Universidade Federal de Santa Maria

Centro de Tecnologia

Curso de Ciência da Computação

Sistemas de Tipos para Capturar Informação de Contexto em Computação Pervasiva

Trabalho de Graduação

Santa Maria, RS, Brasil

2010

Sistemas de Tipos para Capturar Informação de Contexto em Computação Pervasiva

por

Jorge Luiz Titoneli Pinto Filho

Trabalho de Graduação apresentado ao curso de Ciência da Computação da Universidade Federal (UFSM, RS), como requisito parcial para obtenção do grau de

Bacharel em Ciência da Computação

Orientador: Prof[a] Dr[a] Juliana Kaizer Vizzotto

Trabalho de Graduação, N[o] 311

Santa Maria, RS, Brasil

2010

Universidade Federal de Santa Maria

Centro de Tecnologia

Curso de Ciência da Computação

A comissão Examinadora, abaixo assinada,

aprova o Trabalho de Graduação

Sistemas de Tipos para Capturar Informação de Contexto em Computação Pervasiva

Elaborado por

Jorge Luiz Titoneli Pinto Filho

como requisito parcial para obtenção do grau de

Bacharel em Ciência da Computação

COMISSÃO EXAMINADORA:

_____
Prof[a] Dr[a] Juliana Kaizer Vizzotto
(Presidente/Orientador)

_____
Prof[a] Dr[a] Deise Brum Saccol (UFSM)

_____
Prof Dr Giovani Librelotto (UFSM)

Santa Maria, 09 de dezembro de 2010

AGRADECIMENTOS

Agradeço primeiramente a Deus, por ter tornado tudo na minha vida possível, por todos os momentos incríveis que vivenciei, por todas as pessoas maravilhosas que colocou no meu caminho, que fez tudo parecer destinado a dar certo, enfim, por ter me dado forças para chegar até esse momento.

À minha família, minha mãe, minhas irmãs, minha madrinha, e principalmente ao meu pai, que sempre serviu de suporte, uma grande fonte de conhecimento e orientação, de forma atenciosa e paciente. Uma família que sempre me apoiou e acreditou em mim, em todos os momentos, mesmo nos mais difíceis, em que eu não sabia que caminho seguir.

Agradeço meus colegas de curso, tanto da UFSM quanto da UFSC, por tantas horas de companheirismo, fazendo trabalhos, tirando dúvidas, até mesmo jogando conversa fora, onde pude aprender alguma coisa com cada um.

Aos meus amigos e minha namorada, por terem sido fundamentais e tornarem esses anos de curso suportáveis e mais divertidos. Sempre pude contar com eles em todos os momentos, para me distrair um pouco, sair um pouco da rotina de estudos que o curso exige. Algumas pessoas que infelizmente estão longe tenho muito carinho e saudades.

Aos mestres, que tiveram a preocupação em transmitir conhecimento e desafiar nossos limites. Tiveram muita importância em nos conduzir até o fim da faculdade. Em especial agradeço à professora Juliana, que com muita paciência e dedicação me orientou na realização deste trabalho.

Com certeza existem muitas outras pessoas que sou grato, que tiveram grande influência na minha vida até este momento. Por isso peço desculpa por não citá-los aqui, mas tenham certeza que agradeço tudo o que fizeram por mim.

"Nossas dúvidas são traidoras e nos fazem perder o que poderia ser nosso pelo simples medo de tentar."

- William Shakespeare

RESUMO

Trabalho de Graduação

Curso de Ciência da Computação

Universidade Federal de Santa Maria

Sistemas de Tipos para Capturar Informação de Contexto em Computação Pervasiva

Elaborado por: Jorge Luiz Titoneli Pinto Filho

Orientador: Prof[a] Dr[a] Juliana Kaizer Vizzotto

Local e data da defesa: Santa Maria, 09/12/2010

A Computação Pervasiva já é uma realidade no cotidiano das pessoas. Todos os dias novos dispositivos são criados visando à segurança, à comodidade, ao entretenimento e ao bem-estar geral do indivíduo. Os ambientes inteligentes estão se tornando frequentes e se aproximando celeremente do cidadão comum. A demanda por softwares que gerenciem esta interação homem-ambiente-dispositivo também cresce concomitantemente, exigindo mecanismos que auxiliem a construção dos mesmos. A captura e organização correta dos tipos de dados que deverão ser manipulados pelos aplicativos presentes no ambiente constitui fator fundamental para o sucesso das respostas requeridas pelos usuários. Daí a ideia de se realizar este estudo sobre um sistema de tipos que possa estruturar as variáveis presentes no contexto de um ambiente com implementação da computação pervasiva.

Palavras-chave: Computação pervasiva, sistema de tipos

ABSTRACT

Trabalho de Graduação

Graduate Program in Computer Science

Universidade Federal de Santa Maria

Type Systems to Capture Context Information in Pervasive Computing

Author: Jorge Luiz Titoneli Pinto Filho

Advisor: Prof[a] Dr[a] Juliana Kaizer Vizzotto

The Pervasive Computing is already a reality in daily life. Everyday new devices are designed aiming at the safety, convenience, entertainment and general welfare of the individual. Smart environments are becoming frequent and swiftly approaching the ordinary citizen. The demand for software to manage this interaction man-environment-device grows concomitantly requiring mechanisms that contribute to the construction of them. The capture and correct organization of the types of data to be manipulated by applications in the environment is a fundamental factor for the success of the responses required by users. Hence the idea of conducting this study on a type system that can structure the variables present in an environment with implementation of pervasive computing.

Keywords: Pervasive computing, type system

Lista de Figuras

FIGURA 1.	Computação Pervasiva baseada em *Tele-home Healthcare System*..	16
FIGURA 2.	Trânsito inteligente utilizando RFID.....	18
FIGURA 3.	Exemplo de modelagem do contexto da aplicação.....	20
FIGURA 4.	A Ontologia do Núcleo.....	42
FIGURA 5.	A Teoria do Contexto.....	43
FIGURA 6.	Classes e propriedades de Contexto na Arquitetura CoBrA, baseadas em Ontologia...	44
FIGURA 7.	A formação de CRTs.....	48
FIGURA 8.	A arquitetura global.....	49
FIGURA 9.	Entidades da situação.....	53
FIGURA 10.	Sala de estar mobiliada.....	55
FIGURA 11.	Ontologia da sala de estar.....	56

Lista de Abreviaturas

ATN - Augmented Transition Networks

CPU - Central Processing Unit

CRT - Context Record Types

DRT - Dependent Record Types

FOL - First-Order Logic

HD - Hard Disk

ITT - Intuitionistic Type Theory

ORM - Object-Role Modeling

PC - Personal Computer

RAM - Random Acess Memory

RFID - Radio-Frequency IDentification

UML - Unified Modeling Language

WSMO - Web Service Modeling Ontology

Sumário

1. **INTRODUÇÃO** 11
1.1 OBJETIVOS 12
1.2 JUSTIFICATIVA 12
1.3 METODOLOGIA 13
1.4 ESTRUTURA DO TEXTO 13

2. REVISÃO BIBLIOGRÁFICA 15
2.1 COMPUTAÇÃO PERVASIVA 15
2.2 CONTEXTO EM COMPUTAÇÃO PERVASIVA 16
2.3 FORMALISMOS PARA CAPTURAR CONTEXTO 18
2.3.1 ISAMadapt 20
2.3.2 Modelo Key-Value 21
2.3.3 Modelo Esquema de Marcação 21
2.3.4 Modelos Gráficos 22
2.3.5 Modelos Orientados a Objeto 22
2.3.6 Modelos Baseados em Lógica 22
2.3.7 Modelos Baseados em Ontologia 23
2.4 SISTEMAS DE TIPOS 24
2.4.1 Erros de Execução 25
2**.4.2 Propriedades Esperadas de um Sistema de Tipos** 26
2**.4.3 Sistemas de Tipos Especializados** 27
2.4.4 Linguagens Tipadas e Não Tipadas 29
2.4.5 Formalização de um Sistema de Tipos 30
2**.4.6 A Linguagem de um Sistema de Tipos** 32
2.5 TIPOS DE REGISTRO DE DEPENDENTES 36
2.6 TIPOS DE REGISTRO DE CONTEXTOS 37
2.7 SUBTIPOS COM CONTEXTOS 40
2.8 A ONTOLOGIA DE DOMÍNIO 41

3. TRATANDO CONTEXTO ATRAVÉS DE SISTEMA DE TIPOS 45
3.1 CONSCIÊNCIA DE CONTEXTO EM SISTEMA DE TIPOS 45
3.2 CONTEXTO EM INTELIGÊNCIA ARTIFICIAL 46
3.3 RECURSOS DA IMPLEMENTAÇÃO 47
3.4 EXEMPLO DE APLICAÇÃO – SMART PHONE 50

4. ESTUDO DE CASO 54
4.1 DESCRIÇÃO DA SITUAÇÃO 54
4.2 ONTOLOGIA 55
4.3 SISTEMA DE TIPOS 56
4.4 CONCLUSÃO DO ESTUDO DE CASO 59

5. CONCLUSÃO E FUTUROS TRABALHOS 61

REFERÊNCIAS 63

1. INTRODUÇÃO

Na Era da Informação houve progresso prodigioso nas relações humanas no que se refere às possibilidades de comunicação mais eficiente entre as pessoas, não importando as distâncias ou a quantidade de interlocutores. A internet veio coroar esta era, inaugurando a utilização dos chamados Computadores Coletivos (CCs), no qual as informações podem ser compartilhadas por milhares de pessoas ao mesmo tempo, mesmo estando aquelas dentro de apenas uma máquina.

A Computação Móvel [Luca Cardelli, 1998] fez com que a interação entre as pessoas se desse em qualquer momento e de qualquer lugar. Os telefones móveis, cada vez mais cheios de recursos, as redes *Wi-Fi*, as casas comerciais que fornecem acesso à internet, permitem que toda pessoa possa estar sempre conectada. Sobre este fato convém verificar-se o que foi dito por [Weinberger, 2008]:

> Não estamos na era da informação. Não estamos na era da Internet. Nós estamos na era das conexões. Ser conectado está no cerne da nossa democracia e nossa economia. Quanto maior e melhor forem essas conexões, mais fortes serão nossos governos, negócios, ciência, cultura, educação...

Hoje, o importante é estar conectado, i.e., ter a possibilidade de obter no mais curto prazo a informação desejada.

Surgiu recentemente e constitui tendência atual a utilização de computadores "invisíveis", em que a interação homem-máquina é feita de forma menos tradicional que a utilização de teclados, telas sensíveis ao toque ou outras formas que utilizam diretamente os periféricos. Os computadores são dispostos no ambiente de modo a responder aos estímulos do usuário sem que este tenha, necessariamente, contato com a máquina. Este é um conceito aproximado de Computação Pervasiva. As máquinas são distribuídas no ambiente, de forma perceptível ou não, e através de sensores ou outros meios de comunicação como RFID (acrônimo para o termo em inglês "*Radio-Frequency IDentification*") ou *Bluetooth*, se comunicam com os usuários, captando as informações destes, analisando-as e executando o processamento de acordo com as necessidades.

A Computação Pervasiva deve ser sensível ao contexto, apresentando soluções diferentes para situações diferentes. Uma situação diferente pode ser caracterizada pela mudança do usuário ou usuários, mudança nas condições climáticas ou de tempo (horário, por exemplo), ou do contexto computacional [Schilit e outros, 1994].

Formalismos para capturar informações do contexto são utilizados para auxiliar na construção de ambientes, equipamentos, softwares, entre outros. Existem diversos formalismos empregados para capturar informações do contexto, como por exemplo: i) o modelo de *Key-Value*, que é muito usado em *frameworks* de serviços distribuídos; ii) o modelo de esquema de marcação, onde é comum existir uma estrutura de dados hierárquica consistindo de *tags* de marcação com atributos e conteúdo; iii) os modelos gráficos, onde o UML (*Unified Modeling Language*) e o ORM (*Object-Role Modeling*) podem ser empregados; iv) o modelo orientado a objetos, que tem a intenção de empregar os benefícios principais da orientação a objetos, como o encapsulamento e reusabilidade; v) os modelos baseados em lógica, onde é comum o alto grau de formalidade; e vi) os modelos baseados em ontologias, que são instrumentos promissores para especificar conceitos e interrelações.

Uma abordagem formal alternativa, que constitui fator fundamental para a boa implementação do software de controle do contexto, baseia-se na estruturação dos tipos de dados de ambiente a serem manipulados. Sistema de Tipos [B. Pierce, 2002] é um *framework* sintático tratável para a classificação de frases de acordo com os tipos de valores que elas computam.

1.1 OBJETIVOS

O objetivo deste trabalho é realizar um estudo de caso sobre a utilização dos sistemas de tipos na computação pervasiva, em que é verificada a validade dessa utilização para a captura de informações de contextos em ambientes pervasivos. Para este fim, foi tomado como base o artigo de [Dapoigny e Barlatier, 2007], no qual é implementado um sistema de tipos calcado em alguns conceitos como ontologias, Tipos de Registros Dependentes (DRT) e Tipos de Registros de Contexto (CRT).

1.2 JUSTIFICATIVA

Como a computação pervasiva está cada vez mais disseminada no mercado, verifica-se a necessidade crescente de softwares que atendam à maior demanda dos aparelhos “em qualquer lugar” [Domingues, 2008]. A criação desses softwares de forma acelerada impõe a necessidade de utilização de mecanismos que aperfeiçoem o esforço dos programadores, visando à criação rápida dos produtos, sem, contudo, perder a eficácia dos mesmos.

Os formalismos facilitam a vida do programador, definindo uma estrutura e a forma como os programas devem ser feitos. Sistema de Tipos surge como um formalismo muito utilizado, pois antecipando as regras e padronizando os procedimentos, conduz o raciocínio lógico dentro de um escopo mais limitado, em face da grande quantidade de variáveis presentes em ambientes pervasivos. Em cada ambiente podem se suceder frequentes mudanças de contextos, o que torna muito complexa a captura e análise das variáveis, justificando a necessidade daquela padronização.

1.3 METODOLOGIA

A metodologia aplicada para o desenvolvimento do trabalho foi a seguinte. Primeiramente, estudou-se a computação pervasiva, seus conceitos, definições, e um exemplo foi utilizado para o melhor entendimento. O contexto, também foi estudado, pois sua definição é fundamental na computação pervasiva, para, no estudo dos formalismos, permitir-se ter uma ideia de como as informações poderiam ser capturadas. Sistema de tipos e suas regras também foi alvo de estudo, para assim demonstrar seu funcionamento, um pouco de suas vantagens e alguns de seus usos. DRTs, CRTs e ontologia de domínio foram demonstrados para se ter uma base para o melhor entendimento da abordagem utilizada no artigo alvo de estudo do presente trabalho. Completando o trabalho foi realizado um estudo de caso empregando os conceitos anteriormente abordados.

1.4 ESTRUTURA DO TEXTO

O presente trabalho está estruturado em cinco capítulos.

No Capítulo 2 é apresentada a revisão bibliográfica com os conhecimentos necessários para o melhor entendimento deste estudo. É apresentada a noção de contexto, e sua importância para a Computação Pervasiva. Os formalismos para capturar informação do contexto são especificados mais detalhadamente, para assim expor a motivação para o uso de sistemas de tipos.

No Capítulo 3 é apresentado um estudo de um exemplo de aplicação usando sistema de tipos baseado no modelo proposto por Richard Dapoigny e Patrick Barlatier, no qual os autores exploram CRT, DRT e ontologias na formalização dos tipos de dados.

No Capítulo 4 é apresentado um estudo de caso, envolvendo a sala de estar de uma casa inteligente (*Smart House*), explicitando a praticidade dos conceitos e da abordagem apresentada.

No Capítulo 5 é apresentada a conclusão do trabalho, realçando a validade da implantação dos sistemas de tipos e finalizando com informações sobre possíveis trabalhos futuros.

2. REVISÃO BIBLIOGRÁFICA

Este capítulo apresenta a revisão bibliográfica deste trabalho. A seção que se segue apresenta uma introdução à computação pervasiva, em cujo escopo é feito o presente estudo. O uso de linguagens de programação para a computação é abordada em seguida, com um resumo de suas necessidades e características. Discute-se, a partir de então, a noção de computação sensível ao contexto e seus conceitos. Finalmente, discorre-se sobre noção de sistemas de tipos.

2.1 COMPUTAÇÃO PERVASIVA

O termo Computação Pervasiva foi proposto por Mark Weiser [Weiser, 1991], no início dos anos noventa, como uma nova tendência da computação. Ela visa tornar o computador transparente para o usuário, fornecendo computação em qualquer lugar, através da virtualização de serviços, informações e aplicações. Em um ambiente pervasivo, uma pessoa, por exemplo, poderia acionar uma máquina de lavar através do celular [Dey, 2000]. Computação Pervasiva consiste de uma grande variedade de dispositivos, móveis ou fixos, e serviços interconectados. Além disso, os computadores teriam sistemas inteligentes que estariam conectados ou procurando conexão o tempo todo, dessa forma tornando-se onipresentes.

A Computação Pervasiva se propõe a ser mais abrangente em relação às conexões entre o homem e o ambiente. Os cinco principais sentidos humanos terão maior atuação na comunicação que, na maior parte das vezes, far-se-á de forma transparente para o indivíduo. Serão os sensores e outros dispositivos ligados aos diversos computadores que irão captar as variações do ambiente, mudanças de atitudes do usuário e mesmo suas intenções e necessidades e responderão da forma apropriada.

Pelo menos dois passos devem ser considerados pela computação pervasiva para esta ser considerada onipresente: o primeiro passo, que se refere à necessidade de se chegar ao nível de interação invisível para o usuário, seria utilizar interfaces naturais, como a fala, gestos, para assim não ser mais necessário o uso de teclados e mouses; e o segundo passo diz respeito à geração de uma computação sensível ao contexto, na qual será possível que os dispositivos capturem as alterações do ambiente automaticamente, ou seja, de forma dinâmica.

