChange</Value>
  </Property>
  <Property>
    <Key>Type</Key>
    <Value>Edit</Value>
  </Property>
  <Property>
    <Key>Current-Size</Key>
    <Value>344</Value>
  </Property>
  <Property>
    <Key>Class-Name</Key>
    <Value>br.ufpr.inf.bravo.server.ClassToRefactor</Value>
  </Property>
  <Property>
    <Key>Current-Statements</Key>
    <Value>2</Value>
  </Property>
  <Property>
    <Key>Current-Methods</Key>
    <Value>3</Value>
  </Property>
```

```
26 </Properties>
```

O conteúdo da Listagem 5.8 resultou no desenvolvimento de uma ontologia nomeada "Ontologia Alteração de Estado", apresentada na Figura 10.

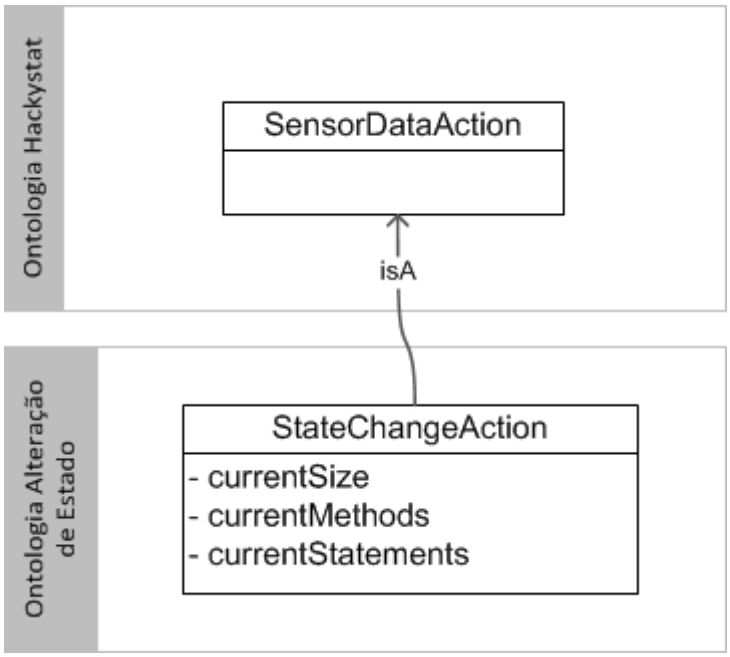

Figura 10: Ontologia Alteração de Estado

Fonte: O Autor

A Ontologia Alteração de Estado usa a Ontologia *Hackystat* e é composta de uma única classe. A classe *StateChangeAction* representa uma alteração de estado em um recurso associado. Indivíduos dessa classe possuem propriedades funcionais que descrevem o tamanho atual em bytes *(currentSize)*; a quantidade atual de métodos *(currentMethods)* e a quantidade de instruções *(currentStatements)*. Estas propriedades foram definidas de acordo com as chaves (conteúdo das *tags* `<Key>`) geradas pelo sensor.

Neste trabalho foram apresentados apenas os dados coletados pelos sensores específicos dos experimentos descritos anteriormente. Porém, muitos outros dados foram coletados. A Tabela 2 apresenta um resumo dos dados coletados durante o desenvolvimento deste trabalho.

Como pode ser observado na Tabela 2, foram coletados 78582 dados considerados ações para os sensores. Dentre estes dados, 23058, o que representa 29,3 % das ações coletadas, são ações reconhecíveis pela arquitetura proposta. São consideradas reconhecíveis as ações para as quais foram desenvolvidas ontologias utilizando a Ontologia de Integração, descrita anteriormente na seção 4.1.2.

Tabela 2: Dados Coletados

Quantidade de Ações Coletadas	78582
Ações Reconhecíveis	23058
Recursos	858
Ontologias Implementadas para Sensores	3
Total de Ontologias implementadas	5

Há um conjunto de ações reconhecíveis já detectadas mas não incluídas nas ontologias apresentadas nesta implementação da arquitetura proposta neste trabalho. As ações são: (1) adicionar método, (2) adicionar atributo, (3) mover classe (entre pacotes), (4) remover classe, (5) remover método, (6) remover atributo, (7) abrir arquivo, (8) fechar arquivo, (9) versionamento para repositório de código *(commit)*, (10) construção de software *(Build)* e (11) transição entre recursos (ação de sair de recurso A e ir para o recurso B). Recurso pode ser uma classe java ou qualquer outro arquivo relacionado com o software em desenvolvimento.

A implementação atual não considerou a atividade de depuração de código, mas sim a atividade de desenvolvimento de código. Apenas para complemento, as atividades de depuração representaram 64,7% das ações coletadas pelos sensores, ou seja, quase 2/3 dos dados coletados. Como cada ação possui um recurso associado, foram registrados 858 recursos. E finalmente, foram desenvolvidas e descritas nestes trabalho três ontologias para sensores. Outras duas ontologias foram desenvolvidas, mas estão além do escopo desta pesquisa[1].

Com os experimentos 1 a 6, foram desenvolvidas as ontologias Refatoração e Alteração de Estado que foram descritas nas seções anteriores. A partir destas ontologias, foram instanciadas classes que mostram que a arquitetura, através de consultas SPARQL[2] executadas com o motor de inferência, consegue inferir sobre ações de usuário utilizando ontologias. Desta maneira, a arquitetura proposta atinge o objetivo para o qual ela foi desenvolvida.

5.8 COMPARAÇÃO

Para comparar a arquitetura proposta neste trabalho, foi adotado um subconjunto dos itens caracterizados como relevantes por Bolchini et al. (2007). Utilizando um conjunto de

[1]São ontologias que representam processos no sistema operacional e atividades de versionamento de código *(commit)*.

[2]O apêndice D apresenta as consultas.

categorias, foram comparados modelos contextuais, caracterizando-os a partir de diversas perspectivas. Oberhauser (2010) usou por completo o conjunto definido por Bolchini et al. (2007). Porém, para este trabalho, um subconjunto dos itens definidos foi considerado e mais os seguintes itens: (a) Utilização de sensores variados, (b) Regras de inferências mutáveis, (c) Identificação de ações e (d) Código fonte aberto .

A Tabela 3 apresenta a comparação entre as arquiteturas propostas, descritas anteriormente na seção 2.5, e a arquitetura proposta neste trabalho.

Tabela 3: Comparando Arquiteturas

Categoria	**Chen and Nugent [2009]**	***Software Trajectory* [2010]**	**CoSEEEK [2010]**	**eXTadEi**
Tempo	Suportado	Suportado	Suportado	Parcial
Espaço/Tempo	Não Identificado	Não Identificado	Suportado	Previsto
Histórico de Contexto	Não Identificado	Não Identificado	Suportado	Previsto (com o uso de sensores)
Sujeito	Objetos e Atores	Não Identificado	Diversos Itens	Extensível
Perfil do Usuário	Suportado	Não Identificado	Suportado	Parcial
Tipo de Formalismo/Método de Captura	Mineração de Dados em Dados de Sensores - Ontologias em Objetos e Atores	Mineração de Dados	Ontologia	Ontologia (em todos os dados)
Extensibilidade	Somente Casa Inteligente	Desenvolvimento de Software	Desenvolvimento de Software	Suportado
Granularidade Variável do Contexto	Não Suportado	Não Identificado	Suportado	Parcial

continua na próxima página

continuação da página anterior

Categoria	**Chen and Nugent [2009]**	***Software Trajectory* [2010]**	**CoSEEEK [2010]**	**eXTadEi**
Construção do Contexto	Centralizado	Centralizado	Centralizado	*Stand-Alone*, mono-usuário.
Raciocínio sobre o contexto	Não Identificado	Não Identificado	Previsto	Suportado com SPARQL
Monitoração de qualidade da informação do contexto	Não Identificado	Suportado sem Ontologias	Não Identificado	Suportado utilizando Ontologias
Gerenciamento de incompletude e ambiguidade	Não Identificado	Não Identificado	Não Identificado	Previsto
Características de Aprendizagem Automática	Previsto	Não Identificado	Previsto	Previsto
Modelagem Multi-Contextual	Não Identificado	Não Identificado	Não Identificado	Futuro
Utilização de Sensores Variados	Apenas em utensílios e atores	Apenas do Hackystat	Apenas do Hackystat	Conforme ontologia
Regras de Inferência mutáveis	Não Identificado	Não Identificado	Não Identificado	Inferência sobre ontologias
Identificação de Ações	Não Suportado	Não Suportado	Não Identificado	Suportado
Aberto para a Comunidade	Modelo Conceitual	Modelo Conceitual	Modelo Conceitual	Disponível

Na primeira coluna são apresentados os itens em relação a arquitetura proposta por Chen e Nugent (2009); na segunda coluna os itens relacionados com a arquitetura *"Software Trajectory" Framework*, proposto por Senin (2010); na terceira coluna são apresen-

tados os itens relacionados ao CoSEEEK, proposto por Oberhauser (2010), e a quarta coluna apresenta os itens em relação a arquitetura eXTadEi, proposta neste trabalho e descrita anteriormente no capítulo 4.

Os itens do subconjunto utilizado neste trabalho foram descritos por Bolchini et al. (2007) da seguinte maneira:

Tempo. Se o modelo do contexto permite a representação de aspectos temporais.

Espaço/Tempo. Os parâmetros de espaço e tempo são representados de formas absolutas (usando coordenadas de um GPS, por exemplo) ou relativas (perto de algo, depois disto, por exemplo).

Histórico de Contexto. Se o estado atual de um contexto depende do estado anterior do contexto.

Sujeito. Quem ou o que é o sujeito descrito pelo contexto;

Perfil do Usuário. O perfil do usuário é representado no contexto.

Tipo de Formalismo/Método de Captura. Classe da ferramenta conceitual utilizada para capturar o contexto.

Extensibilidade. Bolchini et al. (2007) chamaram esse item de **Flexibilidade**, que significa se um modelo possui habilidades para fácil adaptação em diferentes contextos.

Granularidade Variável do Contexto. A habilidade do modelo de representar características do contexto em diferentes níveis de detalhes.

Construção do Contexto. Destaca se a descrição do contexto é distribuída ou centralizada.

Raciocínio sobre o contexto. Indica se o modelo de contexto permite raciocínio sobre os dados do contexto para inferir propriedades ou informações mais abstratas sobre o contexto para que possa, por exemplo, deduzir atividade do usuário combinando leituras de sensores.

Monitoração de qualidade da informação do contexto. Indica se o sistema considera e gerencia explicitamente qualidade da informação recuperada pelo contexto, por exemplo, quando os dados do contexto são percebidos por sensores.

Gerenciamento de incompletude e ambiguidade. Capacidade do sistema de saber manipular condições de incompletude e ambiguidade.

Conteúdo de Aprendizagem Automática. Considera-se que, por observação do comportamento do usuário, experiências individuais anteriores com outro usuário ou com o ambiente, o sistema pode extrair conhecimento a partir do contexto.

Modelagem Multi-Contextual. Capacidade de representar mais de um contexto em um único sistema.

Utiliza Sensores Variados. A arquitetura consegue identificar eventos que venham de sensores variados? Sensores variados no sentido de que a arquitetura não é dependente de um tipo ou tipos particulares de sensores.

Regras de Inferência mutáveis. As regras de inferência são definidas no código fonte da implementação?

Identificação de Ações. Identificação das ações executadas pelo desenvolvedor.

Aberto para a Comunidade. O código fonte de alguma implementação está disponível para uso pela comunidade?

Como pode ser observado na Tabela 3, a categoria **Tempo**, embora considerada suportada pelas demais arquiteturas, não é totalmente suportada pela arquitetura proposta neste trabalho. O suporte é considerado **parcial** porque a arquitetura **eXTadEi** não usa a ontologia DOLCE que suporta, por exemplo, o conceito de ciclos ou intervalos de tempo.

