**Rodrigo Campiolo**

# *Aspectos de Modelagem de Ambientes de Computação Ubíqua*

Florianópolis – SC

Junho / 2005

**UNIVERSIDADE FEDERAL DE SANTA CATARINA**
**PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO**

**Rodrigo Campiolo**

# Aspectos de Modelagem de Ambientes de Computação Ubíqua

Dissertação submetida à Universidade Federal de Santa Catarina como parte dos requisitos para a obtenção do grau de Mestre em Ciência da Computação

**Prof. Dr. João Bosco Mangueira Sobral**

Florianópolis, junho de 2005

# ASPECTOS DE MODELAGEM DE AMBIENTES DE COMPUTAÇÃO UBÍQUA

**Rodrigo Campiolo**

Esta Dissertação foi julgada adequada para a obtenção do título de Mestre em Ciência da Computação na Área de Concentração Sistemas de Computação e aprovada em sua forma final pelo Programa de Pós-Graduação em Ciência da Computação.

---

Prof. Dr. Raul Sidnei Wazlawick
Coordenador do PPGCC

Banca Examinadora

---

Prof. Dr. João Bosco Mangueira Sobral
Orientador

---

Prof. Dr. Joni da Silva Fraga

---

Prof. Dr. Bernardo Gonçalves Riso

---

Prof. Dr. Paulo José de Freitas Filho

*Dedico aos que por mérito merecem,*
*por abdicarem parte de suas vidas*
*para dedicarem aos seus filhos.*
*Ofereço com amor e carinho*
*aos meus queridos pais.*

# *Agradecimentos*

Primeiramente, agradeço aos meus pais, Flávio e Maria, e a minha família por todo suporte e apoio durante o tempo que estive longe de casa e pelo carinho nos momentos de proximidade. Muito obrigado pelo incentivo e motivação para vencer os desafios e a distância.

Agradeço ao professor João Bosco pela orientação e por acreditar em minha capacidade para realizar este trabalho. Em especial, agradeço pela sua confiança e amizade. Convivemos na mesma sala durante dois anos em harmonia. Obrigado pelas oportunidades e pelos ensinamentos.

Agradeço ao PPGCC - Programa de Pós-Graduação em Ciência da Computação e à UFSC - Universidade Federal de Santa Catarina por disponibilizarem o curso e a estrutura para o desenvolvimento do trabalho. Agradeço à CAPES pelo apoio financeiro, viabilizando assim, a minha total dedicação ao aprendizado.

Agradeço a Clytia e a Emílio pela amizade, pelas revisões e críticas sobre o trabalho e pela companhia. Agradeço a William e a Benincá, os criciumenses, pela convivência diária, dividindo problemas, alegrias e frustrações. Em especial, pela amizade e parceria adquiridas ao dividir o mesmo teto.

Agradeço pela colaboração do professor Emílio Takase sobre neurociência, a John Barton e a Dominik Heckmann pela atenção e material disponibilizado sobre seus projetos. Agradeço aos professores das disciplinas cursadas, pois em diversos momentos os conhecimentos obtidos foram aplicados nesta dissertação.

Por fim, agradeço a todos os amigos e colegas, próximos ou distantes, que ambientaram e dividiram comigo as felicidades, as tristezas e o cotidiano. Em especial, agradeço a Deus pela força recebida, mesmo sem merecer, em todos os momentos e situações.

*"Faça as coisas o mais simples que puder,*
*porém não as mais simples."*
***Albert Einstein***

# *Resumo*

Computação ubíqua, ou *pervasive computing*, introduz o uso de dispositivos móveis e/ou embutidos em objetos e ambientes, com capacidade de comunicação e computação, interagindo diretamente com o homem. Modelar e simular cenários baseados nessa filosofia implica em sobrepor uma série de desafios, a maioria associada à complexidade e heterogeneidade dos elementos presentes em tais cenários. Este trabalho especifica e discute os requisitos, as características e algumas funções essenciais para modelagem e simulação de ambientes de computação ubíqua, auxiliado por uma linguagem de especificação formal, Object-Z. Para alcançar esse objetivo, foi desenvolvido um estudo minucioso dos elementos, características, desafios, projetos e tecnologias associadas à computação ubíqua. Paralelamente, um estudo sobre os conceitos essenciais de modelagem e projetos relacionados à simulação foram pesquisados e analisados. Baseada nessas pesquisas e análises, a especificação foi desenvolvida e avaliada sob três distintos cenários exemplos de computação ubíqua. Como conseqüência, verifica-se que o modelo provê estruturas para representar, analisar e discutir os aspectos para modelagem e simulação em diversos cenários de computação ubíqua. Através da representação e discussão desses aspectos, é possível não apenas modelar cenários, mas investigar em profundidade, desde as questões básicas até as mais complexas, os desafios, soluções, características e comportamentos nos ambientes.

# *Abstract*

Pervasive computing (also called ubiquitous computing) is a trend toward mobile and/or embedded devices in objects and environments. It provides a great computation power, connection to other devices, and direct interaction with humans. The complexity and heterogeneity of elements is a research challenge to model, and simulate ubiquitous computing environments. Therefore, there is not a simulator to achieve many important aspects of pervasive computing philosophy till now. A detailed study about features and relations in the environments is necessary to reach this aim. This dissertation specifies and argues about requirements, features, relations, and behaviors to model, and to simulate ubiquitous computing environments. A formal specification language (Object-Z) is used to design a specification model to aggregate all these aspects. As a result, the model provides structures to represent, analyze, and discuss several issues about modeling, and simulation in ubiquitous computing scenarios. By the discussion and representation of the aspects, it is possible to model scenarios, and investigate basic and relevant issues about challenges, solutions, features and behaviors in the environments.

# *Sumário*

# *Lista de Figuras*

# *Lista de Siglas*

DARPA – Defense Advanced Research Projects Agency
DAML – DARPA Agent Markup Language
GPS – Global Positioning System
IrDA – Infrared Data Association
ISO – International Organization for Standards
OIL – Ontology Inference Layer
OWL – Web Ontology Language
PDA – Personal Digital Assistant
PIN – Personal Information Number
RDF – Resource Description Framework
RFID – Radio Frequency Identification
UML – Unified Modeling Language
UPnP – Universal Plug and Play
XML – Extensible Markup Language
WLAN – Wireless Local Area Network
WMAN – Wireless Metropolitan Area Network
WPAN – Wireless Personal Area Network
WWAN – Wireless Wide Area Network

# 1 Introdução

Computação ubíqua, ou *pervasive computing*, é um paradigma promissor, e realidade em alguns casos, para tornar a computação pessoal muito mais que a simples interação com microcomputadores, mas para libertá-la das restrições estáticas e pessoais e inseri-la no ambiente do homem, rompendo as barreiras físicas e/ou virtuais. Fornece uma perspectiva interessante para acessar a informação em qualquer local e a qualquer hora, através da criação de ambientes impregnados de dispositivos com capacidades computacionais e de comunicação, interagindo diretamente com o homem. Esses dispositivos, denominados dispositivos ubíquos, têm se tornado mais poderosos e menores, facilitando a sua inserção em diversos tipos de objetos e introduzindo controle, computação e comunicação para acessar e compartilhar informação com um conjunto infinito de outros dispositivos.

Casas e carros inteligentes, ambientes que se adaptam ao comportamento do usuário, dispositivos embutidos em roupas, acessórios, eletrodomésticos ou em qualquer outro objeto, dispositivos portáteis que se comunicam entre si e com o ambiente, sensores espalhados em ambientes coletando informações, reconhecimento de voz, face e gestos são exemplos de tecnologias desse cenário. Para prover e promover todas essas características, computação ubíqua impõe uma busca por soluções em software e hardware. Os sistemas devem ser onipresentes, transparentes, conectados, intuitivos, portáveis, seguros e constantemente disponíveis. Os dispositivos devem ser pequenos, embutidos, econômicos, portáteis e ergonômicos. Os usuários não devem se incomodar ou se distrair com a tecnologia, mas a utilizarem inconscientemente na realização de suas tarefas.

Atingir as características supracitadas requer sobrepor uma série de desafios para a construção dos cenários que suportam a filosofia descrita por computação ubíqua. Satyanarayanan (2001) afirma que muitos desafios são derivados de sistemas distribuídos e computação móvel, tais como tolerância a falha, comunicação remota,

alta disponibilidade, segurança distribuída, redes móveis, acesso a informação móvel e aplicações adaptativas; outros são específicos, tais como uso efetivo de espaços inteligentes, distração mínima do usuário, escalabilidade de localização e mascaramento de condições adversas. Hild (AHMED; THOMPSON, 2004) afirma que são três os principais desafios. Primeiro, a segurança, devido à conectividade de informação. Segundo, a energia, pois, devido às dimensões e interações, os dispositivos despendem mais energia. Terceiro, a otimização em larga escala, devido à grande quantidade de dados monitorados, coletados, armazenados e processados. Há outros problemas relacionados a implicações comportamentais e sociais, privacidade, questões de interface, manutenção dos ambientes e dispositivos, ambientes e protótipos de testes, desenvolvimento de aplicações úteis baseadas nas tecnologias atuais ou novas, entre outros.

Os desafios citados são apenas uma amostra da dificuldade para tornar efetiva a visão de computação ubíqua. Entender, representar e propor soluções para os problemas é uma atividade complexa, pois requer análise minuciosa do ambiente e de suas restrições, criação de protótipos, testes de aplicações e dispositivos, além de avaliação das implicações práticas da solução. Todo esse processo consome um longo tempo para alcançar os requisitos necessários e enfim avaliar o problema e a solução em si. Muitas vezes, ao chegar ao final do processo, são observadas novas questões não previstas ou não consideradas anteriormente. Uma das soluções para auxiliar, acelerar e evitar problemas é o uso de modelagem e simulação. Os modelos abstraem, através de uma representação simplificada do sistema real, as questões importantes a serem investigadas e viabilizam uma análise dos elementos existentes em um cenário. A simulação viabiliza a percepção, medição e avaliação do comportamento do sistema sob diferentes situações.

Barton (2002) justifica o uso de simulação baseado na dependência do ciclo para construção de dispositivos e aplicações. Ele alega que construir hardware adequado depende da adequação da aplicação de computação ubíqua e a aplicação não pode ser desenvolvida sem o hardware adequado. Ubiwise é um projeto de simulador, não terminado, que tenta endereçar esse problema. Heckmann (2005) propõe uma ontologia para modelar partes do mundo real, em especial, o usuário. UbisWorld é o resultado desse modelo. O objetivo é auxiliar a resolução de questões de pesquisa e modelagem de usuários. Também pode ser usado para simulação, inspeção e controle do mundo real. Morla (2004) propõe um novo ambiente para testar e validar questões

relacionadas a rede e a sistemas em aplicações baseadas em localização antes de seu desenvolvimento real. Ele defende que testes e validação complexos e caros podem comprometer ou inibir o desenvolvimento custo-efetivo de aplicações de computação ubíqua.

A modelagem e a simulação de ambientes de computação ubíqua podem ser usados em diversos escopos. Na educação, pois os recursos para construir uma infra-estrutura envolve muitos gastos. Na redução de custos de desenvolvimento em empresas, onde o tempo para fornecer produtos de qualidade são críticos. Na averiguação de compatibilidade, pois averiguar e comparar com uma diversidade de produtos reais é complicado. Em pesquisas, para testes e validação de experimentos, tais como dispositivos, ambientes e protocolos. Na avaliação de qualidade, pois estatísticas de simulação permitem inferir sobre uma quantidade expressiva de variáveis em ambientes. No projeto de novos dispositivos, os quais não podem ser construídos plenamente por impossibilidade de implementação. Na verificação de impactos sociais e comportamentais sobre os usuários. Enfim, em uma série de atividades e projetos em que os modelos reais ainda não existem ou não podem ser aplicados devido às restrições.

Desenvolver um simulador para cenários reais é, no entanto, uma tarefa complexa e envolve um estudo minucioso da área de aplicação. Como exemplo, os simuladores de vôos precisam reproduzir com fidelidade todos os aspectos e situações reais ao qual uma aeronave pode ser exposta. A partir dessa premissa, a construção de um simulador para ambientes de computação ubíqua envolve um estudo aprofundado sobre as características e relações nos ambientes de computação ubíqua. Através da representação e discussão desses aspectos a partir da especificação de um modelo, é possível não apenas modelar ambientes, mas investigar em profundidade, desde as questões básicas até as mais complexas, os desafios, soluções, características e comportamentos nos ambientes.

Considerando a importância de modelar e simular ambientes de computação ubíqua e a dificuldade e inexistência de um simulador genérico, isto é, com a capacidade de representar e simular uma diversidade de ambientes, esse trabalho tem como objetivo especificar e discutir os requisitos, características e algumas funções para modelagem e simulação de ambientes de computação ubíqua. Para alcançar esse objetivo, diversos tópicos relevantes para computação ubíqua são abordados.

Em específico, a conceituação e qualificação da área, a discussão de suas premissas e desafios, a compreensão e representação dos elementos envolvidos nos ambientes, a especificação e representação dos usuários, a discussão dos problemas e soluções para modelar e simular os ambientes, a aplicação e análise da especificação em um ambiente específico.

Para atingir os objetivos utiliza-se uma linguagem de especificação formal para representar as características, relações e associações entre os elementos em ambientes de computação ubíqua. Entretanto, inicialmente deve-se compreender todas as temáticas referentes à composição dos ambientes e como elas se relacionam. Entre as principais temáticas temos a representação dos elementos, a conectividade e as características e desafios em computação ubíqua. Com um conhecimento fundamentado, essas questões são associadas com a teoria de modelagem e simulação. A partir dessa composição, especifica-se um modelo com a linguagem formal de acordo com os requisitos obtidos e avaliados. O conteúdo da especificação é justificado e analisado através de seus requisitos e uma aplicação. Ao final, o trabalho é comparado com trabalhos similares.

Apesar da abrangência do tema abordado, há itens que devem ser explicitados para não originarem discordâncias quanto ao conteúdo e métodos empregados. O formalismo é utilizado apenas para representar e discutir a especificação e não para provar assunções sobre os ambientes. A implementação de um simulador não é um objetivo, logo não são abordadas as técnicas e questões para sua viabilidade. Nem todos os elementos são caracterizados detalhadamente, em alguns casos, são somente consideradas as propriedades mais relevantes. A modelagem se concentra em especificar as características e comportamentos importantes para ressaltar as interações, isto é, as relações entre os elementos.

Este trabalho evidencia duas importantes contribuições. A primeira é a especificação e discussão dos aspectos relevantes para modelagem e simulação de ambientes de computação ubíqua. A segunda é a representação através de uma linguagem formal das características, relações e comportamentos dos elementos envolvidos para o projeto e desenvolvimento de um simulador. Em especial, isso é inovador, visto que não existe um simulador genérico e para implementá-lo é necessário conhecer profundamente o ambiente de aplicação. A especificação, mesmo não provendo um modelo de implementação, colabora para tornar realidade este projeto.

A dissertação está organizada em seis capítulos, entre eles a introdução e conclusões. O capítulo dois apresenta e introduz computação ubíqua. Apresentam-se questões históricas, definições, relações com outras áreas, características, desafios e projetos. Sua principal contribuição, no entanto, refere-se à definição e qualificação das entidades que compõem os ambientes de computação ubíqua e aos meios de comunicação utilizados para prover a conectividade. Também são apresentados os serviços e aplicações que se utilizam da infra-estrutura dos ambientes e um exemplo de cenário para fundamentar os conceitos abordados. A combinação de toda essa fundamentação constitui a fonte primária de informação para a especificação do modelo. O capítulo três aborda as questões referentes à modelagem e simulação através da apresentação dos conceitos e teoria sobre essa área. Neste capítulo é apresentada a linguagem formal Object-Z, utilizada na especificação, e o estudo mais detalhado de dois trabalhos direcionados à modelagem e simulação de ambientes de computação ubíqua. O capítulo quatro apresenta a especificação e a discussão do modelo para representar os elementos, associações e comportamentos nos ambientes de computação ubíqua. Esse capítulo é o principal capítulo da dissertação, pois durante a construção do modelo, os aspectos relevantes para modelagem e simulação são examinados e discutidos no momento de sua especificação. O capítulo cinco é a aplicação, análise e discussão em cenários reais para observar e avaliar alguns detalhes sobre modelagem e simulação não perceptíveis na especificação. São avaliadas também a especificação e sua comparação com outros trabalhos. O capítulo seis apresenta as conclusões, as expectativas e os projetos futuros.

# 2 Computação Ubíqua

Não é de hoje que se discute sobre sociedades que convivem com dispositivos integrados ao ambiente, os quais se comunicam com dispositivos móveis e fornecem uma série de facilidades para a comodidade do homem. Os escritores, os cineastas e os filósofos sempre adoraram abordar esse assunto, tratando de cenários futurísticos, onde dispositivos colaboram com a sociedade, possuem inteligência similar à humana, ou no mínimo interagem através da fala, entendem o contexto das situações e executam ações que vão além das capacidades de qualquer pessoa.

A ciência ainda está muito longe de alcançar tais perspectivas, mas há muito trabalho sendo desenvolvido para tornar essa visão realidade. Este capítulo apresenta um breve histórico, definições, elementos, características e desafios em computação ubíqua. Ao final, são apresentadas aplicações e projetos que contemplam elementos e diversas características desejáveis em ambientes de computação ubíqua.

## 2.1 Histórico

Computação ubíqua teve seu início em meados de 1987, nos laboratórios de eletrônica e imagem da Xerox, no centro de pesquisa de Palo Alto (WEISER; GOLD; BROWN, 1999). O termo utilizado pelos pesquisadores para referir-se a essa nova área era *Ubiquitous Computing*. A visão dos pesquisadores se distanciava do tradicional esquema: uma pessoa, um computador pessoal. O objetivo da equipe era distribuir dispositivos com capacidades computacionais pelo ambiente, mas sem que a presença destes fosse notada, daí o termo Computação Invisível.

A partir desta pesquisa, diversos laboratórios resolveram participar, inclusive de outras áreas como Antropologia, responsável por observar e analisar os impactos no comportamento das pessoas pela introdução de dispositivos eletrônicos em

seu ambiente. Todo esse interesse resultou, em 1988, na criação do programa de Computação Ubíqua do Laboratório de Computação (CSL)[1].

Em 1991, Mark Weiser, pesquisador da Xerox e considerado o idealizador da computação ubíqua, publicou na *Scientific American* o primeiro artigo que abordava o assunto, intitulado *The Computer for the 21th Century* (WEISER, 1991). Neste artigo, são discutidos três tipos de dispositivos: tabs - dispositivos pequenos (ex.: cartão), pads - dispositivos medianos (ex.: palmtops), board - dispositivos grandes (ex.: telas ou letreiros digitais). Contudo, a principal contribuição do seu artigo foi qualificar como deveria ser implementada e qual a expectativa com relação à computação ubíqua.

No ano de 1993, Weiser publicou, com enfoque mais científico, o artigo *Some Computer Science Issues in Ubiquitous Computing* (WEISER, 1993) em *Communications of ACM*. Este artigo discute questões relevantes de computação ubíqua. São abordadas questões de hardware, rede, aplicações, privacidade, métodos computacionais e interação.

Apesar destas publicações e de outras pesquisas, o assunto não se destacou durante a década de 90. O enfoque das pesquisas nessa década foram os sistemas distribuídos, principalmente impulsionados pela Internet e pela necessidade das grandes corporações. Porém, a computação ubíqua começou a engatinhar, pois muitas das limitações, principalmente referentes a questões de comunicação, foram vencidas em virtude das pesquisas em sistemas de telecomunicações e sistemas distribuídos.

A partir do final da década passada, o assunto emergiu novamente, impulsionado pela computação móvel e pela evolução dos dispositivos eletrônicos, cada vez menores e com capacidades computacionais mais elevadas. Assim, poderiam ser transportados, embutidos em eletrodomésticos, ou até mesmo implantados sob a pele.

Um passo importante para a evolução da área foi o surgimento de eventos científicos específicos, como por exemplo, o *Ubicomp - ACM*[2], abordando o assunto não mais com ceticismo e como pura ficção, e o lançamento de revistas específicas, destacando-se a *IEEE Pervasive Computing*[3].

[1] http://www.parc.com/research/csl/
[2] http://ubicomp.org
[3] http://www.computer.org/pervasive

Um artigo importante para a formalização da área é *Pervasive Computing: Vision and Challenges* (SATYANARAYANAN, 2001) que introduz definições, campos relacionados, desafios e linhas de pesquisas na área.

Sumariando, ao final do século 20 e começo do novo século, muitos grupos de pesquisa trabalham com a área, tornando impossível atribuir os méritos a todos os projetos que estão sendo desenvolvidos.

## 2.2 Definições

Há muitas definições de computação ubíqua, as quais nem sempre contemplam toda a abrangência do seu significado. Por exemplo, os termos *pervasive* e *computing* não traduzem realmente as potencialidades dessa área. Em português, computação permeada, impregnante ou simplesmente, para os que querem transformá-la corriqueira usando um neologismo, computação pervasiva. Computação ubíqua, utilizada como sinônimo, significa computação onipresente.

Não há um consenso quanto à utilização de *ubiquitous computing* como sinônimo de *pervasive computing*. Na maioria dos artigos, os termos aparecem como sinônimos, porém há alguns pesquisadores que os utilizam com significados diferentes. Nestes casos, *pervasive computing* é usado para expressar computação embutida, móvel e invisível enquanto *ubiquitous computing* é usado para expressar computação onipresente, no sentido de acessar recursos em qualquer hora e lugar.

Um exemplo mais heterogêneo quanto à nomenclatura pode ser encontrado em Lyytinen (2002), onde são diferenciados computação móvel, *pervasive computing* e computação ubíqua. Lyytinen afirma que os termos são indevidamente usados em contextos iguais, mas, na verdade, seus significados são conceitualmente diferentes e empregam diferentes idéias de organização e serviço. Para Lyytinen, computação móvel é a capacidade de mover fisicamente conosco serviços de computação; *pervasive computing* é fornecer aos dispositivos a capacidade de obter informação do ambiente em que estão embutidos, utilizá-la para dinamicamente construir modelos de computação e fornecer ao ambiente a capacidade de detectar e se auto-configurar para esses dispositivos; computação ubíqua é a junção de *pervasive computing* e um grau elevado de mobilidade.

Segundo a visão de Weiser (WEISER, 1991, 1993), computação ubíqua é a criação

de ambientes impregnados de dispositivos, com capacidade de computação e comunicação, os quais devem apresentar-se de maneira invisível ao usuário. Ele afirmava que as tecnologias que mais se destacam são aquelas que desaparecem ao tornar-se parte do cotidiano, impossibilitando notar sua presença.

Os cientistas da IBM definem *pervasive computing* como um acesso conveniente a informações relevantes, através de uma nova classe de ferramentas e aplicações com a habilidade para facilmente realizar ações sobre elas quando e onde for necessário (HANSMANN et al., 2003).

Em nosso estudo consideramos que *pervasive computing* e *ubiquitous computing* (computação ubíqua) representam as mesmas idéias. Tal consideração é feita com base na nomenclatura aceita pelo IEEE, conforme pode ser conferido em Satyanarayanan (SATYANARAYANAN, 2001).

Particularmente, pode-se dizer que computação ubíqua é a evolução da computação centrada no microcomputador para a computação em qualquer hora e lugar, utilizando como suporte, além dos microcomputadores, uma legião de novos dispositivos e tecnologias emergentes. Em outras palavras, é a computação onipresente, acessível em qualquer ambiente; é a computação através da comunicação entre dispositivos, usuários e ambientes; e, é a computação que simplifica a realização das tarefas do usuário.

## 2.3 Entidades

Entidade é uma palavra de origem latina que significa algo que constitui a essência de uma coisa, um ser, um grupo que dirige as atividades de uma classe, ou ainda, tudo quanto existe ou pode existir (FERREIRA, 2004).

No presente contexto, o termo entidade é utilizado para referenciar todas as instâncias que de alguma forma colaboram para a formação e definição de um ambiente de computação ubíqua. Entidades podem ser pessoas, dispositivos e softwares.

Além das entidades, os ambientes de computação ubíqua são compostos pelos meios de comunicação, responsáveis pela conectividade, e por objetos sem propriedades computacionais, os quais interferem diretamente no comportamento das entidades nos ambientes ubíquos.

Durante as interações nos ambientes, dependendo do nível de abstração, há situações em que entidades podem estar desempenhando simultaneamente o mesmo papel. Por exemplo, no caso de uma operação com um palmtop. Neste caso, não é possível identificar unicamente qual entidade está interagindo. Pode ser o dispositivo, o software ou a pessoa que requisitou a operação.

As entidades desempenham os principais papéis nos cenários de computação ubíqua. Para ilustrar sua importância, nesta seção são apresentados uma nomenclatura, uma classificação e os papéis dessas entidades.

### 2.3.1 Pessoas

A participação do ser humano em ambientes de computação ubíqua desempenha um dos principais aspectos para a definição desses ambientes. Entender e representar sua participação é uma atividade complexa, pois não há uma classificação e nem uma nomenclatura para definir pessoas; relevantes são os comportamentos e as relações com as outras entidades.

Uma pessoa relaciona-se com outras entidades através de interfaces. As interfaces são responsáveis por fornecer os meios adequados para que essa interação seja o mais transparente possível. Outro requisito é a qualidade. Se as relações não atendem as expectativas, as pessoas podem interferir diretamente no funcionamento do sistema ou no seu desuso.

O comportamento define como as pessoas podem reagir a determinadas circunstâncias. Por exemplo, uma pessoa com domínio de informática aceita e aprende com maior facilidade um sistema desconhecido, enquanto outras podem simplesmente desistir ou usar o sistema incorretamente.

Pessoas são seres autônomos e móveis. Essas características implicam diretamente em diversos aspectos no ambiente de computação ubíqua. A autonomia influencia nas relações, pois as outras entidades têm regras bem definidas de funcionamento. Logo, se uma relação é iniciada, nem sempre pode ser interrompida, ou caso seja, pode levar a resultados não esperados. A mobilidade afeta a comunicação, pois pessoas em movimento, carregando seus dispositivos, podem interferir ou aumentar a complexidade das comunicações que estão sendo realizadas.

### 2.3.2 Dispositivos

Dispositivos com capacidade de comunicação e/ou computação são, com certeza, os alicerces para a existência e a crescente ascensão da computação ubíqua. São referenciados como dispositivos ubíquos ou *pervasive devices*.

Os primeiros produtos que apareceram com essas características eram muito simples, por exemplo, os primeiros celulares. Atualmente, a quantidade e sofisticação desses dispositivos é superior aos primeiros produtos comercializados. Mas, o mais inacreditável são as promessas que ainda não são produtos e podem vir a se tornar realidade nos próximos anos.

Devido a grande variabilidade desses dispositivos ubíquos, torna-se difícil descrever separadamente as características de cada um. Para tal, é proposta uma divisão em três grandes grupos, os quais ainda são subdivididos em categorias mais específicas, como pode ser observado na figura 2.1.

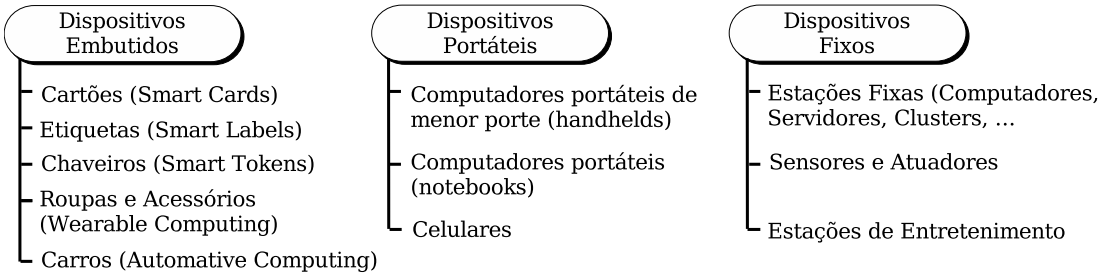


Figura 2.1: Classificação em categorias dos dispositivos

Dispositivos embutidos são dispositivos de hardware inseridos dentro de um objeto, os quais introduzem a capacidade de computação e comunicação. Nesses dispositivos existe um sistema que realiza todo o controle, por isso geralmente na literatura essa área é designada como sistemas embutidos. Os sistemas embutidos são programados dentro desses dispositivos e dependendo de sua sofisticação possuem diversas características que se aproximam de sistemas computacionais tradicionais. Uma outra característica é a possibilidade de comunicação e de inserção de um certo grau de autonomia e inteligência nesses dispositivos, conduzindo para o que é proposto por computação ubíqua. O termo dispositivo embutido implicitamente engloba em seu significado a existência um dispositivo inserido dentro de um objeto e que este dispositivo tem um sistema embutido que o controla. Por exemplo, o forno de microondas possui um microcontrolador embutido com um software de controle.

O software fornece a interface das funções do microondas e gerencia a execução das tarefas.

Dispositivos portáteis são dispositivos que por sua natureza são projetados para realizar tarefas computacionais e podem ser transportados pelo usuário. Esses dispositivos fornecem aos seus usuários poder de computação superior aos dispositivos embutidos e uma variedade maior de funções e possibilidade de expansão. Os dispositivos portáteis são classificados como dispositivos que possuem um porte maior, uma interface com mais funções, um sistema operacional mais sofisticado e capacidades computacionais para processamentos mais complexos.

Dispositivos fixos são os computadores, servidores, clusters que fornecem o mais alto grau de computação para seus usuários. Esses dispositivos são estações fixas, devido ao seu tamanho e a necessidade de alimentação de energia constante. Desempenham papéis importantes como auxiliares para computação e comunicação entre dispositivos menores. Além dos computadores, todo dispositivo que se apresenta em posições fixas no ambiente e não se encaixa nas outras categorias faz parte desse grupo, como é o caso dos sensores e atuadores.