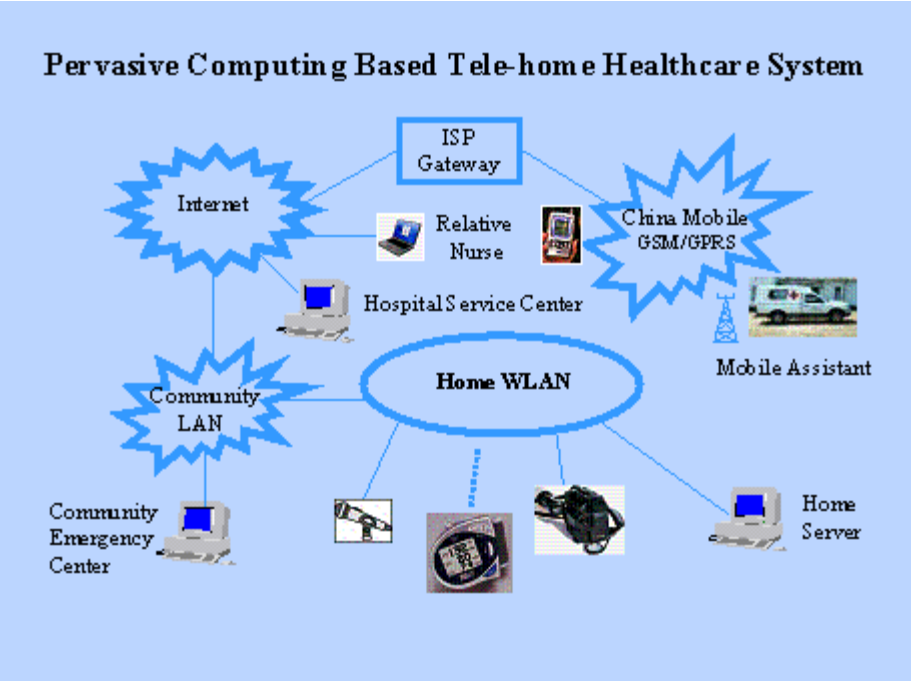

Figura 1 – Computação Pervasiva baseada em *Tele-home Healthcare System*
Fonte: http://www.jucs.org/jucs_12_1/a_pervasive_multimodal_tele/jucs_12_01_0099_0114_miao.html

A figura acima mostra um exemplo de emprego da computação pervasiva. Na casa de um usuário existe uma série de equipamentos de captura de informações (microfones, câmeras, sensores) ligados a uma rede local que recebe os dados do usuário e, através da internet, transmitem os mesmos para um hospital. De acordo com a análise realizada, um sistema de ambulâncias pode ser acionado por telefone móvel para atendimento ao usuário. Um paciente cardíaco, por exemplo, pode acionar o sistema sem a necessidade de telefonar ou enviar uma mensagem via teclado do computador. A análise do ambiente, realizada pelos equipamentos e sensores, permite que o computador local tome as decisões que o caso requeira.

2.2 CONTEXTO EM COMPUTAÇÃO PERVASIVA

Um contexto pode ser caracterizado por qualquer informação que é inerente a uma situação e que, dependendo de seu valor, pode alterar o resultado de uma ação.

O contexto, segundo [Dey, 2001], é qualquer informação que pode ser usada para caracterizar a situação de uma entidade. Entidade é uma pessoa, objeto ou local que é considerado relevante para a interação entre um usuário e aplicação. Em [Che, 2000] foram definidos quatro tipos de contexto: Contexto da Computação (redes e recursos), Contexto do Usuário (pessoas, lugares e objetos), Contexto Físico (luz, odor, temperatura) e Contexto Temporal (hora, dia, mês). Um exemplo de contexto é a

capacidade de um dispositivo detectar a temperatura em um determinado ambiente e atuar sobre equipamentos (condicionadores de ar) para fornecer o valor ideal de temperatura para os usuários.

Outra definição é a de [Satyanarayanan, 2001], que diz que o contexto de um usuário de uma aplicação sensível ao contexto consiste de atributos como localização física, estado fisiológico, estado emocional, histórico pessoal, padrões diários de comportamento, entre outros, que se fornecidos para um assistente humano, podem ser usados para tomada de decisões sem necessidade de interromper o usuário a todo momento.

Não é difícil perceber que não se pode enumerar todos os aspectos relevantes em todas as situações, o que invalida as definições específicas de contexto.

Os dados que caracterizam um contexto podem vir do mundo físico como do mundo virtual e algumas vezes se misturam.

As pessoas normalmente não percebem ambientes físicos (escritório, piso de uma loja, estádios) e virtuais (desktop de um computador, funcionalidades de um telefone celular) como partes separadas, já que objetos e processos podem ser representados nos dois universos. Assim, se faz necessário projetar estruturas capazes de representar elementos tanto do mundo real - podem ser elementos físicos ou apenas conceitos -, como do mundo virtual, da maneira mais genérica possível, que possibilite a criação de ambientes que suportam melhor as atividades físicas e virtuais relacionadas.

Na figura seguinte pode-se acompanhar um emprego simples de computação pervasiva reagindo com o contexto. Os sinais de trânsito são colocados em locais de grande fluxo de veículos e o tempo de mudança de permissões é regulado segundo a teoria das filas [Donald e Harris, 1998], baseado no histórico do volume de carros que circulam. Conforme a quantidade em cada sentido, é regulada maior ou menor permanência de uma cor do semáforo. No entanto, nem sempre a previsibilidade funciona. Há situações que podem gerar congestionamentos simplesmente porque a lógica estabelecida não está condizente com a situação. O emprego de sensores, nos carros e nas esquinas, permite que uma rede de computadores possa analisar cada contexto em particular, alterando os tempos de abertura dos sinais de acordo com o

volume em cada uma das vias. Esta alteração simples poderá diminuir os engarrafamentos em horários de grande circulação.

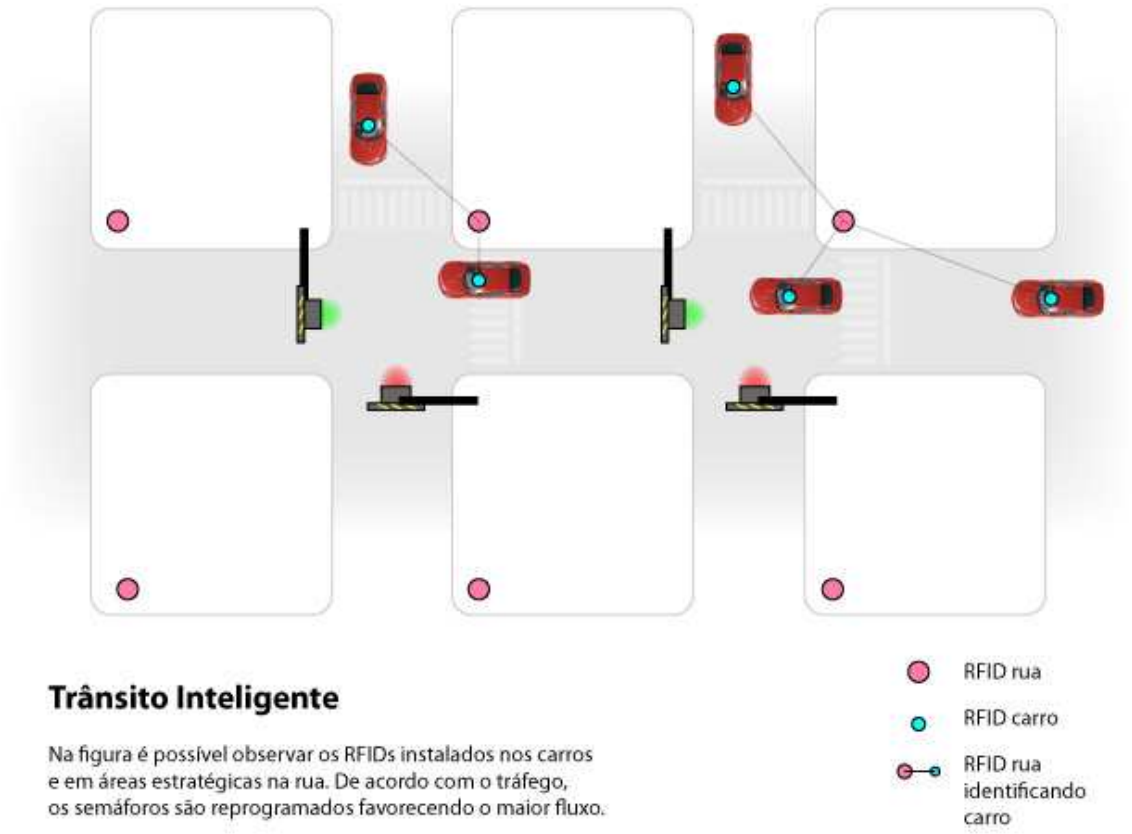

Figura 2 – Trânsito inteligente utilizando RFID
Fonte: http://www.faberludens.com.br/pt-br/taxonomy/term/170?page=3

No presente contexto as entidades a serem tratadas serão os veículos, o semáforo, possíveis pedestres e as vias. Em relação aos veículos, os sensores deles próprios e os posicionados nas esquinas irão definir várias propriedades, tais como: a quantidade de veículos em cada uma das ruas, o sentido, a definição dos que irão dobrar à direita ou vão prosseguir, a velocidade do escoamento, etc. O semáforo informará a cor atual, e terá outras propriedades, como o tempo de cada cor, pré-configuradas na rede de computadores. As vias darão informações de localização e sentido, bem como a existência de faixas de pedestres, estado da pista de rolamento (umidade, buracos, etc). O tratamento dessas variáveis permitirá a alternância inteligente das interrupções e aberturas do sinal em cada sentido.

2.3 FORMALISMOS PARA CAPTURAR INFORMAÇÃO NO CONTEXTO

Tem surgido uma ampla variedade de pesquisas sobre diferentes modelos de contexto, pois um modelo bem desenvolvido é um atributo chave para qualquer sistema

sensível ao contexto. Enquanto modelos mais antigos eram principalmente dirigidos a criarem o contexto no que diz respeito a uma aplicação ou uma classe de aplicação, modelos de contexto genéricos são mais interessantes, já que muitas aplicações podem se beneficiar deles.

Segundo [Thomas Strang, 2003], sistemas pervasivos possuem grandes exigências para a modelagem de contexto, como:

- c**omposição distribuída**: qualquer sistema de computação pervasiva é derivado de um sistema de computação distribuída, que carece de uma instância central responsável pela criação, implantação e manutenção de dados e serviços, nas descrições do contexto particular. Em vez disso, composição e administração de um modelo de contexto e seus dados variam de acordo com a dinâmica, nomeadamente em termos de tempo, topologia de rede e fonte;

- validação parcial: é altamente desejável que se possa validar parcialmente o conhecimento contextual sobre a estrutura, bem como no nível de instância em relação a um modelo de contexto em uso. Isto é feito mesmo quando não há um só lugar ou momento em que o conhecimento contextual esteja disponível em um nó, como resultado da computação distribuída. Isto é particularmente importante devido à complexidade das interrelações contextuais, que tornam qualquer intenção de modelagem propensa a erros;

- riqueza e qualidade da informação: a qualidade de uma informação entregue por um sensor varia de acordo com o tempo, bem como a riqueza de informações fornecidas por diferentes tipos de sensores que caracterizam uma entidade em um ambiente pervasivo de computação podem ser diferentes. Assim, o modelo de contexto adequado para o uso na computação pervasiva deve suportar a indicação de qualidade e riqueza;

- incompletude e ambiguidade: o conjunto de informações contextuais disponíveis em qualquer momento e que caracterizam entidades relevantes em ambientes de computação pervasiva, geralmente é incompleto ou ambíguo, em particular se esta informação é recolhida de uma rede de sensores. Isso deve ser abordado pelo modelo, por exemplo, por interpolação de dados incompletos em nível de instância;

- **níveis de formalidade**: é sempre um desafio descrever fatos contextuais e interrelações de forma precisa e de maneira rastreável. É altamente desejável que cada uma das partes que participa da interação na computação pervasiva compartilhe da mesma interpretação de dados trocados; e

- **aplicabilidade em ambientes existentes**: a partir da perspectiva de implementação, é importante que um modelo de contexto seja aplicável junto com a infra-estrutura de um ambiente de computação pervasiva, por exemplo, um framework de serviços, como um *Web Service*.

Existem na literatura diversas abordagens para modelar o contexto em aplicações pervasivas. A seguir descreve-se um leque de abordagens interessantes de modelagem de contexto.

2.3.1 ISAMadapt

O *ISAMadapt* [Augustin, 2004], é um ambiente de desenvolvimento de aplicações móveis com comportamento adaptativo ao contexto em um ambiente de Computação Pervasiva. No ambiente pervasivo a localização corrente do usuário determina o contexto de execução da aplicação.

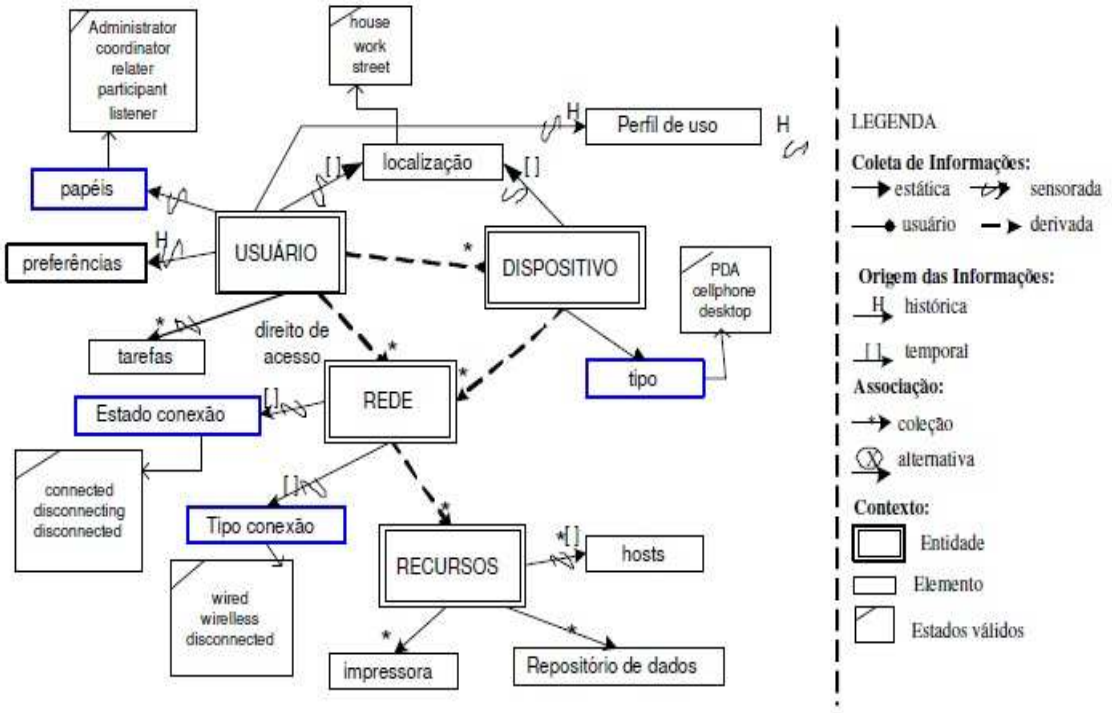

Figura 3: Exemplo de modelagem do contexto da aplicação
Fonte: Augustin, Iara; Yamin, Adenauer; da Silva, Luciano Cavelheiro; Real, Rodrigo; Geyer, Claudio Resin 2004 ISAMadapt – um Ambiente de Desenvolvimento de Aplicações para a Computação Pervasiva

Codificar uma aplicação pervasiva é especificar qual código deve executar no ambiente em que a aplicação está no momento. Assim, tem-se a necessidade de adaptação dinâmica do contexto. No artigo em estudo, os autores projetaram um modelo de adaptação contendo abstrações como: contexto, adaptadores, comandos e políticas de adaptação. A abstração de contexto permite focar em alguns aspectos que são relevantes em uma situação particular. No ISAM, contexto é definido como “toda a informação relevante para a aplicação e que pode ser obtida por esta”. O programador explicitamente identifica as entidades e define seus atributos, os quais integram o contexto da aplicação.

2.3.2 Modelo *Key-Value*

O modelo *Key-Value*: [Schilit et al., 1994] já utilizava pares chave-valor para modelar o contexto, fornecendo o valor de uma informação de contexto - por exemplo, informações de localização -, para uma aplicação com um ambiente variável. A abordagem de modelagem de *Key-Value* é freqüentemente utilizada em estruturas de serviço distribuídas (por exemplo, [Capeus, 2001]). Em tais estruturas, os serviços em si são normalmente descritos como uma lista simples de atributos de uma forma *Key-Value*, e o processo de descoberta de empregados do serviço, por exemplo, SLP, Jini, opera um algoritmo de correspondência exata sobre estes atributos. Em particular, os pares *Key-Value* são fáceis de gerir, mas faltam recursos para um sofisticado estruturamento para permitir algoritmos eficazes de recuperação de contexto.

Segundo [Thomas Strang, 2003], o modelo *Key-Value* é bem simples e é um avanço para a gestão e análise de risco de erros, porém é desvantajoso quando a qualidade da meta-informação ou ambiguidade deve ser considerada.

2.3.3 Modelo de Esquema de Marcação

Os modelos de esquema de marcação (*markup schema*) têm como representantes típicos de modelagem de contexto os *profiles*. Eles geralmente são baseados em derivados da SGML (*Standard Generic Markup Language*), a superclasse de todas as linguagens de marcação, como o XML (*eXtensible Markup Language*).