As categorias **Espaço/Tempo** e **Histórico de Contexto** são suportadas apenas pela arquitetura CoSEEEK, proposta por Oberhauser (2010). Para a arquitetura proposta neste trabalho a característica é prevista, considerando sensores e suas respectivas ontologias que descrevam esta categoria.

Apenas a arquitetura proposta por Senin (2010) não suporta explicitamente a categoria **Sujeito**. Porém, as demais arquiteturas suportam um grupo limitado ou definido de sujeitos. Apenas a arquitetura proposta neste trabalho permite que novos sujeitos sejam agregados. Porém, na implementação descrita neste trabalho o sujeito é o Processo de Software.

O item **Perfil do Usuário** não é suportado apenas pela arquitetura *Software Trajectory*, proposta por Senin (2010). A arquitetura proposta neste trabalho suporta par-

cialmente, pois está limitada ao conceito de papéis *(Roles)* através da Ontologia *Role*, descrita na seção 2.3.2. As demais arquiteturas suportam por completo.

Na categoria **Tipos de Formalismo/Métodos de Captura**, tanto a arquitetura proposta por Chen e Nugent (2009) quanto a *Software Trajectory*, proposta por Senin (2010), utilizam mineração de dados. A arquitetura proposta por Chen e Nugent (2009) utiliza ontologias para a representação de atores, sensores e objetos. A arquitetura CoSEEEK, proposta por Oberhauser (2010), também utiliza ontologias para representar alguns itens. Apenas a arquitetura proposta neste trabalho utiliza ontologias para representar os dados coletados por sensores.

A categoria **Extensibilidade** é uma categoria considerada importante para este trabalho. Na arquitetura proposta por Chen e Nugent (2009), o contexto está limitado ao ambiente de uma casa inteligente. Na arquitetura *Software Trajectory*, proposta por Senin (2010), e na arquitetura CoSEEEK, proposta por Oberhauser (2010), o contexto está limitado ao Processo de Software. Na arquitetura proposta neste trabalho, embora a implementação atual esteja também limitada ao Processo de Software, o contexto não está limitado ao definido para esta implementação, pois as ontologias de nível superior permitem a aplicação em outros contextos.

A categoria **Granularidade Variável do Contexto** é suportada apenas pela arquitetura CoSEEEK, proposta por Oberhauser (2010). A arquitetura proposta por Chen e Nugent (2009) não suporta esta categoria. A arquitetura *Software Trajectory*, proposta por Senin (2010), não descreve explicitamente esta informação. A arquitetura proposta neste trabalho suporta parcialmente este item, pois está inicialmente limitada ao projeto do ator e tem como menor granularidade o recurso associado com a ação executada pelo ator.

Todas as arquiteturas avaliadas definem a categoria **Construção de Contexto** como centralizada. Porém, a arquitetura proposta neste trabalho foi concebida considerando ambientes onde exista a distribuição de conhecimento, ou seja, que as informações não sejam centralizadas como as demais arquiteturas. A implementação apresentada neste trabalho é orientada para um ator, não existindo comunicação com outros atores.

A arquitetura CoSEEEK, proposta por Oberhauser (2010), prevê o uso futuro da categoria **Raciocínio sobre o Contexto**. A arquitetura proposta por Chen e Nugent (2009) e a arquitetura *Software Trajectory*, proposta por Senin (2010), não definem isto de forma explícita. A arquitetura proposta neste trabalho suporta o raciocínio através do componente "Motor de Inferência", descrito anteriormente na seção 4.1.3.

A arquitetura *Software Trajectory*, proposta por Senin (2010), suporta a categoria **Monitoração de qualidade da informação de contexto**, porém sem utilizar ontologias. Os dados coletados por sensores são representados através do uso de ontologias apenas na arquitetura proposta neste trabalho. A arquitetura proposta por Chen e Nugent (2009) e a arquitetura CoSEEEK, proposta por Oberhauser (2010), não definem isto de maneira explícita.

Apenas a arquitetura proposta neste trabalho prevê a implementação futura do **Gerenciamento de incompletude e ambiguidade**, utilizando para isto ontologias. As demais arquiteturas não definem isto de maneira explícita.

A característica **Aprendizagem Automática** é prevista para implementação futura em todas as arquiteturas comparadas, exceto na arquitetura *Software Trajectory*, proposta por Senin (2010), que não define isto de maneira explícita.

A característica de **Modelagem Multi-Contextual** não é considerada nas arquiteturas avaliadas, exceto na arquitetura proposta neste trabalho, onde as Ontologias de Nível Superior, descritas na seção 4.1.1, não limitam o número de contextos. A implementação atual, porém, está limitada ao processo de desenvolvimento de software.

Em relação à **Utilização de Sensores Variados**, a arquitetura proposta por Chen e Nugent (2009) suporta apenas sensores relacionados ao contexto de uma casa inteligente. A arquitetura *Software Trajectory*, proposta por Senin (2010), e a arquitetura CoSEEEK, proposta por Oberhauser (2010) suportam apenas os sensores da ferramentas *Hackystat*. Apenas a arquitetura proposta neste trabalho suporta vários tipos de sensores, pois a dependência da arquitetura é em relação às ontologias de sensores e não aos sensores propriamente.

O item Regras de Inferência mutáveis refere-se à condição de que as regras de inferência estão codificadas de maneira imutável no código fonte da implementação da arquitetura. No que diz respeito à esse item, a arquitetura proposta por Chen e Nugent (2009) e a arquitetura *Software Trajectory*, proposta por Senin (2010), não possuem implementação, sendo estas consideradas modelos conceituais. A arquitetura CoSEEEK, proposta por Oberhauser (2010), possui uma implementação que não está aberta, na época desta escrita, nem para a comunidade científica nem para a comunidade corporativa, o que faz com que esse item não possa ser identificado. Em relação à arquitetura proposta nesta pesquisa, as Regras de inferência não estão codificadas de maneira imutável no código fonte, e estão explícitas através das ontologias.

O item Identificação de Ações não é tido como objetivo para a arquitetura proposta por Chen e Nugent (2009), nem para a arquitetura *Software Trajectory*, proposta por Senin (2010). A arquitetura CoSEEEK, proposta por Oberhauser (2010), tem como objetivo o reconhecimento de atividades, mas por não existir, na época desta escrita, uma versão pública para avaliação, foi considerado não identificado. Apenas a arquitetura proposta neste trabalho busca e identifica as ações do ator.

Finalmente, o item Aberto para a Comunidade trata da condição da existência de implementação e de código fonte aberto para que possa ser livremente avaliado. A arquitetura proposta por Chen e Nugent (2009) e a arquitetura *Software Trajectory*, proposta por Senin (2010), são modelos conceituas e não apresentam implementações atualmente. A arquitetura CoSEEEK, proposta por Oberhauser (2010), possui uma implementação que não está aberta para avaliação pública. Apenas a arquitetura proposta nesta trabalho está aberta para avaliação.

5.9 CONCLUSÃO

Ao avaliar as informações apresentadas na tabela 3 e a discussão da seção anterior, não foram consideradas a arquitetura proposta por Chen e Nugent (2009) e arquitetura *Software Trajectory*, proposta por Senin (2010), por não existir uma implementação que permita uma comparação com a implementação da arquitetura proposta neste trabalho. A arquitetura CoSEEEK, proposta por Oberhauser (2010), suporta muitos itens apresentados na Tabela 3. Porém, a única implementação do modelo conceitual não está disponível publicamente, segundo Oberhauser (2011).

A arquitetura proposta neste trabalho foi pensada para contribuir com o ator com papel de desenvolvedor de software. A partir do ponto de vista do desenvolvedor, a intenção é que o ator tenha controle do seu conhecimento. A arquitetura proposta neste trabalho é a única que garante este comportamento através da característica **Construção de Contexto**, que não é centralizada, mas sim orientada para um único usuário e com funcionamento independente. A arquitetura foi concebida considerando um ambiente onde exista a distribuição de conhecimento.

Outro ponto que pode ser considerado um diferencial em relação às demais arquiteturas é a existência de uma implementação com código aberto, o que permite que a arquitetura seja avaliada por outros.

5.10 CONSIDERAÇÕES FINAIS

Neste capítulo foram apresentados os experimentos realizados com o objetivo de avaliar o reconhecimento de ações executadas pelo ator utilizando ontologias. Para cada experimento, foram inspecionados os dados coletados pelos sensores e, a partir destes dados, foram desenvolvidas ontologias. Tais ontologias, nomeadas Ontologias de Sensores, foram detalhadas e tiveram classes instanciadas que foram utilizadas como conteúdo para consultas SPARQL realizadas com o motor de inferência. Ao final do capítulo, foram comparadas as arquiteturas propostas anteriormente com a arquitetura descrita neste trabalho. No próximo capítulo são descritas as conclusões e os trabalhos futuros.

6 CONCLUSÕES E TRABALHOS FUTUROS

Os trabalhos mais evoluídos em relação ao reconhecimento de sequências aplicadas à engenharia de software foram apresentados na seção 2.5. Os trabalhos apresentaram propostas de arquiteturas ou modelos conceituais aplicados em um ambiente com sensores para coleta de dados. Porém, nem todos os trabalhos descritos estavam disponíveis publicamente no momento desta escrita. Por esse motivo, optou-se por implementar uma nova arquitetura, descrita no capítulo 4, que atendesse os objetivos propostos. Este capítulo finaliza o presente trabalho com o relato da contribuição, relevância e possíveis trabalhos futuros.

Com base nos experimentos realizados, descritos no capítulo 5, foi possível verificar que a arquitetura consegue reconhecer ações executadas por um ator, em um dado momento, utilizando ontologias. Algumas características diferenciais entre a arquitetura proposta neste trabalho, descrita no capítulo 4, em relação às arquiteturas propostas anteriormente, descritas na seção 2.5, foram demonstradas na Tabela 3. A característica mais relevante da arquitetura proposta neste trabalho em relação às demais é a extensibilidade, uma vez que permite a adição de novos sensores sem a alteração do núcleo da arquitetura proposta. Esta característica é obtida através da Ontologia de Integração, descrita na seção 4.1.2.

Para que novos sensores sejam agregados à arquitetura proposta, foi utilizada uma técnica na qual novas ontologias de sensores devem ser uma herança da Ontologia de Integração. A Figura 11 apresenta algumas possibilidades de sensores que podem ser acoplados à arquitetura. A seção 6.3 cita exemplos de sensores que podem ser implementados para arquitetura proposta.