### Cartões (Smart Cards)

*Smart cards* são cartões com microprocessador ou memória embutidos que, junto com um leitor, podem ser utilizados para diferentes tipos de aplicações. Podem ser empregados para guardar informações críticas, como por exemplo, chaves criptográficas; ser utilizado como carteira digital, ou seja, armazenar dinheiro e realizar transações financeiras; fornecer serviços de autenticação e assim por diante.

Podem ser de dois tipos: cartão memória, onde sua única função é armazenar dados e fornecer uma segurança opcional, ou cartão processador, capaz de manipular uma memória interna e realizar diferentes tipos de processamento, tais como operações criptográficas. O software dos cartões é dividido em duas partes: on-card, processado pelo cartão (ex.: cifrar ou decifrar um dado) e off-card, que atua somente quando o cartão está conectado ao leitor. A comunicação com o leitor pode ser por contato ou sem contato.

### Etiquetas (Smart Labels)

Etiquetas são dispositivos que utilizam identificação por freqüência de rádio (RFID), tipicamente de 125kHz, 13,56 MHz ou 800-900 MHz, para rotular unicamente um objeto. São utilizadas para controle de estoque, identificação e localização de pacotes, liberação de produtos, identificação de pessoas.

São constituídas por um microcircuito que contém uma antena de rádio freqüência conectada. São envolvidas por um plástico e sua espessura é bastante fina. Podem ser ativas (com bateria) ou passivas (sem bateria). As passivas não necessitam de fornecimento de energia elétrica; obtêm a energia do campo de rádio freqüência emitido pelo dispositivo de leitura.

### Chaveiros (Smart Tokens)

Os chaveiros possuem a mesma forma de atuação dos cartões, contudo são encapsulados diferentemente para atender a aplicações específicas que exigem um modelo mais adequado para transportar ou usar o dispositivo. Outra vantagem é a possibilidade de utilizar materiais mais resistentes para proteger o equipamento, como plástico ou metal.

### Roupas e acessórios (Wearable Computing)

*Wearable computing* é a computação inserida no espaço pessoal do usuário, controlada e sempre acessível ao usuário (MANN, 1998). Mais especificamente, são dispositivos constantemente presentes com o usuário, seja em suas roupas (calças, jaquetas, sapatos) ou acessórios pessoais, como por exemplo, relógios, óculos, brincos, entre outros. A vantagem de usar dispositivos carregados no corpo é explorar os princípios de computação ubíqua imediatamente, ao invés de esperar o desenvolvimento da infra-estrutura freqüentemente associada ao campo (STARNER, 2002).

*Wearable computing* tem o objetivo de fornecer ao usuário disponibilidade, isto é, fornecer acesso constante e imediato; fornecer o aumento da realidade, ou seja, tornar a computação tarefa secundária para auxiliar nas necessidades locais e físicas do usuário; criar uma sinergia entre o usuário e o dispositivo, isto é, o usuário executa a tarefa que melhor desempenha enquanto o computador executa a tarefa a qual está mais apto. Um exemplo sofisticado do uso de roupas e acessórios é como

assistente digital pessoal (PDA) (STARNER et al., 1997).

As tecnologias nesse campo podem ser classificadas em duas categorias: *Smart Clothing*, refere-se aos dispositivos que podem ser usados no corpo e *Wearable Applications*, refere-se especificamente aos sistemas de navegação e interação embutidos nesses dispositivos (BILLINGHURST, 2002).

### Carros (Automotive Computing)

O termo *automotive computing* é atribuído a inserção de sistemas eletrônicos e computacionais nos automóveis. Sensores de vários tipos detectam os mais variados parâmetros: pressão, pneus, movimento, direção, posição e segurança dos passageiros. Há dispositivos para fornecer serviços sofisticados, tais como serviços de localização baseados em GPS, controle de equipamentos através de painéis ou voz, sistemas de entretenimento e comunicação, entre outros.

A inserção de tanta tecnologia introduziu novos conceitos relacionados à computação em automóvel (MOTOROLA, 2004):

- Unidades de controle: são responsáveis por controlar as funções elétricas, estabilidade e transmissão dos sinais pelo barramento.
- Sistemas de navegação: permite ao motorista obter informação em tempo real sobre condições do tráfego, estradas e outras informações de interesse.
- Telemática: são dispositivos de comunicação sem fio projetados para automóveis que fornecem aos motoristas informações personalizadas, serviços de mensagem, entretenimento, informações sobre a viagem e serviços de segurança.
- Auto barramento: permite em uma rede de veículos a troca de informações, dados e controle de mensagens.

### Objetos e eletrodomésticos (Smart Appliances)

São objetos e eletrodomésticos que possuem, além dos tradicionais processadores e sistemas, a capacidade de se comunicar com outras entidades no ambiente e tomar decisões autônomas. São exemplos geladeiras que informam ao usuário a

falta de algum produto básico, relógios conectados a Internet e que se adaptam ao comportamento do usuário, acionamento automático de objetos (ex.: lâmpadas, cafeteira).

Esses objetos e eletrodomésticos são geralmente associados a casas inteligentes, isto é, casas que possuem um sistema inteligente responsável por gerenciar os dispositivos elétricos na casa. Contudo, esses objetos podem estar presentes no dia-a-dia, como por exemplo, no escritório ou em ambientes públicos.

### Computadores portáteis de menor porte

São todos os dispositivos que possuem capacidade de computação e comunicação e que podem ser transportados. Além disso, possuem tamanho equivalente ao de uma mão humana e interfaces mais avançadas e, utilizam baterias. Estão inclusos nesse grupo os handhelds (palm, pocketpc, entre outros), câmeras digitais, assistentes digitais pessoais, equipamentos de entretenimento (MP3 player, mini video games).

Estes dispositivos permitem maior interação entre o usuário e os ambientes de computação ubíqua, pois além de possuírem sistemas operacionais mais complexos, possuem diversos tipos de comunicação sem fio, como por exemplo, infravermelho e rádio. Devido à mobilidade, os dispositivos estão sempre ao alcance de diferentes sensores e outros dispositivos espalhados no ambiente, o que lhes permite obter muitas informações sobre o contexto local.

### Computadores portáteis

São dispositivos que possuem capacidades computacionais (disco, memória e processador) e arquiteturas praticamente idênticas aos microcomputadores, podem ser transportados, utilizam bateria e possuem tamanho maior que os computadores de mão. Como exemplo, os notebooks e laptops.

### Celulares

A comunicação móvel é o principal atrativo da telefonia celular. Com o avanço das tecnologias, os celulares tornaram-se verdadeiros computadores portáteis. A

quantidade de opções oferecidas por celulares vai desde visores coloridos a sofisticados sistemas para entretenimento com câmera, vídeo e música. Os celulares acessam a Internet, realizam computação e são portáteis. Duas vantagems são seu sistema próprio e global de comunicação, e os serviços de mensagens, que podem ser usados para controle de objetos remotamente.

### Estações fixas

Correspondem aos dispositivos de maior porte e que necessitam estar conectados a algum tipo de fonte de alimentação constante. Enquadram-se nessa categoria os computadores, servidores e clusters. Estes dispositivos atuam para dar o suporte necessário aos dispositivos menores e aos ambientes de computação ubíqua.

### Estações de entretenimento

São dispositivos de computação ubíqua de grande porte dedicados ao entretenimento, tais como aparelho de som, aparelho de DVD, televisão, video game, entre outros. Todos esses equipamentos, além de atender a função específica a que se destinam, estão equipados com dispositivos que permitem conexão em rede, controle via voz, processamento de conteúdo digital e muito mais.

Devido às características citadas, um novo mercado de aplicações está surgindo para atender o mercado de entretenimento. Entre as novidades estão TV digital, video e som em demanda, navegação avançada na Internet e controle de outros equipamentos conectados.

### Sensores e atuadores

Sensores são dispositivos que respondem a estímulos físicos (calor, luz, som, imagem, pressão, magnetismo, movimento, entre outros) e transmitem um impulso resultante. São exemplos os sensores de pressão, sensores de temperatura, sensores de posição, sensores de luminosidade, alto-falantes, sensores de pH, sensores de gás, câmeras.

Atuadores são dispositivos mecânicos que executam ações sobre o mundo físico. São exemplos de atuadores os motores, válvulas, chaves, acionadores elétricos (ligar/desligar, travar/destravar).

Os sensores permitem aos dispositivos de computação ubíqua obter uma percepção do mundo e os atuadores, modificar o mundo baseado nestas percepções. Os sensores e atuadores se encontram fixos nos dispositivos e ambientes, ou seja, só se movem se o dispositivo for móvel. Eles podem ser considerados, respectivamente, como os sentidos e os membros para a imersão dos computadores nos ambientes físicos.

Para promover essa imersão, um conceito que vem se expandindo são as redes de sensores, em especial, as redes de sensores sem fio (*wireless sensor neworks*). As redes de sensores combinam processamento, percepção e comunicação embutidos em dispositivos muito pequenos, os quais usam um protocolo de comunicação ponto-a-ponto, onde os dados são roteados entre todos os nós (HILL et al., 2004). Estas redes permitem observar e interagir com o meio físico em tempo real e podem ser usadas para uma variedade de aplicações, tais como dinâmica de ecossistemas, proteção de propriedades, operação e gerenciamento de máquinas, segurança de perímetros e construções, proteção de pacotes e containers, monitoramento, sistemas de observação médica, entre outros (CULLER; HONG, 2004).

### 2.3.3 Softwares

No topo da estrutura para suportar ambientes de computação ubíqua se encontram os softwares. Eles provêem os mecanismos para programação e funcionalidade dos dispositivos nesses cenários. Podem ser classificados em sistemas operacionais, ambientes e aplicações. Os sistemas operacionais fornecem a estrutura para acesso ao hardware dos dispositivos; os ambientes são as camadas intermediárias que fornecem estrutura para a programação; e as aplicações fornecem os serviços para o usuário.

Infelizmente, construir software para atender aos requisitos dos ambientes de computação ubíqua não é uma tarefa fácil. Há muitas restrições que devem ser respeitadas, entre elas a capacidade computacional dos dispositivos, ambientes dinâmicos, vivacidade, entre outros.

A capacidade computacional restrita exige que os sistemas, ambientes e aplicações sejam pequenos e otimizados, o que limita muito a complexidade das aplicações e o uso de interfaces avançadas. Ambientes dinâmicos implicam em interoperabi-

lidade e adaptação a mudanças, o que pode ser alcançado com o uso de camadas intermediárias, como por exemplo, uma máquina virtual. A vivacidade implica que o software deve estar sempre disponível, ou seja, não pode parar ou falhar.

Nigel Davies (2002) aponta em seu trabalho alguns outros desafios para o desenvolvimento de sistemas de computação ubíqua:

- Interação de componentes: componentes e plataformas devem ser projetados para interagir com os diferentes componentes existentes em ambientes saturados com dispositivos.
- Sensibilidade ao contexto e adaptação: os ambientes são altamente mutáveis, logo as aplicações e os componentes devem ser projetados para pressentir e se adaptar a essas mudanças.
- Gerenciamento apropriado de mecanismos e políticas: como o número de componentes é alto e diferenciado, o gerenciamento torna-se problemático. Há a necessidade de estabelecer políticas e interfaces padrões para gerenciar essa quantidade expressiva de componentes heterogêneos.
- Associação de componentes e análises de tarefas: determinar as intenções do usuário e implementar associações entre os componentes para ajudá-lo a alcançar seu objetivo.
- Modelos economicamente viáveis e infra-estrutura de suporte: usuários podem até pagar para viver em um mundo com computação ubíqua, mas poucos estarão dispostos a pagar para ter acesso a determinados componentes ou serviços.
- Interface com o usuário: como há diversas aplicações simultâneas no ambiente, definir e apresentar a interface ativa ao usuário é um problema, conseqüentemente há a necessidade de integração e negociação dessas interfaces.
- Soluções técnicas, legais e sociais para privacidade e segurança: as legislações não abordam questões de privacidade no contexto da computação ubíqua, nem o usuário é consciente da importância de suas informações. Logo, medidas devem ser implementadas para endereçar essa situação.

Existem diversos projetos que tentam abordar essas questões, os quais são discutidos mais adiante. Um exemplo de ambiente distribuído que oferece serviços e meios para o desenvolvimento de aplicações em computação ubíqua é o JINI (EDWARDS, 1999; SUN, 2003).

## 2.4 Meios de Comunicação

Os meios de comunicação são responsáveis por estabelecer a conectividade entre todas as entidades que compõem os ambientes de computação ubíqua. Basicamente, são divididos em duas categorias: por cabo e sem fio (TANENBAUM, 2003; HANSMANN et al., 2003).

Neste tópico são enfatizados os meios de comunicação sem fio, como a comunicação por rádio, infravermelho, satélite, laser, campos elétricos e magnéticos. Além desses tipos de comunicação é abordada a comunicação utilizando a rede elétrica e telefônica.

### 2.4.1 Satélites

Satélites de comunicação são repetidores de sinais, ou seja, um sinal na forma de microondas é enviado, amplificado e retornado pelo satélite. Podem ser usados para transmitir qualquer tipo de informação, desde que possa ser configurada na forma de ondas. A área de cobertura do satélite é um dos seus principais atrativos, pois podem ser utilizados para cobrir uma área extensa específica ou quase a superfície terrestre inteira.

O satélite é um equipamento composto por uma série de transponders que atuam como repetidores de sinal. Eles recebem os sinais das estações terrestres em determinadas freqüências, amplificam e enviam novamente através de difusão em uma freqüência diferente para não causar interferência com a freqüência de entrada. Devido ao seu porte, há um grande consumo de energia, o qual é garantido pelos painéis solares. A altitude define o período e a área de cobertura de um satélite. Há três regiões onde eles podem ser posicionados considerando essas questões e a segurança devido às partículas carregadas emitidas pelo campo magnético terrestre.

Os satélites conhecidos como geoestacionários tem uma órbita de 24 horas e estão

posicionados a 35.800 km de altitude sobre o equador. Devido a essa característica parecem estar imóveis com relação à superfície terrestre, o que facilita posicionar as antenas das estações terrestres. Podem ser alocados 180 satélites nessa região, pois há a necessidade de espaçamento mínimo entre eles para não ocasionar interferências. Os satélites de órbita média estão posicionados a altitudes que variam de 5.000 a 15.000 km, logo para uma cobertura global são necessários no mínimo 10 satélites. As antenas das estações terrestres devem acompanhar a sua órbita. Os satélites de órbita baixa estão posicionados próximos à superfície, logo são necessários muitos para uma cobertura global. A vantagem é o baixo custo para transmissões e velocidade de retorno, devido a sua proximidade das antenas.

Há muitas aplicações em computação ubíqua onde o uso dos satélites é mais adequado ou é a única solução para a realização da comunicação. Por exemplo, as aplicações que precisam sincronizar algum serviço fornecido para seus clientes podem usar a vantagem da difusão fornecida pelos satélites. Outra aplicação é a transmissão de informações específicas sem a dependência do sistema telefônico. A aplicação que mais se destaca é o GPS (*Global Positioning System*).

### 2.4.2 Rádio

A freqüência de rádio tem muitas vantagens entre os meios de comunicação sem fio. Seus benefícios são a facilidade de geração dos sinais, o alcance da transmissão, o não bloqueio do sinal por construções e a omnidirecionalidade, ou seja, as ondas são propagadas em todas as direções. Há também diversos problemas, como por exemplo, a interferência, a reflexão e a absorção dos sinais. Nas freqüências baixas e médias, as ondas de rádio trafegam próximas ao solo e atravessam objetos sólidos, porém a potência do sinal e a largura de banda são reduzidas. Nas freqüências altas, as ondas de rádio próximas ao solo são absorvidas, enquanto as mais altas são refletidas pela ionosfera, permitindo um longo alcance.

Apesar dos problemas quanto ao uso da freqüência de rádio, elas são interessantes para formar qualquer tipo de rede sem fio. Podem ser usadas para construir redes pessoais (WPANs), redes locais (WLANs), redes metropolitanas (WMANs) e redes globais (WWANs).

As WPANs são redes sem fio utilizadas para interconectar dispositivos que se

encontram no espaço do usuário. Geralmente, uma WPAN usa tecnologias tais como Bluetooth[4] e IrDA[5], as quais permitem comunicações em poucos metros, aproximadamente 10 metros de distância. O padrão IEEE 802.15, originalmente baseado na arquitetura Bluetooth, define os padrões para conectar dispositivos nessas redes. As bandas utilizadas são 2.4 GHz e inferiores, ou seja, bandas não regulamentadas. Devido às restrições dos dispositivos envolvidos para a formação dessas redes, características como baixo consumo de energia, tamanho reduzido e custos são considerados por essa padronização. Conseqüentemente, essas redes são essenciais e ideais para a comunicação em ambientes de computação ubíqua.

As WLANs são redes sem fio que possuem uma área de cobertura limitada a uma área local ou a um espaço específico. Nessas redes, os dispositivos podem mover-se dentro e entre áreas de cobertura sem que a conexão seja interrompida. O padrão IEEE 802.11 define os padrões para a formação dessas redes. Esse padrão inclui dois tipos de redes locais sem fio: redes ad hoc e redes de infra-estrutura. As redes ad hoc ou redes espontâneas são redes onde não existe uma estrutura fixa para estabelecer a comunicação entre as máquinas que a compõem, ou seja, as próprias máquinas roteiam os pacotes (WU; STOJMENOVIC, 2004). As redes de infra-estrutura são redes que conectam estações móveis aos pontos de acessos, os quais possuem antenas para enviar e receber os sinais e uma conexão Ethernet.

As WMANs são redes sem fio para oferecer acesso de banda larga para uma área geográfica extensa, sem os altos custos envolvidos na criação de uma infra-estrutura com ligações por cabo, além de permitir um acesso mais espontâneo e onipresente (EKLUND et al., 2002). O padrão 802.16 especifica e define os padrões para estas redes. Neste padrão, uma WMAN fornece acesso de rede para construções através de antenas externas de comunicação com estações base centrais. Desse modo, um usuário em tais construções acessa a WMAN através de sua rede local. Outra abordagem é fornecer acesso seguro diretamente ao usuário, ou seja, o usuário através de seu dispositivo acessa uma estação base. Para a evolução da computação ubíqua esse último caso seria ideal, pois permite que a conexão esteja sempre disponível em qualquer local e em todo momento.

As WWANs são redes sem fio que abrangem conexão global. São compostas por uma infra-estrutura que envolve redes locais sem fio, redes metropolitanas sem fio,

[4]www.bluetooth.com
[5]www.irda.org

satélites e conexões por fibra ótica. Exemplos dessas redes são sistemas militares e o sistema de celular.

### 2.4.3 Infravermelho

O espectro infravermelho ou luz infravermelha é usado para comunicações em curta distância. Seu comprimento de onda varia de 700 nm e 100 $\mu\, m$, contudo para as comunicações a banda utilizada está entre 780 nm e 950 nm. Por esse motivo, comunicações infravermelhas referem-se à propagação de ondas de luz na banda próxima a infravermelha (KAHN; BARRY, 1997).

O espectro infravermelho é direcional e as ondas se propagam em forma de cone. A comunicação entre dispositivos que usam o meio infravermelho pode ser realizada de duas maneiras. Pode ser entre dois dispositivos portáteis ou através de um ponto de acesso ou estação base (CARRUTHERS, 2002).

Apesar do espectro infravermelho ser direcional, há dois tipos de ligação: ponto-a-ponto e difusão. Na ligação ponto-a-ponto, o transmissor e o receptor devem estar alinhados e nenhum corpo obstruindo o caminho. Na ligação por difusão, os transmissores e os receptores não precisam estar alinhados, pois os transmissores emitem um feixe extenso de luz e os receptores possuem um extenso campo de visão. Podem ainda ser empregadas estruturas para refletir as ondas para que a conexão esteja sempre disponível.

As vantagens do uso de infravermelho são o seu baixo custo, largura de banda não regulamentada, não interferência por sinais externos ao ambiente e a segurança, visto que a luz não atravessa objetos opacos. As desvantagens são a criação de redes entre ambientes fechados distintos, necessitando instalar pontos de acesso, e os ruídos internos de determinados ambientes, como luz solar e fluorescente.

### 2.4.4 Laser

A comunicação por laser é similar à infravermelha, contudo utiliza outro comprimento de onda e alcança uma distância bem maior. Seus principais problemas são a necessidade de alinhamento perfeito e a sensibilidade a interferências.

Um sistema de comunicação que usa este meio deve consistir de uma fonte

para o laser, um modulador para adicionar a informação ao feixe gerado, um meio de transmissão, no caso sem fio, o espaço, um detector para captar e converter a transmissão para um sinal elétrico e um demodulador para extrair a informação deste sinal.

### 2.4.5 Campos Elétricos

A comunicação com campos elétricos utiliza o corpo humano como condutor para a informação. Pequenos dispositivos colocados próximos ou no próprio corpo modulam um campo elétrico e induzem pequenas correntes através do corpo (ZIMMERMAN, 1995, 1999). As conexões são estabelecidas por toque ou por proximidade, com um alcance aproximado de 2 metros.

Neste meio de comunicação há diversos inconvenientes, pois os sinais podem ser bloqueados pelo próprio corpo ou por objetos aterrados. O simples ato de tocar no dispositivo pode bloquear o campo elétrico e impedir que o dispositivo inicie ou faça parte em uma comunicação. Segundo Zimmerman (1999) um local ideal para utilizar um dispositivo que utiliza esse meio de comunicação seria próximo ao solo, no tênis. Entretanto, há ainda restrições quanto a largura de banda, o que implica na utilização dessa comunicação para poucas atividades, tais como identificação e aplicações onde o tamanho do conteúdo transmitido é desprezível.

### 2.4.6 Campos Magnéticos

As comunicações utilizando campos magnéticos, ao contrário do campo elétrico, não sofrem a interferência do próprio corpo ou de outros corpos. Em tais sistemas, a comunicação não se baseia no fluxo da energia. Nesse caso, um campo magnético modulado é gerado e permanece localizado ao redor do dispositivo gerador.

O principal inconveniente é o tamanho dos equipamentos para geração do campo magnético, os quais são incompatíveis com o tamanho dos dispositivos móveis. O consumo de energia para transmissão é baixo. Podem ser empregados para transferir informações de tamanho muito pequeno.

### 2.4.7 Rede Elétrica

Redes de instalação elétrica usam a estrutura existente da rede elétrica como um meio para transportar informações. Os sistemas existentes têm uma taxa de transferência inferior a 14 Mbps, pois há muito ruído e diversidade de materiais utilizados para compor a linha (HANSMANN et al., 2003). Todavia, avanços nas metodologias de comunicação e modulação, processamento de sinais digitais adaptativos, detecção e correção de erros têm proporcionado uma velocidade e desempenho semelhante a redes cabeadas tradicionais.

A principal motivação para a utilização da rede elétrica são as *home networks*, redes que permitem aparelhos eletrônicos e eletrodomésticos em uma residência se comunicar um com o outro, através de um controlador central, ou com entidades externas. Fornecem suporte para controle e automação simples ou complexa, além de permitir a conexão com um ponto externo, ligando a rede local a uma rede externa.

A motivação para o uso do sistema elétrico deriva do fato que ele provê as ligações, visto que todo aparelho eletrônico está conectado à rede elétrica. Logo, não é necessário instalar novos pontos de acesso, pois estes se encontram distribuídos pela casa. O custo para instalação é relativamente baixo comparado com outras tecnologias utilizadas para a mesma finalidade.

O padrão *HomePlug*, uma aliança composta pela indústria, especifica e implementa os protocolos necessários para a comunicação usando a rede elétrica. Em sua atual especificação, prevê taxas de aproximadamente 200 Kbps, além de fornecer um protocolo de QoS para aplicações de tempo real. Com essa taxa de transferência, é possível transportar conteúdo multimídia de alta qualidade.

As bandas usadas para enviar os sinais são regulamentadas. Uma linha de energia é projetada para transferir sinais de 50 ou 60 Hz, e não para transmitir sinais em alta freqüência necessários para a comunicação. A largura de banda usada pelo padrão *HomePlug* é de 25 MHz.

### 2.4.8 Redes de Linhas Telefônicas

Redes de linhas telefônicas utilizam as linhas de telefone existentes para a formação de uma rede local. Igualmente as redes elétricas, são interessantes para a construção de *home networks*. Transmitem suas informações em altas freqüências, entre 5,5 a 9,5 MHz, e podem atingir taxas de transmissões de até 32 Mbps.

O padrão *HomePNA*, uma aliança composta pela indústria, regulamenta e estabelece padrões e protocolos para a formação das redes de linhas telefônicas. A vantagem do uso dessas redes é a infra-estrutura existente, bastando apenas adquirir os adaptadores e as placas. A desvantagem é o número limitado de tomadas telefônicas nas residências.

## 2.5 Áreas Relacionadas

Diversas áreas contribuem para a composição e definição de computação ubíqua. Sistemas distribuídos e computação móvel são apresentados como as raízes e os alicerces. Com certeza, são duas áreas fundamentais, porém há outras áreas que participam e colaboram diretamente para a criação de ambientes de computação ubíqua. Nesta seção, um esquema e uma breve discussão são apresentados para completar essa visão restrita e exaltar a contribuição de outras importantes áreas.

Sistemas distribuídos contribuem com os esforços para sistemas tolerantes a falhas, replicação para alta disponibilidade, suporte a comunicação remota, questões relacionadas à segurança da informação, suporte para heterogeneidade, compartilhamento da informação e o uso efetivo da tecnologia de agentes.

Computação móvel contribui com questões de gerenciamento de redes móveis, acesso a informação móvel, localização e consumo de energia, protocolos para acesso a informação móvel, infra-estrutura para comunicação móvel, dispositivos e tecnologias de telecomunicação.

Inteligência artificial fornece suporte para adaptação de sistemas baseados no perfil do usuário, gerenciamento inteligente de energia e de ambientes impregnados com dispositivos, planejamento para melhor executar uma tarefa, enfim, para todas as questões onde o sistema deve ter autonomia para tornar a computação o mais transparente possível ao usuário.

Projeto e arquitetura de computadores fornecem dispositivos miniaturizados, portáteis, com eficientes sistemas de gerenciamento de energia e dissipação de calor, alto poder de processamento e interoperacionais, para atender a necessidade dos softwares que são executados sobre estes.

Sistemas multimídia colaboram com pesquisas em processamento de imagem e áudio. Processamento de imagens fornece meios para identificar e localizar entidades em um ambiente. Processamento de áudio permite que as pessoas interajam e controlem os dispositivos através da voz, sem ter que se preocupar com interfaces e controles complexos.

Interação homem-máquina e engenharia de interfaces fornecem meios para se construir dispositivos de fácil utilização, transparentes ao ambiente, com o objetivo de minimizar os impactos sociais e comportamentais e, principalmente, tornar amigável a presença da tecnologia.

Sistemas operacionais fornecem as abstrações para que diferentes serviços possam interagir e possam ser executados sobre sua arquitetura. Em conjunto com o hardware é responsável por gerenciar eficientemente todos os recursos dos dispositivos portados pelo usuário.

Não apenas dessas áreas a computação ubíqua é composta, mas a partir delas é possível construir um esquema (figura 2.2) que represente a intersecção e ao mesmo tempo o espaço existente para a concretização e materialização de algumas das premissas e desafios de computação ubíqua.

O modelo apresentado na figura 2.2 tem a forma de um quebra-cabeça com algumas peculiaridades. Algumas faces não são seqüenciais, ou seja, não formam uma figura uniforme. Os encaixes não são complementares, ou seja, não se encaixam apesar de possuírem a mesma face. Essas características têm por objetivo enfatizar que a computação ubíqua tem características de diversas áreas, mas ainda falta modelar alguns encaixes e projetar algumas faces para que seja construída uma realidade que suporte esse paradigma.

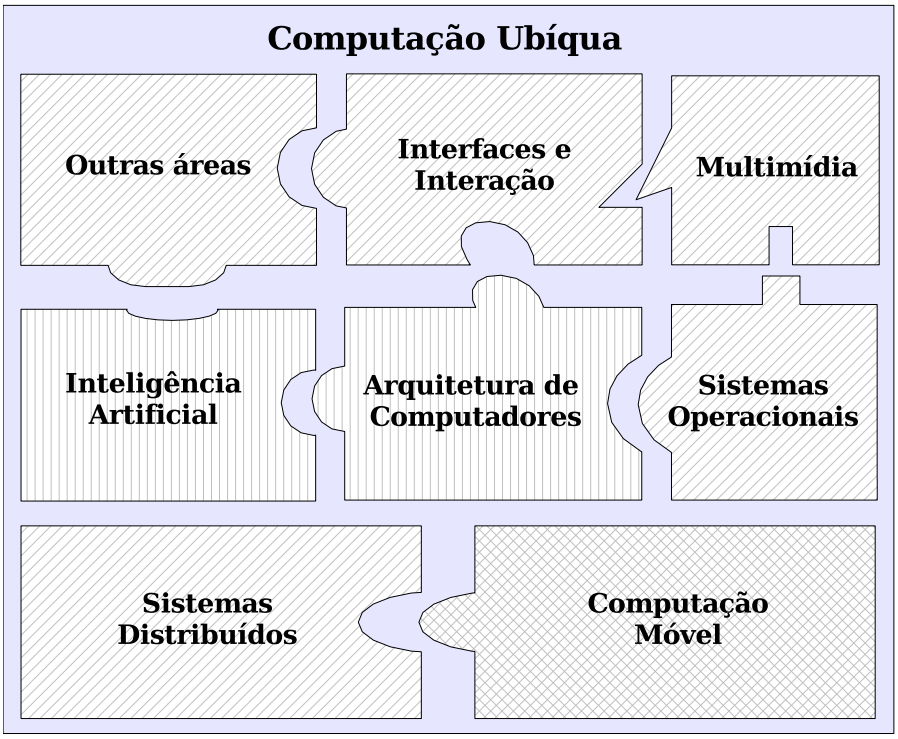


Figura 2.2: Quebra-cabeça da composição de computação ubíqua

## 2.6 Características

Na seção anterior foi dissertado sobre as diversas áreas que contribuem com a computação ubíqua, sendo que muitas das características das áreas apresentadas estão intrinsecamente presentes nas características individuais de computação ubíqua.