Segundo [Thomas Strang, 2003], esse modelo é forte para validação parcial, mas é incompleto e ambíguo devendo ser tratado no nível de aplicação.

2.3.4 Modelos Gráficos

Um modelo gráfico muito conhecido de propósito de geral é o UML, que possui um forte componente gráfico e por ter uma estrutura genérica é muito apropriado para a modelagem de contexto.

Outro exemplo de modelagem gráfica é o ORM, no qual a modelagem básica é o *fato* (*fact*); a modelagem de um domínio usando ORM envolve identificar apropriadamente os tipos de fato e as funções que os tipos de entidades desempenham neles.

Segundo [Thomas Strang, 2003], os pontos fortes desse modelo estão no nível de estrutura, que são usados principalmente para descrever um formato de conhecimento contextual de modo a derivar algum código ou algum modelo ER (Entidade Relacionamento), que é valioso no sentido da exigência da aplicabilidade.

2.3.5 Modelos Orientados a Objeto

Nos modelos orientados a objetos os detalhes do processamento do contexto são encapsulados em um nível de objeto, portanto, oculto para outros componentes. O acesso à informação contextual é fornecido através de interfaces específicas.

Segundo [Thomas Strang, 2003] esse modelo é interessante no que diz respeito à exigência de composição distribuída. Novos tipos de informação contextual (classes), bem como novas ou instâncias atualizadas (objetos) podem ser tratados no sistema de forma distribuída.

2.3.6 Modelos Baseados em Lógica

Nos modelos baseados em lógica, a lógica define as condições em que uma expressão de conclusão ou fato pode ser derivada - um processo conhecido como raciocínio ou inferência - de um conjunto de outras expressões ou fatos. Para descrever essas condições em um conjunto de regras um sistema formal é aplicado. Em um modelo de contexto baseado em lógica, o contexto, consequentemente, é definido como fatos, expressões e regras. Normalmente a informação contextual é adicionada, atualizada e excluída de um sistema baseado em lógica a partir dos fatos, ou inferidos a

partir das regras do sistema, respectivamente. O alto grau de formalidade é comum a todos os modelos baseados em lógica.

Nesse modelo existem algumas abordagens distintas, como a *Formalização de Contexto* proposta por [McCarthy, 1993], onde o contexto é introduzido como entidade matemática abstrata com propriedades úteis para a inteligência artificial. Ele impediu que fosse dada uma definição do que o contexto é, para, ao invés disso, dar uma receita que permite a formalização de simples axiomas para os fenômenos de senso comum.

Já o foco principal da abordagem de [Giunchiglia, 2001, 1993], é muitas vezes conhecido como *Sistemas de Multiconteúdo*, que é menos modelado no contexto do que no raciocínio de contexto. Ele pega o contexto para ser um subconjunto específico do estado completo de uma entidade individual, usado para raciocinar sobre determinado objetivo e que é visto como uma teoria (parcial) do mundo, que identifica uma perspectiva individual subjetiva sobre ele.

Outra abordagem é a *Teoria Estendida da Situação* por [Akman e Surav, 1997], que é baseada na *Teoria da Situação* de [Barwise e Perry, 1983]. Barwise e Perry tentaram cobrir uma semântica modelo-teórica de uma linguagem natural em um sistema de lógica formal. Akman e Surav usaram e ampliaram esse sistema para modelar o contexto com tipos de situações representando situações comuns e objetos.

Segundo [Thomas Strang, 2003], modelos baseados em lógica possuem validação parcial e são difíceis de manter. Seu nível de formalidade é extremamente alto, mas sem validação parcial a especificação do conhecimento contextual dentro desse modelo é muito suscetível a erros.

2.3.7 Modelos Baseados em Ontologia

Os modelos baseados em ontologia são especialmente adequados em partes do projeto para descrição de informações e são usados em nosso cotidiano em estruturas de dados utilizadas em computadores.

Uma das primeiras abordagens propostas foi a de [Otzturk e Aamodt, 1997], na qual foi proposto um modelo de contexto baseado em ontologias devido às suas vantagens no campo da normalização e formalidade, depois de um estudo psicológico sobre a diferença entre recordação e reconhecimento.

A abordagem do sistema CoBrA proporciona um conjunto de conceitos ontológicos para caracterizar entidades como pessoas, lugares e vários tipos de outros objetos com seus contextos. Este sistema usa uma arquitetura *broker-centric* para fornecer suporte de execução para sistemas sensíveis ao contexto.

A arquitetura CoBrA prevê compartilhamento de conhecimento, raciocínio de contexto, e suporte para a proteção de privacidade em sistemas pervasivos conscientes do contexto.

Os modelos ontológicos são considerados frameworks promissores para especificar conceitos e suas interrelações, podendo fazer o uso de compartilhamento de conhecimento, lógica de inferência e reuso de capacidades de conhecimento [Richard Dapoing, Patrick Barlatier, 2007].

Segundo [Thomas Strang, 2003], essas abordagens são fortes a respeito da exigência de composição distribuída. Validação parcial é possível, e existe um amplo conjunto de ferramentas de validação.

2.4 SISTEMAS DE TIPOS

Sistema de tipos é o ramo da matemática e da lógica que se preocupa com a classificação de entidades em conjuntos chamados tipos.

Com o surgimento de poderosos computadores programáveis, e o desenvolvimento de linguagens de programação para os mesmos, a Teoria dos Tipos [Alonzo Church, 1940] tem encontrado aplicação prática no desenvolvimento de sistemas de tipos para estas linguagens. Definições de “sistemas de tipos” no contexto de linguagens de programação variam, mas a seguinte definição dada por [Benjamin C. Pierce, 2002] corresponde, aproximadamente, ao consenso corrente na comunidade de Teoria dos Tipos:

> Um sistema de tipos é um método sintático tratável para provar a isenção de certos comportamentos em um programa através da classificação de frases de acordo com as espécies de valores que elas computam. (*Types and Programming Languages, MIT Press,* 2002). Em outras palavras, um sistema de tipos divide os valores de um programa em conjuntos chamados tipos (isso é o que é denominado uma “atribuição de tipos”), e torna certos comportamentos do programa ilegais com base nos tipos que são atribuídos neste processo. Por exemplo, um sistema de tipos pode classificar o valor “*hello*” como uma cadeia de caracteres e o valor 5 como um número, e proibir o programador de tentar adicionar “*hello*” a 5, com base nessa atribuição de tipos.

Existem duas formas de “tipagem”: a estática e a dinâmica. A tipagem estática verifica em tempo de compilação o tipo de um dado, enquanto a tipagem dinâmica verifica em tempo de execução.

Os dados também podem ser fortemente tipados ou fracamente tipados. Nos dados fortemente tipados todas as variáveis têm um tipo específico que deve ser explicitado pelo programador, enquanto na tipagem fraca a conversão não se faz necessária, sendo realizada implicitamente pelo compilador ou pelo interpretador. Uma linguagem fortemente tipada é mais consistente, pois fornece uma segurança maior para o programador.

Na computação pervasiva um sistema de tipos deve ser muito bem estruturado, em face da grande quantidade e da diversidade de dados que irão ser operados a todo momento. Cada entidade componente do contexto, das quatro já mencionadas, produzirá tipos de dados diferentes ao mesmo tempo, o que exigirá receptores eficientes para o tratamento correto das informações.

2.4.1 Erros de Execução

Sistema de Tipos tem como propósito fundamental prevenir a ocorrência de erros durante a execução de um programa, e para isso se torna necessário definir o que seria um erro de execução.

Um erro de execução pode ser percebido quando uma falha inesperada no software ocorre, podendo ser causada por uma instrução ilegal ou por uma referência ilegal à memória, por exemplo. Porém, existem tipos mais sutis de erros de execução que podem causar dados corrompidos, sem serem imediatamente percebidos, como uma divisão por zero, que normalmente não é prevenida por um Sistema de Tipos.

Os erros de execução levantam a questão da segurança e do comportamento dos programas, por isso eles são diferenciados em dois tipos: os que fazem o computador parar imediatamente, e aqueles que passam despercebidos até que um comportamento arbitrário ocorra posteriormente. O primeiro é chamado de *trapped errors*, e o segundo é chamado de *untrapped errors* [Luca Cardelli, 2007].

Um exemplo de *untrapped error* é haver um salto, por um período arbitrário de tempo, para um endereço errado de memória; a memória pode ou não representar um fluxo de instruções.

Exemplo de um *untrapped error* em Java:

```
private void cmdUntrapped_Click(object sender,
System.EventArgs e)
{
     short i = 1234;
     short j = 0;
     short k = -1;
     k = Convert.ToInt16(i / j);
     // Note que você nunca verá a declaração MsgBox
abaixo
     MessageBox.Show("Your results are: " +
k.ToString(), this.Text, MessageBoxButtons.OK,
MessageBoxIcon.Information);
}
```

Fonte: http://www.dotnetspider.com/resources/24381-Untrapped-Error.aspx

Já um exemplo de *trapped error* pode ser uma simples divisão por zero, ou acessar um endereço ilegal da memória, onde o computador iria parar imediatamente.

Um programa é considerado seguro se ele não causa *untrapped errors*. Linguagens em que todos os fragmentos são seguros, são consideradas linguagens seguras.

2.4.2 Propriedades Esperadas de um Sistema de Tipos

Algumas propriedades caracterizam um Sistema de Tipos e podem determinar seu grau de eficiência na previsão e tratamento antecipado de possíveis condições de erro que poderiam ocorrer em um aplicativo. Elas estão listadas a seguir:

- **transparência:** um programador deverá ser capaz de prever se o programa terá checagem de tipos. Se ele falha na checagem de tipos, o motivo da falha deve ser auto-evidente;

- **executabilidade:** declarações de tipo devem ser verificadas estaticamente, tanto quanto possível, e verificadas dinamicamente. A coerência entre as declarações de tipo e seus respectivos programas deve ser rotineiramente verificadas; e

- **verificabilidade:** deve haver um algoritmo (chamado algoritmo de *typecheking*) que possa garantir que um programa é *bem comportado*. A finalidade de um sistema de tipos não é simplesmente aplicar as **intenções** do programador, mas capturar ativamente os erros de execução antes que eles ocorram.

2.4.3 Sistemas de Tipos Especializados

Existem os sistemas de tipos especializados, que são usados em determinados ambientes para certos tipos de dados em análise de programas estáticos. Frequentemente são baseados em ideias da teoria de tipo formal e só estão disponíveis como parte de um protótipo de sistemas de pesquisas [Wikipédia, 2010]. Alguns exemplos de Sistemas de Tipos especializados estão listados abaixo:

- Tipos Dependentes: são baseados na ideia da utilização de escalares ou valores para descrever com mais precisão o tipo de outro valor. Por exemplo, "matriz (3,3)" pode ser o tipo de uma matriz 3 × 3. Podemos, então, definir regras de tipagem, tais como a seguinte regra para multiplicação de matrizes:

$$\text{matrix\_multiply} : \text{matrix}(k,m) \times \text{matrix}(m,n) \rightarrow \text{matrix}(k,n)$$

onde k, m, n são valores inteiros positivos arbitrários. Uma variante de ML[1], chamado ML Dependente, foi criada com base neste tipo de sistema, mas como a verificação de tipos convencional de tipos dependentes é indecidível, nem todos os programas que as utilizam podem ser de tipo verificado (*type-checked*) sem qualquer espécie de limites. ML Dependente limita o tipo de igualdade que pode decidir a Aritmética Presburger[2]. Outras linguagens, como Epigram [Conor McBride e James McKinna, 2004], tornam o valor de todas as expressões da linguagem decidível, para que a verificação de tipo possa ser decidível. Também é possível fazer a Linguagem Turing completa, através de verificação de tipo indecidível, como em Cayenne [Lambert M. Surhone, Mariam T. Tennoe e Susan F. Henssonow, 2010].

- Tipos Lineares: baseiam-se na teoria da lógica linear e estão estreitamente relacionados aos tipos de exclusividade. São tipos de valores destinados a ter como propriedade uma e apenas uma referência a eles em todos os momentos. Estes são valiosos para descrever grandes valores imutáveis, como strings, imagens, e outros, porque qualquer operação que, simultaneamente, destrói um objeto linear e cria um objeto similar (tal como 'str = str + "a") pode ser otimizada em uma alteração local. Normalmente isso não é possível porque essas alterações podem causar efeitos

[1] Linguagem Funcional de proposta geral desenvolvida por Robin Milner e outros no final dos anos 1970

[2] **Aritmética Presburger** é a teoria de primeira ordem dos números naturais com a adição , nomeada em honra de Mojżesz Presburger , que a introduziu em 1929. A assinatura da aritmética Presburger contém apenas a operação de soma e igualdade, omitindo a operação de multiplicação inteiramente.

colaterais em algumas partes do programa com outras referências para o objeto, violando a transparência referencial. Eles também são utilizados no protótipo de sistema operacional *Singularity* para comunicação entre processos, assegurando estaticamente que os processos não podem compartilhar objetos da memória compartilhada, a fim de evitar problemas em condições de corrida. A linguagem limpa - uma linguagem como Haskell, por exemplo -, usa este tipo de sistema, a fim de ganhar rapidez, enquanto o mantém seguro.

- Tipos de Intersecção: são tipos que descrevem os valores que pertencem a dois outros tipos de dados com sobreposição de conjuntos de valores. Por exemplo, na maioria das implementações de C o *signed char* tem a escala de -128 a 127 e o *unsigned char* tem escala de 0 a 255, então o tipo de intersecção entre esses dois tipos têm escala de 0 a 127. Neste caso, um tipo de interseção poderia seguramente ser passado em funções esperando tanto *signed* ou *unsigned chars*, pois ele é compatível com ambos os tipos.

Tipos de intersecção são úteis para descrever tipos de função sobrecarregada: Por exemplo, se "*int → int*" é o tipo de uma função, tendo um argumento inteiro e retornando um inteiro, e "*float → float*" é o tipo de uma função, tendo um argumento *float* e retornando *float*, então a interseção desses dois tipos pode ser usada para descrever as funções que fazem um ou outro, com base no tipo de entrada que é dada. Essa função pode ser passada para outra função que espera um "*int → int*" sendo garantido que ela funcione de forma segura; ela simplesmente não iria usar a funcionalidade "*float → float*".

Em uma hierarquia de subclasses, o cruzamento de um tipo e um tipo ancestral (como seu pai) é o tipo mais derivado (subclasse inferior). A interseção de tipos irmãos é vazia.

- Tipos de União: tipos de união são tipos que englobam valores que pertencem aos dois tipos da expressão. Por exemplo, em C, o *signed char* tem a escala de -128 a 127, e o *unsigned char* tem escala de 0 a 255, então a união destes dois tipos teria faixa de -128 a 255. Qualquer função para lidar com este tipo de união teria de lidar com números inteiros nesta faixa completa. Em termos mais gerais, as únicas operações para um tipo de união são as operações que são válidas em ambos os tipos que estão sendo unidos. Uniões em C possuem um conceito semelhante a tipos de união, mas não é *typesafe* porque permite operações que são válidas em qualquer tipo, ao invés de dois.

Tipos de união são importantes na análise do programa, no qual são usados para representar valores simbólicos cuja natureza exata (valor ou tipo, por exemplo) não é conhecida.

Em uma hierarquia de subclasses, a união de um tipo e um tipo de ancestral (como seu pai) é o tipo ancestral. A união dos tipos irmãos é um subtipo de seu ancestral comum, ou seja, todas as operações permitidas em seu ancestral comum são permitidas no tipo de união, mas eles também podem ter outras operações válidas em comum.

2.4.4 Linguagens Tipadas e Não-tipadas

Também existem linguagens que não empregam um sistema de tipos, mas, no entanto, as falhas passam despercebidas. Por isso é importante verificar o que seria uma linguagem tipada e uma linguagem não-tipada.

As linguagens de programação tipadas são linguagens nas quais podem ser atribuídos tipos às variáveis, como por exemplo: uma variável x pode ser do tipo *float* (ponto flutuante), quando é suposto que ela possa assumir apenas valores *float* durante a execução do programa. As linguagens tipadas podem ser fortemente ou fracamente tipadas. Nas linguagens fortemente tipadas, como o Java, a declaração de tipo é obrigatória. Elas testam se os valores que se quer atribuir a uma variável são exatamente do mesmo tipo da variável. Por exemplo, ao se atribuir a uma variável do tipo *float* “n” o valor “2”, seria exigido “n = 2.0”, ao invés de “n = 2”. Já as linguagens fracamente tipadas são aquelas em que a declaração de tipo é opcional, as variáveis podem ser interpretadas de forma diferente, dependendo do contexto. Em C, por exemplo, é possível somar duas variáveis *chars*, com letras, e atribuir o resultado a uma variável *float*.

Já as linguagens não-tipadas contêm um único tipo universal que pode conter todos os valores. Nessas linguagens, operações podem ser aplicadas para argumentos inapropriados: o resultado pode ser um valor ajustado arbitrariamente, uma exceção, uma falha, ou um efeito inesperado. O cálculo lambda puro é um caso extremo de linguagem não tipada onde falhas nunca ocorrem: a operação única é a aplicação de uma função e, uma vez que todos os valores são funções, a operação nunca falha. A linguagem Perl, por exemplo é não-tipada. O nome de uma variável escalar começa com “$”, como “$x” ou “$nome”. Uma variável escalar Perl pode guardar qualquer tipo de

dado: uma string, um número inteiro, um número de ponto flutuante, um valor lógico, etc.

Linguagens não-tipadas podem aplicar segurança através da realização de verificações em tempo de execução (verificação dinâmica). Linguagens tipadas podem aplicar segurança em tempo de compilação, rejeitando todos os programas que são potencialmente perigosos (verificação estática). Linguagens tipadas também podem usar uma mistura de verificação em tempo de execução e tempo de compilação (ou interpretação).