A arquitetura proposta neste trabalho não é um sistema especialista, mas sim uma demonstração de que, a partir de dados coletados por sensores, é possível criar ontologias que dão significado aos dados. Cada ação executada pelo ator que seja capturada pelo sensor e que tenha uma ontologia associada é um conceito de ação, e um conjunto de ações pode descrever uma atividade. É possível, com o uso de ontologias, descrever todos os conceitos, mas não a ordem na qual eles são aplicados. Nos experimentos, foram considerados conceitos de engenharia de software, mas é possível representar, por exemplo,

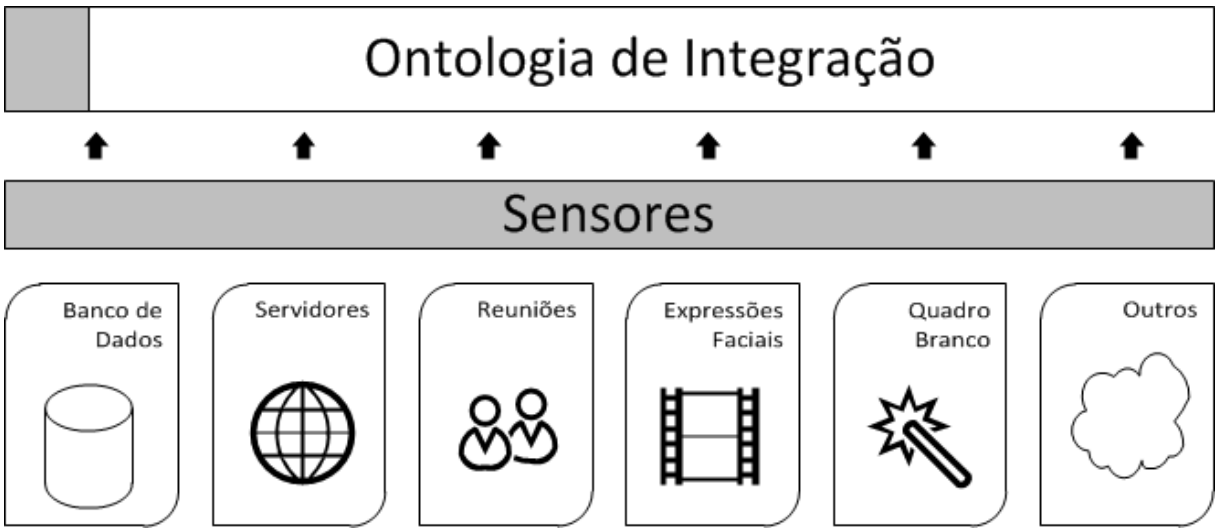

Figura 11: Ontologia para diversos sensores

Fonte: O Autor

sequências de páginas da internet visitadas.

A implementação atual da arquitetura proposta neste trabalho executa algumas tarefas de maneira semi-automatizada. Para a execução destas tarefas, algumas instruções precisam ser passadas manualmente para a implementação. A ação de importar os dados do *Hackystat* precisa ser iniciada manualmente, da mesma forma que a ação de converter os dados para o formato intermediário também precisa. A conversão para ontologias, que é executada de forma automatizada, também precisa ser iniciada manualmente pela implementação atual. As consultas SPARQL devem ser executas manualmente contra o motor de inferência. Todas as ações iniciadas manualmente na implementação atual, podem ser iniciadas automaticamente em intervalos regulares em outras implementações.

6.1 CONTRIBUIÇÕES

Em relação ao ambiente computacional, já foram implementados sensores que inferem sobre processos de sistema em execução no computador do ator, o que permite inferir se algumas atividades foram executadas, como a leitura de documentos de um projeto de software. A Figura 17 do apêndice A apresenta as classes desta ontologia. Os detalhes acerca da "Ontologia de Processos de Sistema" estão além do escopo deste trabalho. Outra contribuição é a demonstração da possibilidade de representar e dar significado a dados coletados por sensores através do uso de ontologias sem a intervenção do usuário.

O uso da semântica nos dados coletados por sensores apresentado neste trabalho

também é considerado uma contribuição, pois foi demonstrado que esta técnica permite evoluir para um modelo conceitual mais amplo e detalhado, pois a modelagem é aplicada em ontologias e não em dados. Os dados, por sua vez, estarão sempre sendo coletados e disponibilizados. Em outro cenário, no qual a forma como um banco de dados é organizado não está disponível como, por exemplo, em um arquivo com dados posicionais de tamanho fixo; utilizando ontologias, os dados não sofrem alterações, mas a semântica, por sua vez, pode ser explorada com o objetivo de representar mais de um significado. Por exemplo, um dado representado através de ontologias pode representar uma tema em um contexto, e outro tema em outro contexto. Esta característica não está disponível, ao menos facilmente, no banco relacional.

Outra contribuição é que, na arquitetura proposta, o modelo conceitual está explícito em ontologias e não implícito no código ou nos dados coletados, o que torna possível a modelagem de novas ontologias sem alteração no núcleo da arquitetura proposta e sem a necessidade de reimplementações em código de linguagem de programação. Além disso, usa-se um motor de inferência para ontologias, e não mais um motor de regras com uma linguagem particular, como é usado no Zorro. Assim, o processo de modelagem de conceitos e regras se torna mais fácil de representar. As ontologias apresentadas nesse trabalho podem ser aplicadas em evoluções da arquitetura proposta e em outras arquiteturas ou, até mesmo, em trabalhos que envolvam processo de software e uso de sensores.

6.2 LIMITAÇÕES

O presente trabalho não pretende redefinir o tipo de informação, nem mesmo, alterar o formato dos dados capturados pelos sensores da ferramenta *Hackystat*. Porém, por simplicidade na conversão, os dados são transformados e armazenados em uma base de dados intermediária. Além disso, este trabalho também não pretende capturar dados de todos os atores em todos os possíveis papéis que este possa assumir. A pesquisa não pretende implementar todas as prováveis ontologias para todos os prováveis sensores, incluindo todas as ações coletadas através dos sensores da ferramenta *Hackystat*.

Embora tenha sido desenvolvida uma ontologia para a ação *Commit*[1], exibida na Figura 18 do Apêndice A, as ações do tipo *"Commit"* não foram consideradas pelo presente trabalho, pois estas não são capturadas discretamente pelos sensores do *Hackystat*. A coleta destes dados ocorre com o uso de um sensor que é acionado apenas quando o

[1]Submissão de código fonte para o repositório de código.

desenvolvedor executa explicitamente tarefas de construção com a ferramenta apropriada[2]. Os detalhes acerca da "Ontologia *Commit*" estão além do escopo deste trabalho.

Ainda que, nesta pesquisa, os experimentos descritos no capítulo 5 sejam caracterizados como "refatoração", uma implementação mais precisa é possível usando técnicas e conceitos de reflexão[3] nos artefatos submetidos para o repositório de código fonte. Levando em conta, por exemplo, não apenas o número de linhas adicionadas, removidas ou total de linhas, número de métodos e instruções correntes em um artefato, mas também o nome e quantidade de métodos e atributos em um determinado artefato. Com a implementação de um sensor com tais características, uma nova ontologia deve ser desenvolvida, considerando sempre o uso da "Ontologia de Integração". A linguagem usada nos experimentos realizados, descritos no capítulo 5, permite que outras métricas sejam aplicadas para caracterizar refatoração. Por exemplo, quantidade de métodos adicionados, quantidade de métodos removidos, quantidade de campos públicos, acoplamento entre classes, entre outros descritos por Li et al. (2010).

Para a arquitetura proposta, não existe a pretensão de gerar informações com objetivos gerenciais, mas apenas as que tragam benefícios para o ator, principalmente os atores com papel de desenvolvedor, para evitar o cenário descrito por Johnson (2007). Desta forma, minimiza-se qualquer desconforto ou insegurança para o ator durante a execução de suas atividades. Da mesma forma que não pretende-se implementar um assistente que, com base nos dados inferidos de acordo com o contexto e com o papel, possa auxiliar outros atores.

6.3 TRABALHOS FUTUROS

Outras implementações da arquitetura podem utilizar outros tipos de dados coletados por outros sensores, como registros de eventos de sistemas operacionais, registros de eventos em servidores de aplicações para a Internet, ou qualquer tipo de dado capturado por qualquer tipo de sensor, desde que uma ontologia de integração e uma consulta sejam definidas para o sensor. Alguns exemplos de sensores são apresentados na Figura 11.

Outros sensores relacionados a ambientes computacionais podem ser referentes ao histórico de navegação feito pelo ator na Internet, o que pode demonstrar, por exemplo, pesquisas e downloads efetuados pelo ator. Sensores que representem falhas em servidores de aplicações, banco de dados, entre outros, baseado no conteúdo dos registros efetuados

[2]No caso do Hackystat é utilizado um sensor para o Apache Ant (http://ant.apache.org).
[3]Engenharia reversa em código fonte.

por estes sistemas onde ações preventivas ou corretivas podem ser tomadas para evitar condições não esperadas. Se dados dos registros podem ser interpretados, então notificações de prováveis problemas podem ser feitas de forma automática.

Além da área computacional, podem ser implementados, por exemplo, sensores que podem capturar vários tipos de dados em vários cenários diferentes. Alguns possíveis cenários são:

- Reunião com diversos atores ou partes interessadas em uma sala. Sensores para este evento podem, mas não estão limitados a, capturar dados como:
  - Digitais com as identificações das pessoas que adentraram a sala;
  - Gravações das discussões mantidas durante a reunião;
  - Anotações feitas em um quadro;
- Sensores que capturem movimentações em uma sala;
- Sensores que capturem expressão facial de um ator;
- Situação na qual o ator responde a uma série de e-mails em um período, uma evolução da arquitetura poderia relembrar o ator que um dado e-mail já foi enviado anteriormente ou que deve ser enviado para mais destinatários.
- Situação na qual o ator escreve um caso de uso e uma evolução da arquitetura detecta que uma sequência de texto já está definida em outro caso de uso, baseado em dados coletados por sensores;

Ontologias refinadas, para casos de refatorações especializadas, por exemplo, podem ser construídas juntamente com novos sensores. Estas construções podem permitir a inferência mais detalhada de atividades de refatoração. Isso deve auxiliar em conclusões como: o ator A refatora apenas renomeando métodos para nomes mais intuitivos, enquanto o ator B faz refatorações que diminuem o acoplamento do código.

A implementação da arquitetura proposta é um agente que monitora e infere atividades de um ator. No entanto, no futuro, ela pode interagir com o ator, sugerir outras sequências ou prioridades de atividades.

A arquitetura, na forma como foi proposta, pode, em algumas situações, atuar na condição de um objeto ouvinte descrito por Gamma et al. (2000), aguardando por notificações de dados de sensores. Em outros momentos, a arquitetura pode atuar na condição

de um objeto observável, também descrito por Gamma et al. (2000) que, após inferir informações relevantes para um determinado aspecto, notifica agentes interessados que observam as inferências efetuadas pela arquitetura.

A arquitetura **eXTàdEi** têm conhecimento sobre o contexto de uma ação através do uso de sensores, porém executa em um único ambiente na forma e sob a condição de um agente independente. Evoluir a arquitetura para um cenário no qual o compartilhamento de ações e atividades, bem como sequências reconhecidas, ocorra entre agentes distribuídos, pode elevar os agentes para a condição de assistentes do ator. Por exemplo, um ator com papel de desenvolvedor começa a implementar um algoritmo, e seu "Agente A" detecta que a sequência em execução já foi previamente detectada ou reconhecida por um "Agente B". O "Agente B" consegue concluir que a sequência executada anteriormente não foi satisfatória. O "Agente B" notifica o "Agente A" com o resultado obtido anteriormente. Este ator pode ser notificado, por meio de seu "Agente A", com a explicação de que talvez aquele não seja o melhor caminho para a solução do problema. Essa condição torna favorável o uso da arquitetura e de agentes de forma colaborativa. Uma sequência de ações pode ser representada por um grafo de ações ou eventos e o resultado de sucesso ou falha na execução da atividade do ator pode ser um protocolo que conecte as arquiteturas e os agentes em um ambiente distribuído colaborativo.

REFERÊNCIAS

ABOWD, Gregory D. Software engineering issues for ubiquitous computing. **Proceedings of the International Conference on Software Engineering (ICSE'99)**, ACM, New York, NY, USA, p. 75–84, 1999. Disponível em: <http://doi.acm.org/10.1145/302405.302454>.

ABOWD, Gregory D.; MYNATT, Elizabeth D. Charting past, present and future research in ubiquitous computing. **ACM Transactions on Computer-Human Interaction**, v. 7, p. 29–58, 2000.

BERTOLLO, Gleidson. **Definição de Processos em um Ambiente de Desenvolvimento de Software**. Dissertação (Mestrado) — Universidade Federal do Espírito Santo, 2006.

BOLCHINI, Cristiana et al. A data oriented survey of context models. **SIGMOD Record**, v. 36, p. 19–26, 2007.