Nesta seção são discutidas as características que se destacam em computação ubíqua. As características apresentadas são estudo de áreas específicas, contudo são elas que definem as principais características dos ambientes de computação ubíqua e tornam essa área muito mais complexa do que a simples intersecção de conhecimentos desenvolvidos anteriormente em outras áreas.

### 2.6.1 Invisibilidade

Utopicamente, é o total desaparecimento da percepção do usuário a tecnologia utilizada. Na prática, uma razoável aproximação é a distração mínima do usuário com a tecnologia. A tecnologia não pode falhar e principalmente, incomodar o usuário na realização de suas tarefas. Ela deve ser apresentada de maneira simples,

de modo que o usuário trabalhe em nível subconsciente, como se estivesse fazendo algo natural como escrever ou falar.

Invisibilidade não é um termo empregado com freqüência em computação; na maioria dos casos é utilizado o termo transparência, definido como o ocultamento da complexidade de um sistema ao usuário e/ou programador. Também é empregado em computação ubíqua com esse sentido, pois sistemas e ambientes de computação ubíqua necessitam ser transparentes em diversos aspectos, especialmente aqueles relacionados à computação distribuída. Entretanto, difere de invisibilidade, pois ao manusear um sistema que se apresente transparente, o usuário sabe que está usando um sistema.

Em uma situação de invisibilidade, o usuário está consciente do uso sistema, mas durante sua realização não há uma atenção direcionada para o sistema, ou seja, a atenção está toda direcionada para o trabalho. Por exemplo, comparando um músico amador, o qual tem que conscientemente pensar sobre todas as notas e posições dos dedos, com um músico profissional, o qual conhece a ferramenta tão bem e tão inconscientemente, o músico profissional realiza melhor sua tarefa, pois dedica toda sua atenção a qualidade da música.

A invisibilidade pode ser atingida em ambientes de computação ubíqua se estes atenderem todas as expectativas do usuário para a realização de uma determinada tarefa e não apresentarem nenhum inconveniente, como por exemplo, executar demoradamente um processo simples ou executar parcialmente uma atividade exigindo esforço do usuário para concluí-la.

### 2.6.2 Ambientes inteligentes

Ambientes inteligentes são ambientes saturados com dispositivos eletrônicos com capacidade de computação que podem se comunicar entre si e com outros dispositivos que se apresentem a eles. Um ambiente pode ser representado por uma sala fechada ou por uma área aberta com fronteiras bem definidas. Ambientes inteligentes devem ser capazes de determinar a presença de outros dispositivos, autoconfigurar-se para atender as demandas em um determinado instante, fornecer interfaces padrões para comunicação, não violar a privacidade dos usuários e se apresentar com uma aparência amigável.

Há muitos exemplos de ambientes inteligentes. Alguns são mais simples, como casas que permitem controle das condições do ambiente, tais como temperatura e luzes, outros mais complexos, como ambientes que possuem uma infra-estrutura para monitorar o usuário e executar tarefas mais sofisticadas, tais como controlar dispositivos através da Internet.

Além da classificação básica, como ambientes abertos e fechados, os ambientes podem ser classificados como públicos e privados. Uma casa é um ambiente privado, um shopping ou uma praça é um ambiente público. Esta classificação implica em restrições que devem ser impostas para que a segurança e a privacidade não sejam violadas.

Um exemplo de um ambiente inteligente é a *Intelligent Room*[6], um ambiente interativo que observa e participa de eventos. Equipado com câmeras, microfones e telas projetoras, os participantes podem interagir com o ambiente através da fala e gestos. O ambiente em contrapartida é responsável de fornecer o suporte para as reuniões, tais como registrar discussões, localizar e projetar informações.

### 2.6.3 Sensibilidade à localização

Localização para sistemas ubíquos é de fundamental importância, pois permite que os dispositivos gerenciem largura de banda, consumo de energia, qualidade de comunicação e, principalmente, não interfiram no funcionamento de outros dispositivos.

Hightower e Borriello (2001) apresentam algumas propriedades e características, independentes de tecnologia, para classificar e avaliar os sistemas de localização para computação ubíqua:

- Posição física e simbólica: posição física indica através de um valor de precisão a localização de uma entidade. Por exemplo, a "entidade x" está localizada a 30 graus norte, a 100 metros de altitude. A posição simbólica indica através de idéias abstratas a localização de uma entidade. Por exemplo, a "entidade x" encontra-se na cozinha.

- Localização absoluta e relativa: localização absoluta utiliza uma grade de re-

[6]http://aire.csail.mit.edu/

ferência compartilhada para todos os objetos localizados, ou seja, dois objetos colocados em um ponto comum retornam leituras de posição equivalentes. Na localização relativa cada objeto tem sua própria grade de referência, ou seja, um objeto colocado em um ponto retorna leituras de posição relativas a algum outro ponto.

- Computação localizada da posição: o objeto que está sendo localizado é responsável por computar sua própria posição. Este modelo de sistema garante a privacidade, pois o objeto é quem publica sua posição. Em determinadas situações o objeto deve periodicamente informar por difusão sua posição para os dispositivos de localização.

- Exatidão e precisão: sistemas de localização informam as posições baseados em medidas e a freqüência que essa medida pode ser obtida. Por exemplo, um sistema de localização indica a presença de uma entidade com exatidão de 3 metros em 98% das medidas. Neste sistema, a exatidão é 3 metros e a precisão é obtida em 98% dos casos.

- Escala: permite estimar quantos dispositivos podem ser localizados em uma determinada dimensão de um ambiente por intervalo de tempo. Logo, a escala define a quantidade máxima de objetos que podem ser localizados e o intervalo de tempo para que não haja congestionamento da largura de banda utilizada no processo de localização.

- Identificação: objetos localizados devem ter uma identificação para que o sistema possa reconhecê-los ou classificá-los. A identificação corresponde geralmente a números ou nomes únicos para distinguir os objetos.

- Custo: permite estimar através de algumas variáveis o custo de um sistema de localização. Os custos de tempo estão relacionados ao processo de instalação e administração. Os custos de espaço ao tamanho dos equipamentos e do ambiente. Os custos capitais ao preço por unidade dos elementos de localização.

- Limitações: consiste em considerar as características fundamentais da tecnologia que implementa o sistema de localização para avaliar as limitações funcionais, tais como restrições de freqüência, espaço e tempo.

Há diversos sistemas de localização, entre os principais estão o GPS (LEICK, 1995), Active Badges (WANT et al., 1992) e Active Bats (WARD; JONES; HOPPER, 1997). O GPS é um sistema de localização global, constituído por satélites estacionários, que por meio de medições e cálculos da combinação de no mínimo três satélites informa a posição de um corpo na superfície. Active Badges é um sistema de localização local que utiliza difusão de infravermelho a cada 10 segundos para atualizar um servidor central, responsável por identificar o local onde foi emitido o sinal. O Active Bats é uma modificação que trabalha com ondas de rádio, onde uma solicitação é enviada pelo servidor e os clientes emitem um pulso ultra-sônico. O pulso é captado por receptores e sincronizado com o servidor central responsável por calcular então a localização.

### 2.6.4 Sensibilidade ao usuário

Ambientes e dispositivos ubíquos devem trabalhar em função das necessidades de seus usuários, logo devem fornecer mecanismos para armazenar, recuperar e prever possíveis tarefas, comportamentos e até mesmo o estado emocional das pessoas envolvidas.

Sensibilidade ao usuário implica em uma mudança de comportamento dos sistemas para tornarem-se pró-ativos, conduzindo a uma mudança da computação centrada no homem para a computação supervisionada pelo homem. Com isso, a interação manual entre ambos é diminuída, os sistemas ganham em autonomia e expandem a sua área de atuação para cenários maiores e mais complexos.

Tennenhouse (2000) denomina esse campo como computação pró-ativa. Ele descreve três pontos para serem avaliados para atingir esse objetivo: sistemas pró-ativos devem, através de sensores e atuadores, monitorar o usuário e o ambiente para explorá-lo; computadores pró-ativos devem responder a estímulos externos muito mais rápido que um humano responderia; e a interação com os sistemas seria necessária somente para supervisão, ou seja, remover a atenção humana da computação e direcioná-la para o acompanhamento da tarefa desempenhada.

Um exemplo do uso efetivo dessa propriedade é em atividades rotineiras, onde um usuário sempre executa um conjunto de tarefas bem definido. Um sistema pró-ativo pode se adiantar ao usuário e executar uma tarefa antes da emissão de um

comando expresso. Entretanto, essa pró-atividade deve ser bem administrada para não causar efeito reverso, isto é, distrair o usuário, como no caso citado, se a tarefa pretendida fosse outra e não a prevista pelo sistema.

### 2.6.5 Contextos de ambientes

Em um espaço podem existir diversos tipos de ambientes que podem ser classificados por sua sofisticação ou efetividade. A mudança entre estes ambientes pode causar desconforto a um usuário. Logo, é necessário que os dispositivos do usuário possam minimizar esses impactos através da adaptação ao contexto.

Para que uma adaptação ocorra, é necessário que os dispositivos tenham ou possam obter conhecimento do local onde se encontram. Esse conhecimento refere-se aos recursos físicos, como objetos, dispositivos e redes, e aos recursos virtuais, como largura de banda, disponibilidade de comunicação, memória e processamento de algum outro dispositivo, entre outros.

Computação ciente de contexto é responsável por coletar e utilizar a informação de contexto, ou seja, a informação que atua diretamente no comportamento dos serviços computacionais (ABOWD et al., 2002). Pois, da mesma maneira que humanos usam qualquer situação de contexto para adequadamente se comportar, espera-se que os serviços virtuais possam reagir de maneira mais apropriada no ambiente em que são invocados.

A coleta da informação de contexto depende de diversos fatores, os quais vão desde questões de privacidade, no caso de dados pessoais do usuário e sua localização, a questões de quais informações são relevantes para a definição do contexto. Essas questões podem ser explícitas, ou seja, perceptíveis pelo usuário, ou implícitas, aquelas que são observadas apenas pelo sistema.

Apesar da coleta ser dependente do ambiente, um fator mais problemático é como tomar a ação apropriada para o contexto obtido, pois tomar ações é algo complexo e requer considerável inteligência. Erickson (2002) destaca que computadores são eficientes para coletar e agrupar dados e humanos são eficientes para interpretar o contexto e determinar o que é apropriado.

De qualquer forma, o objetivo dos sistemas conscientes de contexto em computação ubíqua é prover mecanismos para que os dispositivos, principalmente com

capacidades limitadas, estendam ou adequem seus recursos àqueles disponíveis no ambiente local (EBLING; HUNT; LEI, 2001).

### 2.6.6 Segurança e privacidade

Segurança refere-se a segurança de informações digitais, como também dos dispositivos físicos, pois, em computação ubíqua, as aplicações estão intrinsecamente ligadas a infra-estrutura física do ambiente onde executam.

A privacidade pode ser violada pelos dispositivos e sensores, pois estão distribuídos no ambiente, recolhendo informações sobre localização, dados pessoais, freqüência de uso, entre outros, muitas vezes, sem o conhecimento do usuário que fornece estes dados.

As características que tornam os ambientes de computação ubíqua convenientes e poderosos, também são responsáveis por torná-los vulneráveis a uma séria de novos ataques a segurança e a privacidade (CAMPBELL et al., 2002). Stajano (2002a) destaca que quando o tamanho de um problema cresce centenas de vezes, não há garantias que as soluções antigas tornem-se escaláveis.

Uma tarefa simples como autenticação pode ser um grande problema em computação ubíqua. Com computadores pessoais somente o dono do computador tem o acesso, mas em computação ubíqua há uma série de dispositivos que são compartilhados por diversas pessoas. Isto conduz ao princípio do *Big Stick* (STAJANO, 2002b), ou seja, todos que têm acesso ao dispositivo têm permissão para assumir seu controle. Isso acontece com vários dispositivos ubíquos, como por exemplo, etiquetas e cartões inteligentes, geladeiras, entre outros.

A privacidade em ambientes de computação ubíqua pode ser ameaçada por diversas aplicações. Um exemplo clássico é a privacidade de localização, onde ambientes ou dispositivos tem acesso a localização de um indivíduo. Outra questão é quanto ao conhecimento da identidade do usuário por aplicações, as quais coletam informações para construir perfis, que podem ser usadas intrusivamente por organizações.

### 2.6.7 Impactos sociais, culturais e psicológicos

A computação ubíqua não está se tornando embutida somente em ambientes físicos, mas na cultura, sociedade e história (SENGERS et al., 2004).

Os impactos sociais, culturais e psicológicos são importantes para avaliação dos dispositivos em computação ubíqua, contudo têm sido desconsiderados em muitas pesquisas. Por estarem relacionados com as áreas de psicologia e sociologia, a maioria dos cientistas e engenheiros não tem estrutura para lidar com essas questões. Conseqüentemente, suas observações e avaliações não contemplam totalmente questões referentes à aceitação, influência na sociedade, mudança no comportamento das pessoas, esforço de pesquisa versus aplicação, dentre outros.

Apesar de todas novas tecnologias introduzirem essas questões, em computação ubíqua elas devem receber muito mais atenção. Isto se deve ao fato de que a tecnologia está muito mais presente na vida diária das pessoas do que em outras áreas. Essa presença obriga, de certa maneira, o usuário a interagir com freqüência com dispositivos, o que inevitavelmente influencia no seu cotidiano.

Dryer, Eisbach e Ark (1999) apresentam um modelo teórico para avaliar os impactos sociais introduzidos em computação ubíqua. No modelo são apresentados quatro grupos de relacionamento: projeto do sistema, atribuição social, comportamento humano e conseqüências da interação. Após a aplicação do modelo, Dryer conclui que dispositivos ubíquos podem inibir em vez de facilitar a interação social, e ainda acrescentou que dispositivos ubíquos podem afetar os mecanismos que determinam quando relações são satisfatórias e produtivas.

Há muitas pesquisas sendo desenvolvidas sobre essas características. O exemplo citado apenas ressalta a relevância em considerar essa propriedade nos ambientes de computação ubíqua.

## 2.7 Desafios e Soluções

Computação ubíqua necessita de novas pesquisas e soluções para preencher os requisitos de suas características. Nesta seção, são apresentados os principais problemas e possíveis investigações que podem solucionar ou contribuir para a implementação inicial de ambientes que suportem as exigências de computação ubíqua.

Os desafios discutidos a seguir foram apresentados por Satyanarayanan (2001).

### 2.7.1 Intenções do usuário

Com o objetivo de alcançar a pró-atividade, ou seja, a capacidade de prever as ações do usuário, deve-se armazenar informações pessoais e seus comportamentos. As questões vão desde o que e como deve ser armazenado até questões mais cruciais de viabilidade deste procedimento.

Para ilustrar essa situação, suponha que um usuário está realizando uma comunicação de duração prolongada com seu dispositivo e recebe a solicitação para iniciar uma comunicação de prioridade maior. O sistema pode optar por encerrar a primeira conexão, compartilhar a banda para as duas comunicações ou ainda pedir para o usuário tomar a decisão. De qualquer forma, se a tarefa executada não for a que o usuário desejava, o sistema terá violado a propriedade de invisibilidade.

Esse é um dos grandes problemas do sistema em tentar ser pró-ativo, isto é, realizar ações que vão ao desencontro às expectativas do usuário. Isso não é raro de acontecer, pois por diversas limitações é difícil um sistema tomar decisões que envolvam um grande número de variáveis, principalmente se as variáveis forem abstratas e não puderem ser quantificadas pelo sistema.

### 2.7.2 Empréstimo computacional

Devido à limitação da capacidade dos dispositivos e comunicação móvel, a idéia é dinamicamente ampliar os recursos computacionais de um computador móvel sem fio, explorando a infra-estrutura de hardware com fio. Este sistema pode até ser comparado com as redes espontâneas, onde computadores aderem momentaneamente e depois deixam a rede da mesma forma.

No caso citado, há representantes, os quais seriam responsáveis, temporariamente, por suportar conexões desses dispositivos móveis, executar processamento ou fornecer comunicação e, ao final da conexão, limpar os vestígios do usuário. O processo se daria da seguinte maneira: um usuário móvel entra na área de cobertura de um representante, negocia seu uso e utiliza os recursos para realizar suas tarefas. Acredita-se que futuramente será comum lugares públicos fornecerem recursos para este tipo de serviço.

Questões importantes devem ser consideradas para tornar esse processo efetivo, entre elas estão a escolha da tecnologia utilizada para descoberta (JINI, UPnP), como estabelecer uma relação de confiança com os representantes, como balancear a carga, como fornecer suporte ao usuário que sai de uma área de cobertura, dentre outras.

### 2.7.3 Estratégias de adaptação

Devem ser empregadas quando há uma significante variação entre o fornecimento e a demanda dos recursos. Há diversas estratégias para tratar o problema, tais como: adaptação pelo cliente (dispositivo) às condições do ambiente, adaptação pelo ambiente (balanceamento para atender a demanda), intervenção do usuário com ou sem sugestão de seu dispositivo.

No primeiro caso, o cliente reduz a utilização do recurso escasso, o que ocasiona perda de fidelidade da aplicação. No segundo caso, o cliente solicita ao ambiente para garantir o nível aceitável para a utilização do recurso. No terceiro caso, o cliente sugere ao usuário uma ação corretiva para solucionar ou melhorar o uso do recurso.

Entre as questões que devem ser consideradas para o uso de adaptação, tem-se a escolha da estratégia sob uma determinada situação, o impacto da estratégia sobre o usuário, definir quais os recursos devem ser gerenciados para melhorar o desempenho sem comprometer outra aplicação, como evitar que as adaptações sejam intrusivas.

### 2.7.4 Gerenciamento de energia

O gerenciamento de energia não se deve destinar somente aos componentes eletrônicos, os softwares devem estar equipados para tratar deste problema em alto nível. Dependendo da tarefa iniciada, um sistema não poderá mantê-la satisfatória justamente por falta de energia, ou ainda, acabará por consumir toda a energia deixando outras tarefas pendentes.

Esse é um dos grandes problemas para viabilidade efetiva das visões em computação ubíqua. Não existe ainda uma fonte de energia, capaz de alimentar um sistema móvel e atender as demandas dos dispositivos existentes. As soluções são utilizar gerenciamento em alto-nível para tentar economizar o máximo possível deste

recurso indispensável para o funcionamento de qualquer dispositivo.

As questões de interesse nesse tópico são como construir dispositivos, softwares e protocolos de comunicação que desperdicem pouca energia. Isto vai ao desencontro do que se é pretendido na maioria das outras características, as quais exigem dos dispositivos constante interação com o ambiente e, conseqüentemente, uso constante de energia.

### 2.7.5 Densidade do cliente

Densidade refere-se ao poder computacional que um dispositivo ubíquo deve possuir, ou seja, a capacidade de processamento, de disco, energia, memória, dentre outros. Estas questões determinam as restrições de hardware para um cliente. Logo, o cliente pode ser denso ou leve. Clientes densos tendem a ser maiores, pesados, requerem uma bateria maior e dissipam mais calor. Portanto para usuários móveis isto não é muito aceitável.

Para uma dada aplicação, a densidade mínima aceitável do cliente é determinada pelo pior caso das condições ambientais sobre os quais a aplicação deve executar satisfatoriamente. Um cliente leve é adequado somente se ele pode contar sempre com uma largura de banda larga e baterias recarregáveis ou substituíveis facilmente, caso contrário, o cliente terá que ser o suficientemente denso para compensar algum destes problemas. Com um cliente de densidade moderada, pode-se executar com eficiência tarefas simples localmente, enquanto tarefas mais complexas, que exigem maiores recursos, pode-se usar uma estrutura remota.

### 2.7.6 Consciência de contexto

O contexto do usuário e ambiente pode ser bastante rico, consistindo de atributos tais como localização física, estado psicológico, estado emocional, história pessoal, padrões comportamentais diários, e assim por diante. Um sistema tem consciência de contexto se ele usa informações de contexto para fornecer informações relevantes ou serviços para o usuário.

Consciência de contexto é uma das características de computação ubíqua, mas também é considerado um grande desafio para os pesquisadores. O desafio principal é obter a informação necessária para que uma ação possa ser tomada. Em alguns

casos, a informação pode ser parte do espaço de computação pessoal do usuário, em outros, depende de informações dinâmicas que só fazem sentido em tempo real.

Outras questões que norteiam a pesquisa em consciência de contexto são como obter, armazenar, consultar e descartar as informações de contexto e como balancear o contexto e a usabilidade em um dispositivo com capacidades limitadas.

### 2.7.7 Privacidade e confiança

Consiste em estabelecer a confiança entre o sistema e o usuário, ou seja, fornecer um sistema minimamente intrusivo que preserve a invisibilidade. Baseado nesta idéia, não é aceitável que os usuários sejam interrompidos freqüentemente com opções e alertas para configurar, aceitar ou rejeitar algum tipo de comunicação do ambiente que seja intrusiva.

Os sistemas ubíquos estão sempre monitorando o usuário, trocando informações com outros dispositivos e isso, dependendo do nível, pode ser visto como invasão de privacidade. Os problemas são como garantir e provar que um ambiente não está monitorando os usuários em todos os lugares e obtendo informações que não lhe foram autorizadas.

## 2.8 Aplicações e Serviços

Aplicações e serviços são responsáveis por fornecerem ao usuário final todas as vantagens dos ambientes de computação ubíqua. Não há razão para tanta pesquisa se ela não produz benefícios. Esses benefícios só são perceptíveis quando são materializados em serviços que podem ser utilizados para facilitar e solucionar problemas diários.

Uma variedade de aplicações e serviços desenvolvidos é dirigida para o suporte e o uso de ambientes inteligentes. Entre todos os ambientes, o mais acessível e próximo aos usuários em geral são as casas automatizadas. Nessas casas há diferentes tipos de serviços, tais como serviços de controle dos equipamentos, como luz, aquecimento, refrigeração e eletrodomésticos; serviços de entretenimento, como televisão digital e jogos; serviços de segurança, como monitoramento via Internet; serviços de energia, como o fornecimento e gerenciamento de eletricidade; entre outros.

Os serviços de informação são responsáveis por disponibilizar aos usuários conteúdo para auxiliá-lo em tomadas de decisões, sejam em negócios ou simplesmente em alguma atividade mais simples, como por exemplo, dirigir. São exemplos os serviços de troca de informações, executados independente da posição do usuário; os serviços para veículos, como informação sobre tráfego, localização, distância de outros veículos e proteção contra acidentes; serviços de localização e consulta, como verificar onde encontrar um determinado produto; entre outros.

As organizações comerciais podem usufruir da estrutura dos ambientes ubíquos para acessar seus clientes de uma maneira mais personalizada e com maior impacto. Entre os serviços que podem ser disponibilizados estão os serviços de difusão de propagandas, ou seja, o envio de mensagens com promoções ou com produtos de interesse do usuário ao se aproximar de uma loja; serviços de atendimento, ou seja, ao entrar em um estabelecimento são objetidas informações sobre o usuário para melhor atendê-lo; serviços de pagamento, como contabilizar compras por etiquetas inteligentes e pagamento através de dinheiro digital; entre outros.

Devido à variedade de aplicações torna-se inviável enumerá-las, pois há aplicações específicas para cada área, como por exemplo, serviços para atendimento remoto de pacientes, serviços para auxiliar em construções, serviços para educação, enfim, uma infinidade de opções.

## 2.9 Projetos

Há grandes equipes e projetos envolvidos com as questões de pesquisa em computação ubíqua e muitos deles já têm obtido avanço em direção às características e aos problemas discutidos. A seguir, são apresentados três projetos importantes, os quais desenvolvem tecnologias, soluções e protótipos para ambientes de computação ubíqua.

### 2.9.1 Projeto Aura

O projeto Aura (GARLAN et al., 2002; SOUZA; GARLAN, 2002, 2003) é desenvolvido em Carnegie Mellon University e abrange uma série de outras pesquisas. Todos os esforços são direcionados para obter um sistema onde o recurso mais im-

portante não seja mais o processador, a memória ou o disco, mas a atenção humana. O objetivo do projeto Aura é criar um ambiente adaptável ao contexto e às necessidades do usuário de modo que os recursos computacionais se tornem invisíveis, causando o mínimo possível de distração.

Aura é um ambiente para computação ubíqua que engloba comunicação móvel, computadores portáteis, computadores embutidos em roupas e espaços inteligentes. As pesquisas desenvolvidas abrangem desde o desenvolvimento em nível de hardware e sistema operacional até aplicações e seus usuários. Aura atua como um representante (proxy) para o usuário móvel. Aura, ao perceber que um usuário entra em um novo ambiente, gerencia os recursos para atender e suportar as tarefas do usuário.

A arquitetura de Aura apresenta um framework com três características fundamentais: as tarefas do usuário tornam-se entidades de primeira classe e são representadas explícita e independentes de um ambiente específico; as tarefas do usuário são representadas como serviços unificados; os ambientes são equipados para automonitorar e renegociar suporte das tarefas na presença de variações de capacidades e recursos.

Através de sua infra-estrutura, Aura explora o conhecimento sobre a tarefa do usuário para configurar automaticamente o ambiente ubíquo, utilizando uma arquitetura em três camadas (figura 2.3). A camada de gerenciamento de tarefa (task management) monitora as tarefas, o contexto e as intenções do usuário, mapeia para serviços no ambiente e executa tarefas complexas como planejamento, decomposição e dependência de contexto. A camada de gerenciamento do ambiente (environment management) monitora os recursos e as capacidades do ambiente, mapeia as necessidades dos serviços e otimiza a utilidade do ambiente conforme a tarefa do usuário. Por fim, a camada de ambiente (environment) suporta a execução da tarefa do usuário.

A figura 2.3 ilustra as camadas e os quatro principais componentes da arquitetura: o Task Manager, conhecido como Prisma, atua como o representante, coordenando a reconfiguração do ambiente; o Context Observer fornece informações sobre o contexto e reporta eventos para o Prisma; o Environment Manager prepara os serviços requeridos pelas tarefas do usuário; e o Supplier fornece os serviços para as tarefas (editores, reprodutores de mídia ou áudio, etc.).

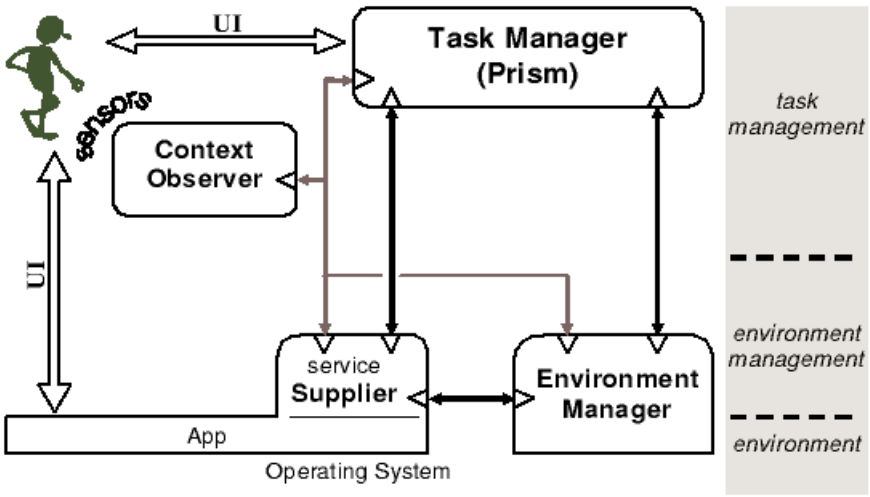


Figura 2.3: Arquitetura de Aura (SOUZA; GARLAN, 2003)

### 2.9.2 Projeto one.world

O projeto one.world (GRIMM et al., 2004), desenvolvido em University of Washington, fornece um framework para construir aplicações para ambientes altamente dinâmicos, permitindo que estas se adaptem às mudanças que podem ocorrer. A arquitetura é baseada em um modelo de aplicação simples para facilitar o compartilhamento de dados e em um conjunto de serviços para atuar em redes dinâmicas.

São considerados três requisitos básicos para suportar as aplicações ubíquas. Primeiro, mudança de contexto, as aplicações devem estar cientes da mudança de contexto e se adaptar automaticamente a um novo ambiente. Segundo, composição ad hoc, o sistema deve compor aplicações, serviços e dispositivos dinamicamente, pois o usuário espera que seus dispositivos e aplicações trabalhem em conjunto, ou seja, estejam disponíveis em qualquer ambiente que se encontrem. Terceiro, reconhecimento de compartilhamento como padrão, o acesso a informação deve estar acessível em qualquer lugar e momento, pois em um mundo ubíquo a troca de informações deve ser natural.

Para atender esses requisitos, a arquitetura foi estruturada seguindo quatro princípios básicos: serviços base, responsáveis por tratar dos três requisitos fundamentais; serviços de sistema, derivados dos serviços base para construir aplicações; divisão do kernel e espaço do usuário, onde os serviços base e de sistema são tratados pelo kernel enquanto bibliotecas, utilidades do sistema e aplicações executam no espaço do usuário; liberdade para o desenvolvimento e implementação das aplicações.