Sistema de Tipos é o componente de uma linguagem tipada que acompanha os tipos de variáveis e, em geral, os tipos de todas as expressões do programa. Sistemas de Tipos são usados para determinar quando os programas são bem comportados. Apenas programas que fazem uso de um sistema de tipos podem ser considerados verdadeiros programas de linguagens tipadas; os outros programas devem ser descartados mesmos antes de serem executados.

Uma linguagem é tipada quando existe um sistema de tipos para ela, sendo que os tipos podem ou não aparecer na sintaxe do programa. Linguagens tipadas são explicitamente tipadas se os tipos são parte da sintaxe, ou implicitamente tipadas quando não o são. Nenhuma das linguagens mais populares são puramente implicitamente tipadas, embora linguagens como o Haskell[3] suportem escrever grandes fragmentos de programas onde a informação de tipo é omitida; os sistemas de tipos dessas linguagens ajustam automaticamente os tipos nesses fragmentos de programas.

2.4.5 Formalização de um Sistema de Tipos

Como já foi discutido, sistemas de tipo são usados para definir a noção de *tipar bem*, que é em si uma aproximação estática de bom comportamento (incluindo a segurança). Segurança facilita a depuração por causa de falhas que travam o computador, e permite a coleta de lixo, protegendo as estruturas de tempo de execução. Uma boa tipagem facilita o desenvolvimento do programa ainda mais por interceptar os erros de execução antes do tempo de execução.

[3] Linguagem de programação puramente funcional, de propósito geral, nomeada em homenagem ao lógico Haskell Curry [Wikipédia, 2010]

Para garantir que os programas bem tipados são também bem comportados, um sistema de tipos formal é expresso pelas caracterizações matemáticas de um sistema de tipos informal que são descritas em manuais de linguagens de programação. Quando um sistema de tipos é formalizado pode-se tentar provar o teorema da *solidez de tipo*, afirmando que os programas bem tipados são bem comportados. Se o teorema é verificado, pode-se dizer que o tipo é sólido [Luca Cardelli, 2007]. O tipo é considerado sólido quando está completamente determinado, de acordo com as regras do sistema de tipo.

Um sistema de tipos pode ser formalizado através de algumas regras, como a descrição de sua sintaxe. Para a maioria das linguagens populares isto se reduz a descrever a sintaxe de tipos e termos. Tipos expressam conhecimento estático sobre programas, nos quais os termos (declarações, expressões, ou outros fragmentos do programa) expressam o comportamento algorítmico.

Também é importante definir as regras de escopo da linguagem, que inequivocamente associam ocorrências de identificadores para os seus locais de ligação (os locais onde os identificadores são declarados). O escopo necessário para linguagens tipadas é invariavelmente estático, no sentido de que locais de ligação dos identificadores precisam ser determinados antes do tempo de execução. Locais de ligação muitas vezes podem ser determinados unicamente através da sintaxe de uma linguagem, sem qualquer análise mais aprofundada; escopo estático é então chamado de escopo léxico. A falta de escopo estático é chamada de escopo dinâmico.

Escopos podem ser formalmente especificados pela definição do conjunto de variáveis livres de um fragmento de programa (que envolve a especificação de como as variáveis são limitadas por declarações). A noção associada de substituição de tipos ou modalidades de variáveis livres pode então ser definida.

Após esses entendimentos, pode-se proceder à definição das regras tipo da linguagem. Elas descrevem uma relação *tem-tipo* (*has-type)* da forma M:A. Um entre os termos de M e do tipo A. Algumas linguagens também exigem uma relação de *subtipo-de* da forma A <:B entre os tipos, e muitas vezes uma relação *tipo-igual* (*equal-type)* da forma A = B do tipo de equivalência. O conjunto de regras de tipo de uma linguagem forma seu sistema de tipos.

As regras de tipo não podem ser formalizadas sem que antes seja introduzido outro ingrediente fundamental que não se reflete na sintaxe da linguagem: ambientes de tipagem estática. Estes são usados para gravar os tipos de variáveis livres durante o processamento de fragmentos de programas, pois eles correspondem rigorosamente à tabela de símbolos de um compilador durante a fase de *checagem de tipos (typechecking)*. As regras de tipo são sempre formuladas em relação a um ambiente estático para o fragmento ter seu tipo checado *(typechecked)*. Por exemplo, a relação *tem-tipo* M:A é associada a um ambiente de tipagem estática Γ que contém informações sobre as variáveis livres de M e A. A relação está escrita na sua totalidade como Γ ⊢ M:A o que significa que M tem o tipo A, em um ambiente Γ.

O passo final na formalização de uma linguagem é definir a sua semântica como uma relação *tem-valor (has-value)* entre os termos e uma coleção de resultados. A forma dessa relação depende muito do estilo de semântica que é adotada. Em qualquer caso, a semântica e o sistema de tipos de uma linguagem estão interligados: os tipos de um termo e de seu resultado deve ser o mesmo (ou adequadamente relacionados), o que é a essência do teorema de solidez.

As noções fundamentais do sistema de tipos são aplicáveis a praticamente todos os paradigmas computacionais (funcional, imperativo, concorrentes, etc.). Regras de tipo individuais podem freqüentemente ser adotadas de forma inalteradas por diferentes paradigmas. Por exemplo, as regras de tipo básicas para funções são as mesmas se a semântica é *call-by-name* ou *call-by-value* ou, ortogonalmente, funcional ou imperativo.

2.4.6 A linguagem de um Sistema de Tipos

Um sistema de tipos especifica as regras de tipo de uma linguagem de programação independente do algoritmo de *checagem de tipos* utilizado. Isto é análogo a descrever a sintaxe de uma linguagem de programação por uma gramática formal, independentemente de algoritmos de análise particular.

É conveniente e útil separar os sistemas de tipos de algoritmos de *checagem de tipos*: sistemas de tipos pertencem a definições de linguagem, enquanto algoritmos pertencem aos compiladores. É mais fácil explicar os aspectos de tipagem de uma linguagem por um sistema de tipos do que de um algoritmo usado por um compilador dado. Além disso, os compiladores diferentes podem usar diferentes algoritmos de *checagem de tipos* para o mesmo sistema de tipos.

- Julgamentos: sistemas de tipos são descritos por um formalismo especial. A descrição de um sistema de tipo começa com a descrição de uma coleção de declarações formais chamadas julgamentos (*judgments*). Um julgamento tem a forma:

$\Gamma \vdash \mathfrak{I}$ $\mathfrak{I}$ é uma afirmação; a variável livre de $\mathfrak{I}$ é declarada dentro de Γ

Dizemos que Γ *implica* $\mathfrak{I}$. Aqui Γ é um *ambiente de tipagem estática*, por exemplo, uma lista ordenada de variáveis distintas e seus tipos, da forma ϕ, x_1: A_1, ...A_n, x_n.

O ambiente vazio é denotado por ϕ, e o conjunto de variáveis x_1... x_n declarado em Γ é indicado por *dom*(Γ), o domínio de Γ. A forma da asserção $\mathfrak{I}$ varia de julgamento para julgamento, mas todas as variáveis livres de $\mathfrak{I}$ devem ser declaradas em Γ.

A decisão mais importante, para os propósitos atuais, é o julgamento de tipagem, que afirma que um termo M tem um tipo A com relação a um ambiente de tipagem estática para as variáveis livres de M. Ele tem a forma:

$\Gamma \vdash M : A$ M tem o tipo A em Γ

Exemplos:

$\phi \vdash true : Bool$ *true* tem o tipo *Bool*
$\phi, x{:}Nat \vdash x+1 : Nat$ $x+1$ tem o tipo *Nat*, contanto que *x* tenha o tipo *Nat*

Outras formas de julgamento são muitas vezes necessárias; uma comum afirma simplesmente que um ambiente está **bem formado**:

$\Gamma \vdash \Diamond$ Γ é bem formado (isto é, foi construído corretamente)

Qualquer julgamento pode ser considerado *válido* (por exemplo, $\Gamma \vdash true: Bool$) ou *inválido* (por exemplo, $\Gamma \vdash true: Nat$). Validade formaliza a noção de programas bem tipados. A distinção entre julgamentos válidos e inválidos poderia ser expressa em uma série de maneiras, no entanto, surgiu uma forma altamente estilizada de apresentar o conjunto de julgamentos válidos. Esse estilo de apresentação, com base em regras de tipo, facilita, afirmando e provando teoremas e lemas técnicos sobre os sistemas de tipos. Além disso, as regras de tipos são altamente modulares: regras para as diferentes construções podem ser escritas separadamente (em contraste com um algoritmo de

checador de tipos monolítico). Portanto, as regras de tipo são relativamente fáceis de ler e entender.

- Regras de Tipo: regras de tipo afirmam a validade de certos julgamentos com base em outros que já são conhecidas por serem válidos. O processo se inicia por algum julgamento intrinsecamente válido (normalmente: $\phi \vdash \Diamond$, indicando que o ambiente vazio é bem formado).

$$\frac{\textit{(Nome da Regra)} \quad \textit{(Anotações)} \qquad \Gamma_1 \vdash \Im \quad \ldots \quad \Gamma_n \vdash \Im_n \ \textit{(Anotações)}}{\Gamma_1 \vdash \Im}$$

Cada regra de tipo é escrita como uma série de julgamentos de premissa $\Gamma_i \vdash \Im_i$ acima de uma linha horizontal, com um único julgamento de *conclusão* $\Gamma \vdash \Im$ abaixo da linha. Quando todas as premissas são satisfeitas, a conclusão deve esperar; o número de premissas pode ser zero. Cada regra tem um nome. (Por convenção, a primeira palavra do nome é determinada pelo julgamento de conclusão; por exemplo, nomes de regras da forma "(Val ...)" são para as regras cuja conclusão é um julgamento de tipagem do valor). Quando necessário, condições restritivas para a aplicabilidade de uma regra, assim como abreviaturas utilizadas na regra, são anotadas ao lado do nome da regra ou das premissas.

Por exemplo, a primeira das duas seguintes regras afirma que todo o numeral é uma expressão do tipo *Nat*, em algum ambiente bem formado Γ. A segunda regra afirma que duas expressões M e N denotando números naturais podem ser combinadas em uma expressão maior M+N, que igualmente denote um número natural. Além disso, o ambiente Γ para M e N, que declara o tipo de toda variável livre de M e de N, é transportado para o M + N.

$$\frac{\Gamma \vdash \Diamond}{\Gamma \vdash n : Nat}\ (\text{Val } n)\ (n = 0,1,\ldots) \qquad \frac{\Gamma \vdash M : Nat \quad \Gamma \vdash N : Nat}{\Gamma \vdash M+N : Nat}\ (\text{Val } +)$$

Uma regra fundamental diz que o ambiente vazio é bem formado, sem pressupostos:

$$\frac{}{\phi \vdash \Diamond}\ (\text{Env } \phi)$$

Uma coleção de regras de tipos é chamada de um sistema de tipos (formal). Tecnicamente, os sistemas de tipos se encaixam no framework de *sistema de provas*

formal: coleções de regras usadas para realizar deduções passo-a-passo. As deduções realizadas nos sistemas de tipos se referem aos tipos dos programas.

- Derivações de Tipo: uma derivação em um sistema de tipos determinado (*given type system*) é uma árvore de julgamentos com as folhas na parte superior e uma raiz na parte inferior, no qual cada julgamento é obtido por aqueles imediatamente acima dele em função de uma regra do sistema. Um requisito fundamental em sistemas de tipo é que deve ser possível verificar se uma derivação está ou não construída adequadamente. Um **julgamento válido** é aquele que pode ser obtido como a raiz de uma derivação em um sistema de tipos determinado. Ou seja, um julgamento válido é aquele que pode ser obtido mediante a aplicação das regras de tipos. Por exemplo, usando as três regras dadas anteriormente, pode-se construir a derivação a seguir, que estabelece que $\phi \vdash 1+2$: *Nat* é uma sentença válida. A regra aplicada em cada etapa é exibida à direita de cada conclusão:

$$\dfrac{\dfrac{\dfrac{}{\phi \vdash \Diamond}\ \text{by (Env } \phi\text{)}}{\phi \vdash 1 : \mathit{Nat}}\ \text{by (Val } n\text{)} \qquad \dfrac{\dfrac{}{\phi \vdash \Diamond}\ \text{by (Env } \phi\text{)}}{\phi \vdash 2 : \mathit{Nat}}\ \text{by (Val } n\text{)}}{\phi \vdash 1+2 : \mathit{Nat}}\ \text{by (Val +)}$$

- Boa tipagem e inferência de tipo: em um sistema de tipos determinado, um termo M é bem tipado para um ambiente Γ, se existe um tipo A, tal que a $\Gamma \vdash M:A$ é um julgamento válido, ou seja, se o termo M pode receber algum tipo.

A descoberta de uma derivação (e, portanto, de um tipo) por um termo é chamada de problema de **inferência de tipo**. Em um sistema de tipos simples consistindo de regras (ENV ϕ), (Val *n*), e (Val +), um tipo pode ser *inferido* para o termo 1+2 no ambiente vazio. Este tipo é *Nat*, pela derivação anterior.

Suponha-se que agora se queira adicionar uma regra de tipos com a premissa $\Gamma \vdash \Diamond$ e conclusão $\Gamma \vdash$ *true*: *Bool*. No sistema de tipos resultante não se pode inferir qualquer tipo para o termo 1+*true*, porque não existe uma regra para somar um número natural com um booleano. Devido à ausência de derivações para 1+*true*, pode-se dizer que 1+*true* é *não tipável*, ou que está mal tipado, ou que tem um **erro de tipagem**.

É possível acrescentar ainda mais uma regra de tipo, com a premissa $\Gamma \vdash M$: *Nat* e $\Gamma \vdash N$: *Bool*, e com a conclusão $\Gamma \vdash M + N$: Nat (por exemplo, com a intenção de interpretar *true* como 1). Nesses sistemas de tipo, um tipo pode ser inferido para o termo 1+*true*, que agora seria bem tipado.

Assim, o problema de inferência de tipo para um determinado termo é muito sensível ao sistema de tipos em questão. Um algoritmo para inferência de tipos pode ser muito fácil, muito difícil, ou impossível de encontrar, dependendo do sistema de tipos. Se encontrado, o melhor algoritmo pode ser muito eficiente, ou demasiadamente lento. Embora os sistemas de tipos sejam expressos e muitas vezes desenvolvidos de forma abstrata, a sua utilidade prática depende da disponibilidade de bons algoritmos de inferência de tipo.

- Solidez de Tipo: quando se aprofunda em regras de tipo, deve-se ter em mente que um sistema de tipos sensível é mais do que apenas uma coleção arbitrária de regras. Boa tipagem é destinada a corresponder a uma noção semântica de bom comportamento do programa. É costume se verificar a consistência interna de um sistema de tipos provando um teorema de solidez de tipos. Este é o lugar onde os sistemas de tipos encontram a semântica. Para semântica denotacional, espera-se que se $\phi \vdash M:A$ é válido, então $[[M]] \epsilon [[A]]$ é mantido (o valor de M pertence ao conjunto de valores indicados pelo tipo A) e, para a semântica operacional, espera-se que se $\phi \vdash M:A$ e M reduz-se para M', então, $\phi \vdash M':A$. Em ambos os casos o teorema de solidez de tipo afirma que programas bem tipados irão operar sem erros de execução.

2.5 TIPOS DE REGISTRO DEPENDENTES (DRT)

Tipos de registros dependentes (DRTs) são os tipos de registros nos quais o tipo de um campo depende do valor de um anterior.

Por exemplo, se Nat e Vect(n) são os tipos de números naturais e vetores de tamanho n, respectivamente, {n: Nat, v: Vect(n)} é o tipo de registro dependente com objetos (chamadas registros) como { n = 2, v = [5, 6]}, onde a dependência é respeitada: o vetor [5, 6] deve ser do tipo Vect(2)[Yangyue Feng, 2010].

Como regra, as letras maiúsculas denotam tipos de contexto e as letras minúsculas denotam os símbolos de contexto:

Definição: *Um tipo de registro dependente é uma seqüência de campos em que rótulos l_i correspondem a certos tipos de T_i, ou seja, cada campo sucessivo pode depender dos valores dos campos anteriores:*

$$C = \begin{cases} l_1 : T_1 \\ l_2 : T_2(l_1) \\ \ldots \\ l_n : T_n(l_1 \ldots l_{n-1}) \end{cases}$$

onde o tipo de T_i pode depender dos rótulos anteriores l_1, ..., l_{i-1}.

Uma definição semelhante vale para os símbolos de registro nos quais uma seqüência de valores é tal que o valor v_i pode depender dos valores dos campos anteriores l_1, ..., l_{i-1}:

$$c = \begin{cases} l_1 = v_1 \\ l_2 = v_2 \\ \ldots \\ l_n = v_n \end{cases}$$

Pode-se também fazer uso da notação tabular para representar registros. A sequência vazia <> é um tipo de registro dependente e o tipo T_i é uma família de tipos sobre o tipo de registro l_1: T_1, ..., l_{i-1}: T_{i-1}. Supondo-se que Γ é um contexto válido, podemos expressar as regras de formação de tipos de registros fornecidos contanto que l não esteja já declarado em R:

$$\frac{}{\Gamma \vdash <>: \text{record} - \text{type}}$$

$$\frac{\Gamma \vdash R : record - type \quad \Gamma \vdash T : record - type \rightarrow type}{\Gamma \vdash < R, l : T >: record - type}$$

Assume-se também que as regras gerais de construção para os tipos *Set* e *Prop* são válidas e que as construções sintáticas primitivas (isto é, a igualdade, aplicação funcional e abstração lambda) funcionam.