BRÉZILLON, P. Context in artificial intelligence: IA survey of the literature. **Computer & Artificial Intelligence**, v. 18, p. 321–340, 1999.

BUNNINGEN, A. **Context Aware Querying - Challenges for data management in ambient intelligence**. [S.l.], 2004.

CERAVOLO, Paolo et al. A ontology-based process modelling for xp. **Tenth Asia-Pacific Software Engineering Conference (APSEC'03)**, p. 236–242, 2003.

CHEN, Guanling; KOTZ, David. **A Survey of Context-Aware Mobile Computing Research**. Hanover, NH 03755 USA, 2000.

CHEN, Liming; KHALIL, Ismail. Activity recognition: Approaches, practices and trends. **Activity Recognition in Pervasive Intelligent Environments**, v. 4, p. 1–31, 2011.

CHEN, Liming; NUGENT, Chris. Ontology-based activity recognition in intelligent pervasive environments. **International Journal of Web Information Systems**, v. 5, p. 410–430, 2009.

CHEN, L.; NUGENT, C.D.; WANG, H. A knowledge-driven approach to activity recognition in smart homes. **IEEE Transactions on Knowledge and Data Engineering Name**, 2010.

COMPANJEN, Ben. Classification methods for activity recognition. **11th Twente Student Conference on IT**, June 2009.

DEKKERS, Paul. **CORDYS – Simplifying Business**. Dissertação (Mestrado) — Radboud University Nijmegen, 2007.

DEY, Anind K.; ABOWD, Gregory D.; SALBER, Daniel. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. **Human-Computer Interaction Journal**, v. 16, p. 97–166, 2001.

ECLIPSE. **Eclipse - The Eclipse Foundation open source community website**. 05 2010. Disponível em: <http://www.eclipse.org>.

ESCOBEDO, Edgardo Paul Ponce. **Modelagem de Contexto Usando Ontologias**. Dissertação (Mestrado) — Escola Politécnica da Universidade de São Paulo, São Paulo, 2008.

FALBO, Ricardo De Almeida. **Integração De Conhecimento Em Um Ambiente De Desenvolvimento De Software**. Tese (Doutorado) — Universidade Federal do Rio de Janeiro, 1998.

FENSEL, D. **Ontologies: A Silver Bullet for Knowledge Management and Eletronic Commerce**. 1st. ed. New York: Springer, 2001. 138 p.

FERNÁNDEZ, Mariano; GÓMEZ-PÉREZ, AsunciÓn; JURISTO, Natalia. Methontology: From ontological art towards ontological engineering. p. 33–40.

FOWLER, Martin. **Refatoração – Aperfeiçoando o Projeto de Código Existente**. Porto Alegre: Bookman, 2004.

GAMMA, Eric et al. **Padrões de Projeto**. Porto Alegre: Bookman, 2000.

GANGEMI, Aldo et al. Sweetening ontologies with dolce. Springer, p. 166–181, 2002.

GRUBER, T. R. A translation approach to portable ontologies. **Knowledge Aquisition**, v. 5(2), p. 199–220, 1993.

GRUNINGER, Michael; FOX, Mark S. Methodology for the design and evaluation of ontologies. April 1995.

GUARINO, Nicola. Formal ontology and information systems. **FOIS'98**, p. 3–15, June 1998.

GUIZZARDI, Giancarlo. **Uma Abordagem Metodológica de Desenvolvimento para e com Reuso, baseada em Ontologias Formais de Domínio**. Dissertação (Mestrado) — Universidade Federal do Espírito Santo, 2000.

HACKYSTAT. **hackystat - A framework for collection, analysis, visualization, interpretation, annotation, and dissemination of software development process and product data**. 05 2010. Disponível em: <http://www.hackystat.org>.

HENRICKSEN, Karen; INDULSKA, Jadwiga; RAKOTONIRAINY, Andry. Modeling context information in pervasive computing systems. In: **Proceedings of the First International Conference on Pervasive Computing**. London, UK: Springer-Verlag, 2002. (Pervasive '02), p. 167–180. ISBN 3-540-44060-7. Disponível em: <http://portal.acm.org/citation.cfm?id=646867.706693>.

HORRIDGE, Matthew et al. A practical guide to building owl ontologies using the protégé-owl plugin and co-ode tools edition 1.0. Agosto 2004.

HORSTMANN, Cay. **Padrões e Projeto Orientados a Objetos**. Porto Alegre: Bookman, 2007.

JAVA. **Oracle Technology Network for Java Developers**. 05 2010. Disponível em: <http://www.oracle.com/technetwork/java/index.html>.

JENA, Apache. **Apache Jena - Reasoners and rule engines: Jena inference support**. 07 2012. Disponível em: <http://jena.apache.org/documentation/inference/>.

JOHNSON, Philip M. Requirement and design trade-offs in hackystat: An in-process software engineering measurement and analysis system. **First International Symposium on Empirical Software Engineering and Measurement**, p. 81–90, 2007.

JOHNSON, Philip M.; KOU, Hongbing. Automated recognition of test-driven development with zorro. **Computer Society**, 2007.

JOHNSON, Philip M. et al. Beyond the personal software process: Metrics collection and analysis for the differently disciplined. **IEEE Computer Society**, 2003.

KOU, Hongbing. Studying micro-processes in software development stream. **Collaborative Software Development Laboratory at the University of Hawaii**, July 2005.

KOU, Hongbing. **Automated Inference of Software Development Behaviors: Design, Implementation and Validation of Zorro for Test-Driven Development**. Tese (Doutorado) — University of Hawai, DECEMBER 2007.

KOU, Hongbing; JOHNSON, Philip M. Automated recognition of low-level process: A pilot validation study of zorro for test-driven development. **Springer-Verlag Berlin Heidelberg**, p. 322–333, 2006.

KOU, Hongbing; JOHNSON, Philip M.; ERDOGMUS, Hakan. Operational definition and automated inference of test-driven development with zorro. **Automated Software Engineering**, 2009.

LI, Lin et al. The measurement and analysis of software change based on software repository. **2nd International Conference on Software Engineering and Data Mining (SEDM), 2010**, p. 289–294, June 2010.

MAES, Pattie. Agents that reduce work and information overload. **Communication of the ACM**, v. 37, n. 7, p. 31–40, July 1994.

MAES, Pattie. Intelligent software: Easing the burdens that computers put on people. **IEEE Expert**, p. 62–63, December 1996.

MCBRIDE, Brian. Jena: A semantic web toolkit. **IEEE Internet Computing**, p. 55–59, November / December 2002.

MORSE, D. R.; ARMSTRONG, S.; DEY, A. K. The what, who, where, when and how of context-awareness. 2000.

MURPHY-HILL, Emerson; PARNIN, Chris; BLACK, Andrew P. How we refactor, and how we know it. **ICSE'09**, p. 287–297, 2009.

NETO, Renato de Freitas Bulcão. **Um processo de software e um modelo ontológico para apoio ao desenvolvimento de aplicações sensíveis ao contexto**. Tese (Doutorado) — USP - São Carlos, November 2006.

NOY, Natalya Fridman; HAFNER, Carole D. The state of art in ontology design: A survey and comparative review. **AI Magazine**, v. 18, p. 53–74, 1997.

NOY, Natalya F.; MCGUINNESS, Deborah L. Ontology development 101: A guide to creating your first ontology. **Stanford Knowledge Systems Laboratory Technical Report KSL-01-05 and Stanford Medical Informatics Technical Report SMI-2001-0880**, March 2001. Disponível em: <http://www.ksl.stanford.edu/people/dlm/papers/ontology101/ontology101-noy-mcguinness.html>.

NUNES, Vanessa Tavares; SANTORO, Flávia Maria; BORGES, Marcos R.S. Capturing context about group design processes. **Proc. of the 11th International Conference on Computer Supported Cooperative Work in Design (CSCWD'07)**, p. 18–23, 2007.

OBERHAUSER, Roy. Leveraging semantic web computing for context-aware software engineering environments. 2010.

OBERHAUSER, Roy. **CoSEEEK [mensagem pessoal]**. Mensagem recebida de <roy.oberhauser@htw-aalen.de>, em 04 de Agosto de 2011.

PEDROSO, Bruno. **Besouro: Aprimorando a Aferição Automática da Conformidade das Atividades de Desenvolvimento com TDD**. Dissertação (Mestrado) — (Draft) Universidade de Brasília, 2011.

PINTO, H. Sofia; STAAB, Steffen; TEMPICH, Christoph. Diligent: Towards a fine-grained methodology for distributed, loosely-controlled and evolving engineering of ontologies. IOS Press, p. 393–397, 2004.

SCHILIT, Bill; ADAMS, Norman; WANT, Roy. Context-aware computing applications. **IEEE Computer Society**, p. 85–90, 1994.

SCHLESINGER, Frank; JEKUTSCH, Sebastian. Electrocodeogram: An environment for studying programming. **TeamEthno-online Issue 2**, p. 67–76, June 2006.

SCHMIDT, Albrecht; BEIGL, Michael; GELLERSEN, Hans-W. There is more to context than location. **Computer and Graphics (Elsevier)**, v. 23, p. 893–901, 1999.

SEBESTA, Robert W. **Conceitos de Linguagens de Programação**. Porto Alegre: Bookman, 2011.

SENIN, Pavel. Software trajectory analysis: An empirically based method for automated software process discovery. **5th International Doctor Symposium on Empirical Software Engineering**, September 2010.

SENSOR, Hackystat Eclipse. **A Hackystat sensor for the Eclipse IDE**. 05 2010. Disponível em: <https://code.google.com/p/hackystat-sensor-eclipse/>.

SHAW, Ray. Rescu – on-line real-time artificial intelligence. **Computer-Aided Engineering Journal**, February 1987.

SMITH, Barry. Ontology. **Blackwell Guide to the Philosophy of Computing and Information**, p. 155–166, 2003. Disponível em: <http://ontology.buffalo.edu/smith/>.

SPARQL. **SPARQL Query Language for RDF**. January 2008. Disponível em: <http://www.w3.org/TR/rdf-sparql-query/>.

STEFANIDIS, Kostas; PITOURA, Evaggelia; VASSILIADIS, Panos. On supporting context-aware preferences in relational database systems. **Proc. of the International Workshop on Managing Context Information in Mobile and Pervasive Environments**, 2005.

TEIXEIRA, Pedro Henriques Dos Santos; MILIDIú, Ruy Luiz. Data stream anomaly detection through principal subspace tracking. **SAC'10**, p. 22–26, 2010.

VAJIRKAR, Pravin; SINGH, Sachin; LEE, Yugyung. Context-aware data mining framework for wireless medical application. **DEXA'2003**, p. 381–391, 2003.

VIEIRA, Vaninha et al. Ariane: An awareness mechanism for shared databases. **Proc of the X International Workshop on Groupware (CRIWG'04)**, p. 92–104, 2004.

VIEIRA, Vaninha et al. Uso e representação de contexto em sistemas computacionais. **Simpósio Brasileiro de Sistemas Colaborativos**, Novembro 2006.

VIEIRA, Vaninha; TEDESCO, Patricia; SALGADO, Ana Carolina. Modelos e processos para o desenvolvimento de sistemas sensíveis ao contexto. **Jornadas de Atualização em Informática 2009 (JAI'09)**, 2009.

VIEIRA, V. et al. Investigating the specifics of contextual elements management: The cemantika approach. **Proc. of the 6th International and Interdisciplinary Conference on Modeling and Using Context, CONTEXT07**, p. 493–506, 2007.

ZIMMERMANN, Andreas; LORENZ, Andreas; OPPERMANN, Reinhard. An operational definition of context. **Proc. of the 6th International and Interdisciplinary Conference on Modeling and Using Context, CONTEXT07**, p. 558–571, 2007.