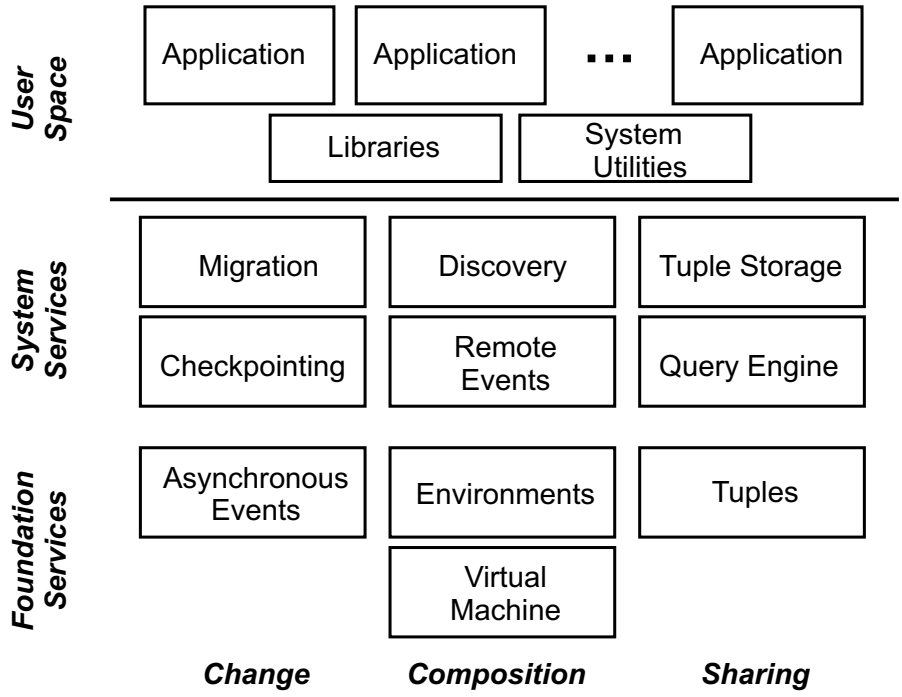


Figura 2.4: Arquitetura de one.world (GRIMM et al., 2004)

Os serviços base consideram os requisitos básicos de mudança, composição e compartilhamento. A máquina virtual fornece um ambiente comum para a execução independente de dispositivo e plataforma. Tuplas definem um modelo de dado comum para compartilhamento. Todas comunicações são tratadas como eventos assíncronos, logo as aplicações são notificadas de mudanças e podem se adaptar. Os ambientes são responsáveis por estruturar e compor as aplicações. Eles também são conteiners para tuplas, componentes e outros ambientes. Toda aplicação é constituída por no mínimo um ambiente, responsável por isolar uma aplicação de outra.

Os serviços de sistema fornecem um conjunto de serviços para a construção de aplicações. Os serviços disponíveis são busca de tuplas (query engine), manipulação de dados nas tuplas (structured I/O), comunicação remota (remove events), localização de serviços (discovery), recuperação de falhas (checkpointing) e migração ou cópia de ambientes entre dispositivos (migration).

No nível do usuário, a arquitetura fornece bibliotecas para implementar as aplicações ubíquas. Na arquitetura one.world, cada dispositivo é executado em uma simples instância. Cada nó é independente e pode ser administrado separadamente. Todas aplicações executadas em um nó compartilham a mesma instância da arquitetura.

Em resumo, uma aplicação em one.world consiste de um ambiente e seu código é executado em resposta a eventos assíncronos gerados pelo sistema, aplicações ou dispositivos. Os dados são armazenados em tuplas em um repositório associativo. O estado da aplicação pode ser armazenado e recuperado. A aplicação pode migrar para diferentes dispositivos. Todos os dispositivos do ambiente devem executar a plataforma one.world.

### 2.9.3 Projeto Oxygen

O projeto Oxygen (MIT, 2004) é desenvolvido no Laboratório de Inteligência Artificial do Massachusetts Institute Technology - MIT e conta com a parceria da DARPA e de grandes empresas como a Acer, Delta Electronics, Hewlett-Packard, Nippon Telegraph and Telephone, Nokia Research Center e Philips Research.

A filosofia do projeto é a construção de um sistema que permita que a computação e a comunicação sejam onipresentes e livres como o ar, ocupando seu espaço naturalmente na vida das pessoas. O projeto é baseado na computação centrada no homem, portanto o sistema deve estar presente em todos os lugares, procurando antecipar necessidades e contribuindo para realizar mais tarefas com tempo e esforço menores.

O projeto conta com a participação de muitos pesquisadores e é composto por cerca de 400 sub-projetos. Conseqüentemente, diversas tecnologias estão sendo desenvolvidas, as quais abrangem dispositivos, redes, softwares, percepção, acesso ao conhecimento, automação e colaboração.

Os dispositivos desenvolvidos são móveis ou estacionários, porém são sempre anônimos e não armazenam configurações de um determinado usuário. Determinados dispositivos são embutidos e utilizados para criar espaços inteligentes, fornecem computação e interface para outros dispositivos. As interações com o usuário ocorrem através da fala ou visão. Outros dispositivos são para computação móvel, aceitam entrada visual ou através da voz, suportam diversos protocolos de comunicação e possuem uma variedade de funções. Têm a capacidade de se autoconfigurar em um ambiente e possuem sistemas para gerenciar o consumo de energia. Como exemplos dos projetos desenvolvidos, tem-se a sala inteligente, um ambiente interativo, e o *Cricket*, um sistema que fornece informações sobre localização em ambientes

fechados.

As redes permitem a formação de regiões de colaboração entre dispositivos móveis e estacionários, suportam uma variedade de protocolos de comunicação e fornecem mecanismos para descoberta de recursos e localização, garantem segurança ao acesso de informações e se adaptam a mudanças das condições, tais como congestionamento, erros, latência, entre outros. Entre as tecnologias desenvolvidas tem-se o *Grid*, um protocolo de roteamento para redes móveis; *Span*, um protocolo para gerenciamento eficiente de energia; *Chord*, um protocolo de pesquisa para redes P2P, entre outros.

Os softwares são projetados para se adaptarem ao usuário, ao ambiente, a mudanças e falhas com mínima intervenção humana. Entre os projetos desenvolvidos tem-se o *Pebbles*, componentes de software independentes de plataforma; *CORE* um ambiente de roteamento orientado a comunicação para computação ubíqua, entre outros.

Em percepção são abordadas as tecnologias que exploram o uso da fala e visão. São desenvolvidos projetos para reconhecimento de voz, objetos, compreensão de linguagem, entre outros. São exemplos o *Galaxy*, uma arquitetura para integrar tecnologias de reconhecimento de voz, e o sistema de rastreamento de pessoas, um sistema que usa uma rede de câmeras para rastrear, estimar trajetórias e outras características de pessoas em um ambiente restrito.

Nas tecnologias do usuário são desenvolvidos projetos para o acesso ao conhecimento, automação e colaboração. São diversos projetos sendo desenvolvidos para atender as necessidades específicas do usuário, como exemplo, o *Haystack*, uma plataforma para criar, organizar e visualizar informação.

# 3 Modelagem e Simulação

Desde os tempos antigos, os pesquisadores e o próprio homem buscam eloqüentemente explicar o funcionamento dos diversos sistemas que os cercam. Entretanto, alcançar tal objetivo, muitas vezes, é uma árdua jornada por caminhos desconhecidos, pois os sistemas são, em sua maioria, tão complexos, que para desvendar seus segredos são necessárias diversas gerações ou uma expressiva colaboração entre os interessados no assunto.

Estes sistemas, criados por homens ou pela natureza, são regidos por leis e variáveis. Mas, há sistemas onde a complexidade de expressão de tais elementos impossibilita quantificar e qualificar a influência exata de cada uma sobre o sistema como um todo. Logo, a solução encontrada foi propor modelos que possam representar o comportamento dos sistemas e através de seu estudo contribuir para o seu entendimento.

Em computação ubíqua, essa complexidade está presente, e este capítulo tem por objetivo apresentar as premissas sobre a construção de modelos e justificar o uso de simulação para sua compreensão. São apresentados uma linguagem formal utilizada no decorrer da dissertação na especificação de modelos e dois projetos que acreditam nos benefícios e importância da modelagem e simulação em ambientes de computação ubíqua.

## 3.1 Modelagem de sistemas

Sistemas podem ser definidos como um conjunto de elementos, materiais ou ideais, entre os quais se possa encontrar ou definir alguma relação. Há diversos tipos de sistemas, por exemplo, sistemas de produção, sistemas de comunicação, sistemas computacionais, sistemas econômicos e sistemas biológicos.

Um modelo é o conjunto de hipóteses ou asserções sobre a estrutura ou comportamento de um sistema físico pelo qual procura-se explicar ou prever, dentro de uma teoria científica, as propriedades do sistema, ou seja, os modelos são, em sua maioria, abstrações simplificadas da estrutura e funcionamento de sistemas reais. Através de um processo de modelagem são realizados os devidos refinamentos para a definição e construção de um modelo.

Há processos e linguagens que fornecem ferramentas e estruturas para descrever e construir modelos. A modelagem gráfica descreve através de diagramas e relações, a modelagem matemática descreve através de fórmulas e relações, a modelagem formal descreve através de linguagens formais, a modelagem estatística descreve através de abstrações estatísticas, e assim por diante.

Qualquer que seja o processo ou a linguagem, em se tratando de modelos que representam sistemas do mundo real, a complexidade para abstração é um obstáculo, pois além do tamanho, o comportamento de sistemas reais pode ser imprevisível. Sistemas de computação ubíqua apresentam essa dificuldade, principalmente, por serem compostos por outros subsistemas e seu comportamento ser dependente do comportamento e ações do homem.

Para a criação de modelos de tais sistemas, deve-se utilizar o recurso de modelos reais ou modelos de simulação, isto é, a criação de modelos que possam representar através de estruturas computacionais a estrutura e o comportamento dos elementos envolvidos no sistema com um grau de abstração e proximidade, permitindo a inferência de verdades coerentes sobre a representação do modelo.

Os modelos de simulação podem ser classificados, segundo Paulo Freitas (FILHO, 2001), de acordo com seus propósitos. Modelos voltados à previsão permitem suposições sobre o comportamento atual e futuro do sistema. Modelos voltados à investigação permitem a busca e o desenvolvimento de hipóteses. Modelos voltados à comparação permitem, através de diversas simulações, avaliar o efeito de mudança sobre o comportamento dos elementos do modelo.

Em geral, os modelos são essenciais para a corretude do processo de estimativa e avaliação em qualquer propósito empregado. Portanto, a construção de modelos que representem as variáveis comensuráveis e, em algumas situações, não comensuráveis, constitui um importante aspecto em qualquer ciência que deseja validar ou demonstrar suas conclusões.

## 3.2 Simulação de sistemas

Simulação é o processo de projetar um modelo computacional de um sistema real e conduzir experimentos com este modelo com o propósito de entender seu comportamento e/ou avaliar estratégias para sua operação (PEGDEN et al., 1995).

O uso da simulação tem crescido consideravelmente devido às facilidades e possibilidades oferecidas. Como vantagens pode-se citar as tomadas de decisões sobre o futuro, observação do desempenho do sistema sobre certas circunstâncias, problemas na execução da simulação não afetando o sistema real, possibilidade de análise e verificação do sistema, entre outras.

No entanto, a simulação pode conduzir a erros grosseiros se não for executada corretamente. Entre as falhas mais comuns estão o pouco conhecimento da ferramenta ou método utilizado, falta de clareza na definição dos objetivos, construção de modelos com excesso de detalhes, conclusões precipitadas sobre os resultados obtidos e falta de planejamento do projeto.

Para não cometer esses erros pode-se seguir alguns princípios utilizados em modelagem. Esses princípios definem como um sistema deve ser organizado. Primeiramente, deve-se construir, sempre que possível, em blocos, ou seja, construir os subsistemas separadamente e então conectá-los. Somente aspectos relevantes devem ser inclusos; muitos detalhes podem interferir no processo de simulação. Toda informação que for utilizada na simulação deve ser precisa, isto é, deve estar correta dentro de sua especificação. Por último, as entidades comuns devem ser agrupadas segundo sua estrutura e comportamento para evitar definições de entidades desnecessárias.

Seguindo esses princípios é possível modelar um sistema de computação ubíqua considerando as suas entidades e o seu funcionamento apesar da complexidade em representá-los de maneira simplificada. Por isso, deve-se ter em mente que a técnica de simulação não tenta isolar todas as variáveis e os relacionamentos particulares entre elas, mas observar a maneira como todas as variáveis do modelo se alteram com o tempo. O relacionamento dessas variáveis deve ser derivado dessas observações (GORDON, 1969).

## 3.3 Object Z

Object Z (SMITH, 1992) é uma extensão da linguagem de especificação formal Z (SPIVEY, 1989) que introduz novos construtores para especificação formal de sistemas orientados a objeto. A principal razão para a extensão são as vantagens e facilidades fornecidas pela orientação objeto, tais como modularidade, entendimento, reuso e manutenção dos sistemas. Essas vantagens se aplicam a especificações formais, pois os sistemas onde essa metodologia é aplicada são, em sua maioria, enormes e complexos.

Dentre as extensões adicionadas, a mais importante é o conceito de esquema de classe. Outros conceitos como herança e polimorfismo também foram adicionados. Uma novidade foi a adição de lógica temporal para representar as propriedades de vivacidade. Nesta seção são apresentadas e brevemente discutidas essas e outras características de Object-Z, necessárias para a compreensão dessa dissertação.

### 3.3.1 Classes

O esquema de classe em Object-Z, baseado na noção de classe em orientação objeto, é responsável por encapsular um esquema de estado com todos os esquemas que podem modificar as suas variáveis. O esquema de classe não é apenas uma extensão sintática, ele pode definir tipos, cujas instâncias são objetos.

A representação sintática do esquema de classe é uma caixa nomeada com zero ou mais parâmetros genéricos. A estrutura básica da classe pode ser visualizada na figura 3.1.

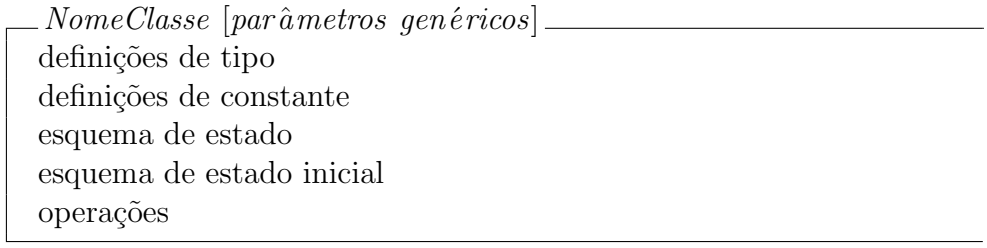


Figura 3.1: Estrutura de classe em Object-Z

As definições de tipo e constante são sintaticamente idênticas ao Z, contudo o escopo é restrito a classe. O esquema de estado armazena variáveis de estado, como

em Z, entretanto não é nomeado. As declarações nesse esquema são as variáveis de estado e o predicado é o estado invariante, o qual restringe os valores das variáveis e das constantes. Em orientação objeto, as variáveis de estado e constantes definem os atributos da classe. O esquema de estado inicial, denominado INIT, é o esquema de operação responsável pela inicialização do sistema, como um construtor. Os esquemas de operação são responsáveis por modificar o estado das variáveis da classe, como métodos. Em sua sintaxe foram adicionadas as delta-lists, as quais indicam as variáveis que se alteram pela operação. Os parâmetros genéricos permitem a definição de classes parametrizadas ou genéricas. A figura 3.2 representa esses conceitos.

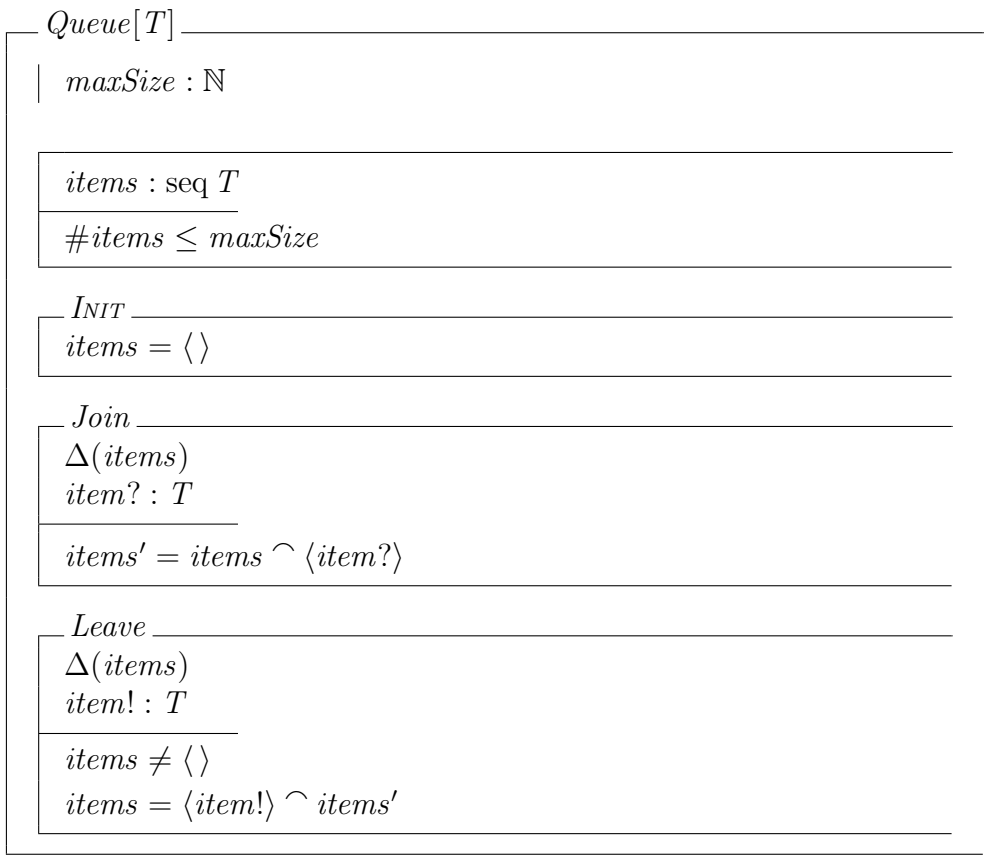


Figura 3.2: Classe em Object-Z que representa uma fila genérica

No exemplo, maxSize define uma constante. A declaração de items está encapsulada no esquema de estado e, é a variável de estado que armazena os elementos da fila. O predicado do esquema de estado garante que ao executar uma operação, o número de elementos de items não será maior que maxSize. O esquema de estado inicial, INIT, inicializa a fila como vazia. Não é necessário incluir o esquema de es-

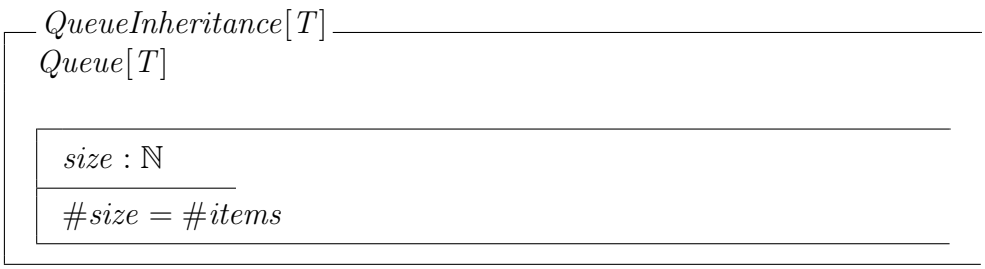


Figura 3.3: Herança em Object-Z

tado, pois ele está implicitamente incluso. Os outros dois esquemas são as operações, onde é necessário especificar, através de delta-list ($\Delta$-list), qual a variável sofrerá mudança. A pós-fixação do ponto de interrogação (?) indica uma entrada e a pós-fixação do ponto de exclamação (!) uma saída.

### 3.3.2 Herança

Herança permite uma classe ser definida a partir das características e funcionalidades de outras classes através da especificação incremental. Object-Z suporta herança simples e múltipla. Para utilizar herança, basta adicionar, antes da definição das constantes em uma classe, as classes herdadas. Como exemplo, herdando do exemplo anterior, temos uma nova classe para manipular filas com a variável de estado size (figura 3.3).

A herança em Object-Z permite adição de atributos, operações, parâmetros de operações e restrições às classes herdada. Também é permitido renomear atributos e operações, para tornar a especificação mais clara para a aplicação a que se destina. Para redefinir totalmente uma operação, ou seja, desconsiderar a implementação herdada, é necessário usar um operador especial, *redef*.

### 3.3.3 Polimorfismo

Polimorfismo é a habilidade de processar objetos diferentemente dependendo de sua classe. Através de polimorfismo pode-se declarar uma variável com a capacidade de armazenar qualquer objeto de uma coletânea de classes associadas por herança. Em Object-Z, isto é alcançado através da declaração $\downarrow$ (polimorfismo) pré-fixada ao tipo do objeto. Por exemplo, seja C uma classe, a declaração $c : \downarrow C$ declara o objeto

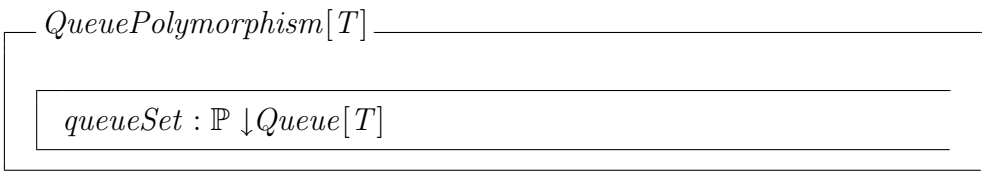


Figura 3.4: Polimorfismo em Object-Z

c para ser da classe C ou qualquer classe derivada de C. A figura 3.4 demonstra o uso do polimorfismo através do atributo queueSet, um conjunto de filas, que pode armazenar qualquer objeto do tipo Queue[T].

### 3.3.4 Invariantes de história

As descrições dos estados e operações, e suas respectivas transições são as propriedades de segurança, ou seja, aquelas que afirmam que algo ruim nunca acontece, isto é, o sistema nunca entra em um estado não aceitável (VASCONCELOS, 1989). Propriedades de vivacidade garantem a ocorrência de operações, probidade e terminação. Probidade garante eventualmente a ocorrência de uma operação e terminação garante eventualmente o estado onde nenhuma operação está ativa.

Object-Z permite a especificação de propriedades de vivacidade associando a classe a uma lógica temporal através de invariantes de história. Essas invariantes de história restringem o conjunto de histórias derivado do estado e operações da classe, ou seja, são predicados que descrevem uma propriedade invariante das histórias de uma classe.

Com o uso de lógica temporal para descrever as invariantes de história, adiciona-se quatro operadores para o Object-Z, os quais são $\Box$(sempre), $\Diamond$(eventualmente), **enabled** (ativado), **occurs** (ocorre). O operador $\Box$ indica que um predicado é verdadeiro no presente e sempre verdadeiro no futuro. O operador $\Diamond$ indica que um predicado é verdadeiro no presente ou em algum momento no futuro. Os operadores **enabled** e **occurs** são usados em conjunto com operações, onde **enabled** significa que a operação está ativada, ou seja, as pré-condições são satisfeitas, e **occurs** que a operação é o próximo evento.

Sintaticamente, as invariantes de história são adicionadas ao final do esquema de classe, separadas do restante por um traço. Um exemplo do uso de invariantes é

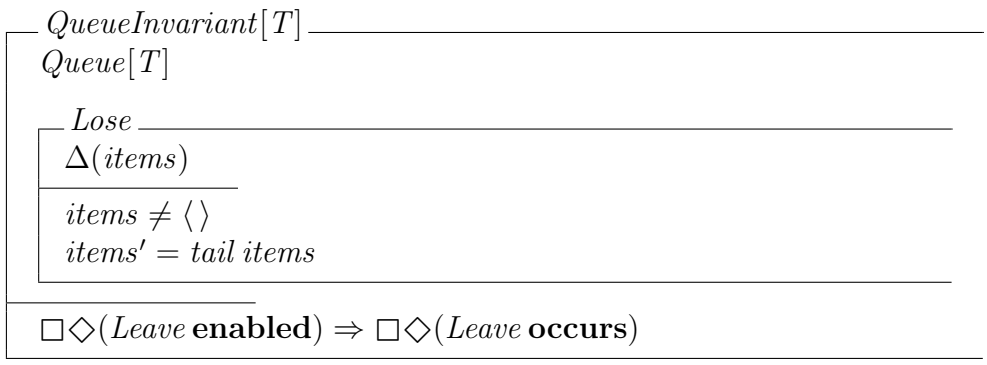


Figura 3.5: Invariantes de história

observado na figura 3.5. Nesta figura, a operação Lose admite a perda do primeiro elemento em qualquer tempo. Definindo uma invariante de história para a operação Leave, é garantido que não há perda infinitas da cabeça da fila, pois sempre, em algum momento, a operação Leave ocorrerá.

### 3.3.5 Instanciação de objetos

Como em Object-Z uma classe pode ser usada como um tipo, conseqüentemente um objeto pode ser declarado como uma instância de uma classe. Por exemplo, a declaração c: C, onde C é uma classe, especifica um objeto c do tipo C. Para aplicar operações a um objeto é utilizada a notação de ponto (.). Outro operador é o paralelo (‖) usado para especificar comunicação entre objetos.

Uma das principais vantagens em instanciar objetos é a possibilidade de agregação, ou seja, agrupar objetos de uma mesma classe. Há dois conceitos em Object-Z para o uso de agregação: os esquemas *framing* ($\Phi$) e o operador *nesting* ($\bullet$). O primeiro são esquemas onde não há operações, mas podem ser combinados com outros esquemas para definir operações. O segundo permite variáveis declaradas em esquemas *framing* serem utilizadas em outros esquemas. Para maiores detalhes consultar Smith (1992).

## 3.4 Validação de modelos

A validação de um modelo para simulação é medida pela proximidade entre os resultados obtidos pelo modelo e os originados do sistema real. Esta tarefa é com-

plexa, pois os modelos introduzem simplificações para representar os sistemas reais e, além disso, na maioria dos casos, os dados do sistema real não estão disponíveis ou não existem para uma comparação.

Devido a essa dificuldade de validação, há três aspectos que podem ser considerados para alcançar um nível de confiança em uma simulação (FILHO, 2001). Primeiro, avaliar as simplificações e as suposições adotadas na modelagem do sistema. Segundo, avaliar os parâmetros de entradas de dados e distribuições utilizadas para aspectos de aleatoriedade. Terceiro, avaliar as análises e conclusões formuladas sobre os resultados obtidos.

Uma outra alternativa para validar modelos é observar e analisar sua aplicação sobre cenários reais. A aplicação permite visualizar problemas e inconsistências na modelagem, pois não há como modelar um cenário sem reflexão durante o processo de modelagem.

## 3.5 Projetos Relacionados

### 3.5.1 Ubiwise

Ubiwise (BARTON; VIJAYRAGHAVAN, 2002) é um simulador para computação ubíqua, que concentra sua atuação na computação e comunicação de dispositivos situados em ambientes físicos. Ubiwise tem a finalidade de simular hardware e software de computação ubíqua com o objetivo de agilizar o processo de testes em pesquisas.

A construção do simulador foi baseada em requisitos e desafios encontrados em ambientes ubíquos, entre os quais destacam-se a possibilidade de experimentar novos conceitos, desenvolvimento de novos serviços, criação acelerada de protótipos, testes para novos sistemas, avaliação da interação entre diversos dispositivos e aplicações.

O simulador apresenta duas visões durante o processo de simulação. Uma visão, em 3D, é a representação do ambiente físico, e a outra visão, em 2D, são os objetos e dispositivos que o usuário pode manipular. Para simular o ambiente físico foi utilizado o motor gráfico de renderização do Quake-Arena III, enquanto para simular os protótipos foi desenvolvido um programa em Java. A sincronização entre as visões ocorre através de mensagens em rede, logo pode ser executado em uma única ou

várias máquinas distintas.

O programa que controla a visão do ambiente é denominado Ubisim e o que controla os dispositivos é denominado Wise. O autor optou por essa divisão para simplificar o processo de simulação, baseado nas premissas que o motor gráfico do Quake alcança alto desempenho e fidelidade visual, enquanto para prototipação, os aplicativos desenvolvidos em Java são mais robustos, rápidos para se desenvolver e podem usar todas as vantagens das bibliotecas existentes.

Uma visão do simulador pode ser observada na figura 3.6. Nesta figura, estão representadas as duas visões discutidas anteriormente e dois usuários distintos, onde cada um possui uma visão do Wise e do Ubisim. No Ubisim, os dispositivos estão organizados em 3D e respondem a eventos físicos, enquanto no Wise, os dispositivos estão organizados em 2D e respondem a eventos do mouse e de rede. A sincronização e a integração entre as visões e entre os ambientes ocorrem através do UbiSim Server, um servidor web.

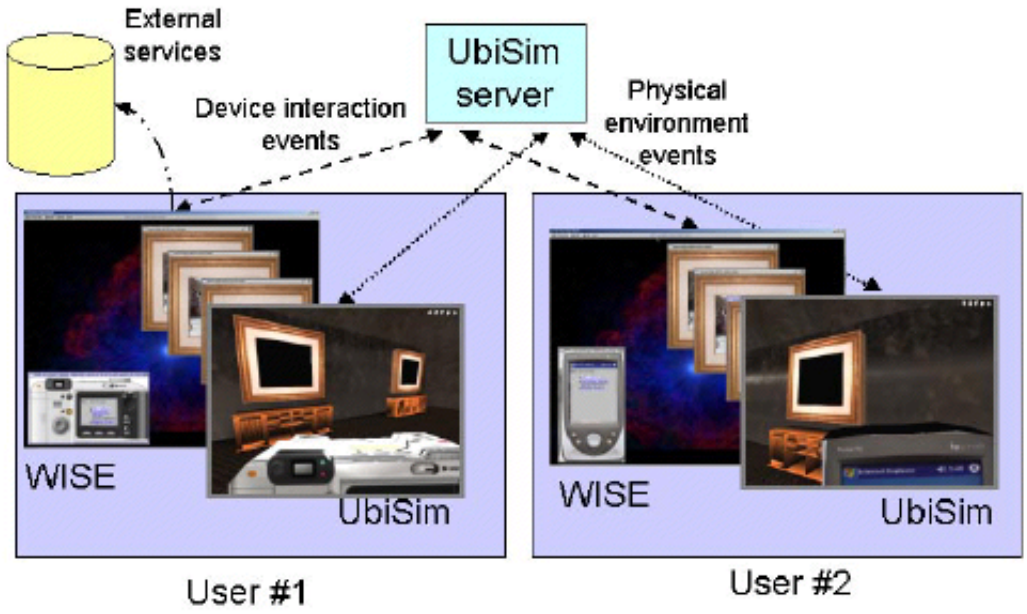


Figura 3.6: Arquitetura do Ubiwise (BARTON; VIJAYRAGHAVAN, 2002)

Para desenvolver uma simulação são necessários diversos passos. A seguir são apresentadas, em tópicos, as etapas para criação de uma simulação. As três primeiras referem-se à programação do Ubisim e as restantes à programação do Wise.