2.6 TIPOS DE REGISTRO DE CONTEXTO (CRT)

A teoria de contexto baseia-se nas obras de [J. Barwise, 1981], que avalia que, em contraste com um “mundo” a determinação do valor de cada proposição e a situação deve refletir a parte limitada da realidade que se percebe e sobre a qual se raciocina. Temas básicos da situação semântica de Barwise foram incorporados em vários frameworks e pode-se também introduzir a sua avaliação através de DRT’s e símbolos. No framework de processos físicos, o Tipo de Registro de Contexto é descrito como um registro no qual os campos detalham conhecimentos físicos (isto é, objetos, suas propriedades e suas restrições). Sua capacidade de fornecer uma estrutura simples que pode ser reutilizada para especificar diferentes tipos de objetos semanticamente estruturados é um elemento chave do modelo. Outro aspecto importante do CRT é que a subtipificação é permitida. Por exemplo, em um contexto simbólico com campos

adicionais não mencionados, o CRT pode variar a partir do contexto de uma única variável envolvida em uma equação física para o contexto de uma situação da vida real.

Em sistemas sensíveis ao contexto, os cenários muitas vezes consideram a informação espacial e temporal. A informação espacial, tal como localização, pode variar de localização geográfica para informações sobre a localização de outros objetos que estão relacionados à localização atual do usuário. Dentro de um edifício, por exemplo, sensores proporcionam informações precisas sobre o local enquanto que, a parte externa do edifício não exige tanta precisão e um serviço de localização por satélite pode ser usado para rastrear a localização do usuário.

A informação temporal, como a hora do dia e a data, são usados principalmente no desencadeamento de ações. Fazendo o componente temporal do contexto explícito permite-se desambiguar alguns eventos dependentes do contexto e descobrir conexões entre eles. Obviamente, a atribuição de valores de tempo para cumprir restrições de tempo real não são muito difíceis - ou seja, o tempo total consumido tanto para a aquisição de um valor de tempo e do processo de verificação de tipo deve ser menor do que o tempo de evolução do sistema. No exemplo a seguir estão incluídas tanto as informações espaciais como as temporais:

$$\begin{pmatrix} x : Vehicle \\ y : RegistrationNumber \\ l_1 : GPSLongitude \\ l_2 : GPSLatitude \\ te : evTime \\ tm : maxTime \\ q_1 : has\_identification(x, y) \\ q_2 : has\_Longitude(x, l_1) \\ q_3 : has\_Latitude(x, l_2) \\ c_1 : has\_lowerV\ alueThan(evTime,maxTime) \end{pmatrix} \quad \begin{pmatrix} x = truck \\ y = 2678KX69 \\ l_1 = 12.0987 \\ l_2 = 67.2365 \\ te = 2/11/06.11 : 33 \\ tm = 2/11/06.12 : 00 \\ q_1 = p_1 \\ q_2 = p_2 \\ q_3 = p_3 \\ c_1 = ct_1 \end{pmatrix}$$

CRT são objetos de primeira classe em que os julgamentos de tipagem correspondentes esperam: *p1* é uma prova de *has_identification* (caminhão, 2678KX69), *p2* é uma prova de *has_Longitude* (caminhão, 12.0987), *p3* é uma prova de *has_Latitude* (caminhão, 67.2365) e *ct1* é uma prova de que o tempo de avaliação é inferior ao tempo máximo (2/11/06.11: 33 <2/11/06.12: 00). Desde que o isomorfismo de Curry-Howard identifica provas com programas, isso pode ser usado para provar uma especificação, ou em outras palavras, para selecionar que definições são necessárias para que a especificação trabalhe corretamente.

- Tipos de Manifesto: (*Manifest Types*) são tipos pré-definidos que podem ser introduzidos com os tipos de manifesto [T. Coquand, R. Pollack e M. Takeyama:].

Definição: *Dado x do tipo T, x: T, um tipo singleton T_x tal como:*

$$y : T_x \text{ iff } y = x$$

Dado um registro, um campo de manifesto é um campo cujo tipo é um tipo *singleton*. Será adotada a notação de [R. Cooper, J. Ginzburg, 1998]:

$$r : \begin{bmatrix} \vdots \\ l = x : T \\ \vdots \end{bmatrix}$$

Os registros são também recursivos, o que significa que um valor correspondente a um rótulo pode ser ele próprio um tipo de registro.

- Tipos de Registro Complexos: campos de manifesto permitem especificar valores constantes para campos do tipo simples, bem como estruturas mais complexas, tais como os tipos de registro. Como conseqüência, qualquer tipo de restrições pode resultar em uma expressão de registro. Seja considerada uma restrição originada em uma determinada situação, essa situação pode ser desenhada como um registro da seguinte forma:

$$\begin{bmatrix} v = \begin{bmatrix} \text{x} : \textit{Vehicle} \\ \text{y} : \textit{RegistrationNumber} \\ p_1 : \textit{has\_identification}(\text{x}, \text{y}) \end{bmatrix} & : \textit{record} - \textit{type} \\ x & : \textit{Ind} \\ p_1 & : \textit{man}(x) \\ p_2 & : \textit{own}(x, v) \end{bmatrix}$$

Este tipo corresponde ao contexto em que o homem é dono de um veículo possuindo um número de registro. A propriedade é verdadeira apenas no caso de existir um registro do tipo:

$$\begin{bmatrix} x : \textit{Vehicle} \\ y : \textit{RegistrationNumber} \\ p_1 : \textit{has\_identification(x, y)} \end{bmatrix}$$

- Intenção: a realização de metas dentro de um processo é dependente do contexto. Como conseqüência, a noção de *contexto-de* formaliza-se com uma relação entre o contexto e o tipo de meta. Contexto deve ser relacionado a um conceito intencional e um campo de aplicação importante relacionado a este aspecto prende-se a processos tais como tarefas ou atividades humanas. A intenção é tornar mais concretas,

por meio de uma expressão de meta (ou seja, uma proposição). A fim de preservar a estrutura do modelo, este objetivo é em si um tipo de registro dependente. A teoria dos tipos de registro como parte da teoria geral do tipo que permite definir funções e tipos de função fornece uma versão do cálculo-λ tipado. A intenção pode ser desenhada como uma função de um tipo de contexto para um tipo de registro que descreve a relação pretendida do objetivo resultante da ação do meio ambiente:

$$\lambda c_{i\,:}\, C_{i.}\,([g : [g_1 : ...)$$

Pode ser visto como um link no qual um agente que observa uma ação agindo no contexto tipo C_i irá prever a existência de um objetivo do tipo $[g_1 : . . . .$

Outro aspecto importante desta modelagem com o CRT é um contexto que pode ter qualquer número de campos (não há limite superior). Como resultado, uma clara compreensão do conceito de subtipagem é necessária.

2.7 SUBTIPOS COM CONTEXTOS

A questão da subtipação exige o conhecimento de todas as coerções possíveis utilizadas por um prazo determinado e os seus efeitos precisos, o que é intratável na prática. Este problema pode ser evitado através da imposição de restrições semânticas sobre coerções [G. Betarte]: este é o caso da subtipagem baseada em registros que será adotada aqui.

Definição: *Dado dois tipos de registro C e C', se C contém pelo menos cada rótulo declarado em C e se os tipos destes rótulos comuns estão na relação de inclusão, então C está incluído em C' que está escrito:*

$$C \sqsubseteq C'$$

A extensão de um contexto físico tipo C para um tipo de contexto C' corresponde ao processo de obtenção de mais informações. Como na teoria dos tipos, o análogo de uma proposição é o julgamento, podemos concluir que o julgamento em C é levantado para o julgamento em C'. Cada registro de símbolo do tipo C é também um símbolo do tipo C', uma vez que contém componentes de formas apropriadas para todos os campos especificados em C'. Inclusão de tipos e provas de regras correspondentes generalizam a inclusão de tipos de registros para os tipos de registros dependentes e podem propagá-lo para todos os tipos dentro da linguagem.

$$\frac{R : record - type}{\Gamma \vdash R \sqsubseteq < >}$$

A equação de qualquer tipo de registro dependente é um subtipo do registro vazio (o registro que não impõe restrições). Contexto, por si só, não faz sentido uma vez que deve estar relacionado a um conceito intencional (que é, ontologicamente falando, um momento universal) [P. Brézillon e J.-Ch. Pomerol, 2001]. Como resultado, o conhecimento do contexto relaciona-se fortemente com um framework ontológico.

2.8 A ONTOLOGIA DE DOMÍNIO

Abstrações de modelagens de contexto são necessárias para apoiar o entendimento comum e para facilitar a comunicação entre agentes em ambientes distribuídos. A representação padrão de contexto de metadados é necessária para frameworks baseados em agentes para garantir a sua interoperabilidade. Ontologia semântica pode fornecer a representação padrão. Sistemas baseados em ontologias são usados em pesquisas atuais em IA para construir o senso comum em máquinas, assim ontologia parece ser a escolha óbvia para tornar os sistemas inteligentes.

Uma abordagem recente de [P. Dockhorn-Costa, J.P. A. Almeida, L. F. Pires, G. Guizzardi e M. van Sinderen, 2006] propôs caracterizar o conceito de contexto dentro de um framework ontológico. Os autores introduziram alguns conceitos, tais como o “contexto de contenção”, que associa várias entidades dentro de um contexto, a exigência de qualidade em que as entidades são portadoras de qualidades (momento intrínseco), as relações formais entre os indivíduos e as relações materiais em que as entidades compartilham uma propriedade comum. Apesar de suas definições interessantes de entidades e contextos, este modelo conceitual de contexto não é suficiente para cobrir todos os aspectos do raciocínio de contexto, pois a lógica subjacente não é explícita. Além disso, o conceito de *contexto* descrito como um *momento universal* não formaliza a noção de *contexto-de*.

Como alternativa, um sistema sensível ao contexto requer a capacidade de representar objetos e manter informações sobre seus atributos e relações com outros objetos. Essa teoria pode ser parcialmente definida através de uma ontologia que descreve as entidades concretas ou abstratas e as suas relações [C.J. Matheus, M.M. Kokar and K. Baclawski, 2003]. A maioria das abordagens orientada a Ontologia ignoram as questões relativas a restrições e relações quantitativas e focam mais nas

metas da partilha de conhecimento [A. Held, S. Buscholz e A. Schill, H. Chen, 2002, T. Finin e A. Joshi, 2003]. A ontologia deve prover um vocabulário para representar o conhecimento sobre o domínio e um entendimento comum da estrutura de informações de contexto. No entanto, capturar contextos em um ambiente de computação pervasiva é uma tarefa difícil. Em vez de definir um "*ContextEntity*" específico, como fizeram [T. Gu, X.H. Wang, H.K. Pung e D.Q. Zhang], é preferível começar com conceitos básicos e simples, como entidade e relação que são a base de todas as ontologias. A idéia principal é substituir o conceito de *contexto* em [P. Dockhorn-Costa, J.P. A. Almeida, L. F. Pires, G. Guizzardi e M. van Sinderen, 2006] com o conceito mais atômico da *propriedade*. Como resultado obtém-se a ontologia do núcleo descrita na Figura 4.

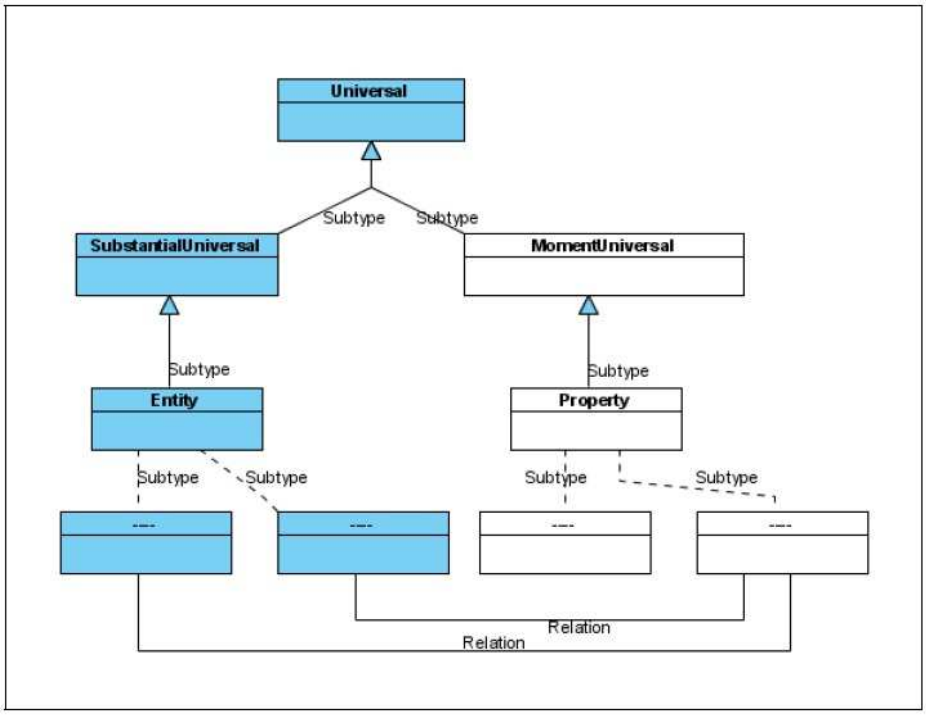

Figura 4: A Ontologia do Núcleo
Fonte: Richard Dapoigny; Patrick Barlatier, "Towards a Context Theory for Context-aware systems", Advances in Ambient Intelligence, Augusto, J. C.; Shapiro, D., IOS, Press 2007

O tipo de possessivo permite relacionar semanticamente o substantivo da entidade a um substantivo da propriedade em uma "*tem_<propriedade>*" ("*has_<property>*") específica de relação. Através das categorias fundamentais da *Entidade* e *Propriedade*, todos os componentes básicos do mundo físico são definidos. Acima destas suposições, o contexto de contenção nada mais é do que uma extensão de contexto (*sub-contexto-de*) transformado dentro da lógica. As propriedades intrínsecas e as relações materiais são representadas por "*tem_<propriedade>*" enquanto as relações

formais são diretamente expressas por proposições na lógica. Relações de subsunção clássica e relações materiais entre entidades e suas propriedades explícitas conceituam de uma forma simples para uma determinada aplicação.

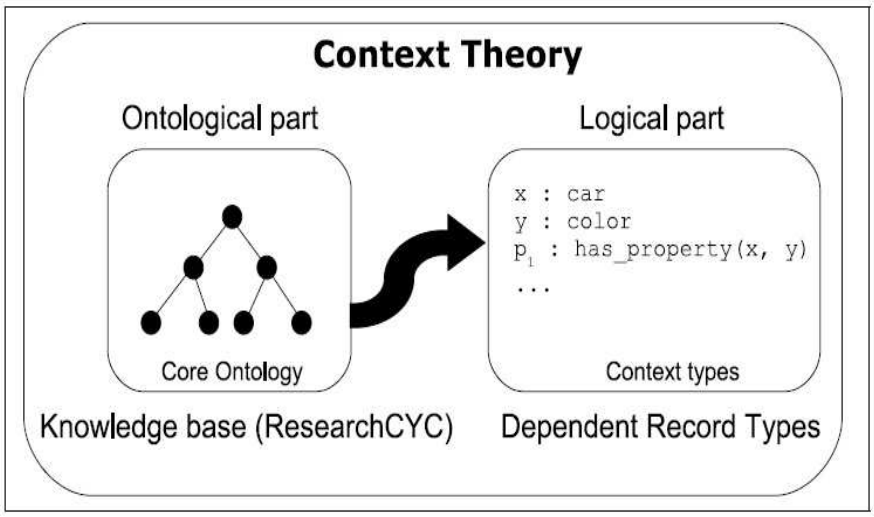

Figura 5: A Teoria do Contexto
Fonte: Richard Dapoigny; Patrick Barlatier, "Towards a Context Theory for Context-aware systems", Advances in Ambient Intelligence, Augusto, J. C.; Shapiro, D., IOS, Press 2007

A ontologia resultante de nível superior não formaliza o contexto, mas seus componentes elementares, ou seja, as entidades e suas propriedades. No entanto, a parte ontológica não é suficiente por si só e deve ser estendida com uma lógica apropriada. Desta forma, a Teoria de Contexto pode ser formulada com um modelo completo (ver Figura 5). O primeiro componente (parte conceitual) construído na etapa de design define a ontologia da aplicação e segue as regras de construção da ontologia de nível superior. O segundo componente (parte lógica) aborda o aspecto lógico relacionando os conceitos definidos na camada anterior através da geração de CRT, incluindo entidades e proposições. É só nesta parte (também construído durante a etapa de design) que o contexto aparece de uma forma dinâmica. De uma perspectiva alargada, a Teoria de Contexto apresenta capacidades tanto para processos mais aplicados ou para uma abordagem mais conceitual. Já que a ontologia tem sido o meio de representação prevalente, existe certa base para se manter nela.