APÊNDICE A – NOVAS ONTOLOGIAS

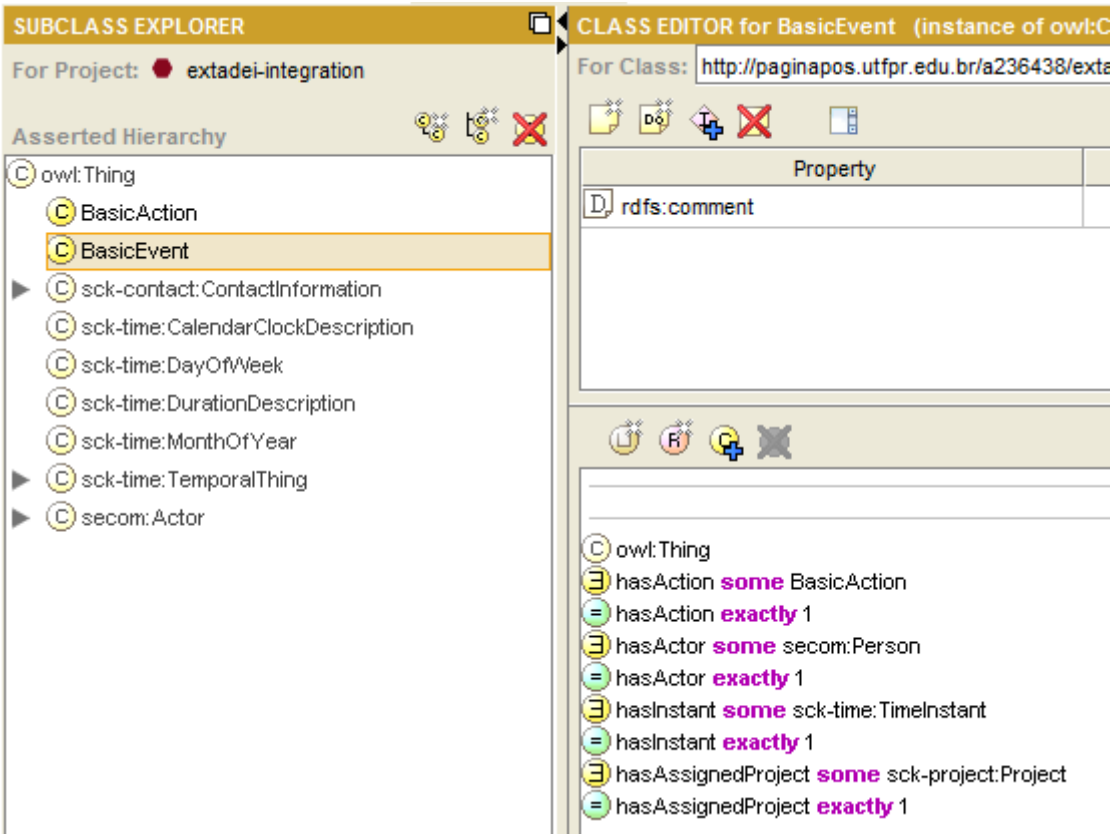

Figura 12: Classes da Ontologia de Integração

Fonte: O Autor

Figura 13: Classes da Ontologia *Hackystat*

Fonte: O Autor

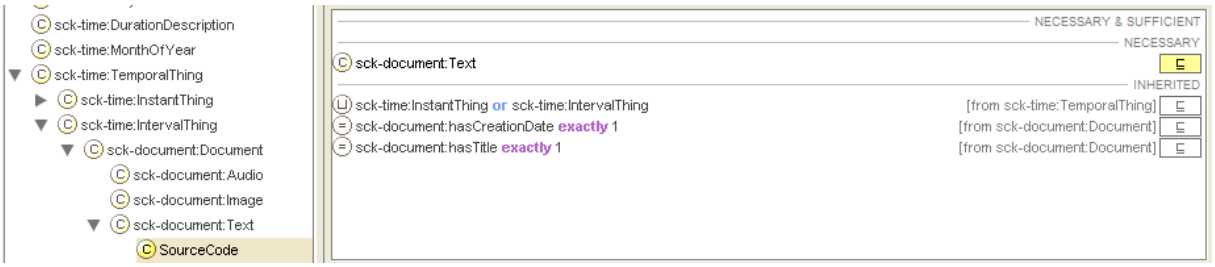

Figura 14: Classes da Ontologia *Hackystat*

Fonte: O Autor

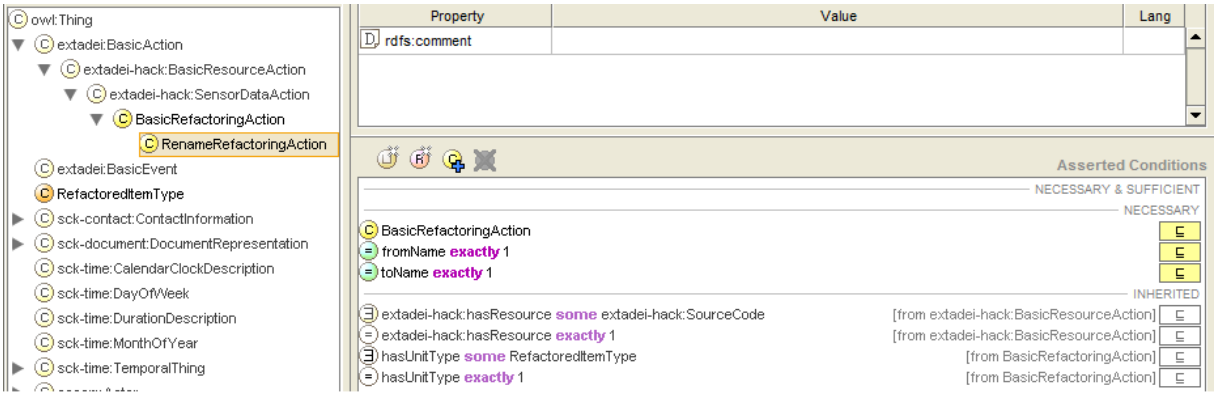

Figura 15: Classes da Ontologia Refatoração

Fonte: O Autor

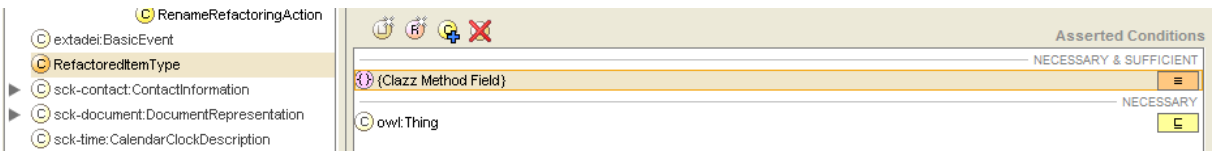

Figura 16: Classes da Ontologia Refatoração

Fonte: O Autor

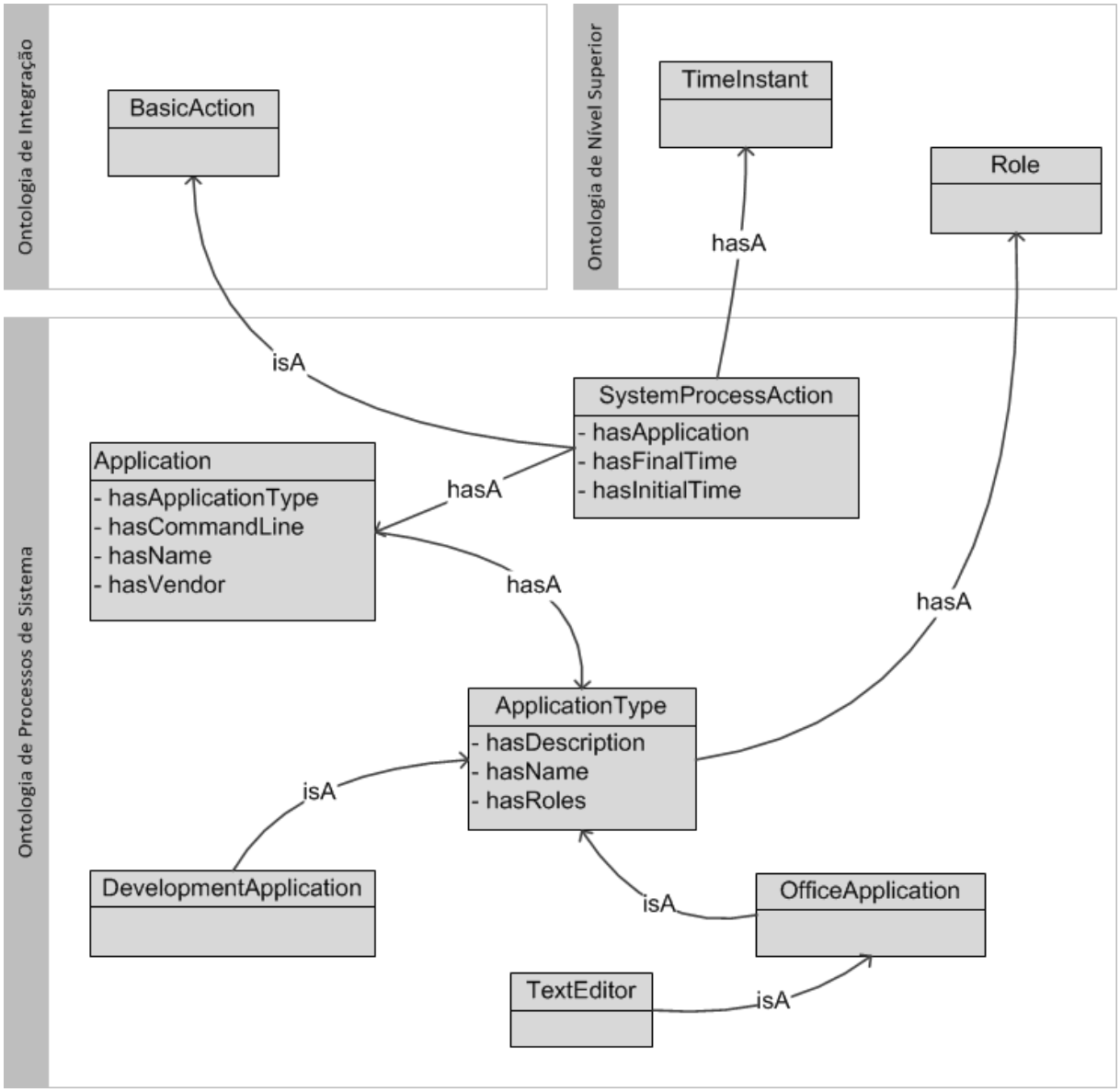

Figura 17: Classes da Ontologia de Processos de Sistema

Fonte: O Autor

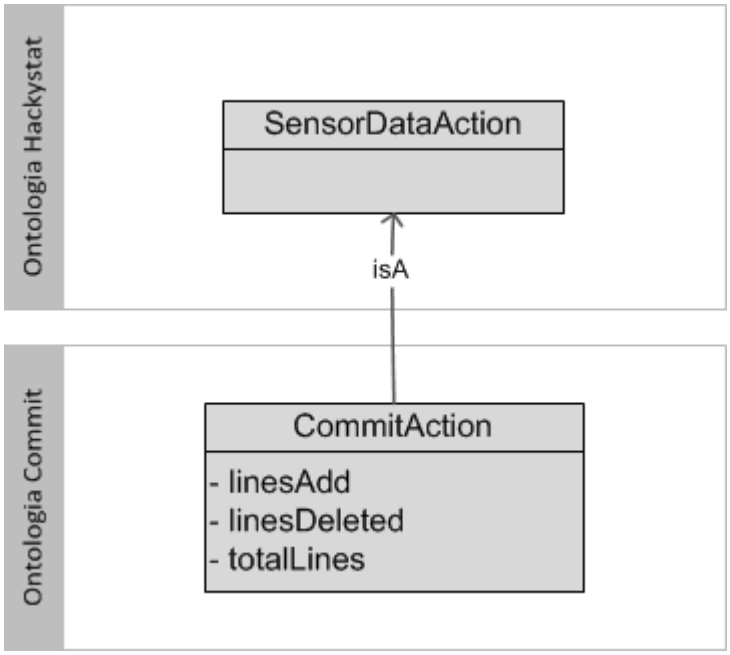

Figura 18: Classes da Ontologia *Commit*

Fonte: O Autor

APÊNDICE B – BIBLIOTECAS DE ONTOLOGIAS

Tabela 4: Bibliotecas de Ontologias

Nome	Ligação
DAML	http://www.daml.org/ontologies/
Ontolingua	http://www.ksl.stanford.edu/software/ontolingua/
ONTOLP	http://www.inf.pucrs.br/ ontolp/index.php
OntoSelect	http://olp.dfki.de/ontoselect?wicket:bookmarkable-Page=wicket0:de.dfki.ontoselect.Home
Protege Ontology Library	http://protege.cim3.net/cgi-bin/wiki.pl?Protege-OntologiesLibrary
SchemaWeb	http://www.schemaweb.info
SWOOGLE	http://swoogle.umbc.edu/

APÊNDICE C – REPOSITÓRIOS DE ONTOLOGIAS EXTADEI

Tabela 5: Repositórios de Ontologias eXTadEi

Nome	Ligação
Ontologia de Integração	http://paginapos.utfpr.edu.br/a236438/extadei/-ontologies/extadei-integration.owl
Ontologia Hackystat	http://paginapos.utfpr.edu.br/a236438/extadei/-ontologies/extadei-hackystat.owl
Ontologia Commit	http://paginapos.utfpr.edu.br/a236438/extadei/-ontologies/extadei-commit.owl
Ontologia Refactoring	http://paginapos.utfpr.edu.br/a236438/extadei/-ontologies/extadei-refactoring.owl
Ontologia de Processos de Sistema	http://paginapos.utfpr.edu.br/a236438/extadei/-ontologies/extadei-system-process.owl

APÊNDICE D – CONSULTAS SPARQL

Listagem D.1: SPARQL para consulta de atores

```
1 PREFIX  act:    <http://linkserver.icmc.usp.br/ckonto/actor#>
2 SELECT
3   ?hasFirstName ?hasSurname ?hasBirthday
4 WHERE {
5   ?element act:hasFirstName ?hasFirstName.