- Criar os objetos e dispositivos: consiste em criar as representações dos dispositivos para ambas visões do Ubiwise. O processo se inicia com uma imagem

digital, como por exemplo uma foto, a qual deve ser convertida para cada visão. Para a visão do Ubisim é necessário criar um modelo 3D wire-frame e para o Wise uma visão 2D da imagem.

- Criar o ambiente: consiste em criar um ambiente e especificar a posição inicial dos objetos. O ambiente pode ser construído modificando modelos pré-existentes com o uso de uma ferramenta (GtkRadiant). Há muitos ambientes disponíveis para a ferramenta, mas mesmo assim, a modificação é um trabalho dispendioso e complexo.

- Adicionar e compilar o código de interação física: consiste em programar o código para simular o comportamento dos dispositivos no ambiente. Para tal, deve-se modificar as bibliotecas de ligação dinâmica do sistema do Quake para que a simulação se comporte conforme programado.

- Criar o arquivo de descrição do dispositivo: consiste em especificar através de um arquivo XML os dispositivos utilizados na simulação. Neste arquivo são especificados informações referentes à interface com o usuário, armazenamento de dados, comunicações em rede, ligação com o código responsável por protocolos e controle, entre outros.

- Definir eventos e código: consiste em programar eventos e códigos para o dispositivo simulado. Eventos podem ser, por exemplo, cliques sobre botões do dispositivo, e o código pode ser, por exemplo, o protocolo de comunicação com um outro dispositivo.

- Programar o servidor web: consiste em programar serviços no servidor para um dispositivo, visto que estes são apenas protótipos, as funções do dispositivo poderiam ser emuladas por serviços no servidor. Esta tarefa é opcional, ou seja, depende do tipo de aplicação desenvolvida. Um exemplo seria a utilização do servidor para fornecer as fotos para um dispositivo câmera.

Finalmente, combinando as duas visões, isto é, executando os dois programas, é possível executar a simulação e avaliar o resultado. O Ubiwise não trabalha através de estatística de variáveis, logo os resultados observados são referentes à interação e usabilidade de protocolos.

### 3.5.2 UbisWorld

UbisWorld (HECKMANN, 2001, 2005) é um projeto coordenado por Dominik Heckmann, Universidade de Saarland, Alemanha, cuja finalidade é representar partes do mundo real e servir como um modelo de mundo virtual para representar, simular e comparar questões de pesquisa em computação ubíqua. Pode ser empregado para representar pessoas, objetos, localizações, tempo, eventos e suas propriedades e características.

A característica principal do projeto é a utilização de ontologias para representação dos elementos do mundo real e suas interações. Em UbisWorld é utilizada uma divisão em seis ontologias que se complementam, denominadas ontologia física, ontologia espacial, ontologia temporal, ontologia de atividade, ontologia de situação e ontologia de inferência. Essas ontologias estão disponíveis em diversas linguagens de representação, tais como RDF, OIL, Daml e OWL.

Além das ontologias (classes e predicados), os conceitos de UbisWorld ainda são compostos por mais dois elementos, os quais são indivíduos e relações, conforme pode ser conferido na figura 3.7. Os indivíduos são instâncias das classes e as relações, em sua maioria enárias, são responsáveis pelas associações entre esses indivíduos.

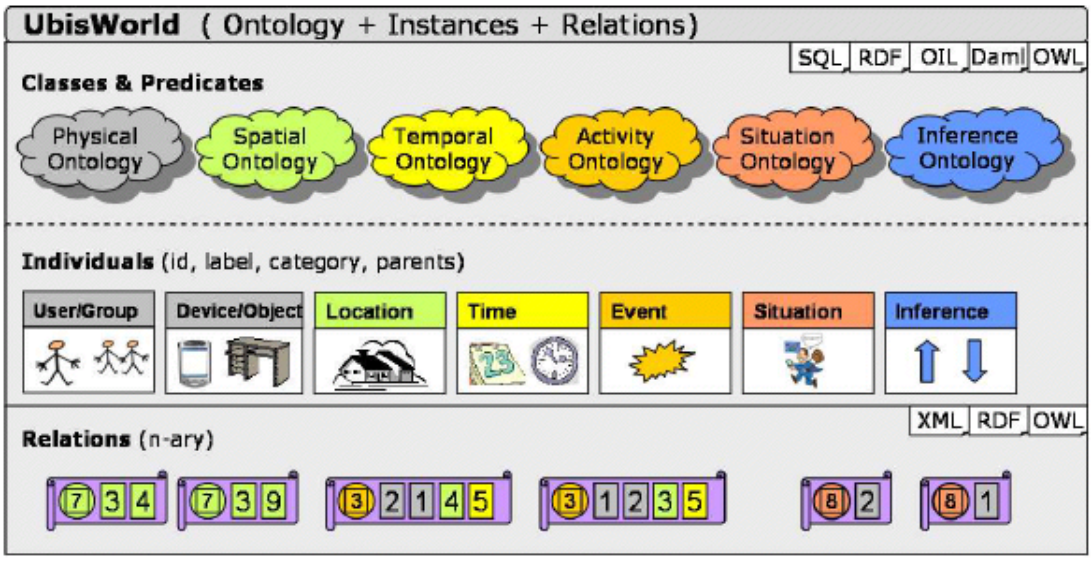


Figura 3.7: Visão da ontologia de UbisWorld (HECKMANN, 2005)

Ontologia física representa os elementos físicos. Esses elementos são classificados como elementos básicos (Basic Element) e elementos agrupados (Grouped Element). Os elementos básicos são divididos em três categorias, ser (Being), coisas (Thing) e

categoria (Category), e os elementos agrupados em grupo do usuário (UserGroup), grupo de sistema (SystemGroup) e grupo misto (MixedGroup). Importantes elementos desses grupos são usuário (User) derivado de ser (Being), dispositivo (Device), mobília (Furniture), veiculo (Vehicle) e outros objetos (Other Objects) derivados de coisas (Thing). Esses elementos ainda possuem mais especificações que detalham com maior precisão a descrição do mundo físico. O elemento categoria (Category) é uma classe onde são herdados o modelo de interesse e as preferências do usuário, como por exemplo esporte, música, ambientes e assim por diante. Os elementos de agrupamento são utilizados para agrupar instâncias relacionadas. A característica mais relevante dessa hierarquia é a definição de herança das propriedades das instâncias; para tal a ontologia suporta herança múltipla.

Ontologia espacial representa localização, topologia e orientação dos objetos e dispositivos físicos. Essa ontologia trata de diversos conceitos espaciais. O primeiro conceito é o nível de granularidade, ou seja, quão precisa é a localização. Por exemplo, um usuário pode estar em uma cidade, rua, edifício, quarto, ou apenas em uma parte do quarto. Outra maneira de endereçar essa questão é especificar a posição do usuário em relação a todas essas localizações. A definição da granularidade está relacionada à interação do usuário com o ambiente. O segundo conceito é a diferenciação de modelos de localização simbólicos (qualitativos) e geométricos (quantitativos). Os autores propõem um modelo híbrido, onde combinam a topologia simbólica e as suas coordenadas em um mapa do ambiente. A ontologia espacial do UbisWorld é baseada no projeto Nexus, onde o modelo do mundo é decomposto hierarquicamente em áreas. O modelo espacial tem dois elementos básicos, localização (Location) e restrições espaciais (SpatialConstraints). A localização pode ser dada em função de um lugar (nível de granularidade), de uma área (extensão), proximidade de um objeto ou localização relativa, como por exemplo, entre dois objetos específicos. As restrições espaciais contêm conexão (Connection) - relaciona duas localidades, aninhamento (Nesting) - relaciona o agrupamento de uma localização por outra, referência em mapa (Map Reference) - relaciona uma localização a um mapa, restrição (Constraint) - relações arbitrárias, e orientação (Orientation) - define a localização em relação a orientação ocular de uma entidade.

Ontologia temporal representa pontos no tempo, intervalos e restrições temporais sobre as situações em ambientes ubíquos. Foi desenvolvido um modelo híbrido para endereçar a granularidade temporal e especificar marcações precisas. Os ele-

mentos temporais são divididos em tempo (Time) e restrições temporais e cronológicas (TemporalConstraints & Chronology). O elemento tempo pode ser um ponto no tempo (Point of Time) - relaciona granularidade temporal, um intervalo (Time Interval) - tempo inicial com duração determinada, um evento (Time by Event) - baseado na ocorrência de um evento, ou uma relação (Time by Relation) - define o tempo baseado em relações temporais. As restrições temporais podem ser de ordem parcial (Parcial Order), aninhamento (Nesting) e restrições arbitrárias (Constraint).

Ontologia de atividade representa as mudanças no mundo, como por exemplo, a mudança de localização. Tem papel importante na descrição das ações sobre o mundo, mas não na definição. Exemplos de elementos dessa ontologia são os eventos, como por exemplo, mover (Move), pegar (Take), dar (Give), colocar (Put), alterar (Change). Cada um desses elementos atua, de alguma maneira, sobre as ontologias anteriores, modificando o que fora definido e conduzindo para uma nova definição.

Ontologia de situação representa atributos, parâmetros ou propriedades sobre usuários, sistemas, localização e atividades, ou seja, é utilizada para descrever o mundo. Como o UbisWorld tem seu foco dirigido para modelagem de usuário, a especificação dos elementos do usuário é detalhada e possui informações sobre contato, demografia, habilidade e proficiência, personalidade, características pessoais, estado emocional, estado fisiológico, estado mental, movimentação, papel, nutrição, entre outras. Ainda há elementos para descrever informação de contexto, tais como localização, ambiente físico e social, produto. Há outros tipos de elementos, como os sensores de dados de nível baixo, os quais definem características mais imperceptíveis nos ambientes.

Ontologia de inferência representa os elementos que definem as regras dentro de ambientes inteligentes. Essas regras permitem comportamento pró-ativo dentro do UbisWorld. Esta parte da ontologia está em desenvolvimento, porém um exemplo de elemento é o relembrar (Remind), o qual é responsável por executar um evento toda vez que uma determinada situação ocorre.

Uma propriedade interessante da ontologia do UbisWorld são os mecanismos para tratar com a questão da privacidade da informação. Os mecanismos são definidos quanto ao acesso, propósito e retenção. O acesso define quem terá direitos sobre a informação, a qual pode ser por exemplo, privada, pública ou compartilhada. O

propósito define a relevância da informação, como por exemplo, comercial, pesquisa ou mínima. A retenção define quanto tempo a informação deve ser mantida antes de ser eliminada, por exemplo, anos, meses ou dias.

# 4 Especificação do Modelo

Para o estudo e a construção de qualquer artefato de software ou hardware para o desenvolvimento de um sistema, uma das primeiras providências é entender a dinâmica do comportamento e as restrições inerentes a esse sistema. Abstrair esses dados não é trivial, principalmente se o sistema em questão é complexo.

As especificações formais usam notação matemática para descrever de forma precisa as propriedades que um sistema computacional deve possuir, sem necessariamente definir como estas propriedades são alcançadas, isto é, elas descrevem o que o sistema deve fazer sem dizer como é para ser feito (SPIVEY, 1989).

Como o presente trabalho se propõe a estudar as questões de modelagem envolvidas em ambientes de computação ubíqua, este capítulo apresenta um modelo que especifica as questões importantes para o assunto estudado, utilizando uma linguagem de especificação formal.

Por meio da modelagem e abstração dos elementos em ambientes de computação ubíqua, visando à simulação das situações, os aspectos de modelagem e simulação são apresentados, detalhados e discutidos informalmente, através de sentenças explicativas, e formalmente, através do modelo em Object-Z.

## 4.1 Visão Geral

A fundamentação apresentada no capítulo 2 constitui a base para a especificação do modelo abordado neste capítulo. Essa fundamentação é utilizada para a construção de um modelo genérico para que um ambiente de computação ubíqua possa ser modelado e simulado computacionalmente. Esta seção apresenta uma visão geral e os termos utilizados para se referir aos principais componentes do modelo.

O modelo identifica oito tipos básicos de elementos, conforme pode ser observado no diagrama 4.1. O diagrama omite determinadas classes e alguns relacionamentos existentes no modelo com o objetivo de apresentar somente as relações mais importantes e fornecer uma visão geral da interação entre esses elementos.

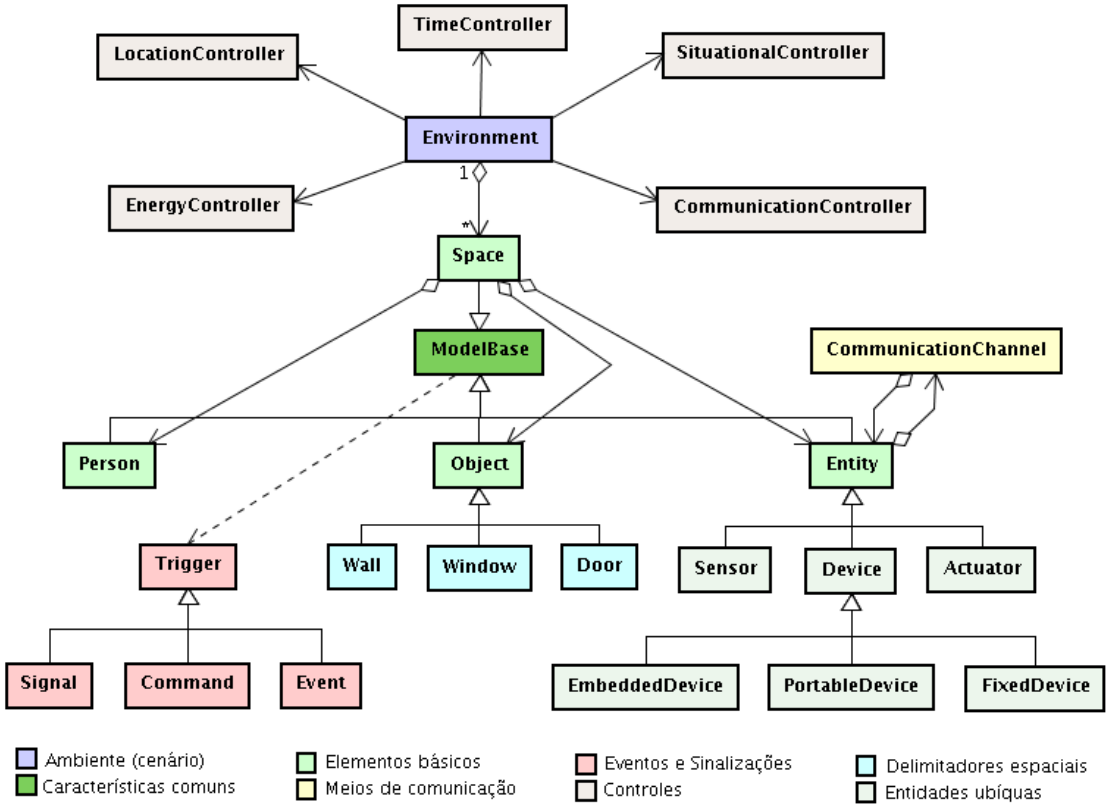


Figura 4.1: Principais componentes da especificação

A notação dos relacionamentos corresponde à notação utilizada nos diagramas de classe em UML (*Unified Modeling Language*) com algumas semânticas redefinidas. São utilizadas as notações de herança, associação, agregação e dependência. As caixas representam as classes essenciais do modelo, herança representa a expansão das classes, a associação representa a ligação entre as classes e a dependência indica a participação direta ou indireta sobre o estado das classes.

Apesar de toda a definição dessa semântica, o aspecto mais relevante do diagrama são as composições, isto é, o que existe na especificação. Há oito tipos de elementos, os quais, auxiliados por outros tipos não presentes no diagrama, permitem a construção de modelos que podem ser simulados através de estruturas computacionais.

No diagrama, o ambiente (roxo) se refere aos cenários de computação ubíqua,

portanto todos os outros elementos fazem parte de sua composição. As características comuns (verde escuro) definem as propriedades compartilhadas pelos principais elementos físicos que compõem o ambiente. Os elementos básicos (verde claro) são os elementos físicos presentes nos ambientes. Meios de comunicação (amarelo) especificam as comunicações entre as entidades. Eventos e sinalizações (rosa) são responsáveis por um tipo especial de comunicação e transporte de informação entre os elementos. Os controles (cinza) são estruturas para tratar questões de gerenciamento de energia, localização, comunicação, tempo e situações no modelo. Os delimitadores espaciais (azul claro) são objetos que possuem propriedades para definir a delimitação dos espaços físicos. As entidades ubíquas (gelo) correspondem às entidades que possuem capacidade de computação e comunicação.

Todas as classes do modelo são descritas com maiores detalhes em tópicos específicos. Para tal, a organização da especificação é apresentada em partes, partindo-se da definição dos tipos básicos até a completa definição do ambiente. Desse modo, o modelo torna-se mais compreensivo, pois se inicia na base até alcançar o todo.

## 4.2 Tipos básicos e definições axiomáticas

A linguagem formal Z permite a especificação de tipos básicos através da declaração do tipo no documento de especificação. Estes tipos básicos são globalmente definidos e seu escopo se estende da definição até o final da especificação. Uma das vantagens de se usar os tipos básicos é a não necessidade de descrever através de estruturas matemáticas ou formais o seu significado, facilitando assim a especificação e não associando o tipo a uma estrutura específica. As definições axiomáticas permitem especificar tipos através de declarações formais e atribuir restrições aos valores especificados. Esta seção apresenta e descreve os tipos básicos e as definições axiomáticas utilizadas na especificação.

Há sete tipos básicos definidos, conforme pode ser observado pela especificação a seguir:

$$[Text, Time, Date, Shape, Data, Action, State]$$

O tipo **Text** representa um conjunto de caracteres para descrever através de linguagem natural algum tipo de propriedade ou característica, ou para ser utilizado

como um identificador. Foi estabelecido com o objetivo de permitir a existência de atributos descritivos nas classes e para rotular e identificar associações através de um nome específico.

Os tipos **Time** e **Date** representam marcadores de tempo. **Time** corresponde a hora, minutos, segundos e centésimos de segundo, enquanto **Date** corresponde a dia, mês e ano. A granularidade foi assim definida para representar com precisão os acontecimentos dependentes do tempo.

O tipo **Shape** representa alguma forma geométrica plana, como por exemplo, retas, círculos, quadrados, triângulos, arcos, ou outro polígono qualquer. Tem por objetivo estabelecer limites para forma de propagação e alcance de comunicação, delimitação de acesso, entre outros.

O tipo **Data** representa qualquer tipo de informação digital. Pode representar uma figura, um texto, um vídeo, dados de áudio, um número, um vetor de números, enfim, toda informação que pode ser representada por uma estrutura de dado.

O tipo **Action** representa uma ação que modifica o estado de um objeto físico. Uma ação pode ser programada para, por exemplo, fechar uma porta, acender uma lâmpada, recolher um tapete, e assim por diante. A ação representa a execução de uma tarefa.

O tipo **State** representa os estados que os elementos possuem ou podem assumir. Exemplificando, no caso de uma lâmpada, os estados podem ser acesa ou apagada, no caso de uma porta, aberta ou fechada, e em exemplos mais complexos, no caso de dispositivos, podem ser valores em registradores específicos, os quais não podem ser descritos devido a diversidade, mas observados em instantes no tempo. O tipo **State** é o tipo básico mais complexo, pois sua sintaxe e semântica são definidas de acordo com o elemento ao qual se refere.

Há duas definições axiomáticas globais:

$$\begin{aligned} &Properties : Text \pfun Data \\ &Direction : \{North, South, East, West, Northeast, Northwest, \\ &\qquad Southeast, Southwest\} \end{aligned}$$

O tipo **Properties**, representado por uma função parcial, associa um identificador a uma informação. Através dessa associação é possível acessar os dados através

de um rótulo nomeado e que identifique adequadamente o que está sendo referenciado. Por exemplo, pode-se definir uma função $f$ : *Properties*, que associa nomes com imagens. Assim, para referenciar a imagem de Ana, tem-se $f(Ana)$.

O tipo **Direction** é um conjunto que define vetores para orientação no ambiente. Os vetores definidos são baseados nos pontos cardeais (norte, sul, leste e oeste) e nos pontos colaterais (noroeste, nordeste, sudoeste e sudeste). Desse modo, a orientação define oito direções sobre um plano. Apesar da simplicidade, através dessa orientação um elemento pode se mover para qualquer ponto em um espaço definido.

## 4.3 Características comuns

Muitas propriedades dos elementos físicos que compõem os cenários em um ambiente de computação ubíqua são comuns. Utilizando o conceito de herança, as características comuns foram extraídas e agrupadas em uma única classe. A classe **ModelBase** define as propriedades comuns de quatro elementos do modelo: pessoas, objetos, entidades e espaços.

*ModelBase*

$ClassType : \{Person, Object, Entity, Space\}$

$identification : \mathbb{N}$
$class : ClassType$
$description : Text$
$position : \mathbb{N} \times \mathbb{N} \times \mathbb{N}$
$mass : \mathbb{R}_1$
$width : \mathbb{N}$
$height : \mathbb{N}$
$shape : \mathbb{P}(\mathbb{R} \times \mathbb{R} \times \mathbb{R})$
$emittedSignal : \mathbb{F}\ Signal$
$location : Location$
$created : TimeStamp$

---

$\forall x, y : identification \bullet x \neq y$
$\forall x, y : identification;\ p1, p2 : position \mid x \neq y \bullet p1 \neq p2$

Figura 4.2: Características da classe ModelBase

A propriedade **identification** identifica unicamente um objeto físico no modelo, ou seja, cada instância tem uma identificação própria e única. A propriedade **class** especifica o tipo de instância a que se refere um objeto, no caso, um valor do tipo **ClassType**. A propriedade **description** permite inserir um texto descrevendo o objeto e suas características, mas tem caráter meramente informativo.

A propriedade **position** define a localização espacial absoluta de uma entidade física no cenário. Todo elemento que possui essa característica pode ser localizado através de coordenadas. Como o mundo real possui três dimensões, a base para as coordenadas é a base ortonormal, ou seja, são utilizados três valores para representar um ponto $(x, y, z)$. Entretanto, na simulação, para a maioria dos cenários, pode ser atribuído o valor nulo ao eixo $z$, tornando o ambiente simulado em duas dimensões ou planar. Desse modo, utilizando um sistema de posicionamento baseado no plano cartesiano, as simulações tornam-se mais simples e facilitam a visualização das interações.

Há quatro propriedades para representar forma e estrutura dos elementos. A propriedade **mass** define a massa em quilogramas de um elemento. Através dessa propriedade pode-se, por exemplo, indicar quais objetos podem ser portados por uma pessoa e definir restrições quanto ao peso. A propriedade **width** define a largura ou comprimento e a propriedade **height** define a altura de um elemento. Essas propriedades influenciam diretamente nas comunicações, pois, por exemplo, um feixe de infravermelho pode ser bloqueado pela presença de um corpo entre a fonte emissora e receptora. A propriedade **shape** define a forma espacial de um elemento. As formas são representadas por formas geométricas espaciais, isto é, por cilindros, cones, esferas, pirâmides, poliedros e prismas. Essas formas podem ser representadas por equações, o que facilita a identificação de pontos pertencentes a elas. Por exemplo, o espaço ocupado por uma pessoa (corpo) pode ser representado por um cilindro, um computador pode ser um cubo, e assim por diante.

Todos os elementos físicos podem emitir algum tipo de sinal. Esse sinal pode ser uma imagem, som, movimento, pressão, cor, e assim por diante. A propriedade **emittedSignal** tem a função de agregar e representar o conjunto de sinais que um elemento pode transmitir em uma simulação. São definidos apenas os sinais relevantes para a simulação em questão. Por exemplo, em um ambiente monitorado por câmeras, o sinal emitido deve ser uma imagem do elemento filmado. Maiores

detalhes sobre sinais são discutidos na seção 4.8.

As propriedades **location** e **created** representam, respectivamente, uma descrição mais detalhada da localização e a data e hora de criação de um elemento no ambiente.

## 4.4 Pessoas

Na seção 2.3.1 descrevemos brevemente a entidade pessoa e alguns exemplos de como sua participação em cenários de computação ubíqua é complexa. As características de autonomia, de mobilidade, de comportamentos e as relações que as pessoas possuem devem ser consideradas em simulação se é desejado alcançar um modelo que represente com fidelidade o que ocorre em ambientes ubíquos. Nesta seção, são descritas as características específicas e necessárias para modelagem de pessoas ou seres vivos.

A classe **Person**, através de seus atributos, representa as características que devem ser consideradas quando se modela uma pessoa para os cenários de computação ubíqua.

Há três tipos básicos definidos, os quais abstraem toda a essência da representação de uma pessoa para quem está modelando um ambiente ubíquo. Os tipos são **Positional**, **Physical** e **Psychological**.

O tipo **Positional** define as posições das pessoas no ambiente e as características que se relacionam com a posição, tais como velocidade, direção e orientação. O tipo **Physical** representa as características físicas, tais como sexo, idade, tamanho, peso, massa, pressão, temperatura, entre outros. O tipo **Psychological** representa características abstratas como emoções, necessidades, interesses, temperamento, enfim, características relacionadas ao psicológico.

A idéia é estabelecer uma relação entre as características definidas por esses tipos e produzir um comportamento baseado na individualidade de cada pessoa modelada para as situações a que são submetidas. Por exemplo, uma pessoa inicia uma comunicação com seu celular com uma central de informações. Se por algum motivo a comunicação demorar, influenciada talvez pela própria posição da entidade, e a pessoa possuir características que conduzem a impaciência, ela pode desistir

*Person*

*ModelBase*

$[Positional, Physical, Psychological]$

$Personality : \{Extraversion, Agreeableness, Conscientiousness, EmotionalStability, Openness\}$

$positional : \mathbb{P}\ Positional$
$physical : \mathbb{P}\ Physical$
$psychological : \mathbb{P}\ Psychological$

$speed : \mathbb{R}$
$direction : Direction$
$orientation : Direction$

$gender : \{male, female\}$
$age : \mathbb{N}$

$personality : Personality \rightarrow \mathbb{N}$

$hasObject : \mathbb{F}\ Object$
$hasEntity : \mathbb{F} \downarrow Entity$

---

$speed \in positional$
$direction \in positional$
$orientation \in positional$
$gender \in physical$
$age \in physical$
$personality \in psychological$
$\mathrm{ran}\ personality \leq 100$

Figura 4.3: Características da classe Person

da comunicação e não realizá-la. Extrair estatísticas sobre esses acontecimentos é importante em simulações.

A grosso modo, o comportamento de uma pessoa no modelo é representada pela seguinte relação:

$$\mathbb{P}\ Positional \times \mathbb{P}\ Physical \times \mathbb{P}\ Psychological \Rightarrow Behavior$$

Na relação acima **Behavior** indica comportamento, isto é, a relação entre as propriedades posicionais, as características físicas e as psicológicas resultam em um comportamento, que nada mais é que uma seqüência de ações, seja no mundo real ou no mundo simulado. Uma seqüência de ações, por sua vez, é movimentação (mo-

bilidade), tomada de decisões (autonomia) e interações com dispositivos (relações).

A classe **Person** em Object-Z apresenta as propriedades mais relevantes e exprime a possibilidade de acrescentar novas propriedades. As propriedades **positional**, **physical** e **psychological** representam genericamente o conjunto dessas propriedades.

As propriedades **speed**, **direction** e **orientation** referem-se à posição e locomoção de uma pessoa no ambiente. A primeira indica a velocidade de uma pessoa no cenário. Se a velocidade é nula, ela está em inércia com relação ao espaço. Como a velocidade pode ser relativa a outros objetos, ela sempre se refere ao ambiente mais restritivo. Por exemplo, se uma pessoa está em um carro, a velocidade com relação ao ambiente é a velocidade do carro, mas em relação ao carro é nula. Nesse exemplo, o objeto mais restritivo é o carro, logo a velocidade é nula. As outras duas propriedades se referem à direção e orientação dentro de um espaço. A direção define o caminho e a orientação a observação desse caminho. Em muitos casos são idênticas, pois as pessoas caminham observando o que está a frente.

As propriedades **gender** e **age** referem-se às características físicas, destacando-se massa, altura e forma, já definidas nas características comuns. A primeira define se a pessoa é do sexo feminino ou masculino e a segunda define a faixa etária, isto é, se é criança, adolescente, jovem, adulto ou idoso, por exemplo. Elas permitem inferências para definir comportamentos tais como interesses, emoções e até em questões quanto ao uso da tecnologia.

A propriedade **personality** está relacionada com as características psicológicas e tenta abstrair através do modelo dos Cinco Grandes Fatores (Big-Five Factors) (GOLDBERG, 1992) a representação da personalidade de uma pessoa. A personalidade é o conjunto de características psicológicas (cognitivas, afetivas, volitivas e físicas) relativamente estáveis que influenciam a maneira pela qual o indivíduo interage com o seu ambiente. Através da personalidade é possível prever, com um grau razoável de certeza, como um indivíduo pode reagir a determinadas situações.