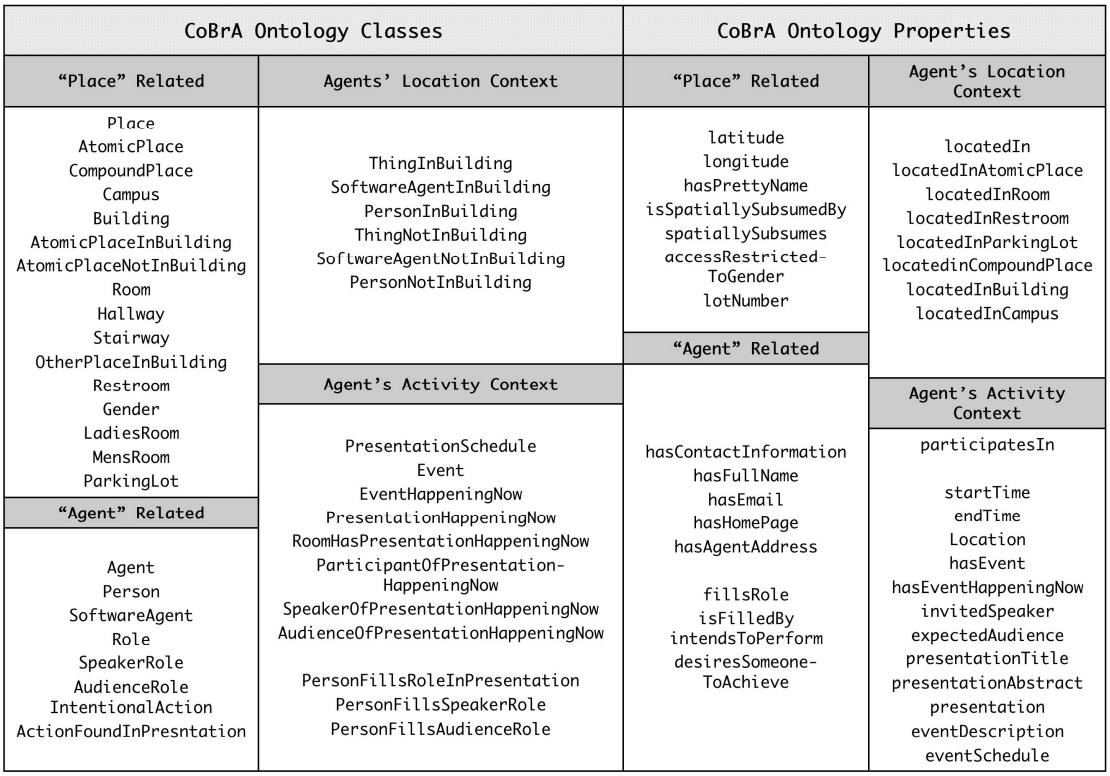

Figura 6: Classes e propriedades de Contexto na Arquitetura CoBrA, baseadas em Ontologia

Fonte: CHEN, Harry; FININ, Tim; JOSHI, Anupam: "An Ontology for Context-Aware Pervasive Computing Environments", 2003

3. TRATANDO CONTEXTO ATRAVÉS DE SISTEMA DE TIPOS

Na Ciência da Computação, Sistema de Tipos pode ser definido como “um *framework* sintático tratável para classificar frases de acordo com tipos de valores a computar” [Benjamin Pierce, 2002], ou seja, associa um valor a um tipo.

Na computação pervasiva o uso do sistemas de tipos, está associado a seu contexto, ou seja, as variáveis que compõem o ambiente em que está inserido, tornando o software a ser desenvolvido sensível ao contexto, que tem como objetivo básico tornar esse software ciente do ambiente e se adaptar às mudanças ocorridas. Para esse efeito, o grande problema é ter um modelo de contexto poderoso. Enquanto formalizações significativas têm sido propostas, modelos de contexto são tanto expressos através de formalismos lógicos quanto com abordagens baseadas em ontologia.

O grande problema dessas abordagens é que elas sofrem com a insuficiência crônica da lógica de primeira ordem para lidar com mudanças dinâmicas e, sobretudo, para resolver o problema de *frame*.

Portanto, a construção de um software com consciência do contexto é uma tarefa complexa, devido à falta de modelos formais apropriados em ambientes dinâmicos.

O modelo formal depende de um conjunto de fatores, como de uma representação de conhecimento em ontologias e em um raciocínio lógico com Tipos de Registro de Dependentes (DRT), com base na Teoria Intuicionista dos Tipos (ITT) e no isomorfismo de Curry-Howard.

3.1 CONSCIÊNCIA DO CONTEXTO EM SISTEMAS DE TIPOS

Modelagem de contexto tem sido alvo de muita pesquisa atualmente, porém não existe um consenso sobre a definição de contexto, sua representação, ou mesmo sua modelagem. Conseqüentemente, um modelo epistemologicamente adequado e sólido é exigido que inclua mecanismos para a integração e o uso do contexto.

Contextos podem ser tratados como *contexto-de (context-of)*, já que estão atrelados ao conceito de propriedade. Com isso podemos ter o conceito de *contexto de uma ação*, que é uma importante subclasse de aplicações do mundo real na qual qualquer processo físico pode ser modelo numa informação contextual. Esta subclasse

inclui processos tais como aplicações de engenharia e controle, engenharia na web, tarefas clínicas e processos de negócios [Richard Dapoing e Patrick Barlatier, 2007].

Por isso, neste trabalho, será feita uma abordagem que focará na forma como o contexto lida com os processos físicos em que o elemento central é uma tarefa. Com isso, o contexto deve ser tratado como *contexto de uma ação* já que é o núcleo intencional da tarefa.

Um grande desafio é fazer o software estar ciente do ambiente (contexto) e se adaptar às mudanças que ocorrem neste ambiente. O contexto pode ser mais bem expresso através de dois formalismos vistos anteriormente neste trabalho, o ontológico e o lógico. Estes modelos são os mais utilizados em Computação Pervasiva e Inteligência Artificial. Em estudos comparativos é demonstrado que modelos baseados em ontologias são mais apropriados e que essas abordagens são bastante complementares, especialmente em composição distribuída e validação parcial.

Enquanto a representação ontológica define as regras e relações, o motor de inferência trabalha sobre estas regras para derivar inferências significativas. O processo de dedução e raciocínio é altamente dependente da forma como as regras estão representadas. Linguagens lógicas podem trabalhar em fórmulas com lógicas de primeira ordem e fórmulas com quantificação. No entanto, o núcleo do problema com as abordagens atuais não é a falta de um poderoso motor de regras, mas sim a falta de meios para facilmente representar cenários do mundo real em termos de regras. Para lidar com este problema, será feito o estudo sobre a adoção da estrutura da lógica intuicionista, que tende a ser mais realista do que a lógica de primeira ordem.

Richard Dapoigny e Patrick Barlatier sugerem o uso de um framework lógico baseado em intuicionismo e na teoria dos tipos para representar contextos a partir de uma perspectiva de Inteligência Artificial.

3.2 CONTEXTO EM INTELIGÊNCIA ARTIFICIAL

Contexto tem crescido em várias áreas da IA, incluindo a representação do conhecimento, processamento de linguagem natural e recuperação de informação inteligente. Em sistemas conscientes do contexto, uma definição conhecida é expressa como: “um sistema é consciente do contexto se ele usa o contexto para prover informação relevante e/ou serviços para o usuário, onde a relevância depende da tarefa

do usuário" [Anind K. Dey e Gregory D. Abowd, 2000]. Essa definição requer categorizar explicitamente o conceito de contexto, conforme visto na seção 2.1.

Um problema que pode ocorrer é definir claramente a forte ligação entre os contextos e situações. No trabalho de [F. Giunchiglia, 1992], isso é destacado, quando três diferentes possibilidades são detalhadas (uma quarta é a combinação dos três tipos básicos). Uma situação dada é relacionada a múltiplos contextos, cada um sendo uma aproximação diferente daquela situação. No artigo, é defendido que esses contextos são diferentes, porque correspondem a diferentes objetivos que se têm em mente, expressos para uma definição de *contexto-de*. Em sua correspondência um-para-um entre situação e contexto, o autor dá um exemplo que envolve uma evolução no tempo de um processo físico. Considerando primeiro que um processo físico tem um objetivo global e, segundo, que esse objetivo pode ser composto de sub-objetivos, cada um deles sendo atingido sucessivamente, resulta que, num dado momento, um único *contexto-de* está disponível (ou seja, o *contexto-de* dos correspondentes sub-objetivos). Uma outra possibilidade, em que um único contexto corresponde a muitas situações, reflete a abordagem tradicional do Cálculo Situacional [John McCarthy 1963, Ray Reiter 1991] em que os contextos não são usados.

Outra questão refere-se à lógica subjacente. Quase todos os trabalhos na literatura sobre o raciocínio de contexto adotam a abordagem da lógica de primeira ordem (*first-order logic*, FOL). No entanto, isso resulta em grande dificuldade de expressar a validação parcial e mudanças dinâmicas.

3.3 RECURSOS DA IMPLEMENTAÇÃO

A implementação deve satisfazer dois conjuntos de restrições, um resultante da fase de design e o outro da fase de *execução* (*run-time*). A fase de design requer, i) verificar a validade semântica dos conceitos, ii) verificar a composição sintática dos CRT.

Na fase de execução, CRT são comparados com a situação atual expressa como uma lista de proposições e entidades, no espírito de [Roy M. Turner e Robert A.G. Stevenson, 1991]. Para a primeira fase, a ontologia *ResearchCyc* é usada (detalhes abaixo na Figura 7). Para a segunda fase, um checador Lisp [John McCarthy, 1958.] é utilizado como provador de teoremas. Em relação à primeira fase, faz-se uso de *Augmented Transition Networks* (ATNs) que são uma forma de analisador utilizado

para análise de linguagem natural. O sistema ATN é implementado como um *front-end* com o banco de dados. Com uma interface ATN-dirigida ligada ao sistema, ao invés de fazer uma consulta formal, os usuários podem fazer perguntas de uma forma restrita em inglês. Como resultado, a ferramenta apresenta uma melhor compatibilidade com os sistemas baseados em NLP [Richard Bandler e John Grinder, 1975].

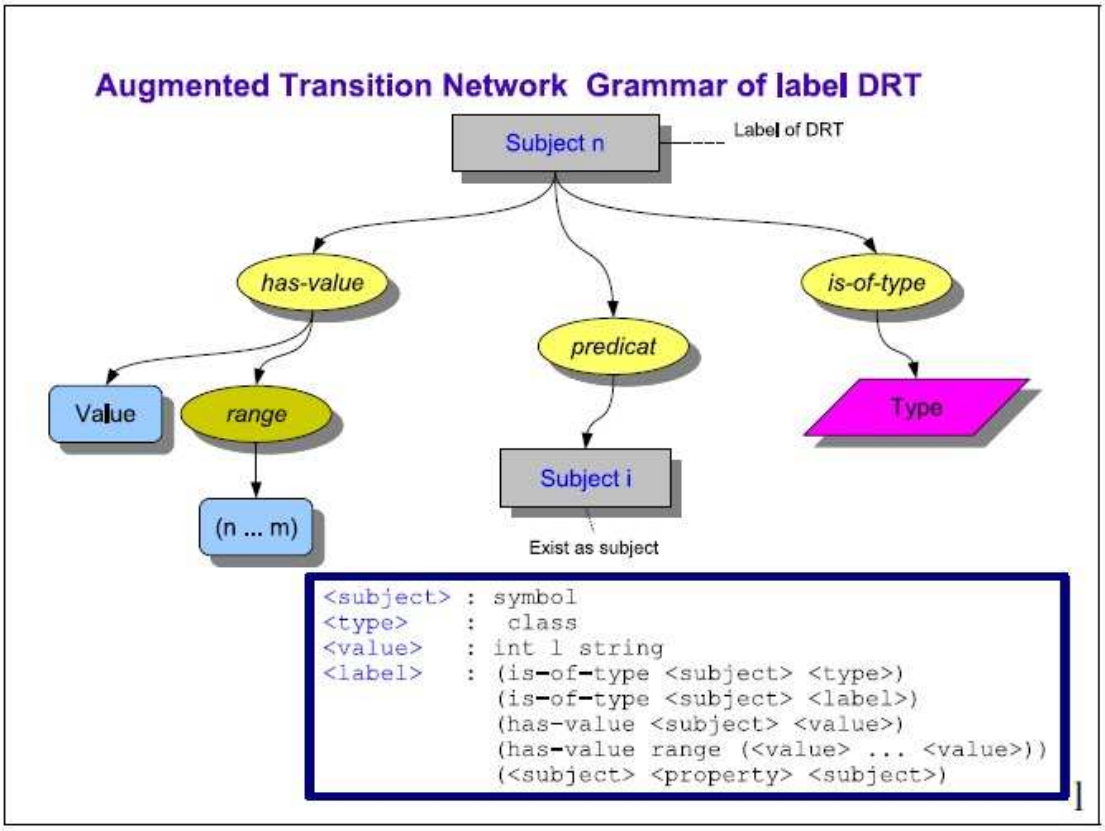

Figura 7: A formação de CRTs
Fonte: Richard Dapoigny; Patrick Barlatier, "Towards a Context Theory for Context-aware systems", Advances in Ambient Intelligence, Augusto, J. C.; Shapiro, D., IOS, Press 2007

A modelagem CRT conta com a propriedade de encerramento que: i) fornece uma maneira poderosa de representar variáveis dependentes em *Lisp*, ii) facilita o design de estruturas DRT que possuem um estado. Os construtores de tipo DRTs utilizados na ferramenta são mônadas que capturam várias noções de computação seqüencial. Sua principal vantagem se prende à restrição das habilidades de programação com funções em vez de variáveis globais. Macros de Lisp geram estruturas DRT que são armazenadas em uma tabela *hash* e dentro de uma ontologia (em linguagem RDF). A ontologia que é expressa em RDF ou OWL descreve as relações de subtipagem entre DRTs. Para esse efeito, a ferramenta *Allegrograph* é usada como um *Datawarehouse* de alto desempenho, permitindo um elevado nível de espaço

de memória e operações de alta velocidade. Com essa ferramenta, os dados são armazenados como *triplas* de RDF e OWL.

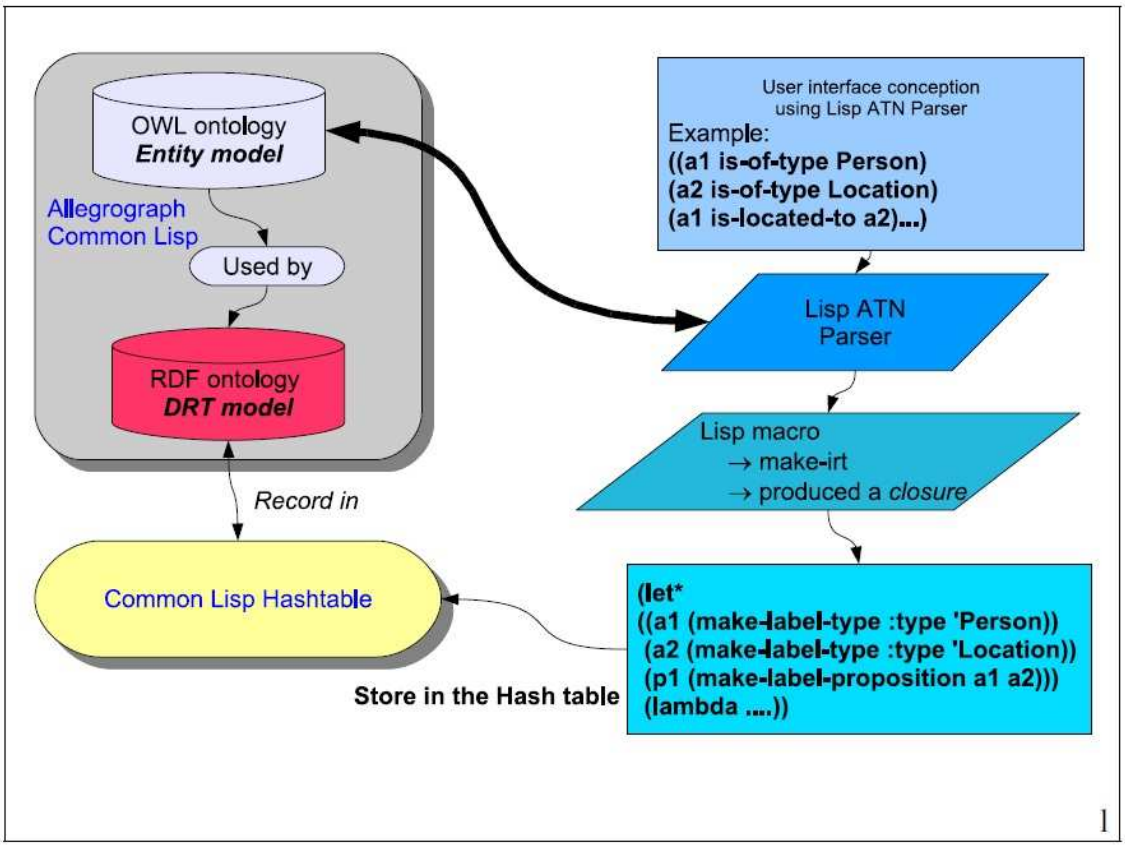

Figura 8: A arquitetura global
Fonte: Richard Dapoigny; Patrick Barlatier, "Towards a Context Theory for Context-aware systems", Advances in Ambient Intelligence, Augusto, J. C.; Shapiro, D., IOS, Press 2007

A base de conhecimento resultante é competitiva, autônoma e compartilhável entre as aplicações. Como programas de ontologias fornecem a possibilidade de acessar um conhecimento compartilhado em uma rede, os programas podem modificar os seus conhecimentos em tempo de execução. As ontologias de aplicação, cuja validade restrita é limitada à execução da tarefa, são uma ferramenta adequada para as especificações dos tipos de contexto das suas relações. A ontologia dinâmica foi concebida, incluindo uma definição intencional de conceitos e suas relações com as operadoras que permitem a composição de novos conceitos e suas relações. A principal vantagem desta abordagem é evitar a explosão combinatória inerente às abordagens estáticas. A Base de Conhecimento é composta de termos - o vocabulário da *Cyc* - e afirmações que se referem a estes termos. Estas afirmações incluem afirmações de base e regras simples. A representação de tipos de contexto no *Cyc* permite a criação de

programas para compartilhar uma base de conceito e permite a inferência e raciocínio de mecanismos que exigem esta base. A arquitetura de software resultante é relatada na Figura 8.

A consistência que as pessoas colocam nos contextos, porém, é de responsabilidade do usuário. A linguagem de programação é poderosa o suficiente para deixar as pessoas ficarem penduradas em um loop, e o sistema de raciocínio prático é poderoso o suficiente para permitir que as pessoas digam coisas que são incompatíveis.