6   ?element act:hasSurname   ?hasSurname.
7   ?element act:hasBirthday  ?hasBirthday.
8 }
```

Listagem D.2: SPARQL para consulta de recursos

```
1 PREFIX secom-document: <http://linkserver.icmc.usp.br/ckonto/document#>
2 SELECT
3   ?hasTitle
4 WHERE {
5   ?element secom-document:hasTitle  ?hasTitle.
6 }
```

Listagem D.3: SPARQL para refatorações realizadas

```
1 PREFIX ext-hack:    <http://paginapos.utfpr.edu.br/a236438/extadei/
    ontologies/hackystat#>
2 PREFIX ext-refact:  <http://paginapos.utfpr.edu.br/a236438/extadei/
    ontologies/refactoring#>
3 SELECT
4   ?hasResource ?fromName ?toName
5 WHERE {
6   ?element ext-hack:hasResource ?hasResource.
7   ?element ext-refact:fromName  ?fromName.
8   ?element ext-refact:toName    ?toName.
9 }
```

APÊNDICE E – DADOS BRUTOS COLETADOS PELOS SENSORES DO HACKYSTAT

Listagem E.1: Dados brutos de Projetos

```
1 <Project LastMod="2011-03-10T00:18:34.489-03:00" Name="codigo-dissertacao
    "><Description>Codigo de dissertacao do mestrado</Description><
    StartTime>2010-03-22T00:00:00.000-03:00</StartTime><EndTime