O modelo dos Cinco Grandes Fatores é um dos modelos mais bem aceitos na teoria de traços de personalidade. Apresenta uma taxionomia baseada em cinco fatores:

- *extraversion* (extroversão): grau que um indivíduo pode tolerar de estimulação

sensitiva de pessoas e situações. Alta extroversão indica preferência em estar acompanhado por muitas pessoas e envolvido em muitas atividades. Baixa extroversão indica preferência para trabalhar sozinho, permanecer quieto e ser uma pessoa privada.

- *agreeableness* (socialização): tendência de ser socialmente agradável; nível de socialização. Alta socialização indica interesses pelas pessoas e pelos seus sentimentos. Baixa socialização indica agressividade, competição e pouca consideração pelo próximo.

- *conscientiousness* (escrupulosidade): grau de organização, persistência e motivação para alcançar um objetivo. Alto grau indica organização, confiança, ambição. Baixo grau indica negligência, não confiável e irresponsabilidade.

- *emotional stability* (estabilidade emocional): estado emocional, nível de estresse, controle emocional. Alto grau indica preocupação, insegurança e alta ansiedade. Baixo grau indica calma, segurança e estabilidade.

- *openness* (intelecto): percepção que um indivíduo tem de sua própria inteligência ou capacidade. Alto intelecto indica criatividade, curiosidade, imaginação. Baixo intelecto indica dificuldade para compreensão de idéias abstratas.

Os fatores estão associados a um percentual (0-100%), que indica o grau que os fatores se manifestam em uma instância de pessoa. A idéia é introduzir um nível de autonomia que possa representar na simulação toda a diversidade inerente ao ser humano e produzir, através de estudos mais aprofundados, os comportamentos reais de pessoas utilizando os ambientes de computação ubíqua. Uma questão a ser considerada é o uso de variáveis aleatórias para alcançar o mesmo resultado, entretanto seu uso pode ser muito mais incerto que o próprio modelo de personalidade.

Por fim, uma pessoa em ambientes de computação ubíqua se relaciona com dispositivos e objetos no ambiente. A relação mais próxima ocorre quando estes são portados pela pessoa. Nesse caso, as propriedades **hasObject** e **hasEntity** indicam quais são os objetos e as entidades que se encontram em posse da pessoa. Objetos podem ser relógios, óculos, tênis e entidades podem ser celulares, palmtops, sensores, e assim por diante. Essas propriedades agregam estes elementos e viabilizam o acesso exclusivo ao indivíduo.

## 4.5 Objetos

Neste trabalho, objetos são os elementos sem capacidade de computação. Apesar de não possuírem tais características, são importantes para a composição dos cenários. Objetos podem ser eletrodomésticos, mobílias, utilitários, roupas, entre outros. Por questões de definição, pode-se dizer que objetos são todos os elementos que não possuem computação e comunicação, a não ser que elas sejam agregadas ou embutidas a eles.

Há três papéis básicos que os objetos desempenham no mundo ubíquo. Primeiro, na mobilidade das entidades. Uma entidade não pode, por exemplo, atravessar um objeto, ela tem que desviar. Segundo, na transmissão dos sinais. Os objetos são uma fonte de interferência para as comunicações sem fio, principalmente a infravermelha. Por último, participando das interações com os dispositivos ubíquos, quando estão equipados com sensores, atuadores ou dispositivos embutidos.

As mudanças físicas no mundo ubíquo são perceptíveis diretamente nos objetos, pois são mudanças nos estados dos objetos. Exemplificando, uma lâmpada pode possuir dois estados: acesa ou apagada. Um evento no ambiente pode alterar o estado de acesa para apagada e vice-versa. Ou ainda, uma porta pode possuir os estados aberta, fechada, travada, destravada. Esses exemplos são simples, há casos em que a quantidade de estados é difícil de ser enumerada, como em um eletrodoméstico, por exemplo. Logo, em simulação, deve sempre ser considerado somente os estados de interesse para não complicar o cenário desnecessariamente.

A problemática de estabelecer os estados dos objetos não se restringe somente à quantidade, mas à especificidade desses estados. Não há como estabelecer um padrão de estados. No entanto, há algumas propriedades comuns evidentes e importantes para a simulação. Temos nesse caso a opacidade e a reflexibilidade. Ambas interferem na transmissão de informações via redes sem fio. Porém, considerar essas propriedades pode apenas aumentar a complexidade das simulações.

No modelo, a principal preocupação é com a mudança de estados e suas conseqüências. A figura 4.4 representa essa situação e as implicações da passagem de um estado para outro.

Na figura 4.4, um objeto é composto por um conjunto de valores que constituem seu estado atual. Ao ocorrer um evento, um desses valores é alterado. Neste caso

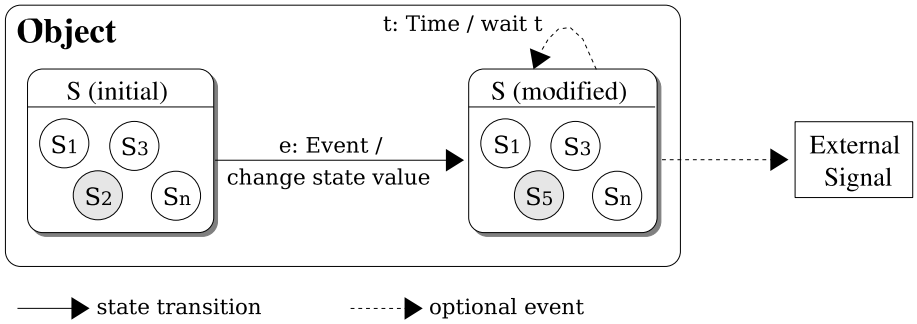


Figura 4.4: Transição de estados em objetos

em especial, o valor $S_2$ transita para $S_5$, conseqüentemente se obtém um novo estado para o objeto. Essa mudança pode implicar na criação de sinalização externa ou ainda em uma transição temporária, isto é, o estado alterado é mantido somente por um período de tempo definido. Avaliando o caso de uma lâmpada, se um evento transitar do estado de apagada para acesa, a lâmpada deve emitir um sinal externo de luminosidade, pois podem existir sensores que reagem e acionam outros mecanismos presentes no ambiente. Além de disparar outros eventos no mundo ubíquo, a mudança de estado pode ser temporária. Por exemplo, se a lâmpada deve permanecer no estado acesa somente durante 10 segundos, um evento interno ao objeto deve mudar esse estado após o esgotamento do tempo. Há casos onde não é necessário emitir nenhum tipo de sinalização interna ou externa. O acesso ao estado dos objetos é realizado através de uma consulta antes de seu uso.

Considerando as questões abordadas, a especificação de um objeto para simulação deve se preocupar com a movimentação no ambiente, interferência nas comunicações, propriedades e estados relevantes, emissão de eventos ou sinais para o mundo externo, associação com entidades (sensores, atuadores e dispositivos embutidos). A classe **Object** aborda essas questões ao apresentar um modelo genérico, estendendo a classe **ModelBase** e provendo atributos para gerenciar estados e associações.

Ao estender **ModelBase**, os objetos herdam propriedades de identificação, forma, localização e emissão de sinais. A propriedade **initialState** é um conjunto de variáveis de estado e define o estado atual do objeto. A propriedade **possibleState** é um conjunto de variáveis de estado e define todos os valores possíveis e válidos para a composição de qualquer estado do objeto. A propriedade **changeState** é uma função de associação entre os eventos que alteram os estados dos objetos com a

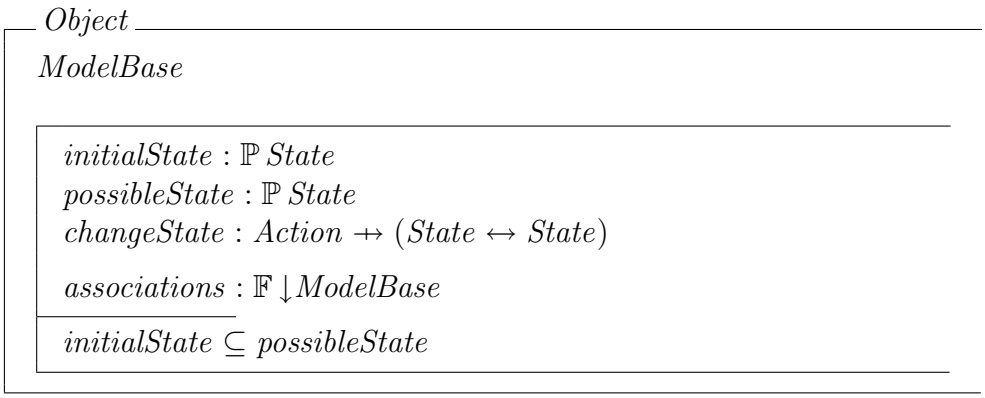


Figura 4.5: Características da classe Object

relação de transição entre o valor atual e o novo valor da variável de estado alterada pelo evento. A classe não possui propriedades descritivas, pois são específicas de determinados objetos.

Para atender o restante dos requisitos, a propriedade **associations** permite a associação de elementos aos objetos. Uma pessoa pode portar um objeto, seja uma roupa ou um relógio, logo é necessário saber quem o porta, pois interfere na localização e interações do objeto. Um objeto pode possuir um sensor e/ou atuador acoplado. Nesse caso, o sensor pode detectar um sinal e o atuador executar uma ação física sobre o objeto ou outros objetos. Por último, o objeto pode estar associado a um dispositivo embutido totalmente responsável pelo controle das funções do objeto. Maiores detalhes dessas associações são discutidos nas seções seguintes.

## 4.6 Entidades

No capítulo 2, entidade define os elementos (pessoas, dispositivos e softwares) responsáveis pela formação dos cenários de computação ubíqua. Na especificação, o termo entidade é usado em caráter mais restritivo, indicando de forma específica os elementos do mundo ubíquo que possuem características e capacidades de computação e/ou comunicação. Entidade é a base para a especificação dos sensores, atuadores e dispositivos. Essa divisão se deve ao fato da existência de características comuns entre os elementos modelados e a necessidade de fornecer uma estrutura que limite o acesso a determinadas propriedades.

A classe **Entity** define uma estrutura abstrata cuja função é fornecer proprie-

dades e restrições aos dispositivos ubíquos, sensores e atuadores. Essa dependência e a estrutura da especificação podem ser observados no diagrama 4.1. A figura 4.6 especifica as características de uma entidade.

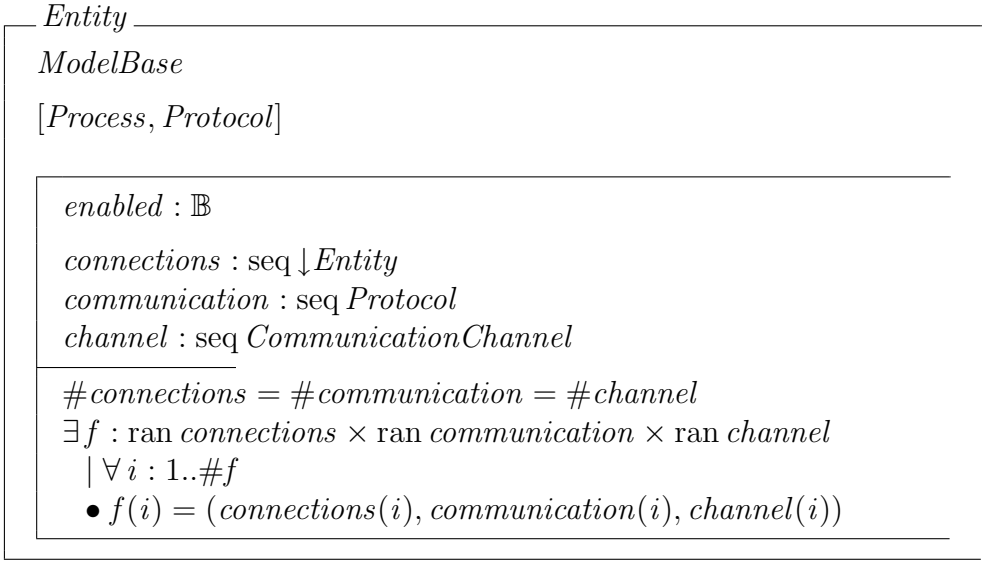


Figura 4.6: Características da classe Entity

A propriedade **enabled** define se a entidade está ativa ou inativa. A maioria das entidades são de alguma forma dependentes de energia ou possuem opções para indicar se estão ativas ou não. Por exemplo nos celulares, se a bateria acabar ou se estiverem desligados é impossível acessar as funções. Essa propriedade implica em restrições de acesso, o que pode interferir diretamente nas simulações, diversificando e complicando os cenários simulados.

Nem toda a entidade executa computação, mas toda entidade se comunica com outras entidades. Essa característica implica na especificação de um mecanismo para que comunicações sejam modeladas e possam ser simuladas. Para tal, há três propriedades, as quais são **connections**, **communication** e **channel**. A primeira define com quais entidades são mantidas conexões, a segunda especifica o protocolo e a terceira descreve o canal físico de comunicação.

A figura 4.7 apresenta uma entidade A que se conecta com outras entidades. Cada conexão é identificada por um valor numérico, iniciando em 1 (um). Como são várias as conexões possíveis, **connections** é representada pela seqüência das entidades que a entidade A está conectada. As conexões necessitam de um meio físico, seja através de conexão com ou sem fio. Logo, cada conexão possui seu próprio

meio representado por **channel**, onde o identificador da conexão se relaciona com o identificador de **channel**. Para representar as regras da comunicação, **protocol** identifica o protocolo para cada conexão utilizando o mesmo identificador de conexão e canal. Tudo isso é garantido por um predicado que define uma função $f$, cujo retorno é a entidade, o canal e o protocolo de qualquer comunicação sendo realizada.


Figura 4.7: Comunicação e conexões entre entidades

O tipo básico **Protocol** define as regras para a comunicação. Em outras palavras, define o sincronismo entre as entidades, como enviar e receber dados por um canal de comunicação, o pré e pós-processamento entre as comunicações. Através desse tipo, é possível simular qualquer protocolo de comunicação entre entidades ubíquas. Não foram estabelecidos protocolos padrões, logo todo protocolo deve ser criado pelo responsável pela simulação. Com o tempo, espera-se criar uma biblioteca com protocolos para reutilização nas simulações.

A especificação de **Entity** não pode ser utilizada diretamente, ela apenas define as características comuns das entidades. Para utilizar os dispositivos ubíquos na simulação é necessário conhecer maiores detalhes de suas especificações. As próximas subseções apresentam os modelos específicos das entidades para instanciação.

### 4.6.1 Sensores

Sensor é a designação comum dada aos dispositivos que percebem e reagem de alguma forma à presença de um determinado tipo de sinal. Devido aos diferentes tipos de sinais, não é possível enumerar todos os sensores existentes, mas como exemplos, pode-se citar os sensores de temperatura, umidade, luminosidade e pressão.

Para especificar um sensor é necessário entender quais são seus componentes e como ele se comporta. Em um ambiente de computação ubíqua há três elementos participando em uma interação de detecção de sinal. Os elementos são a fonte geradora de sinal, responsável por emitir um sinal (áudio, video, calor, pressão, ...), o sinal propriamente dito e o sensor (figura 4.8).

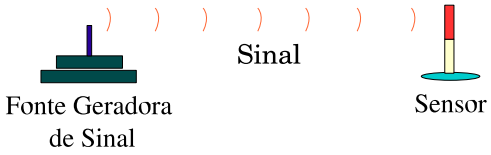


Figura 4.8: Percepção de um sinal por um sensor

A figura 4.8 ilustra a situação de percepção de um sinal por um sensor baseada em cinco passos:

1. Uma fonte (pessoa, objeto, entidade) gera um sinal, por exemplo, ondas eletromagnéticas.
2. O sinal se propaga para uma, várias ou todas as direções.
3. O sensor detecta e reage ao sinal.
4. O estado interno do sensor é modificado temporariamente ou permanentemente.
5. O sensor alerta a mudança de estado às entidades interessadas.

Todos os passos anteriores são os passos referentes ao que realmente ocorre no mundo real, exceto o item 5, pois nem sempre o sensor alerta a mudança de estado. Em geral, há processos que monitoram constantemente os estados do sensor verificando mudanças. Por questões de simulação, vamos sempre assumir que o sensor emite uma notificação de alteração de estado.

Baseado nesses passos é proposta a especificação para o sensor (figura 4.9).

A classe **Sensor** herda todas as características de **ModelBase** e **Entity**. As informações de forma não são importantes, mas as questões de identificação, posição e comunicação são necessárias. A propriedade **type** especifica uma identificação numérica para tratar sensores que reagem simultaneamente ao mesmo sinal.

Sensores só identificam um sinal se este estiver ao alcance de sua cobertura. As propriedades **perceptionShape** e **perceptionDistance** definem a forma e a distância que um determinado sensor percebe a presença de um sinal. A forma pode ser circular, ou seja, cobre toda a região plana, pode ser omnidirecional, linear e assim por diante.

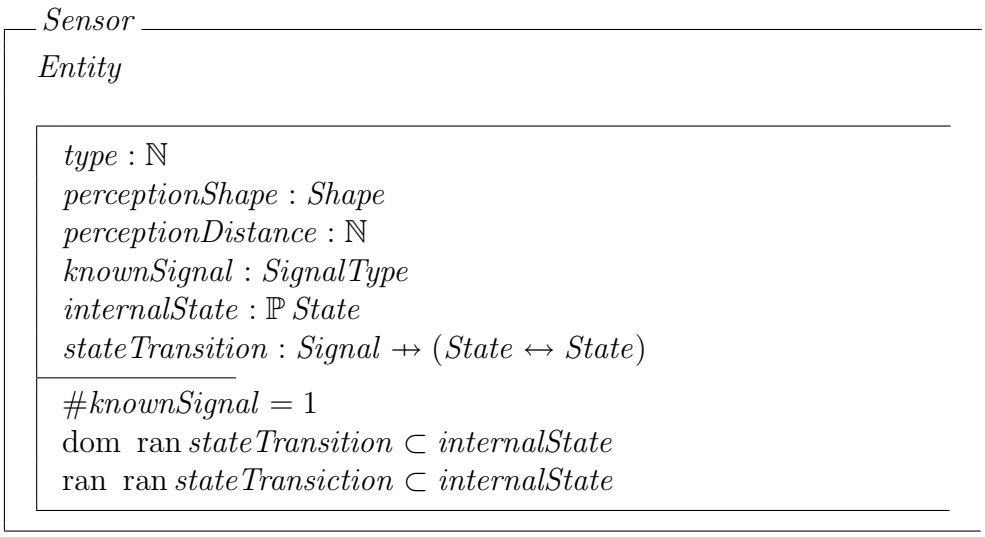


Figura 4.9: Características da classe Sensor

A propriedade **knownSignal** especifica o tipo de sinal reconhecido. Um sensor reconhece no máximo um tipo de sinal. Portanto, para reconhecer vários sinais diferentes são necessários o número de sensores equivalente ao número de tipos de sinais.

Como foi discutido, um sensor ao reconhecer um sinal modifica seu estado interno. Essa modificação é utilizada no modelo para notificar as entidades interessadas. Para tal, há duas propriedades que tratam das variáveis internas do sensor e da transição de estados ao reconhecer um sinal, as quais são **internalState** e **stateTransition**. Essas propriedades controlam os sinais e permitem, em tempo de simulação, avaliar o que está ocorrendo num dado instante.

### 4.6.2 Atuadores

Ao contrário dos sensores, o atuador não reconhece sinais, apenas executa ações sobre objetos no ambiente. São exemplos de ações executadas por atuadores: acender, apagar, travar, destravar, ligar, desligar, acionar, entre outras. Todas essas ações influenciam fisicamente no estado dos objetos reais do ambiente. Por exemplo, para travar a porta do carro, um atuador vai acionar um mecanismo, no caso uma trava, para fisicamente alterar o estado da porta de destravada para travada.

Como os atuadores apenas executam tarefas, a descrição de seu comportamento se resume a dois passos:

1. Recepção de um comando.

2. Execução da ação associada ao comando.

Um atuador pode executar diversas ações sobre diferentes objetos. O primeiro passo consiste na recepção de uma instrução com parâmetros para informar ao atuador qual ação executar e sobre qual objeto. O segundo passo consiste em executar a ação sobre o objeto, ou seja, alterar o estado interno de um objeto.

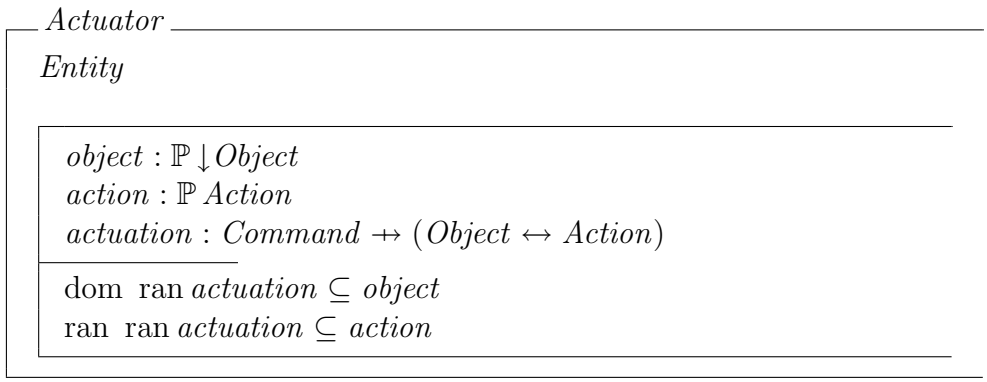


Figura 4.10: Características da classe Actuator

A classe **Actuator** define as propriedades **object**, **action** e **actuation**. A primeira define o conjunto de objetos que o atuador tem acesso, a segunda define as ações que o atuador pode executar e a terceira é uma função que associa um comando, recebido por uma conexão, a um objeto e a ação que modifica seu estado. A execução de uma ação, apesar de ser definida e executada pelo atuador, na simulação sempre é responsabilidade do objeto, pois a mudança de estados em objetos pode ter implicações sobre o mundo ubíquo, como foi apresentado na figura 4.4.

Ilustrando a situação anterior com o exemplo da porta do carro, suponha que o atuador receba um comando *lock_door*. Ao receber o comando, ele consulta através da função $actuation(lock\_door)$ e obtém uma tupla com o objeto e a ação sobre o objeto, por exemplo $(carX.door, lock)$. Ao identificar essa tupla, ele emite um evento para o objeto *door* para que o próprio objeto altere a variável de estado para travada. Se há eventos associados ao travamento da porta, o próprio objeto deve emitir um evento externo. Por exemplo, acionar um sinal sonoro para sinalizar que a porta está trancada e travada.

### 4.6.3 Dispositivos

Dispositivos são os elementos que possuem capacidade de computação e comunicação, os quais ao lado de sensores, atuadores e meios de comunicação, definem a base tecnológica para a criação de ambientes de computação ubíqua. Este tópico trata da especificação dos dispositivos ubíquos, considerando a existência de três classes de dispositivos: dispositivos embutidos, dispositivos portáteis e dispositivos fixos.

Independente da classe do dispositivo, há duas características comuns que mais se destacam, as quais são a capacidade de se comunicar com outros dispositivos e a capacidade de executar computação. A especificação deve definir maneiras para que a comunicação e a computação possam ser modeladas e simuladas.

Comunicação significa utilizar um canal de comunicação, um protocolo e estabelecer conexões com outras entidades que tenham a mesma capacidade. Por computação se entende a entrada, processamento e saída de dados. Não é a definição mais completa, mas abstrair dessa maneira nos permite presumir novas características presentes em dispositivos.

A entrada se refere ao processo de fornecer dados para o dispositivo, em especial para um processo. Os dados podem ser fornecidos por sensores, via teclado, voz, gestos, pela rede, entre outros. O processamento se refere às operações realizadas sobre os dados de entrada. Essas operações são definidas e programadas para atender as necessidades do usuário. A saída é o resultado obtido após o processamento. As saídas podem ser apresentadas na tela, papel, através de áudio e/ou vídeo, transmitidas pela rede, entre outros.

Avaliando a descrição anterior, observa-se a presença de dispositivos que fornecem dados de entrada e dispositivos que exibem dados de saída. Esses dispositivos devem ser levados em consideração na modelagem, pois através deles é possível a interação com os dispositivos ubíquos. São exemplos desses dispositivos os teclados, mouses, microfones, monitores, impressoras, caixas acústicas, entre outros.

Se há processamento em dispositivos ubíquos, há processos. Os processos controlam um conjunto finito de recursos e executam processamento sobre os dados. Por sua vez, há variáveis de estado que definem o estado atual do dispositivo e podem ser alteradas por processos ou por eventos. Processos podem armazenar e recuperar

informações internas ao dispositivo, logo temos que considerar a existência de uma memória. A memória pode ser volátil ou permanente.

Considerando essas questões, a especificação da classe **Device** propõe um modelo genérico de dispositivo, onde essas questões são abordadas de maneira simplificada, acomodando os principais requisitos e visando a simulação.

*Device*

*Entity*

$[InputDevice, OutputDevice]$
$pid, mid : \mathbb{N}$

$model : Text$
$input : \mathbb{P}\, InputDevice$
$output : \mathbb{P}\, OutputDevice$
$entry : InputDevice \pfun \mathbb{P}\, Data$
$exit : OutputDevice \pfun \mathbb{P}\, Data$
$process : pid \pfun Process$
$aliveProcess : \mathbb{P}(Process)$
$state : \mathbb{P}\, State$
$stateTransition : Event \pfun (State \rel State)$
$memory : \mathbb{P}\, Data$
$addressMemory : mid \pfun Data$
$event : \mathbb{P}\, Event$
$eventMap : Event \pfun \mathbb{P}\, Device$

---

$input = \mathrm{dom}\, entry$
$output = \mathrm{dom}\, exit$
$aliveProcess = \mathrm{ran}\, process$
$\mathrm{dom}\ \mathrm{ran}\, stateTransition \subseteq state$
$\mathrm{ran}\ \mathrm{ran}\, stateTransition \subseteq state$
$memory = \mathrm{ran}\, addressMemory$
$event = \mathrm{dom}\, eventMap$

Figura 4.11: Características da classe Device

A classe **Device** herda as propriedades da classe **Entity**, conseqüentemente todo o modelo de comunicação especificado é válido para o dispositivo. Não há necessidade de adicionar novas propriedades para tratar da comunicação, pois todos os elementos classificados como entidades se comunicam entre si, logo a compatibilidade é mantida. Uma outra importante herança é o tipo básico **Process**. Esse

tipo não foi utilizado nas especificações das entidades anteriores, porém desempenha um importante papel para os dispositivos. Um processo representa um conjunto de ações que são realizadas sobre os dados. Para realizar essas ações o processo possui características tais como registradores, estado, prioridade, escalonamento, atributos, blocos de informação, acesso a recursos, entre outros. Para modelagem e simulação, um processo fornece meios para o dispositivo acessar, alocar, processar e modificar variáveis que definem seu estado. Apesar de sua importância, o processo foi definido como um tipo básico, pois analisando a aplicação prática do simulador, não há necessidade de criar uma estrutura que o represente, pois qualquer meio computacional para simulação fornece processos ou *threads* reais.

A propriedade **model** é um rótulo para especificar o modelo do dispositivo. Através desse rótulo é possível classificar, no futuro, modelos padrões de empresas reais para serem utilizados em uma implementação do simulador.

Os tipos básicos **InputDevice** e **OutputDevice** definem os dispositivos que fornecem entradas e saídas de dados para um dispositivo ubíquo. Como os dispositivos podem ter diversas entradas e saídas, as propriedades **input** e **output** especificam o conjunto de dispositivos de entradas e saídas. Apenas especificar os dispositivos não é suficiente, pois as informações relevantes são os dados de entrada e saída. As propriedades **entry** e **exit** definem uma associação entre os dispositivos de entrada e saída com o conjunto de dados. Os dados são, nesse caso, toda a informação produzida ou transmitida para esses dispositivos durante a simulação.

A propriedade **process** associa uma identificação única a um processo, isto é, para obter o processo basta saber a sua identificação. A propriedade **aliveProcess** especifica os processos que estão em execução, ou seja, o conjunto de processos que estão ativos. Estas propriedades apenas definem a existência de processos, não especificam nada sobre o que os processos devem fazer. Essa especificação é de responsabilidade do projetista do modelo. Ações e informações importantes dos processos são: criação de eventos e acesso a variáveis de estado e memória.

Em simulação, as observações dos acontecimentos dependem da mudança de estado segundo a variação do tempo. Observar todas as variáveis ou mesmo definir as variáveis de interesse é complexo para os dispositivos, principalmente se estiverem encapsuladas em processos. Para flexibilizar o modelo, as variáveis de estados são definidas através do tipo básico **State**. A propriedade **state** define o conjunto dessas

variáveis e a propriedade **stateTransition** define como as alterações sobre essas variáveis devem ocorrer. As variáveis de estados podem ser modificadas apenas por eventos, sejam eles internos, produzidos por processos, ou externos, recebidos através de alguma conexão. Uma outra vantagem em definir essas variáveis independentes é a possibilidade de representação de um recurso. Uma variável, por exemplo, pode representar um visor (*display*) de celular.

Além das variáveis de estados, a memória é um outro recurso para armazenar e recuperar valores. As propriedades **memory** e **addressMemory** especificam essa característica. No modelo, a mesma memória é usada para armazenamento permanente ou volátil. A propriedade **memory** representa todos os dados armazenados na memória e a propriedade **addressMemory** permite recuperar esses dados através de um identificador. Na simulação, a memória é importante para prover aos processos um espaço para trabalhar com os dados de entrada e saída, e para gerenciar algumas variáveis internas do processo.