3.4 EXEMPLO DE APLICAÇÃO - SMART PHONE

Este cenário simples é extraído do trabalho de [J. Euzenat, F. Ramparany e J. Pierson, 2006]. Este cenário é criado em um hotel em que os quartos estão equipados com telefones inteligentes, ou seja, telefones que são capazes de receber informações contextuais a partir de sensores (ou outros dispositivos), para agregá-las e raciocinar sobre os contextos, a fim de tomar decisões. Supõe-se que o *Smart Phone* tenha poder de computação suficiente para o processamento de contexto e de raciocínio. Neste cenário, o usuário está dentro do quarto e realiza alguma atividade. O telefone inteligente recebe uma chamada que não é uma comunicação de emergência e deve decidir entre três tarefas: ele deve tocar, produzir apenas uma vibração ou encaminhar a chamada para o correio de voz. Estas ações dependem fortemente da atividade do usuário e, para esse efeito, o telefone inteligente deve agregar informações de meio ambiente, como funcionamento de aparelhos elétricos, detecção de movimento, o estado físico (luminosidade, temperatura, etc). Em outras palavras, o telefone inteligente tem de definir o âmbito da sua missão. Uma vez que tem que lidar com três tarefas, três tipos de contextos estão ligados a essas tarefas como segue:

$$C_1 = \begin{pmatrix} x : hotel - room \\ y : user \\ w : call \\ p_1 : has\_location(y, x) \\ p_2 : has\_LowPriority(w) \\ z : smart - phone \\ s : beep \\ p_3 : has\_activity(y) \end{pmatrix} \longrightarrow \left[g_1 : provide(c_1.z, c_1.s) \right.$$

$$C_2 = \begin{pmatrix} x : hotel - room \\ u_1 : user \\ u_2 : user \\ w : call \\ p_1 : has\_location(y, x) \\ p_2 : has\_LowPriority(w) \\ z : smart - phone \\ s : buzz \\ p_3 : Interact\_with(u_1, u_2) \end{pmatrix} \longrightarrow [g_1 : provide(c_2.z, c_2.s)$$

$$C_3 = \begin{pmatrix} x : hotel - room \\ y : user \\ w : call \\ p_1 : has\_location(y, x) \\ p_2 : has\_LowPriority(w) \\ z : smart - phone \\ v : phone - message \\ p_3 : Asleep(y)\_ \end{pmatrix} \longrightarrow [g_1 : forward(c_3.z, c_3.v) p_2 :$$

Além disso, as informações recebidas do meio ambiente podem extrair tipos de contexto e gravar a partir de bibliotecas da seguinte forma:

- informações de um dispositivo elétrico pode preencher os campos:

$$\begin{pmatrix} x : light \\ q_1 : has\_stateOff(x) \\ ... : ... \end{pmatrix}$$

- informações de um aparelho de TV pode alimentar alguns campos, tais como:

$$\begin{pmatrix} x : tv \\ q_1 : has\_stateOn(x) \\ y : channel \\ v : volume \\ q_2 : has\_selectedChannel(x, y) \\ q_3 : has\_stateOff(v) \\ ... : ... \end{pmatrix}$$

- informações de detectores de infravermelho ou câmeras infravermelhas podem validar os campos:

$$\begin{pmatrix} u_1 : user \\ r : hotel - room \\ q_1 : has\_activity(u_1) \\ q_2 : has\_multipleUsers(r) \\ ... : ... \end{pmatrix}$$

O resultado da agregação de informações do sensor fornece um vetor de conhecimento, descrevendo a situação atual. A partir desta situação, tipos de contexto são verificados e a saída é ajustada (isto é, a tarefa é definida). Trechos de tipos de contexto são:

$$C_{10} = \begin{pmatrix} u_1 : user \\ x : light \\ q_1 : has\_stateOff(x) \\ r : hotel - room \\ q_2 : has\_singleUser(r) \\ t : tv \\ q_3 : has\_stateOff(t) \end{pmatrix} \longrightarrow \lceil g_1 : Asleep(c_{10}.u_1)$$

$$C_{11} = \begin{pmatrix} u_1 : user \\ r : hotel - room \\ q_1 : has\_activity(u_1) \\ q_2 : has\_multipleUsers(r) \\ x : light \\ q_3 : has\_stateOn(x) \end{pmatrix} \longrightarrow \lceil g_1 : Interact\_with(c_{11}.u_1, c_{11}.u_2)$$

Supondo-se agora que os seguintes itens são checados em uma situação particular:

$$\begin{pmatrix} c_{10}.u_1 = Jack \\ c_{10}.x = l\#5677 \\ c_{10}.q_1 = has\_stateOff(x) \\ c_{10}.r = \#1345 \\ c_{10}.q_2 = has\_singleUser(r) \\ c_{10}.t = Panasonic\#1322 \\ c_{10}.q_3 = has\_stateOff(t) \\ c_3.x = \#1345 \\ c_3.y = Jack \\ c_3.w = \#217884 \\ c_3.p_1 = has\_location(y, x) \\ c_3.p_2 = has\_LowPriority(w) \\ c_3.z = Alcatel\#BSX120 \\ c_3.v = \#4365499 \end{pmatrix}$$

Os seis primeiros itens validam o contexto C_{10}, o que, em si, é uma prova de que *Asleep*($c_{10}.u_1$) é válida. Como resultado, estes seis itens podem ser substituídos por *Asleep*($c_{10}.u_1$) e desde que $c_{10}.u_1$ e $c_3.y$ correspondem a um mesmo símbolo da entidade, o contexto torna-se c_3, válido por si mesmo. Finalmente, a ação que conduz à proposição *forward*($c_3.z$, $c_3.v$) é selecionada. Naturalmente, este cenário só é suposto para explicar o mecanismo de verificação de CRT e, em um caso real, mais informações são adicionadas. Observe que os números de identificação de entidades também podem ser substituídos por URIs.

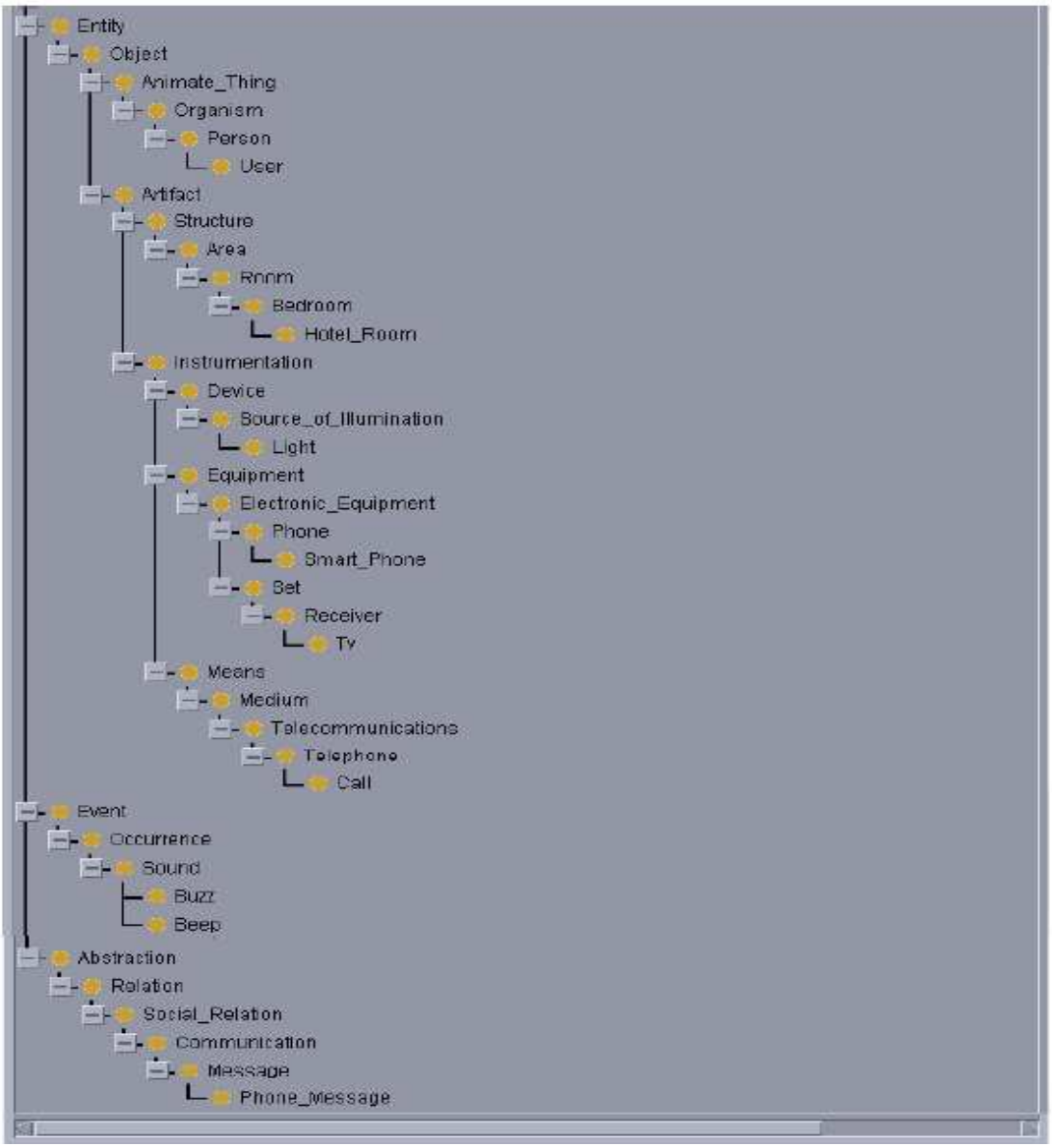

Figura 9: Entidades da situação

Fonte: Richard Dapoigny; Patrick Barlatier, “Towards a Context Theory for Context-aware systems”, Advances in Ambient Intelligence, Augusto, J. C.; Shapiro, D., IOS, Press 2007

4. ESTUDO DE CASO

Para demonstrar o uso de sistema de tipos em ambientes pervasivos será feito um estudo de caso em uma situação factível. Para isso será utilizado um cenário conhecido, como uma “casa inteligente” (*smart house*).

Uma casa inteligente consiste em controles eletrônicos programáveis e sensores que podem regular o aquecimento, refrigeração, ventilação, iluminação, equipamento e operação de equipamentos em conservação de energia e de maneira climaticamente responsiva, proporcionando, com isso, maior conforto para os moradores da casa.

Uma casa inteligente, porém, possui muitas variáveis se forem considerados todos seus cômodos. Em vista disso, vamos nos restringir à sala de estar da casa para fazer o estudo de como um sistema de tipos pode ser utilizado nas variáveis do ambiente.

4.1 DESCRIÇÃO DA SITUAÇÃO

A casa inteligente é um ambiente pervasivo que tem a capacidade de observar os moradores na busca de padrões. Depois que o padrão é encontrado, aparelhos e equipamentos começam a ser executados automaticamente, mudando cada vez que o padrão é alterado, ou seja, o contexto é modificado. A sala de estar, alvo deste estudo, possui uma televisão, um aparelho de som, iluminação, janelas e um condicionador de ar, todos com sensores, ou seja, inteligentes, com a possibilidade de serem configurados.

Figura 10: Sala de estar mobiliada
Fonte: http://arhzine.com/interior/tag/living-room-furnishing/

A televisão irá refletir, em termos de contexto, se o usuário estará interessado em assistir a um determinado programa e qual o canal deve ser selecionado. Isto será regulado por um padrão pré-estabelecido em função das preferências, levando em consideração o horário, dia da semana, ou até pré-configurações realizadas a partir de guias de revistas com a programação dos canais do qual o usuário é assinante.

O aparelho de som também poderá ter uma pré-configuração em função de dias, horários e a existência de mais de uma pessoa no local, visando à criação de um ambiente mais aconchegante. Estações de rádio ou músicas pré-gravadas serão ajustadas para responder às situações do contexto.

A iluminação é regulada de acordo com a atividade do usuário. Ele poderá estar sentado lendo, ou pode estar sentado assistindo televisão. Para capturar essas nuances deverá existir um sensor que tenha condições de verificar a postura do usuário como é feito, por exemplo, pelo Microsoft Kinect[4] do console Xbox 360®. Se o usuário estiver lendo, a iluminação ficará mais intensa e o som ficará mudo ou com uma música suave, e se estiver assistindo televisão a iluminação ficará mais branda, por exemplo.

O condicionador de ar regula a temperatura de acordo com o gosto do usuário e é acionado com a presença do mesmo, ou em horários pré-programados, neste caso preparando o ambiente para o usuário.

4.2 ONTOLOGIA

A Figura 11 apresenta a ontologia da situação descrita, na qual o usuário está no centro, pois suas ações determinam como os objetos deverão se comportar. O condicionador de ar, além da presença do usuário, sofrerá influência da temperatura do ambiente apenas. Os outros equipamentos da sala necessitam saber a localização do usuário para realizar as ações pré-estabelecidas.

[4] O Kinect (Anteriormente chamado de "Project Natal") é o nome de um projeto encabeçado pela Microsoft para seu console de videogame de última geração Xbox 360, que tem ainda como colaboradora a empresa Prime Sense. O projeto visa criar uma nova tecnologia capaz de permitir aos jogadores interagir com os jogos eletrônicos sem a necessidade de ter em mãos um controle/joystick, inovando no campo da jogabilidade, já bastante destacado pelas alterações trazidas pelo console Wii, da Nintendo.

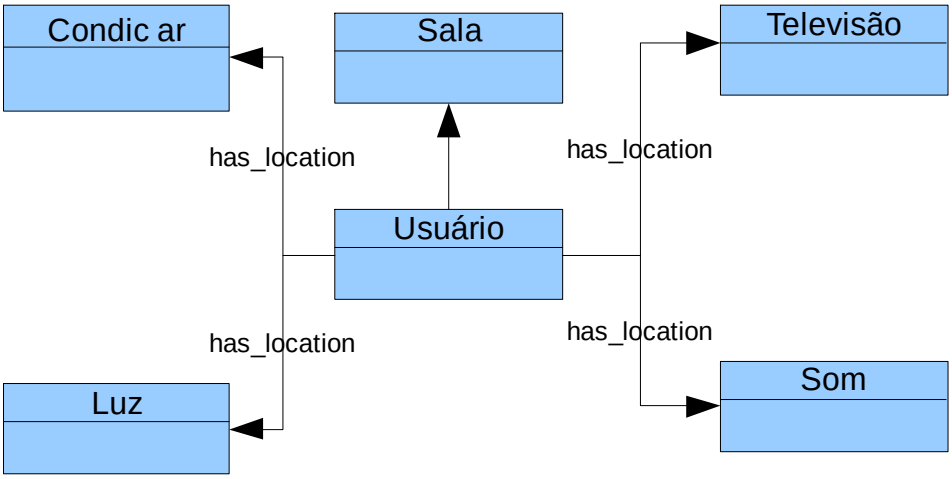

Figura 11: Ontologia da Sala de Estar

4.3 SISTEMA DE TIPOS

Para o estudo de caso proposto, exemplificaremos três tipos diferentes de contexto, entre os vários possíveis, todos com auxílio de sensores.

$$C_1 = \begin{cases} x : sala \\ y : usuario \\ w : entrada \\ p_1 : has\_location(y, x) \\ z : sensor \\ i : luz \\ t : tv \\ h : date\_time \\ s : has\_StateOn(t, h) \\ p_2 : has\_activity(y) \\ d_1 : is\_soft(i) \end{cases} \longrightarrow \begin{cases} g_1 : provide(c_1.t, c_1.s) \\ g_2 : provide(c_1.i, c_1.d_1) \end{cases}$$

Neste contexto, o usuário entra na sala, a televisão é ligada em um canal previamente programado de acordo com o horário e dia da semana. Concomitantemente a luz é acesa com baixa intensidade. Conforme o horário (*h*) a luz pode não acender, em função da luminosidade do ambiente.

$$C_2 = \begin{cases} x : sala \\ y : usuario \\ w : leitura \\ p_1 : has\_location(y, x) \\ z : sensor \\ a : aparelho\_de\_som \\ i : luz \\ t : tv \\ s_1 : has\_StateOff(t) \\ s_2 : has\_StateOn(a) \\ d_2 : is\_high(i) \\ o : Lendo(y) \end{cases} \longrightarrow \begin{cases} g_1 : provide(c_2.i, c_2.\ d_2) \\ g_2 : provide(c_2.a, c_2.s_2) \\ g_3 : provide(c_2.t, c_{2.}s_1) \end{cases}$$

Neste contexto, o usuário está lendo, então, o aparelho de som é ligado com uma música suave previamente configurada e a iluminação fica mais intensa. A televisão é desligada.

$$C_3 = \begin{cases} x : sala \\ u_1 : usuario \\ u_2 : usuario \\ w : reuniao \\ p_1 : has\_location(y, x) \\ p_2 : Interact\_with(u_1, u_2) \\ z : sensor \\ a : aparelho\_de\_som \\ t : tv \\ s_1 : has\_StateOff(t) \\ s_2 : has\_StateOn(a) \end{cases} \longrightarrow \begin{cases} g_1 : provide\ (c_3.t, c_3.s_1) \\ g_2 : provide\ (c_3.a, c_3.s_2) \\ g_3 : provide(c_3.t, c_{3.}s_1) \end{cases}$$

Neste contexto, o usuário está em reunião, ou simplesmente conversando com outra pessoa: a televisão é desligada e o aparelho de som toca uma música mais suave, apropriada para a situação.

$$\begin{cases} x : tv \\ q_1 : has\_StateOn(x) \\ y : channel \\ v : volume \\ q_2 : has\_selectedChannel(x,y) \\ q_3 : has\_StateOff(x) \\ \quad \ldots \end{cases}$$

Em função dos contextos apresentados anteriormente, são mostradas acima algumas propriedades possíveis da televisão. Como por exemplo, se a televisão está desligada ou não, os canais disponíveis e o canal previamente selecionado, o volume, entre outras coisas.

$$\begin{cases} x : aparelho\_de\_som \\ q_1 : has\_StateOn(x) \\ y : station \\ v : volume \\ q_2 : has\_selectedStation(x,y) \\ q_3 : has\_StateOff(t) \\ \quad \ldots \end{cases}$$

Em relação ao aparelho de som, essas são algumas propriedades possíveis, como: se o aparelho está desligado ou não, as estações de rádio disponíveis e as estações previamente selecionadas além do volume ajustado.

$$\begin{cases} x : luz \\ q_1 : has\_StateOn(a) \\ q_2 : is\_high(x) \\ \quad \ldots \end{cases}$$

A iluminação da sala pode apresentar algumas das propriedades acima, como: se está ligada ou não, se é intensa ou suave.

$$\begin{cases} x : condic\_ar \\ q_1 : has\_StateOn(x) \\ y : temperatura \\ s_1 : resfriar \\ \quad \ldots \end{cases}$$

Neste contexto, são apresentadas algumas propriedades possíveis do condicionador de ar, como se está ligado ou não, a temperatura previamente configurada, se está no modo de resfriamento ou de aquecimento.