    >2012-03-22T00:00:00.000-03:00</EndTime><Owner>josivanps@gmail.com</
    Owner><Members/><Invitations/><Spectators/><UriPatterns><UriPattern
    >*/ProcessListener/*</UriPattern><UriPattern>*/ProcessListenerNative
    /*</UriPattern></UriPatterns><Properties/></Project>
```

Listagem E.2: Dados brutos de Sensores

```
1 <SensorData LastMod="2010-04-25T16:07:53.643-03:00"><Timestamp>2010-04-24
    T16:09:45.445-03:00</Timestamp><Runtime>2010-04-24T16
    :09:45.445-03:00</Runtime><Tool>Eclipse</Tool><SensorDataType>
    DevEvent</SensorDataType><Resource>file:/D:/work/FACET2010/src/br/
    facet/si/desenvolvimento/beans/Jogador.java</Resource><Owner>
    josivanps@gmail.com</Owner><Properties><Property><Key>Subtype</Key><
    Value>ProgramUnit</Value></Property><Property><Key>Type</Key><Value>
    Edit</Value></Property><Property><Key>Language</Key><Value>java</
    Value></Property><Property><Key>Unit-Type</Key><Value>Field</Value></
    Property><Property><Key>To-Unit-Name</Key><Value>String ssapelido</
    Value></Property><Property><Key>From-Unit-Name</Key><Value>s</Value