Para completar o modelo, há duas propriedades para tratar os eventos. A propriedade **event** descreve o conjunto de eventos gerados pelo dispositivo. Todo evento gerado deve fazer parte deste conjunto. A propriedade **eventMap** mapeia os eventos para os dispositivos interessados. Portanto, todo dispositivo que tem interesse em um evento remoto, deve se registrar ao dispositivo que origina o evento para ser notificado. Apesar desse relacionamento, os eventos em geral, são de interesse de algum dos processos do dispositivo. A especificação não aborda como o processo faz uso desse evento. A princípio, os processos estão monitorando os estados e, a partir da mudança, eles executam ações sobre as variáveis e os recursos.

A classe **Device** define as características comuns para os dispositivos, porém ela não é utilizada diretamente. Suas propriedades e comportamentos são a base para a definição das três classes de dispositivos, conforme a herança visualizada na figura 4.1.

Os dispositivos embutidos são embutidos em objetos, aparelhos eletrônicos, eletrodomésticos, roupas, carros, e assim por diante. Suas características e comportamentos estão associados a controle e a execução de tarefas. Em determinadas situações age como um atuador, mas sempre se referindo a operações digitais, como por exemplo, corrigir o alarme do relógio através de um comando de voz. Por estar fixo em objetos, o dispositivo embutido pode ter a característica de mobilidade,

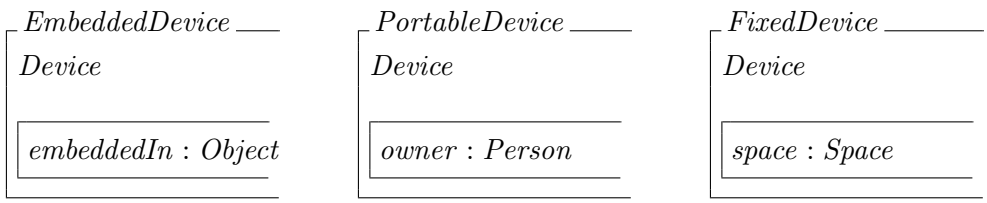


Figura 4.12: Características das classes EmbeddedDevice, PortableDevice e FixedDevice

como no caso de um automóvel ou roupas. Nesse caso, tratar as questões de localização é importante, pois a localização é estática com relação a quem transporta, mas para o ambiente é dinâmica. Logo, a posição deve ser informada baseada na localização do elemento móvel, seja uma pessoa ou um carro.

A classe **EmbeddedDevice** define um dispositivo embutido baseado em herança de **Device** e na propriedade unária **embeddedIn**, que especifica onde ele está embutido. Pela especificação se observa que um dispositivo embutido pode estar associado somente a um único elemento e este elemento deve ser um objeto. Apesar dessa associação, ele possui capacidade de computação e comunicação para estabelecer conexões com outras entidades.

Os dispositivos portáteis são os dispositivos que se aproximam dos microcomputadores e são portáteis, como os celulares, palmtops, notebooks, pdas, e assim por diante. Esses dispositivos possuem sistema operacional e interfaces gráficas mais avançados, um número maior de periféricos e portas de comunicação, entre outros. A localização é sempre dependente de quem os transporta, em geral, uma pessoa, pois são para uso pessoal e privado. As interações com outros dispositivos são, em sua maioria, dependentes das operações definidas pelos programas e suas interfaces, seja digital ou por botões. Devido a essa complexidade, o manuseio do dispositivo ocorre em uma posição fixa, pois a pessoa necessita estar parada para prestar atenção na atividade sendo desempenhada. Há duas questões cruciais para esses dispositivos, a comunicação e a bateria. A comunicação permite a interação com o mundo ubíquo, esteja a entidade parada ou em movimento. Como esses dispositivos desempenham os principais papéis entre a comunicação do ambiente com as pessoas, o gerenciamento da comunicação torna-se destaque no presente e mais ainda no futuro. A bateria é a principal fonte de energia para manter esses dispositivos ligados; se não há energia, o dispositivo não tem como funcionar.

A classe **PortableDevice** define um dispositivo portátil baseado em herança de **Device** e na propriedade unária **owner**, que especifica o dono do dispositivo. Essa propriedade demonstra explicitamente o caráter pessoal e privado do dispositivo portátil. Com isso, associa-se o dispositivo a uma pessoa e as informações de localização podem ser obtidas consultando o portador do dispositivo. A classe não aborda o controle de energia, mas há mecanismos no modelo para gerenciar essa questão. As interfaces podem ser definidas por variáveis de estado e o acesso pode se dar através de eventos.

Os dispositivos fixos são os dispositivos com mais alto grau de computação e comunicação, tais como os computadores, servidores, clusters, entre outros. Devido ao seu porte se encontram em posições fixas no ambiente e necessitam de alimentação constante de energia. Essas entidades participam como provedores de serviços, auxiliares de computação, gerenciadores de ambientes, suporte para protocolos de comunicação, entre outros. Na modelagem, a preocupação está em garantir que esses serviços possam ser atendidos e estabelecer a localização desses dispositivos.

A classe **FixedDevice** define um dispositivo ou estação fixa baseada em herança de **Device** e na propriedade unária **space**, que indica em qual espaço do ambiente a estação se encontra. A classe não acrescenta nenhuma outra propriedade ou característica especial, pois a herança fornece todo o suporte necessário para a programação da comunicação e computação dessas estações.

## 4.7 Comunicação

A comunicação é uma questão crítica e complexa de ser abordada em modelos de simulação. Há muitos simuladores que tratam especificamente dessa questão, como exemplo, o NS-2[1]. Conforme definido nos objetivos, essa dissertação não considera o modelo de comunicação como essencial, pois está concentrada na modelagem e interação das entidades. Contudo, toda interação entre dispositivos acontece através de um canal de comunicação. Prover um modelo, mesmo simplificado, implica em estabelecer o meio físico, os protocolos e as conexões entre as entidades.

A classe **Entity** define para uma entidade a relação entre o meio físico, protocolo e conexão. O meio físico é especificado através da classe **CommunicationChannel**,

[1]http://www.isi.edu/nsnam/ns

o protocolo pelo tipo básico **Protocol** e a conexão por um apontamento para uma entidade.

O meio físico se refere ao canal de comunicação físico e ao tipo de comunicação empregada. Em ambientes de computação ubíqua a maior parte das comunicações são por meios sem fio e os tipos mais empregados são ondas de rádio e infravermelho. Conforme o canal e o tipo há diversos parâmetros a serem considerados, como exemplo interferências, alcance, taxa de transmissão, entre outros. Esses parâmetros são importantes para a simulação, pois interferem sobre o ambiente e no comportamento das pessoas. Por exemplo, se uma comunicação de um celular não é realizada por uma interferência em determinado local, a pessoa que está fazendo a ligação pode se mover para outro lugar ou desistir da chamada.

Como o modelo de comunicação para a simulação é simplificado, as questões quanto às camadas das diferentes tecnologias são descartadas. Simuladores tradicionais contemplam a especificação de camadas, tais como a camada de acesso ao meio, e a pilha de protocolos especificados por estas camadas. O modelo se concentra somente em identificar o canal e inserir restrições de comunicação. A classe **CommunicationChannel** especifica as propriedades e as restrições para descrever a comunicação entre duas entidades.

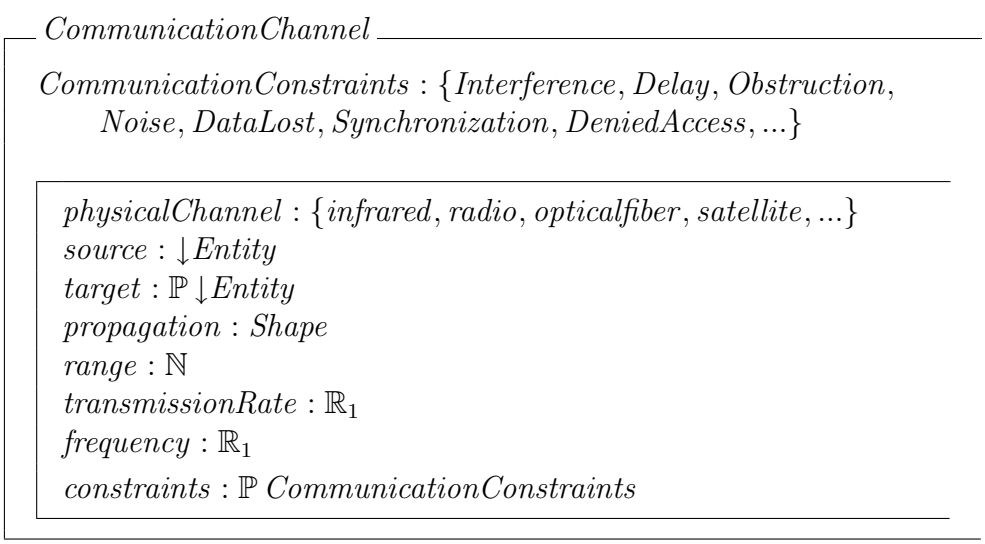


Figura 4.13: Características da classe CommunicationChannel

A propriedade **physicalChannel** define o tipo de canal. Sua função é apenas descritiva, mas pode ser utilizada para implicitamente definir os limites inerentes ao canal. Por exemplo, se o canal é infravermelho e um elemento qualquer obs-

trui a passagem do feixe entre o emissor e o receptor, temos uma interferência na transmissão.

Em um processo de comunicação há no mínimo duas entidades, o transmissor e o receptor. O transmissor envia a mensagem e o receptor recebe através do canal de comunicação. Há a possibilidade de enviar mensagens por difusão, seja para um grupo (multicast) ou para todos (broadcast). Nesse caso, temos um transmissor e diversos receptores. As propriedades **source** e **target** definem, respectivamente, o emissor e o conjunto de receptores de mensagens. Em muitos casos não há como definir os receptores, pois há uma constante entrada e saída de entidades no perímetro das transmissões.

As propriedades **propagation**, **range**, **transmissionRate** e **frequency** definem as características da comunicação. A primeira especifica a forma de propagação do sinal (ex.: circular, omnidirecional, retilínea). A segunda especifica o alcance do sinal e em conjunto com a anterior define a região de cobertura. A terceira especifica a taxa de transmissão, isto é, quantas unidades de dados são transmitidas por unidade de tempo. Pode ser utilizada para estimar o tempo de duração de uma conexão e transmissões não completadas. A quarta especifica a freqüência do canal (ex.: 2.4 Ghz) e pode ser utilizada para verificar restrições definidas para a freqüência.

Um diferencial abordado por esse modelo é quanto às questões de interferência. O tipo básico **CommunicationConstraints** define um conjunto de possíveis restrições ou problemas que podem ocorrer nas comunicações devido a fatores externos, como chuva, ou internos, como políticas de segurança. Essas restrições são representadas pela propriedade **constraints**, que permite definir um conjunto de valores para especificar quais os problemas e a interferência total na comunicação. São exemplos de problemas definidos nesse conjunto as interferências, atrasos, obstruções, ruídos, perda de pacotes, sincronização, acesso negado, e assim por diante.

O protocolo se refere ao protocolo de rede, ou seja, o conjunto de regras e convenções para comunicação entre dispositivos em rede. O tipo **Procotol** apenas especifica sua existência, porém não detalha como deve ser inserido e implementado nos modelos para simulação. No presente modelo, toda comunicação deve possuir um protocolo; este protocolo deve especificar as mensagens e todas as convenções de formatação e troca dessas mensagens. Por exemplo, considere um protocolo em alto nível para estabelecer uma conexão, enviar um dado e finalizar a comunicação

(figura 4.14).

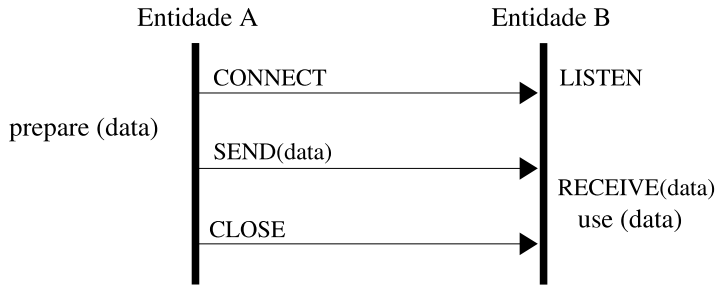


Figura 4.14: Protocolo simples para trocas de mensagens

Seja A a entidade emissora e B a entidade receptora. Os comandos CONNECT, LISTEN, SEND, RECEIVE e CLOSE representam os comandos para solicitar conexão, aguardar conexão, enviar dados, receber dados e finalizar. O protocolo da entidade A é representado pela seqüência de comandos CONNECT, *prepare*, SEND e CLOSE. O protocolo da entidade B é representado pela seqüência de comandos LISTEN, RECEIVE, *use*. O comando *prepare* representa um pré-processamento dos dados e o comando *use* um pós-processamento dos dados. Baseado nesse exemplo, verifica-se a viabilidade para modelar e simular os protocolos de comunicação, desde que sejam estabelecidas as regras e convenções para o seu uso.

Em um sistema computacional as conexões utilizam abstrações conhecidas como portas para estabelecer uma comunicação com o sistema. Na especificação é utilizada uma abordagem similar, onde valores numéricos representam as conexões. A principal diferença é que os valores não são padrões, isto é, não há uma porta especifica para se conectar e iniciar um protocolo de comunicação. O controle de conexões é através de um incremento simples e aloca a próxima porta livre para a conexão.

Apesar das classes **Entity** e **CommunicationChannel** especificarem as propriedades para representar e controlar as comunicações, elas não são suficientes representar as comunicações ativas e para o próprio gerenciamento de uma simulação. Devido as diversas comunicações simultâneas e a descentralização de controle, não é possível observar quais são as comunicações ativas e se há interferências ou problemas entre elas. Outra carência é a limitação de estabelecer restrições de comunicação somente ao canal de comunicação. Em ambientes reais, as interferências podem estar associadas a regiões geográficas, por exemplo, regiões com tunéis, montanhas, poluição, entre outras.

*CommunicationController*

$activeEntities : \mathbb{P} \downarrow Entity$
$activeChannels : \mathbb{P}\, CommunicationChannel$
$connections : CommunicationChannel \times \downarrow Entity$
$area : \mathbb{P}\, Shape$
$restrictArea : Shape \nrightarrow CommunicationChannel.Constraints$

---

$activeEntities = \text{ran}\ connections$
$activeChannels = \text{dom}\ connections$
$area = \text{dom}\ restrictArea$

Figura 4.15: Características da classe CommunicationController

Para auxiliar nessas questões, a classe **CommunicationController** (figura 4.15) especifica propriedades para centralizar o controle e expandir a possibilidade de restrição sobre as comunicações. As propriedades **activeEntities** e **activeChannels** especificam as entidades e os canais ativos em um dado instante. A propriedade **connections** associa as entidades e os canais. Por meio dessa propriedade é possível averiguar a ocupação dos canais pelas entidades e acompanhar as trocas de mensagens nas comunicações ativas. As propriedades **area** e **restrictArea** definem as regiões com restrições ou bloqueio de comunicação.

## 4.8 Eventos e Sinalizações

Eventos são acontecimentos ou ocorrências. Do ponto de vista tecnológico, aplicado a sistemas computacionais, é uma ação iniciada por um usuário ou pelo próprio sistema. São exemplos clássicos os eventos de pressionar as teclas ou algum botão em uma interface e eventos baseados em tempo. Sinalização é o uso de sinais para expressar advertências ou para possibilitar o conhecimento, reconhecimento ou previsão de alguma coisa. Os sinais podem ser qualquer unidade que, em conformidade com as regras de um código, compõem uma mensagem para transmitir ordens, notícias, avisos ou informações em geral.

Na especificação, os recursos de eventos e sinalizações são utilizados para diferenciar e facilitar a compreensão do conteúdo de mensagens trocadas entre os elementos em cenários de computação ubíqua, pois há diversas comunicações simultâneas sendo

realizadas, ocasionando freqüentes e excessivas trocas de mensagens e, conseqüentemente, complicando a transmissão de mensagens que possuem semântica de controle, ordem ou notificação.

Os cenários reais apresentam muitos tipos de comportamentos que devem ser modelados e entendidos como eventos ou sinais em simulação. Por exemplo, a percepção de uma imagem por uma câmera. Todos os elementos físicos possuem uma imagem; não é interessante simular uma câmera que gere uma imagem para todas as entidades, mas que a imagem seja enviada para a câmera pela entidade que a originou. Além de aproximar do que realmente está acontecendo, evita problemas de excesso de carga para determinados elementos. Outra situação, o término de um processo que interessa a outros dispositivos. Não é interessante a verificação constante do processo pelos dispositivos, mas sim a notificação aos interessados quando o processo termina. Temos ainda os eventos ou sinais que representam ordem, ou seja, são comandos para a execução de uma determinada tarefa. Comandos são usados para o controle e, em ambientes reais, podem significar controle em tempo real. Logo, verifica-se a relevância desse tipo de notificação.

Baseados nesses casos, o modelo apresenta uma classificação dos eventos e sinalizações em três classes. As classes representam respectivamente sinais, eventos e comandos. Nesse caso, sinais são informações emitidas por qualquer elemento do mundo ubíquo; eventos são notificações emitidas por entidades, em especial, os dispositivos; e comandos são ordens e parâmetros para modificar o estado de objetos.

Apesar da divisão, há características comuns a todo tipo de evento ou sinal. A classe **Trigger** representa essas características e constitui a base para as três classes específicas de sinalizações.

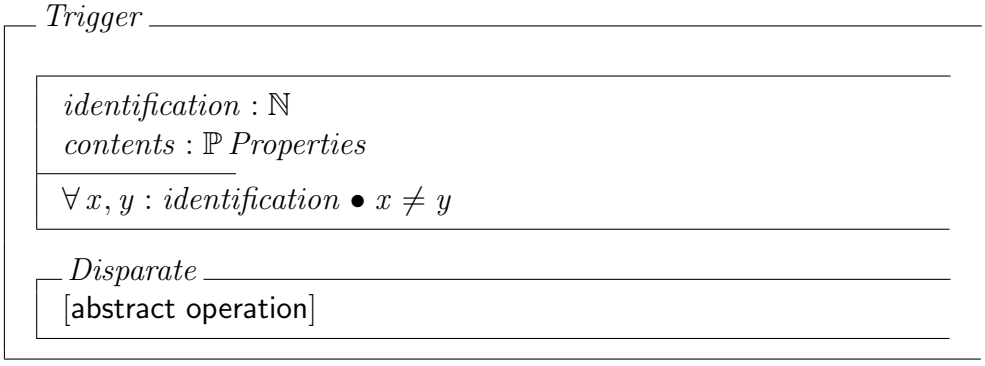


Figura 4.16: Características da classe Trigger

A propriedade **identification** é uma representação única para todo tipo de evento ou sinal. Através dessa identificação é possível identificar e controlar os eventos disparados. A propriedade **contents** é uma representação do conteúdo transmitido. Devido à diversidade dos eventos gerados, as informações são representadas por um conjunto de relações entre rótulos e dados. Por exemplo, um evento que deseja transmitir uma identificação de emissor, um código numérico e uma figura, utilizaria a seguinte estrutura:

$$(source, device\_x), (code, 2564), (image, img.jpg)$$

Conforme apresentado, há três classes que herdam as características de **Trigger**: **Signal**, **Event** e **Command**. Porém, antes de apresentar individualmente cada uma é necessário definir duas classes auxiliares, que são as classes **SignalType** e **EventType**. Estas classes definem uma classificação para cada tipo de sinal e evento. Elas apresentam duas propriedades que descrevem o evento em função de valor numérico e uma descrição textual de seu significado. Desse modo, não há limitações em definir qualquer tipo de evento. Por exemplo, para definir um sinal de vídeo, podemos simplesmente definir "1" e "vídeo" como os valores das propriedades.

Sinais são emitidos por todo e qualquer elemento que faz parte de um ambiente de computação ubíqua. Uma definição especifica para o modelo é definir sinais como qualquer tipo de informação que pode ser reconhecida por um sensor. A classe **Signal** define a estrutura para modelar os sinais e suas propriedades.

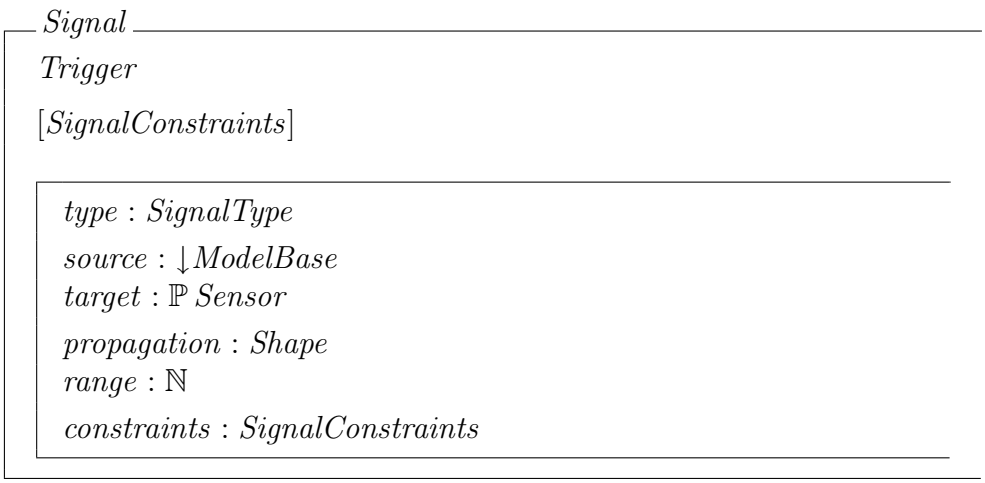


Figura 4.17: Características da classe Signal

A propriedade **type** define o tipo de sinal baseada nas especificações de **Sig-**

**nalType**. As propriedades **source** e **target** definem a origem do sinal e os prováveis receptores, isto é, quem pode acessar o conteúdo. Os receptores devem ser definidos dinamicamente, pois conforme a localidade da entidade há um número específico de sensores que se interessam pelo sinal. As propriedades **propagation** e **range** definem a área de propagação do sinal. Através dessas propriedades se verifica quais sensores podem perceber o sinal em uma dada localização e momento. A propriedade **constraints** permite definir restrições para a emissão de um sinal. Por exemplo, uma imagem captada por uma câmera deve estar no foco da câmera, logo não pode ter objetos entre a fonte emissora e o sensor. No caso de um sensor de áudio esse tipo de restrição não é válido, mas uma restrição de interferência por ruído poderia ser definida. Os sinais não utilizam os canais de comunicação definidos para entidades, por isso as restrições são controladas pelo tipo básico **SignalConstraints**.

Eventos são emitidos somente por entidades, logo seu escopo é muito mais restrito que sinais. Os eventos se referem sempre a dados digitais, isto é, os eventos sempre se referem a acontecimentos que produzem reações em estados de um dispositivo computacional. A sua propagação sempre ocorre por um canal de comunicação. A classe **Event** define a estrutura para modelar os eventos e suas propriedades.

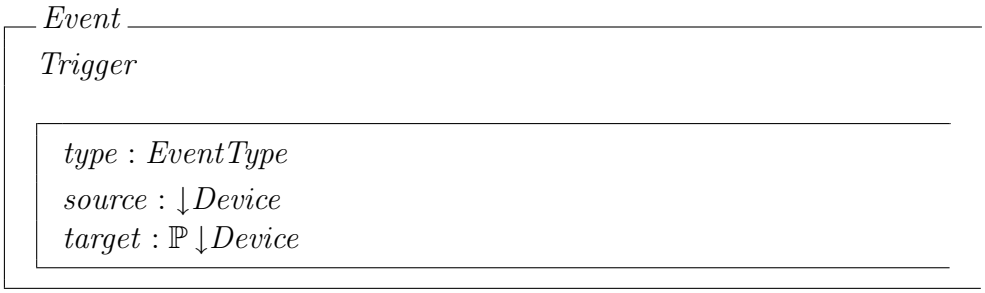


Figura 4.18: Características da classe Event

A propriedade **type** define o tipo de evento baseada nas especificações da classe **EventType**. A propriedade **source** especifica a origem do evento, na maioria das vezes um dispositivo. A propriedade **target** especifica quais os dispositivos notificados da ocorrência do evento. Neste caso, os dispositivos devem estar registrados para serem notificados. No caso de um sensor emitir um evento para um dispositivo, é utilizada a lista de conexões do sensor para notificar os dispositivos interessados.

Comandos são emitidos por entidades e representam uma ordem. No modelo, comandos são definidos como ordens para disparar ações que modificam o estado de

objetos. A classe **Command** define a estrutura para modelar os comandos e suas propriedades.

*Command*

*Trigger*

$source : \downarrow Entity$
$target : \mathbb{P}\, Actuator$
$sintaxe : Data$
$parameters : \mathbb{P}\, Data$

---

$sintaxe \in \text{ran}\, contents$
$parameters \subset \text{ran}\, contents$

Figura 4.19: Características da classe Command

A classe define as propriedades **source** e **target** para representar a origem e o destino dos comandos. O destino sempre é um atuador, pois somente um atuador pode mudar o estado de um objeto e contempla em sua especificação propriedades para associar comandos a ações. As propriedades **sintaxe** e **parameters** definem o comando e seus parâmetros. Elas são definidas para facilitar a recuperação dos dados na propriedade **contents** da classe **Trigger**. Por exemplo, para enviar o comando *lock_door*, com o parâmetro *30 degrees*, torna-se mais simples especificar diretamente nessas propriedades.

## 4.9 Localização

A seção 2.6.3 define a localização como uma questão crucial em ambientes de computação ubíqua. Através da informação de localização das pessoas, objetos e entidades, os ambientes gerenciam os recursos e as interações entre os elementos presentes nos espaços. Para o modelo, a informação de localização deve suportar as características definidas para ambientes reais e, ainda, fornecer mecanismos práticos para gerenciar essa informação.

Independente do elemento, se ele ocupa um lugar no espaço, ele tem uma posição no mundo. A classe **ModelBase** define uma propriedade para representar a posição absoluta nesse mundo. O mundo, nesse caso, refere-se ao ambiente de computação ubíqua ou cenário modelado. A posição absoluta permite localizar, através de uma

grade global de referência, qualquer elemento em um ponto no ambiente. Todos os elementos físicos têm essa propriedade, pois todos são especializações da classe **ModelBase**. Do ponto de vista de aplicação do modelo, é através da posição absoluta que os elementos são dispostos nos cenários. Ao inserir um elemento, a propriedade **position** define um ponto no espaço para representar a localização base de um objeto. Devido à impossibilidade de dois corpos ocuparem a mesma posição em um instante de tempo comum, o modelo compreende uma restrição para garantir a unicidade de localização.

Apesar da posição absoluta garantir a localização de qualquer elemento, há modelos e simulações que necessitam de sistemas de localização mais práticos. Por exemplo, a simulação de uma aplicação de controle de lâmpadas onde identificado os cômodos sem a presença de pessoas, as lâmpadas são apagadas para economizar energia. Nessa aplicação, é necessário obter informação de identificação do cômodo que se encontra a pessoa. Para tal, seria necessário recuperar a informação de posição absoluta da pessoa, recorrer à grade global e identificar o cômodo que agrega tal posição. Para resolver esse problema, o modelo apresenta a classe **SymbolicLocation**, uma estrutura para representar posições simbólicas e abstratas.

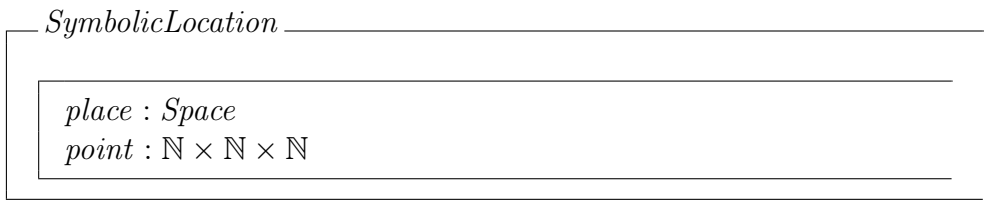


Figura 4.20: Características da classe SymbolicLocation

A propriedade **place** representa o espaço onde um elemento se encontra e a propriedade **point** define um ponto baseado na grade de referência espacial do local. Consultando a primeira propriedade, a recuperação de localização para aplicações que necessitam apenas de informações abstratas se torna eficaz e eficiente, pois não é necessário recorrer a cálculos para definir a localização.

Mesmo utilizando ambas as localizações, absoluta ou simbólica, há outras aplicações que precisam de outro tipo de sistema de localização. Por exemplo, suponha duas pessoas portando seus dispositivos ubíquos e em movimento constante. Uma comunicação é iniciada entre seus dispositivos para trocas de informações, logo os dispositivos devem dispor da localização e distância entre eles. Com a localização

absoluta é simples calcular a distância através da equação de distância entre dois pontos. Entretanto, não é simples manter a referência entre os dois dispositivos e a orientação, isto é, verificar se eles estão alinhados. Com a localização simbólica não há como solucionar o problema. Considerando as situações que necessitam de cálculo de posição com relação a outro elemento, o modelo apresenta a classe **RelativeLocation**, uma estrutura para representar posição e orientação relativa entre elementos.

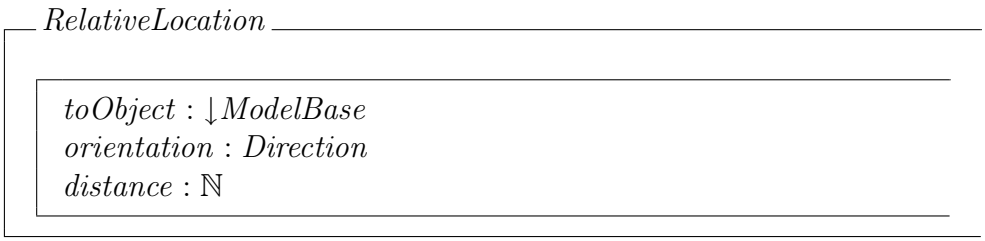


Figura 4.21: Características da classe RelativeLocation

A propriedade **toObject** indica o elemento base para a referência. O elemento base é uma associação direta para a localização, se ele for removido ou não participar mais das interações, a associação deve ser desfeita. A propriedade **orientation** especifica a orientação do elemento com relação ao elemento base. Pode assumir os valores definidos por **Direction**, por exemplo, norte, sudeste, sul. A propriedade **distance** especifica a distância entre os dois elementos.