Agora será apresentado um exemplo de situações com os contextos dos objetos da sala previamente demonstrados.

$$C_7 = \begin{cases} u_1 : usuario \\ x : luz \\ q_2 : is\_high(x) \\ r : sala \\ q_2 : has\_singleUser(r) \\ t : tv \\ q_3 : has\_stateOff(t) \end{cases} \longrightarrow \{ g_1 : Lendo(c_7.u_1)$$

$$C_8 = \begin{cases} u_1 : usuario \\ r : sala \\ q_1 : has\_activity(u_1) \\ q_2 : has\_multipleUsers(r) \\ x : luz \\ q_3 : has\_stateOn(x) \end{cases} \longrightarrow \{ g_1 : Interact\_with(c_8.u_1, c_8.u_2)$$

$$C_9 = \begin{pmatrix} u_1 : usuario \\ r : sala \\ y : temperatura \\ q_2 : has\_multipleUsers(r) \\ x : luz \\ q_3 : has\_stateOn(x) \end{pmatrix} \longrightarrow \{g_1 : Interact\_with(c_9.u_1)$$

Com base nestes contextos podemos ter a seguinte situação:

$$\begin{pmatrix} c_7.u_1 = Pablo \\ c_7.x = l\#8867 \\ c_7.q_1 = has\_stateOff(x) \\ c_7.r = \#sala\_de\_estar \\ c_7.q_2 = has\_singleUser(r) \\ c_7.t = Philips\#0052 \\ c_7.q_3 = has\_stateOff(t) \\ c_2.x = \#sala\_de\_estar \\ c_2.y = Pablo \\ c_2.w = \#lendo \\ c_2.p_1 = has\_location(y, x) \\ c_2.z = Sensor\#sensor \\ c_2.i = luz \\ c2.d = is\_high(i) \\ c2.s2 : has\_StateOn(a) \end{pmatrix}$$

Nesta situação, o usuário está lendo, e como $c_7.u_1$ e $c_2.y$ se referem ao mesmo usuário, o contexto c_2 é validado, levando em conta que eles possuem valores semelhantes nos seus itens de contexto, assim como no exemplo do *smart-phone*. Com base nisso, é possível substituir os valores do contexto c_7 por $Lendo(c_7.u_1)$.

4.4 CONCLUSÃO DO ESTUDO DE CASO

Após a construção da ontologia e do sistema de tipos baseados na descrição da situação, é possível concluir que a modelagem de situações com a abordagem proposta pelos autores Dapoigny e Barlatier é simples e intuitiva. Mesmo em um contexto cheio de variáveis, como a sala de estar de uma casa inteligente, ele se encaixa perfeitamente no modelo proposto.

O sistema de tipos foi empregado para modelar três tipos diferentes de contexto neste caso, onde três situações diferentes poderiam ocorrer. Cada situação possuindo sua particularidade específica, e o seu modo de operação. Assim, foi possível modelar três situações exemplos, para demonstrar melhor como o sistema de tipos poderia ser empregado em uma situação usual do cotidiano do usuário. Por fim, um contexto foi

validado com base em uma ação de um possível usuário, pois esta ação é equivalente ao que estava previsto em um dos contextos descritos.

Sem a utilização de um sistema de tipos, seria mais difícil captar as variáveis do ambiente com precisão, devido às mudanças dinâmicas que ocorrem dentro deste contexto. Por isso com o uso do sistema de tipos proposto é possível captar essas variáveis e utilizá-las, mesmo em aplicações dinâmicas.

5. CONCLUSÃO E FUTUROS TRABALHOS

Com a computação pervasiva evoluindo continuamente e se estabelecendo como um paradigma atual na informática torna-se necessário estabelecer meios de se estruturar e capturar informações para criação de softwares e dispositivos. Uma definição de contexto do ambiente da aplicação se tornou muito importante, pois em computação pervasiva os softwares normalmente são sensíveis a ele.

Surgiram vários trabalhos relacionados com esse propósito, como o modelo baseado em ontologias CoBrA, que procura caracterizar pessoas, lugares ou outros tipos de objetos como entidades dentro dos contextos onde são aplicados. Existem também abordagens lógicas, nas quais informações contextuais são introduzidas, atualizadas e apagadas em termos de fatos ou inferidas através de um conjunto de regras.

A partir do estudo feito pelos autores Dapoigny e Barlatier, foi possível ter o conhecimento de uma abordagem complexa, inovadora e inteligente para capturar informações em ambientes sensíveis ao contexto. Em comparação com as outras abordagens mencionadas no parágrafo anterior, a característica marcante desta é que o contexto é verificado por prova de teoremas. Foi utilizada uma teoria de contexto como um framework para representar as informações contextuais e o fluxo de informações entre contextos. Assim, com a verificação da existência de entidades – que são os substantivos presentes no ambiente - e suas propriedades em tempo de execução pode-se fazer esta abordagem mais apropriada para aplicações sensíveis ao contexto. As abstrações são encarregadas de apresentar os resultados das interações entre as entidades, da análise dos eventos e preparação das respostas para o usuário.

Os autores citados fizeram uso de CRT para assentar a parte lógica da aplicação que foi formalizada utilizando ontologias para a modularização de entidades e propriedades. Como linguagem de programação foi utilizada Lisp, uma linguagem funcional que muito se presta à tipagem dinâmica de dados. Os DRT’s foram utilizados para estabelecer as relações entre os contextos, e fornecer a possibilidade de capturar informações dinâmicas do contexto de uma forma mais eficiente.

Dapoigny e Barlatier procuraram explorar também algumas das principais características dos CRTs, como tipos dependentes, proposições como tipos e os tipos como objetos. CRT tem a capacidade de exprimir as dependências de contexto de forma dinâmica através da extensão do contexto, o que facilita seu emprego em

ambientes com muitas alterações. A arquitetura criada pelos autores, baseada em Lisp, é capaz de suportar todo o processo de harmonização de CRT com as situações. A quantidade de contextos possíveis é muito grande - mudou o ambiente, muda o contexto - o que resulta em modelos de ontologias enormes. Em consequência, há a necessidade de mecanismos de busca eficientes que possam reduzir o tempo da resolução dos problemas advindos das mudanças constantes do contexto, fato bem provido pelos CRT.

Atualmente existem questionamentos sobre se as linguagens de programação devem ser tipadas ou não. Os códigos de linguagens não tipadas, indubitavelmente, possuem mais dificuldades em se manter. Em geral, tipos precisos fornecem informações em tempo de compilação e levam a aplicação a operações apropriadas em tempo de execução, sem a necessidade de testes dispendiosos. Linguagens tipadas diminuem a compilação, pois suas informações de tipo podem ser organizadas em interfaces para módulos de programas, podendo assim ser compilados independentemente uns dos outros, com cada modulo dependendo apenas da interface dos outros. Além disso, um sistema de tipo bem projetado faz com que a checagem de tipos remova uma grande quantidade de erros, eliminando longas sessões de depuração.

Depois do estudo de todos os conceitos e da análise do artigo alvo, é possível concluir que Sistema de Tipos é uma ferramenta essencial, pois transmite muita segurança ao programador. Com o auxílio de ontologias de domínio e DRTs, torna-se uma boa ferramenta para capturar informações de contexto de forma dinâmica, por fornecer uma maior capacidade de representação de variáveis.

O estudo de caso realizado foi importante para se ter uma melhor visão dos conceitos descritos. Partindo de uma situação factível, como a sala de estar de uma casa inteligente, cujas variáveis de ambiente estão presentes no dia-a-dia, foi possível modelar sua ontologia, procurando retratar a situação proposta da melhor maneira, para em seguida ser construído um sistema de tipos sólido, com a capacidade de capturar com segurança as variáveis do ambiente, e com o auxílio dos CRTs foi possível capturar a dinamicidade dessas variáveis, aumentando a precisão das mesmas e facilitando sua utilização pelos sensores dos objetos da sala de estar. Com isso, um exemplo foi modelado para demonstrar a aplicação do Sistema de Tipos construído.

Como um dos futuros trabalhos possíveis com base neste estudo, sugere-se a implementação de um checador de tipos, desde a construção de uma linguagem simples até uma de alto nível.

REFERÊNCIAS

DAPOIGNY, Richard; BARLATIER, Patrick **Towards a Context Theory for Context-aware systems**, Advances in Ambient Intelligence, Augusto, J. C.; Shapiro, D., IOS, Press 2007.

ADOLFO, L. A. **Para Computação Ubíqua, pela computação pervasiva**, 2008. Disponível em: <http://www.andrelemos.info/midialocativa/2008/11/para-computao-ubqua-pela-computao.html>. Acesso em: 20 de agosto de 2010.

VÁSQUEZ, T. **Tecnologia - O que é computação pervasiva**, 2010. Disponível em: <http://www.tomasvasquez.com.br/blog/tecnologia/tecnologia-o-que-e-computacao-pervasiva>. Acesso em: 20 de agosto de 2010.

DOMINGUES, F. L. **Computação Ubíqua**, 2008. Disponível em: <http://www.guiadohardware.net/artigos/computacao-ubiqua>. Acesso em: 20 de agosto de 2010.

CHEN, Guanling; KOTZ, David. **A Survey of Context_aware Mobile Computing Research**. (technical Report TR2000-381). Dartmouth Computer Science. Disponível em: <http://www.cs.dartmouth.edu/~solar/>. Acesso em: 10 Outubro 2010.

DEY K. e ABOWD G. D. **Towards a better understanding of context and context awareness**. In Workshop on The What, Who, Where, When and How of Context-Awareness, 2000.

PIERCE, B. C. **Types and Programming Languages**, MIT Press, 2002.

WIKIPÉDIA 2010. **Context-aware pervasive systems**. Disponível em: <http://en.wikipedia.org/wiki/Context-aware_pervasive_systems>. Acesso em: 20 de agosto de 2010.

WIKIPÉDIA 2010. **Context-awareness**. Disponível em: <http://en.wikipedia.org/wiki/Context_awareness>. Acesso em: 20 de agosto de 2010.

WIKIPÉDIA 2010. **Type System**. Disponível em: <http://en.wikipedia.org/wiki/Type_system>. Acesso em: 20 de agosto de 2010.

STRANG, Thomas; LINNHOFF-POPIEN, Claudia. **A Context Modeling Survey**, 2003.

CARDELLI, Luca. **Type Systems**, 2004.

RIES, Luis Henrique **Estudo de Arquiteturas para Computação Pervasiva**, 2005.

AUGUSTIN, Iara; YAMIN, Adenauer; DA SILVA, Luciano Cavelheiro; REAL, Rodrigo; GEYER, Claudio Resin. **ISAMadapt - um Ambiente de Desenvolvimento de Aplicações para a Computação Pervasiva**, 2004.

AUGUSTO, J. C.; SHAPIRO, D. **Advances in Ambient Intelligence**, 2007.

GIUNCHIGLIA, F. **Contextual Reasoning**, Instituto per la Ricerca Scientifica e Technologica, report 9211-20, 1992.

VALENTINI, S. **Decidability in Intuitionistic Type Theory is functionally decidable**, Mathematical Logic, 42, 300–304, 1996.

HOWARD, W. A.; SELDIN, J.P. e HINDLEY, J.R. **To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism**, The formulae-as-types notion of construction, Academic Press, 479–490, 1980.

DOCKHORN-COSTA, P.; ALMEIDA, J.P. A.; PIRES, L. F.; GUIZZARDI, G. e VAN SINDEREN, M. **Towards Conceptual Foundations for Context-Aware Applications**, Procs. of the AAAI'06 Workshop on Modeling and Retrieval of Context, AAAI Press, 54–58, 2006.

MATHEUS, C.J.; KOKAR, M.M. e BACLAWSKI, K. **A core Ontology for Situation Awareness**, Procs. of the Sixth International Conference of Information Fusion , 1, 545–552, 2003.

CHEN, H.; FININ, T. e JOSHI, A. **Using OWL in a Pervasive Computing Broker**, Procs. of Workshop on Ontologies in Open Agent Systems (AAMAS'03), 2003.

HELD, A.; BUSCHOLZ, S. e SCHILL, A. **Modeling of Context Information for Pervasive Computing Applications**, Procs. of the 6th conf. on Systemics, Cybernetics and Informatics (SCI), 2002.

BARWISE, J. **Scenes and other Situations**, Journal of Philosophy, 77, 369–397, 1981.

EUZENAT, J.; RAMPARANY, F. e PIERSON, J. **Gestion dynamique de contexte pour l'informatique pervasive**, Procs of the 15th French Conference on Pattern Recognition and Artificial Intelligence (RFIA'06), Tours (France), 113–123, 2006.

BRÉZILLON, P. e POMEROL, J.-Ch. **Modeling and using context for system development: Lessons from experiences**, Journal of Decision Systems, P. Humphreys e P. Brézillon Eds, Decision Systems in Actions, 10(2), 265–288, 2001.

MÜHLHÄUSER, Max; Gurevych, Iryna **Introduction to Ubiquitous Computing**. Disponível em: <https://www.igi-online.com/downloads/excerpts/8399.pdf>. Acesso em: 21 de agosto de 2010.

BOLDINI, P. **Formalizing Context in Intuitionistic Type theory**, Fundamenta Informaticae, 42,1–23, 2000.

BOVE, A. e CAPRETTA, V. **Nested General Recursion and Partiality in Type Theory**, TPHOL, R.J. Boulton and P.B. Jackson eds., LNCS 2152, Springer, 121–135, 2001.

COOPER, R. **Mixing Situation Theory and Type Theory to Formalize Information States in Dialogue Exchanges**, in Procs of TWLT 13/Twendial ’98: Formal Semantics and Pragmatics of Dialogue, 1998.

COQUAND, C. e COQUAND, T. **Structured type theory**, Workshop on Logical Frameworks and Metalanguages, 1999.

MONTAGUE, R. **Pragmatics and intensional logic**, Synthèse, 22, 68–94, 1970.

MARTIN-LOF, P. **Constructive Mathematics and Computer Programming**, *Logic, Methodology and Philosophyof Sciences* , 6, 153–175, 1982.

RANTA, A. **Grammatical Framework: A Type-Theoretical Grammar Formalism**, *Journal of Functional Programming*, 14(2), 145–189, 2004.

TURNER, Roy M. e STEVENSON, Robert A.G. **ORCA: An adaptive, context-sensitive reasoner for controlling AUVs**, *Procs. of the 7th Int. Symp. on Unmanned Untethered Submersible Technology* (UUST’91), 423–432, 1991.

KOPYLOV, A. **Dependent Intersection: A New Way of Defining Records in Type Theory**, in Procs. Of the 18th Annual IEEE Symposium on Logic in Computer Science, 86–95, 2003.

BETARTE, G. **Type checking dependent (record) types and subtyping**, *Journal of Functional and Logic Programming*, 10(2), 137–166, 2000.

GINZBURG, J. **Abstraction and Ontology: Questions as Propositional Abstracts in Type Theory with Records**, Journal of Log. and Comput., 15(2), 113–130, 2005.

FENG, Yangyue **A Theory of Dependent Record Types with Structural Subtyping**, Outubro 2010.

WEISER, Mark **The Computer for the 21st Century**,1991.

SURHONE, Lambert M.; TENNOE, Mariam T.; HENSSONOW, Susan F. **Cayenne (programming Language)**, 28 de Setembro de 2010.

WIKIPÉDIA 2010 **Epigram (programming language)**. Disponível em: <http://en.wikipedia.org/wiki/Epigram_%28programming_language%29>. Acesso: 01 de dezembro de 2010.

WIKIPÉDIA 2010 **Situational Calculus**. Disponível em: <http://en.wikipedia.org/wiki/Situation_calculus>. Acesso em: 01 de dezembro de 2010.

WIKIPÉDIA 2010 **Neuro-linguistic programming**. Disponível em: <http://en.wikipedia.org/wiki/Neuro-linguistic_programming#cite_note-1>. Acesso em: 01 de dezembro de 2010.

WIKIPÉDIA 2010 Disponível em:< http://pt.wikipedia.org/wiki/Lisp>. Acesso em: 01 de dezembro de 2010.

GROSS, Donald e HARRIS, Carl M. **Fundamentals of Queuing Theory**. New York: John Wiley & Sons, 1998.

CHURCH, Alonzo **A formulation of the simple theory of types**, The Journal of Symbolic Logic 5(2):56–68, 1940.

WIKIPÉDIA 2010 **Type theory**. Disponível em: <http://en.wikipedia.org/wiki/Type_theory>. Acesso em: 08 de dezembro de 2010.

WIKIPÉDIA 2010 **Kinect**. Disponível em: <http://pt.wikipedia.org/wiki/Kinect>. Acesso em: 14 de dezembro de 2010.