    ></Property><Property><Key>Subsubtype</Key><Value>Rename</Value></
    Property></Properties></SensorData>
```

ANEXO A – ARQUITETURAS PROPOSTAS ANTERIORMENTE

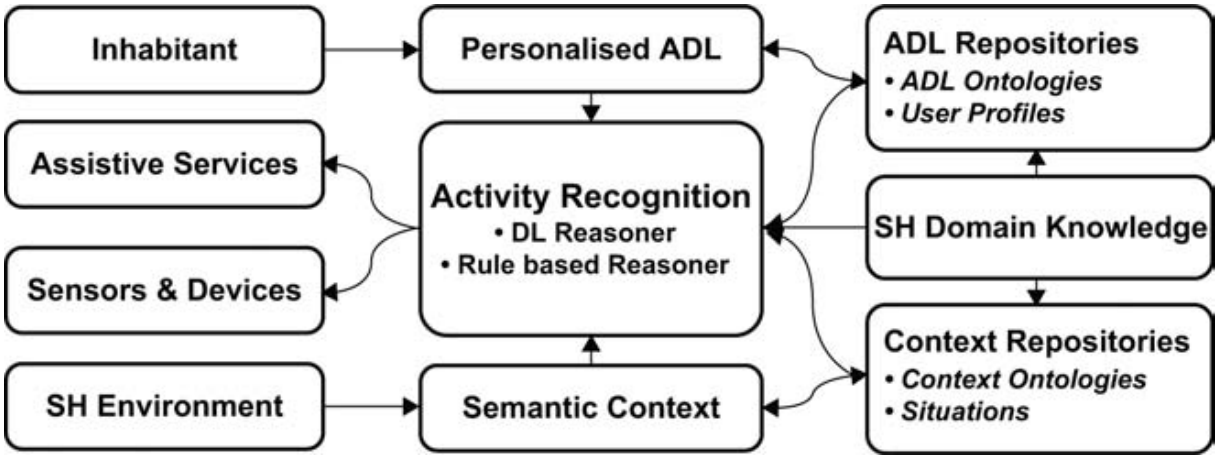

Figura 19: Arquitetura proposta por Chen e Nugent (2009)

Fonte: Chen e Nugent (2009)

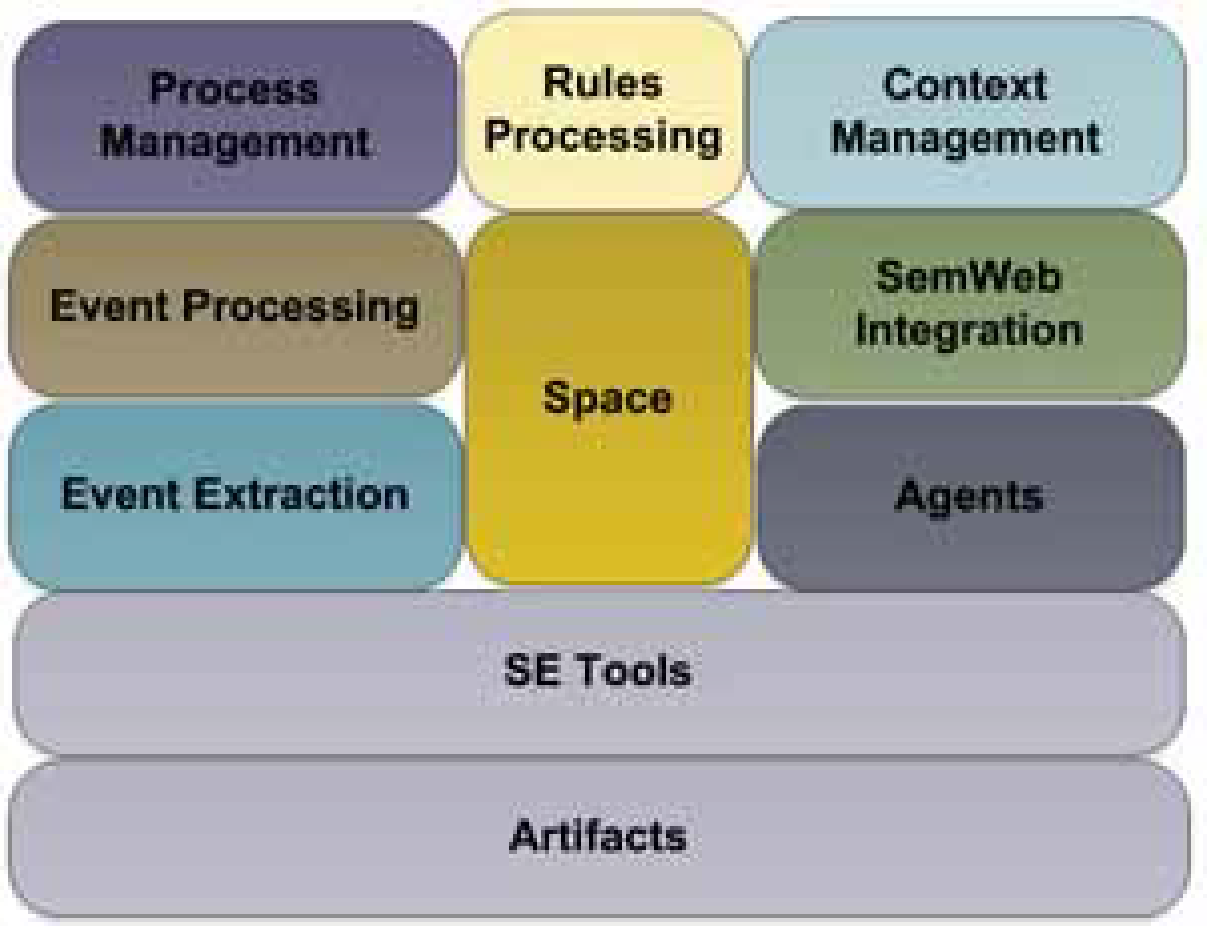

Figura 20: Visão Conceitual do CoSEEEK

Fonte: Oberhauser (2010)

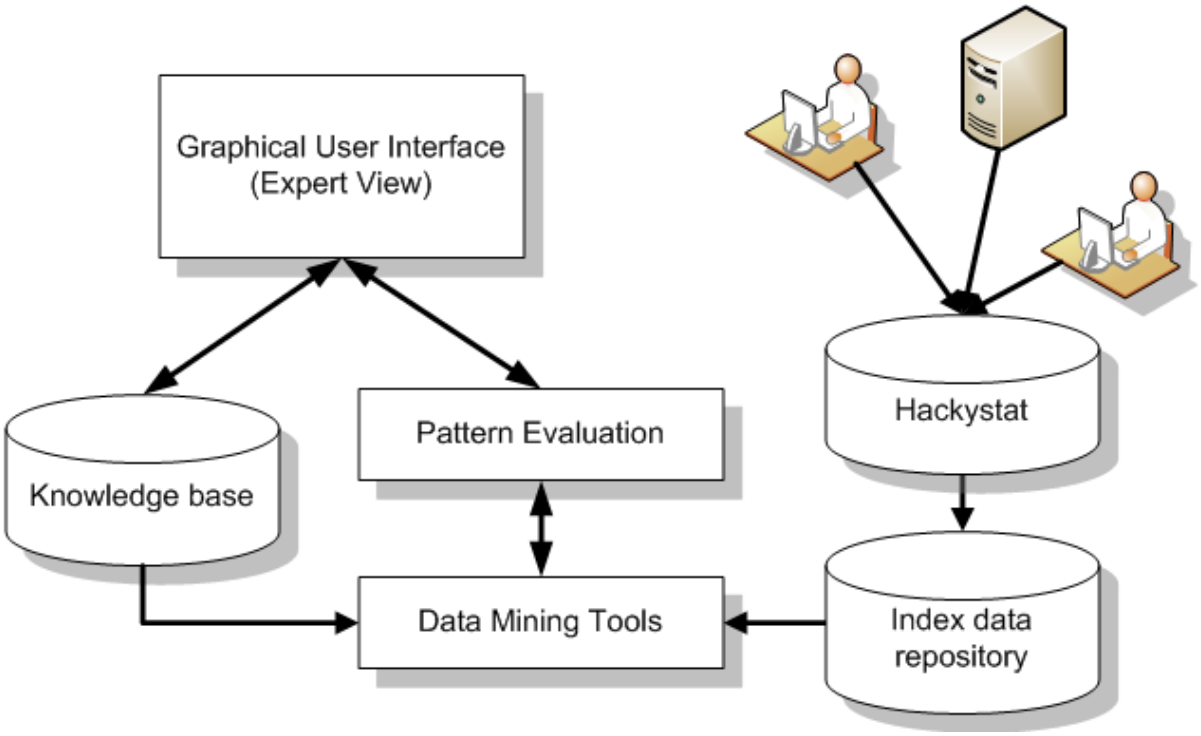

Figura 21: Arcabouço *"Software Trajectory"*

Fonte: Senin (2010)

ANEXO B – ONTOLOGIAS DO MODELO SECOM

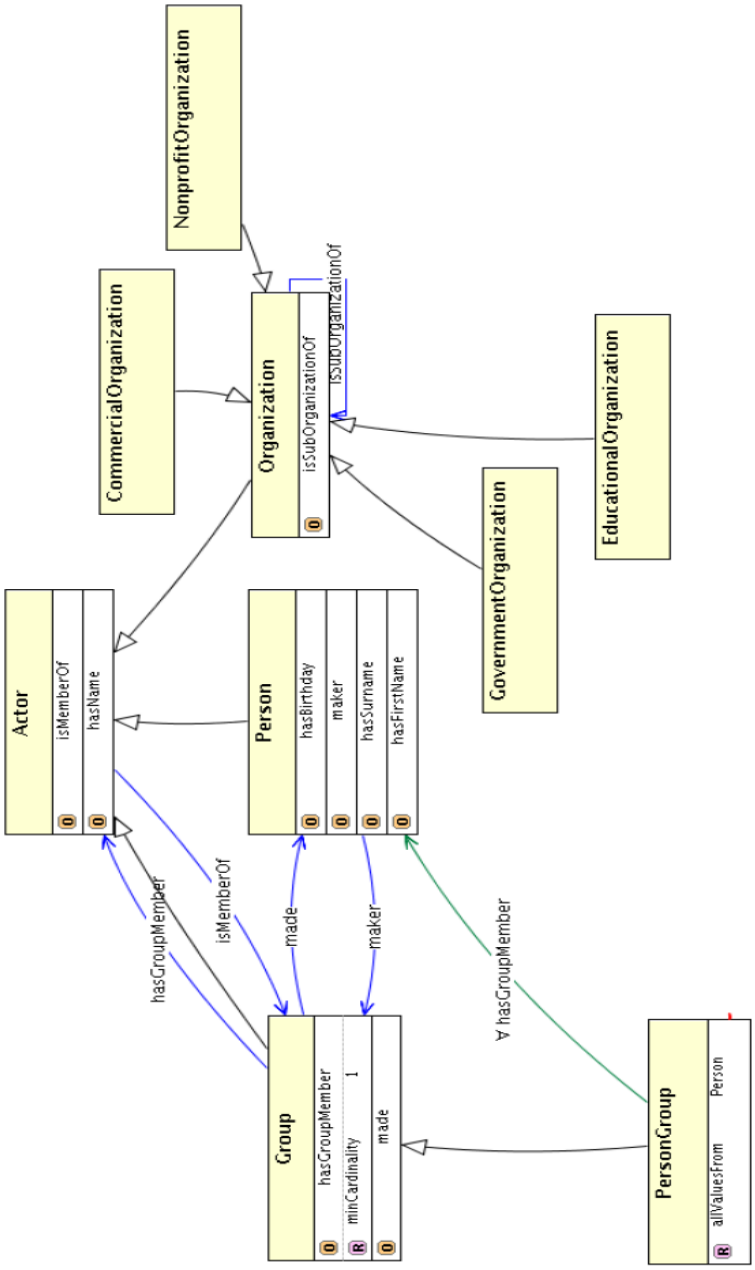

Figura 22: Ontologia *Actor*

Fonte: Neto (2006)

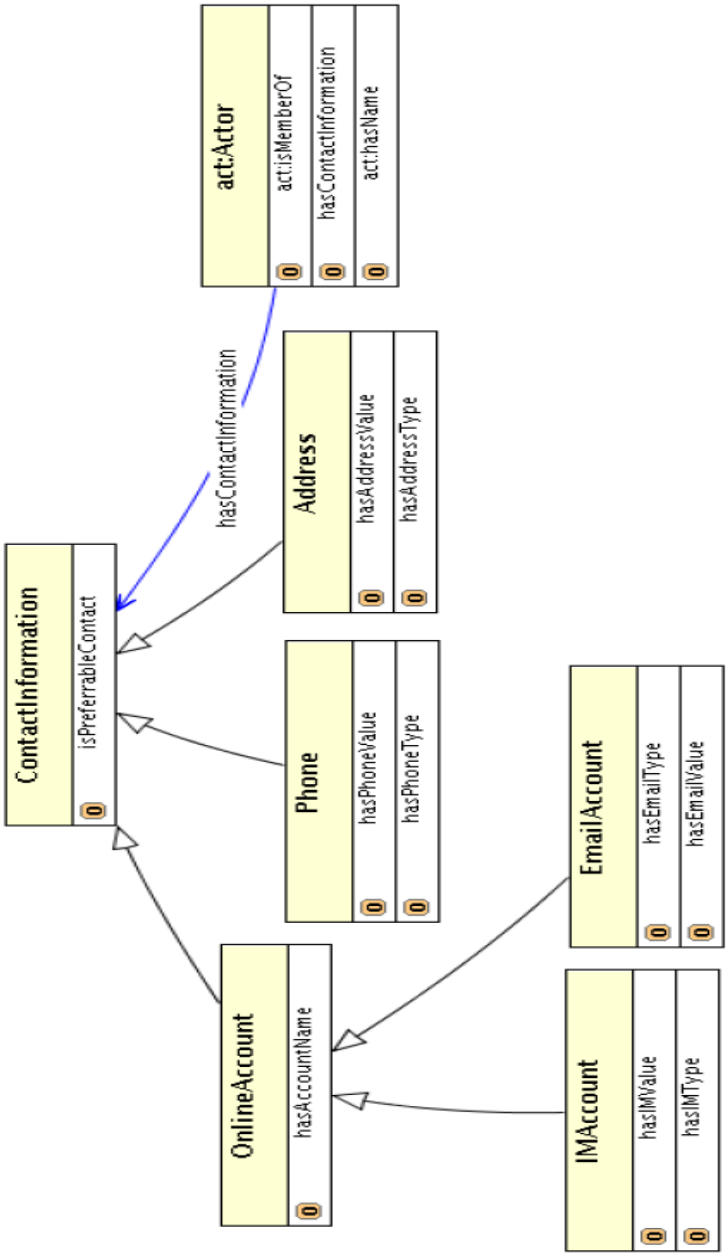

Figura 23: Ontologia *Contact*

Fonte: Neto (2006)

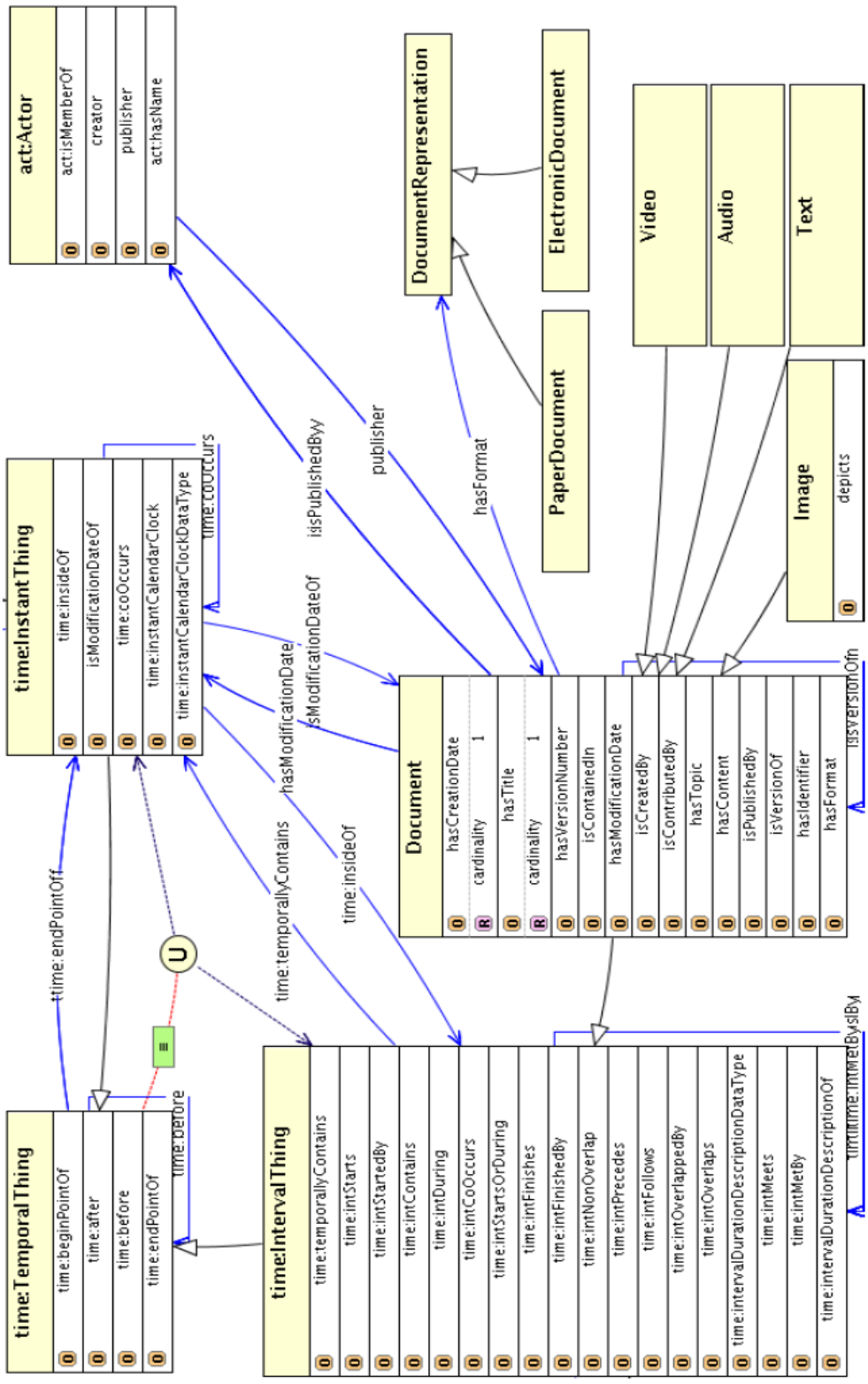

Figura 24: Ontologia *Document*

Fonte: Neto (2006)

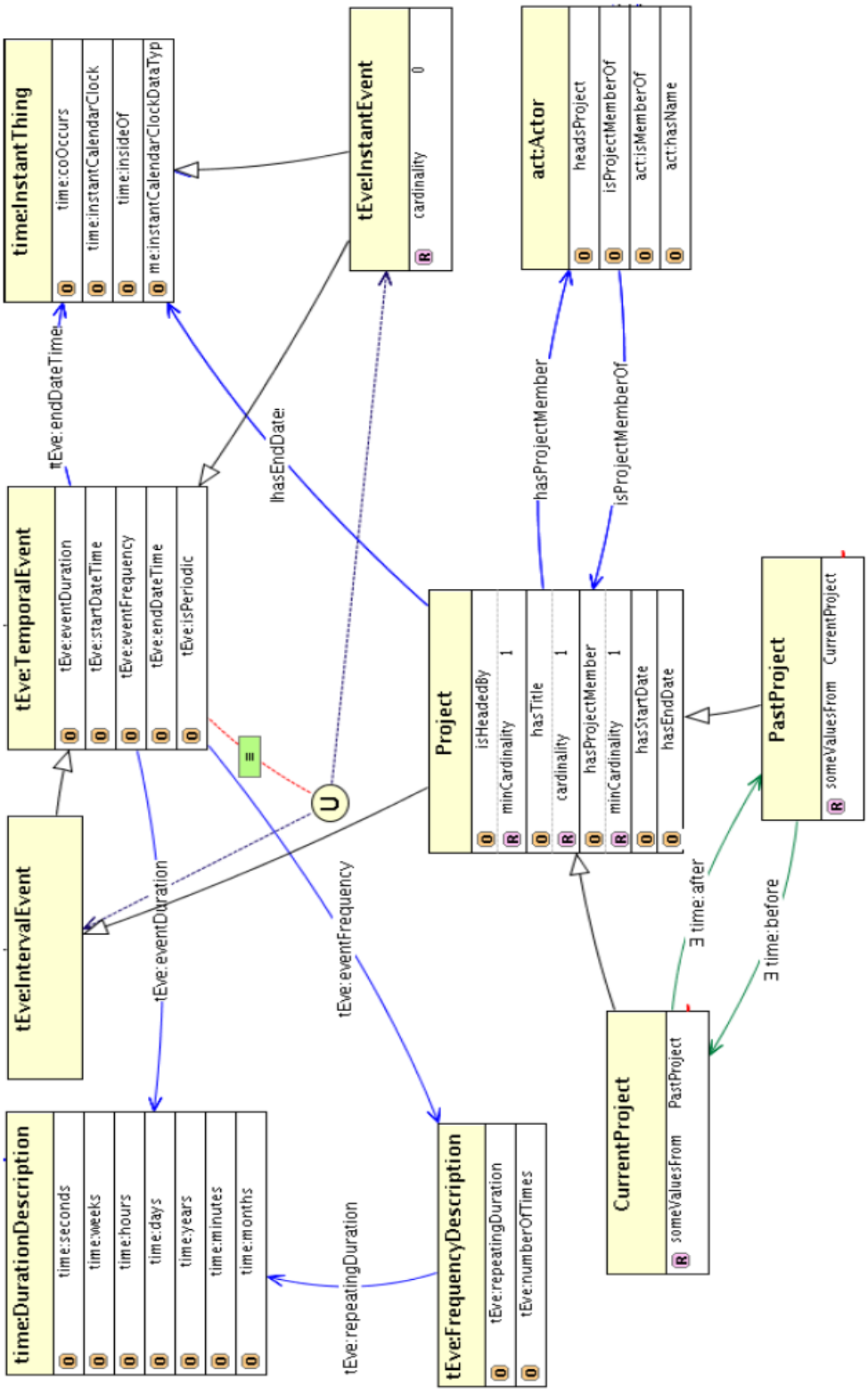

Figura 25: Ontologia *Project*

Fonte: Neto (2006)

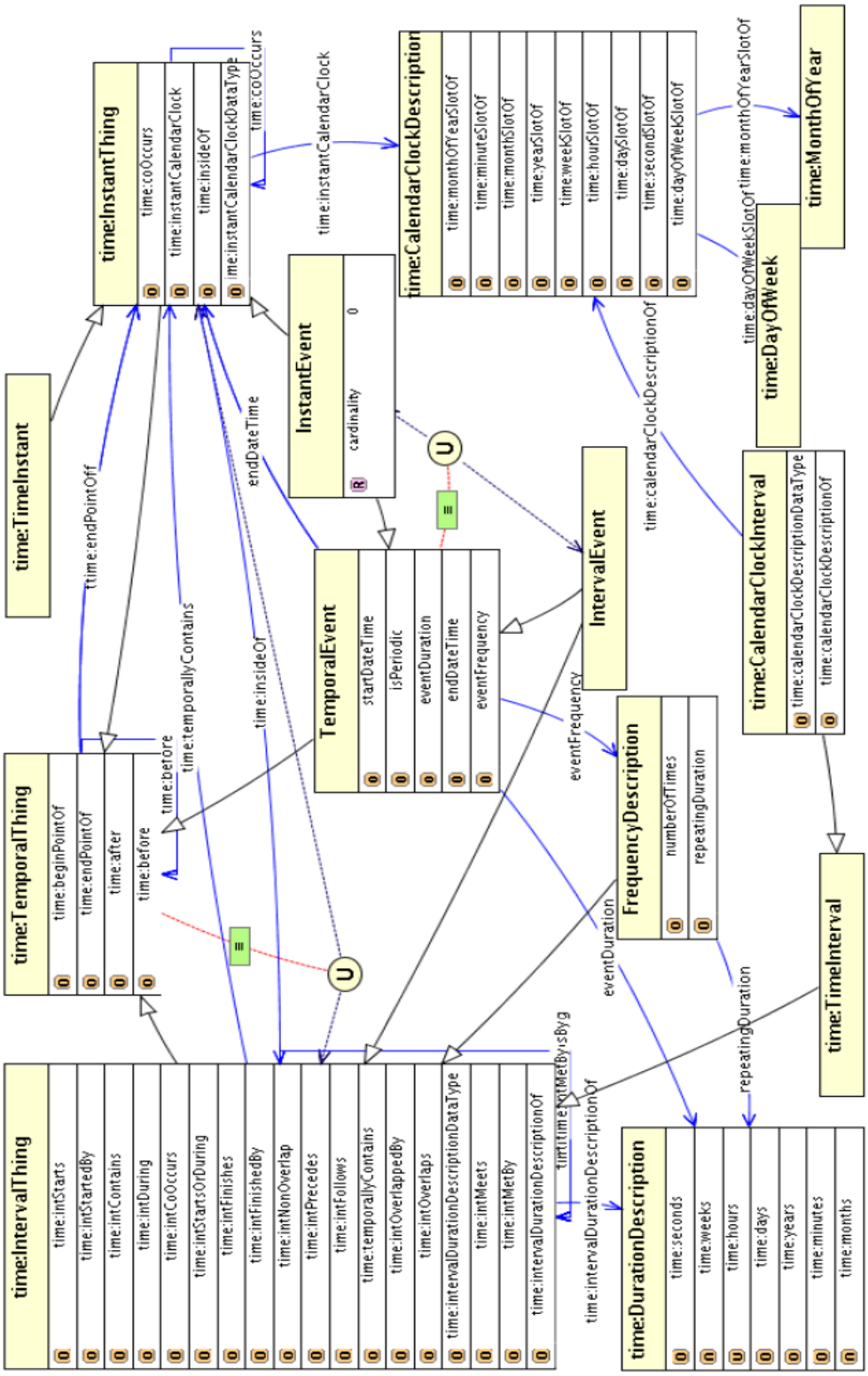

Figura 26: Ontologia *Temporal Event*

Fonte: Neto (2006)

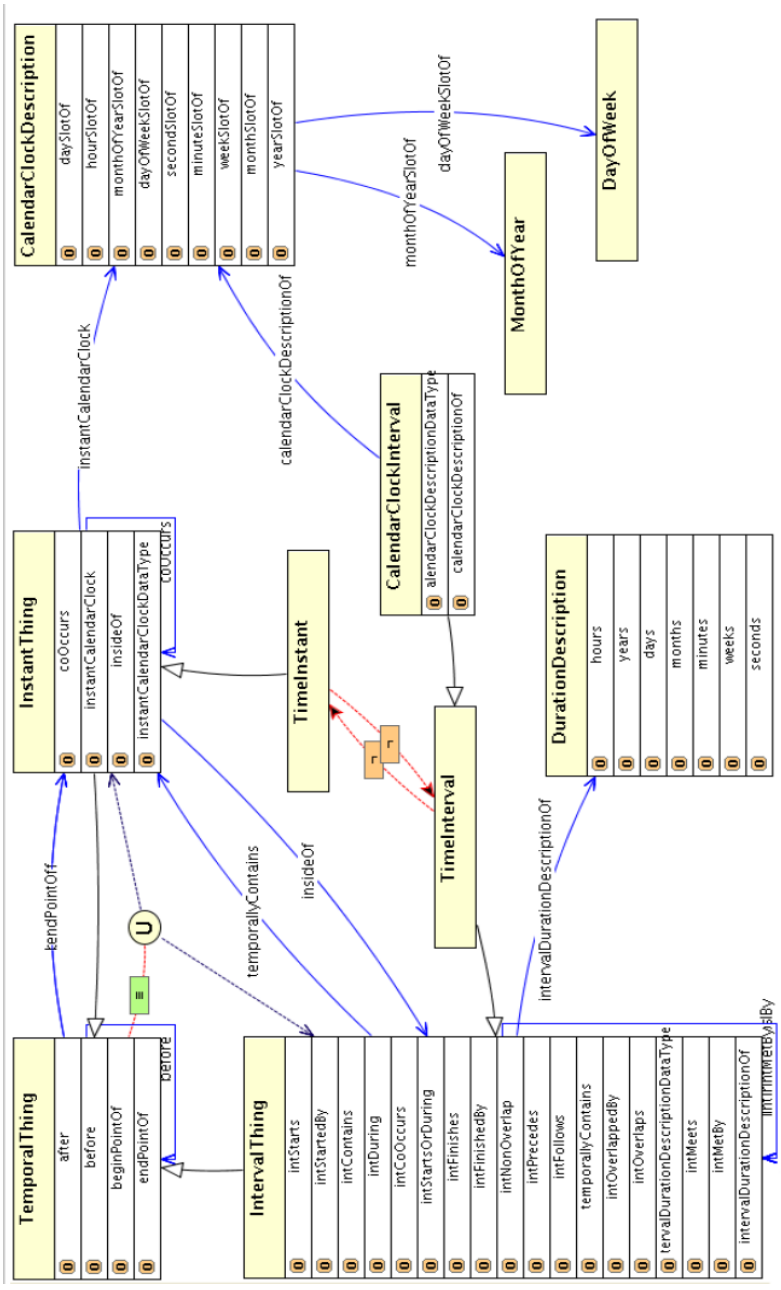

Figura 27: Ontologia *Time*

Fonte: Neto (2006)