A localização absoluta é obrigatória para todos os elementos. A localização simbólica e a relativa são facultativas e podem ser utilizadas em conjunto. A classe **Location** define a estrutura para combinar e aplicar as localizações.

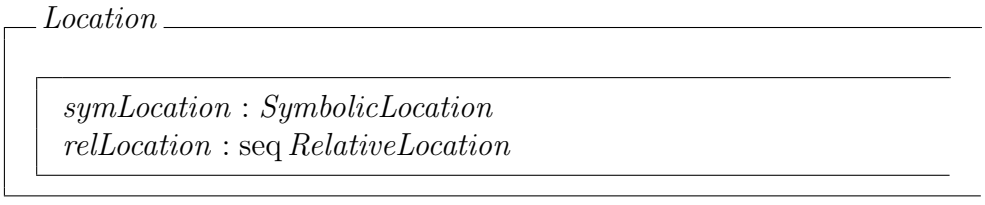


Figura 4.22: Características da classe Location

A propriedade **symLocation** define uma localização simbólica unária, pois é impossível um elemento pertencer a dois espaços ao mesmo tempo. A propriedade **relLocation** define uma estrutura para associar um elemento a diversas localizações

relativas. A classe **ModelBase** define a propriedade não obrigatória **location** que utiliza essa estrutura.

Como é complexo manter a localização das entidades utilizando localização simbólica ou relativa, todo elemento que utiliza esse tipo de localização deve estar registrado no controlador de localização. A classe **LocationController** define uma estrutura para gerenciar e manter esses elementos no sistema.

*LocationController*

$$registeredElement : \mathbb{P} \downarrow ModelBase$$
$$location : \downarrow ModelBase \nrightarrow Location$$
$$registeredElement = \operatorname{dom} location$$

Figura 4.23: Características da classe LocationController

Enquanto no mundo real, os objetos se movem e não precisam se preocupar com sua localização, pois ela é uma implicação intrínseca da locomoção, em simulação, esse processo implica na computação explícita de uma nova posição e, se o ambiente for apresentado visualmente, na atualização das imagens que compõem a visualização. Garantir unicidade de posição em uma confusão como essa é complicado, principalmente quando se trabalha com coordenadas no espaço. Por isso, na prática, é muito mais simples modelar e simular baseado em coordenadas planares e, em algumas situações, desconsiderar as restrições de posição. Outro ponto a considerar é quanto aos dispositivos portados por pessoas ou objetos. Como a posição absoluta desses dispositivos é equivalente à posição dos elementos que os portam, logo não é necessário recalcular sua posição a todo instante, basta recorrer a posição do elemento portador.

## 4.10 Tempo

*"A única razão para o tempo é que nada acontece de uma vez." (A. Einstein)*

Em simulação não é diferente, os acontecimentos ocorrem no decorrer do tempo. A especificação contempla estruturas para suportar, monitorar e controlar o tempo e os acontecimentos dependentes do tempo. Além dessas questões, a contabilização

do próprio tempo é uma variável estatística utilizada para avaliar os resultados em uma simulação.

A classe **TimeStamp** define uma estrutura para armazenar e representar uma marcação de tempo. As propriedades **date** e **time** armazenam a data e a hora.

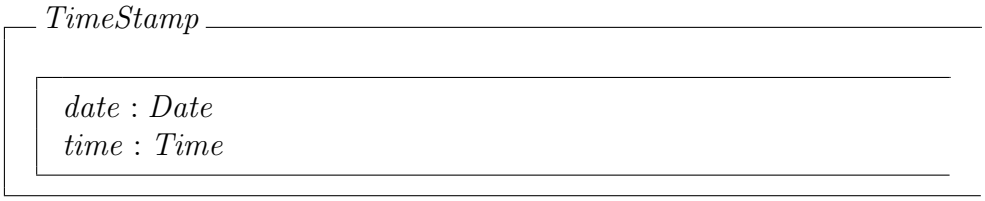


Figura 4.24: Características da classe TimeStamp

O conceito fundamental para o controle de tempo é o relógio de simulação. O relógio de simulação registra o instante corrente e avança segundo as definições para a simulação. Como todos os elementos estão em um ambiente que representa o mundo real, todos são dependentes desse relógio. O avanço de uma simulação ocorre com o incremento do relógio. Duas abordagens conhecidas são por avanço de tempo para o próximo evento e avanço de tempo com incremento fixo. Dependendo do modelo simulado, define-se qual abordagem é mais adequada. Para simular com avanço de incremento fixo, os modelos indicados são aqueles onde quantificar os eventos segundo a unidade de tempo é extensa, como no caso de protocolos de comunicação. A especificação define um relógio de simulação, mas não aborda as questões de avanço.

Os eventos dependentes do tempo utilizam o relógio de simulação para dispararem ou determinar seu tempo de execução. No modelo, os eventos são classificados em três grupos:

- eventos periódicos: são os eventos disparados com uma freqüência definida. Por exemplo, um evento para verificar o gás a cada cinco minutos.
- eventos agendados: são os eventos especificados para dispararem em data e hora fixas. Os eventos estão, na maioria dos casos, nesse grupo. Por exemplo, desligue o forno às cinco horas e dez minutos, ou ainda, mova uma pessoa para a cozinha às oito horas.
- eventos cronometrados: são os eventos com duração de execução fixa. Por exemplo, verifique as comunicações durante cinco minutos.

Considerando a existência de um relógio de simulação e o controle de eventos no tempo, a classe **TimeController** fornece a estrutura para modelar essas questões.

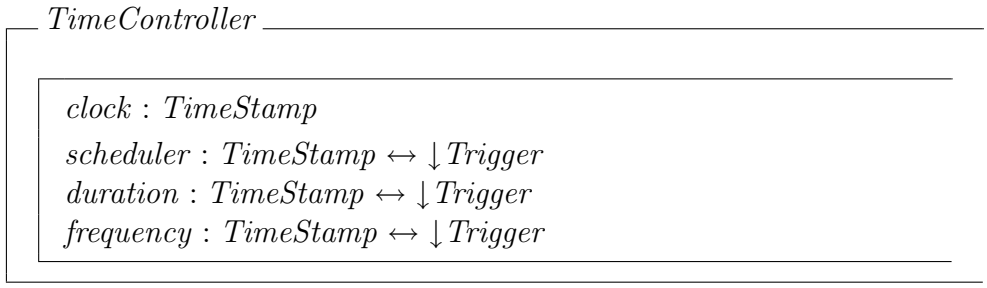


Figura 4.25: Características da classe TimeController

A propriedade **clock** especifica o relógio de simulação. As propriedades **scheduler**, **duration** e **frequency** especificam os eventos dependentes do tempo. As três propriedades de controle de eventos no tempo se apresentam com a mesma estrutura, porém o significado de suas estruturas difere. A primeira define o agendamento através de uma marcação com a data e a hora em que deve ser executado o evento. A segunda define a duração de eventos através de uma marcação que indica tempo de duração. A terceira define a freqüência ou período que o evento deve ser executado. A relação garante a associação da marcação de tempo com o evento específico. Propriedades adicionais, como por exemplo, marcar o momento da última execução de um evento periódico, não estão definidas.

A contabilização estatística do tempo não é incorporada ao modelo, pois é apenas uma medição e registro de variáveis. Contudo, para uma implementação prática de um simulador, devem ser definidas e modeladas.

## 4.11 Situações

Os acontecimentos ou eventos gerados no mundo real, ou mesmo em uma simulação, não são todos dependentes do tempo. Alguns têm relação com o tempo apenas por executarem em um instante aleatório. Apesar da existência de eventos aleatórios, há um subconjunto dependente de uma coleção de estados de determinados elementos. O termo utilizado nesse trabalho para tais eventos é eventos baseados em situações. Formalmente, são eventos disparados quando um conjunto finito de estados é atingido ou ativado e originam uma determinada situação.

Uma situação é caracterizada por um estado global constituído por estados individuais parciais dos elementos que a compõem. O estado individual parcial de um elemento é o conjunto de propriedades válidas ou de interesse para uma situação particular. Baseada nessa definição, uma situação é uma coleção de atributos de elementos distintos ou não, cujos valores atendem aos requisitos estabelecidos para obter a situação. Por exemplo, considere a situação para digirir um carro com segurança: porta fechada, motorista e os passageiros com o cinto e o motorista não deve estar alcoolizado. Essas propriedades caracterizam uma situação baseada nas propriedades parciais desses elementos.

A especificação de situações permite definir ações, eventos e restrições baseadas nos estados dos elementos presentes no ambiente. A classe **SituationController** é uma estrutura que agrega os estados de interesse de uma determinada situação, associa e dispara eventos quando os estados forem ativados.

*SituationalController*

$$situation : \mathbb{F}\, State \pfun \downarrow Trigger$$

$$y : \mathbb{F}\, State \mid y \in \operatorname{dom} situation \bullet (\forall\, x : State \mid x \in y \land x = \text{true} \bullet situation(y).Disparate)$$

Figura 4.26: Características da classe SituationController

A propriedade **situation** associa um conjunto finito de estados a um evento. O predicado garante que quando o conjunto de estados for alcançado, isto é, tornar-se verdadeiro, o evento é disparado. A classe não define o que é verdadeiro para o conjunto de estados, isso deve ser modelado pelo projetista. No exemplo anterior, porta fechada, passageiros com cintos e motorista não alcoolizado era a condição verdade. Para outras situações um novo conjunto de valores é a verdade.

## 4.12 Energia

O consumo de energia é um dos desafios para as pesquisas em computação ubíqua. Os dispositivos com suas baterias restritas, não suportam as características exigidas por muitos tipos de serviços e aplicações. Mesmo com o gerenciamento da energia pelas camadas superiores, tais como o sistema operacional, não é sufi-

ciente para evitar os problemas introduzidos por essa restrição. Esses problemas, por sua vez, introduzem e modificam o comportamento dos elementos envolvidos em interações nos ambientes ubíquos. Por exemplo, suponha um cenário com pessoas, que pelo freqüente uso de seus dispositivos para acessar qualquer informação, estão condicionadas e dependentes da tecnologia. Se, por acaso, essa tecnologia não funcionar por falta de energia e o ambiente não prover outros mecanismos para o acesso da informação desejada, frustração e comportamentos inesperados serão o resultado obtido desta experiência.

O caso anterior e outros casos podem modificar o que seria o comportamento esperado no ambiente. Este comportamento não se restringe somente a pessoas, dispositivos podem iniciar comunicações para troca de informações de autenticação, por exemplo, e não serem capazes de terminar o processo. Além disso, a participação dos dispositivos dependentes de energia para computação e comunicação nos cenários onde estão presentes, se não estiverem com a bateria carregada, é nula.

Considerando essas questões, a modelagem contempla uma especificação de um controlador de energia para gerenciar o consumo de energia e desativar os dispositivos que não têm condições de participar mais da simulação devido a essa restrição. O controlador pode ser utilizado para obter estatísticas sobre quais dispositivos foram desativados, em qual instante e averiguar as implicações sobre o modelo. A classe **EnergyController** apresenta a estrutura para suportar o gerenciamento de energia.

*EnergyController*

$$device : \mathbb{P} \downarrow Entity$$
$$energy : \downarrow Entity \pfun \mathbb{R}$$

---

$$device = \mathrm{dom}\ energy$$
$$\forall x : \downarrow Entity \mid energy(x) = 0 \bullet x.enabled = \mathrm{false}$$

Figura 4.27: Características da classe EnergyController

Como o controle de energia é de interesse para aplicações específicas e nem sempre todos os dispositivos necessitam ser monitorados, a propriedade **device** especifica o conjunto dos dispositivos gerenciados pelo controlador. A propriedade **energy** associa um dispositivo a um valor numérico que representa sua energia em

um instante. Essa propriedade é volátil com relação ao tempo e ao uso do dispositivo, logo assume valores de energia diferentes no decorrer tempo. Quando a energia atinge o valor zero, o dispositivo é desabilitado.

Para decrementar a energia, o modelo simulado deve ter acesso a uma tabela com informações sobre a quantidade de energia despendida nas operações realizadas pelo dispositivo. Esta tabela deve refletir valores que condizem com a realidade, apesar de que suposições podem ser interessantes quando se está desenvolvendo um protótipo de dispositivo.

## 4.13 Espaço e Delimitadores

Se um corpo possui uma massa, ele ocupa um lugar no espaço. Espaços são lugares mais ou menos bem delimitados, cujas áreas podem conter alguma coisa (FERREIRA, 2004). Nas seções anteriores foram definidos pessoas, objetos e entidades; todos têm uma massa e estão contidos em espaços. Essa seção aborda as questões de definição, delimitação, inserção de elementos e gerenciamento do espaço.

Espaços são elementos básicos do modelo. Um espaço especifica um local para conter os elementos e definir os limites para suas interações. Essa delimitação afeta os movimentos, a comunicação e a computação. Os movimentos são restritos ao espaço local e aos espaços conectados ou aos quais a entidade tem permissão de acesso. As comunicações são delimitadas ao espaço local quando há limites que impedem ou bloqueiem comunicações com elementos de espaços externos. As computações são afetadas quando dependem da interação com entidades presentes em um espaço específico.

Os espaços são definidos através da inserção de delimitadores. Os delimitadores são objetos ou marcações imaginárias para especificar uma região fechada, semifechada, semi-aberta ou aberta. Uma região fechada não possui acesso para outro espaço, por exemplo, uma sala sem entradas e saídas. Uma região semifechada possui aberturas, por exemplo, uma sala de reunião, com portas ou janelas, que podem estar abertas ou fechadas. Uma região semi-aberta possui passagens para outros espaços e estão sempre abertas, por exemplo, um túnel. Uma região aberta não possui restrições físicas para acesso a outros espaços, por exemplo, uma praça.

Considerando a figura 4.28 e as questões anteriores, observa-se o uso de objetos

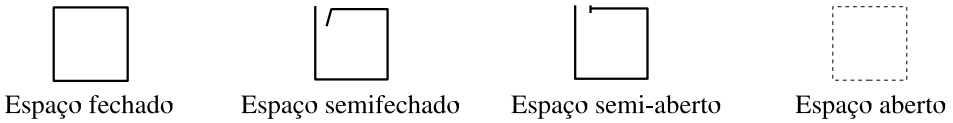


Figura 4.28: Espaço fechado, semifechado, semi-aberto e aberto

para definir espaços. Esses objetos são, em sua maioria, paredes, portas e janelas. As paredes definem os limites para o espaço, enquanto as portas e janelas especificam aberturas para outros espaços. As classes **Wall**, **Door** e **Window** definem as estruturas para especificar as delimitações espaciais.

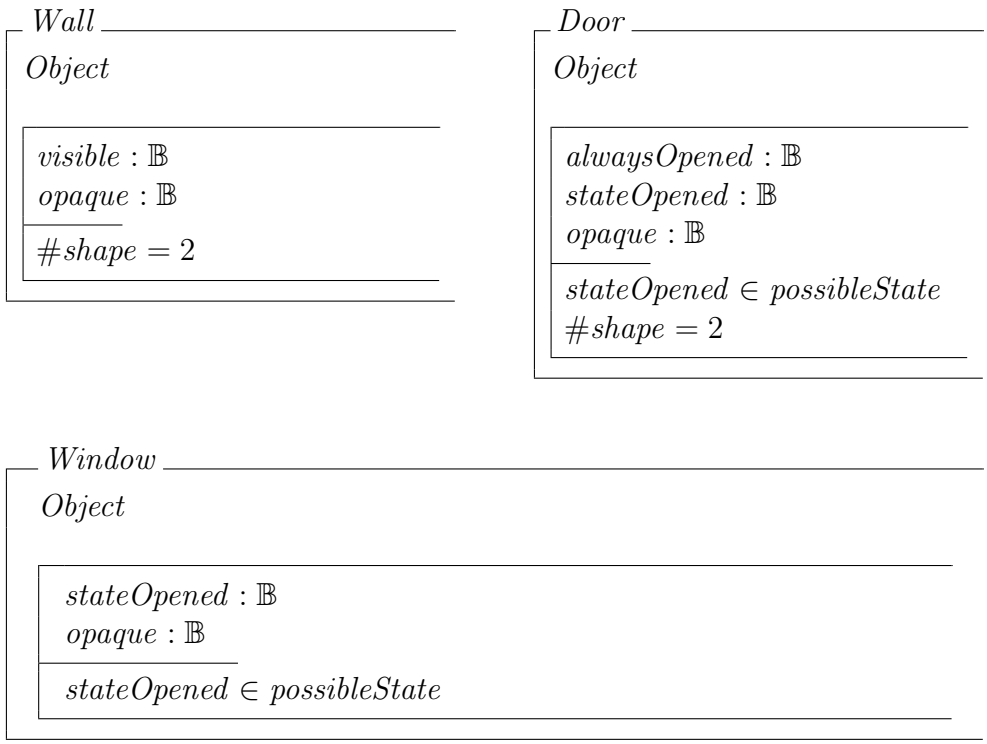


Figura 4.29: Características das classes Wall, Door e Window

Todas as classes herdam de **Object** e definem propriedades e estados para a especificação individual de cada uma. A propriedade **opaque** especifica opacidade: se o objeto é transparente ou não. A transparência influencia na visualização de espaços externos e nas comunicações. A propriedade **stateOpened** indica o estado da porta ou janela como aberto ou fechado. A propriedade **alwaysOpened** define uma abertura na parede e pode ser usada para representar qualquer tipo de abertura, seja uma porta ou até mesmo uma janela. A propriedade **visible** define se as paredes são concretas, para definir espaços fechados, semifechados e semi-abertos, ou imaginárias, para definir espaços abertos.

A posição dos delimitadores espaciais é definida através de coordenadas especificadas pela propriedade **shape**, herdada de **ModelBase**. As janelas podem adquirir qualquer forma, porém para paredes e portas foram definidas restrições para a construção somente de formas retangulares ou quadradas. São usados dois pontos no espaço para modelar tais objetos, conforme demonstrado na figura 4.30. A coordenada $x$ representa o comprimento, a coordenada $y$ a altura e a coordenada $z$ a profundidade.

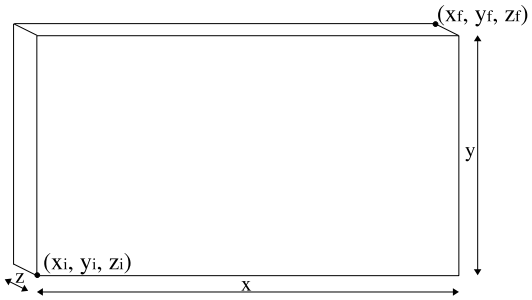


Figura 4.30: Coordenadas para especificar delimitadores espaciais

Devido ao mecanismo de herança, as delimitações podem ser redefinidas para suportar novos tipos de estruturas de espaços, como por exemplo, um espaço circular. Para facilitar a construção dos espaços, podem ser empregados espaços padrões, bastando apenas redimensionar o tamanho. Como exemplos, temos os citados na figura 4.28.

Com a definição de delimitadores espaciais, todos os elementos necessários para projetar um espaço foram abordados. A classe **Space** é a estrutura que encapsula todos os elementos e define um espaço no modelo.

*Space*

*ModelBase*

$name : Text$
$delimiters : \mathbb{F} \downarrow Object$
$persons : \mathbb{F}\ Person$
$objects : \mathbb{F}\ Object$
$entities : \mathbb{F} \downarrow Entity$

Figura 4.31: Características da classe Space

As propriedades para definir localização, tamanho, forma, entre outros são her-

dadas de **ModelBase**. Tais propriedades poderiam substituir os delimitadores do tipo parede, principalmente para espaços abertos. Entretanto, não permitem especificar com precisão propriedades da delimitação, como espessura e material.

Todo espaço possui um nome e a propriedade **name** é responsável por indicá-lo. Há espaço cozinha, sala, quarto, escritório, corredor, elevador, carro, praça, e assim por diante. O nome é uma referência rápida para o entendimento do significado do espaço real. A propriedade **delimiters** especifica os delimitadores, ou seja, os objetos que compõem os limites do espaço. As propriedades **persons**, **objects** e **entities** indicam as pessoas, objetos e entidades contidas no espaço em um determinado momento, visto que esses elementos podem se locomover para outros espaços.

Uma questão importante a ser monitorada é a troca de espaço, isto é, quanto um elemento ultrapassa um delimitador porta. Se o elemento estiver registrado no controlador de localização, uma operação deve atualizar sua posição simbólica para o novo espaço. O restante dos controladores também devem ser checados, pois ao entrar em um novo espaço o elemento se encontra em um novo contexto. Logo, as questões dependentes de energia e situação podem sofrer alterações.

## 4.14 Ambiente

Um ambiente de computação ubíqua fornece computação e comunicação onipresentes para o usuário através de dispositivos embutidos e espalhados em seu espaço. Durante o decorrer da dissertação, o termo ambiente é utilizado para referenciar os ambientes onde é aplicada essa filosofia. Por exemplo, uma casa, um escritório, um shopping, uma região ou até mesmo um carro. Esta seção apresenta e discute a caracterização de ambientes na especificação.

Espaços agrupam pessoas, objetos e dispositivos, impondo os limites físicos para esses elementos. Apesar desse papel, os espaços possuem características de posição e dependência entre si, pois podem fornecer acesso a espaços adjacentes. O ambiente é responsável por agrupar e gerenciar os espaços no modelo. Ele representa o nível mais alto da hierarquia de modelagem, ou seja, o nível máximo de generalização de um modelo. A classe **Environment** especifica a estrutura para representar um ambiente.

A propriedade **name** é um identificador para o ambiente e a propriedade **space**

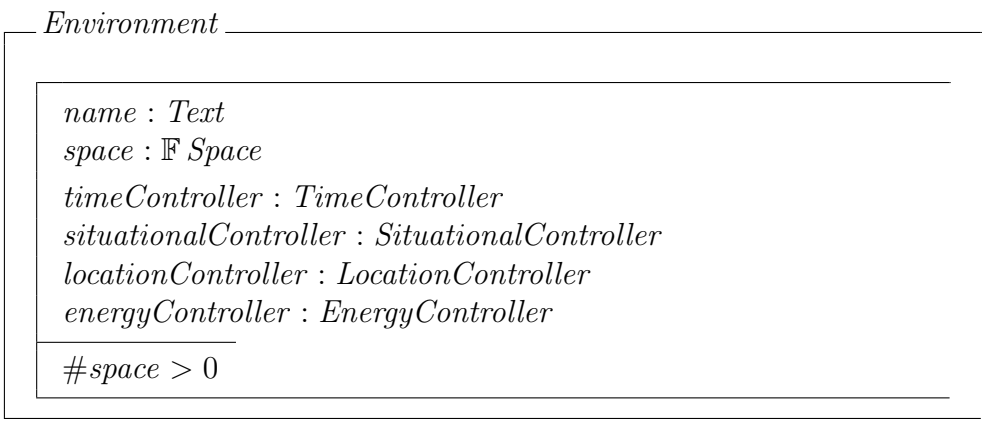


Figura 4.32: Características da classe Environment

agrupa todos os espaços que compõem o cenário modelado. Um ambiente deve ser composto no mínimo por um espaço. O ambiente especifica os controladores de tempo, situação, localização e energia. O controlador de tempo é responsável por controlar e sincronizar o relógio de simulação e para definir os eventos dependentes do tempo. O controlador de situação é responsável por registrar situações que dependem de eventos de elementos presentes em espaços comuns ou distintos. Os controladores de localização e energia registram a localização e a energia de todas as entidades para prover com eficiência acesso e controle sobre elas.

O processo de modelagem está finalizado quando um ambiente é especificado. Não há uma ordem para construir um cenário, mas se deve prestar atenção na definição dos elementos, comunicação, associações, eventos e as mudanças de estados a serem atingidas através da simulação do modelo.

# 5 Aplicação, Análise e Discussão

Na especificação são apresentados e discutidos os elementos necessários para a composição de ambientes de computação ubíqua. Esses elementos são especificados através da análise individual de partes menores e avaliações de situações simples nos ambientes. Através dessa metodologia é possível observar as características, o comportamento e as relações de cada elemento, visando o todo. Entretanto, não permite uma análise concreta de um ambiente completo com diversas situações distintas e simultâneas, e não permite fazer inferências mais detalhadas sobre o processo final de modelagem e simulação de situações, principalmente as mais complexas.

Baseado nesse contexto, para verificar as aplicações, limitações, problemas e interações decorrentes de um processo de modelagem e simulação em ambientes de computação ubíqua, é necessário aplicar, analisar e discutir os cenários originados de ambientes reais e mais complexos. Para tal, a especificação definida neste trabalho é utilizada como instrumento base. Com a sua aplicação é possível não somente avaliar o cenário, mas o próprio modelo provido pela especificação.

Para atingir tal objetivo, são utilizados três exemplos de cenários com diferentes graus de complexidade e quantidade de elementos. O primeiro exemplo é um cenário simples, com poucos elementos e interações. O segundo é um cenário complexo, com diversos elementos distintos, comunicações e interações. O terceiro também é complexo, mas somente algumas questões são discutidas. Ao final do capítulo, a especificação é comparada com dois projetos que trabalham com modelagem e simulação.

Durante a aplicação da especificação para a modelagem dos cenários são consideradas somente as principais características dos elementos envolvidos. A razão dessa simplificação é facilitar a visualização e o entendimento da aplicação de mo-

delagem formal e a não necessidade de representar certas características para todos os elementos. Logo, as especificações são aplicadas a alguns elementos e estendidas a elementos similares, alterando somente as propriedades mais relevantes e as que interferem nas relações e associações.

## 5.1 Cenário: Casa inteligente

A casa inteligente corresponde a um ambiente onde há dispositivos de computação ubíqua auxiliando o usuário em tarefas domésticas ou fornecendo comodidade e conforto. Este cenário é descrito a seguir:

"João, brasileiro, ouve seu rádio-relógio tocar às 05:30 da manhã. Como de costume, as luzes são acesas automaticamente e o barulho se interrompe após 30 segundos. Também, como de costume, João não levanta e dentro de cinco minutos o relógio, que já se adaptou ao comportamento de João, toca novamente, contudo não cessa até que ele o desligue. Sensores na casa percebem que João está acordado e o sistema prepara o ambiente tradicional para as manhãs, ou seja, seu computador é ligado, obtém as notícias de seu interesse, sua geladeira informa o estoque de suprimentos, seu visor de notas exibe a agenda do dia (...) "

Como um primeiro exemplo, esse cenário não é transposto diretamente para o modelo da especificação. A finalidade da aplicação e análise desse cenário é definir os elementos, as relações e os eventos, verificar se a especificação pode suportá-los e quais informações podem ser contabilizadas na simulação. O processo é divido em 4 fases: identificação dos elementos e funções, identificação das relações e associações, identificação dos eventos e preparação, medição e execução da simulação.

### 5.1.1 Identificação dos elementos e funções

Consiste em identificar os elementos, suas respectivas funções no mundo real e representar suas características pelo modelo da especificação. A classe do elemento no modelo é representada pela informação entre parênteses. Todos os elementos identificados nesse tópico correspondem a elementos físicos.

Os elementos diretamente identificáveis observando o cenário são João (Person), rádio-relógio (Object), sensores (Sensor), lâmpadas (Object), geladeira (Object),

computador (FixedDevice) e visor de notas (FixedDevice). A função de João no ambiente é interagir com os elementos; ele é a única entidade móvel presente no cenário. O rádio-relógio tem a função de ligar o aparelho ao atingir um horário programado, logo requer a existência de um dispositivo embutido (EmbeddedDevice) para controlar a hora e o alarme. Há um ou vários sensores verificando se João está dormindo. O sensor pode verificar essa informação através do contorno do corpo de João sobre a cama (Object). Como as lâmpadas são acesas ao alarme do despertador, há atuadores (Actuator) para executar a tarefa. A geladeira informa a falta de algum suprimento, logo há sensores (Sensor) para identificar a falta de determinado produto e um dispositivo embutido (EmbeddedDevice) para receber, informar e controlar o estoque. A casa constitui o ambiente (Environment) e pode ser representada por um único espaço (Space) ou vários espaços, cada um representando um cômodo. Esses espaços devem ser delimitados por paredes (Wall), portas (Door) e janelas (Window). Há outros elementos nesse cenário, mas não são importantes para este exemplo.

### 5.1.2 Identificação das relações e associações

Consiste em identificar as relações e associações entre os elementos físicos identificados na seção anterior. As associações indicam comunicação e composição entre elementos e as relações afetam os estados dos objetos e o comportamento do usuário no ambiente. Nesse estágio, são identificadas novas classes do modelo, em especial, os canais de comunicação.

O usuário, João, não tem nenhuma associação direta com dispositivos ou objetos e suas relações com o ambiente são apenas manuseio e uso de dispositivos fixos no ambiente. O rádio-relógio é associado a um dispositivo embutido. Há um canal de comunicação (CommunicationChannel) entre o dispositivo e o atuador responsável por acender a lâmpada. As associações entre o rádio-relógio e o dispositivo embutido permitem o controle do alarme, das horas e opções como ligar e desligar. Os sensores também estão associados ao rádio-relógio, pois através dos sensores é identificado se João levantou. Os sensores podem estar associados a outros dispositivos, tais como o computador e atuadores em outros cômodos. A comunicação pode estar sendo realizada pela rede elétrica, logo há a necessidade de definir outro canal de comunicação (CommunicationChannel). A geladeira está associada a um dispositivo

embutido e aos sensores. A comunicação é realizada entre os sensores e o dispositivo embutido por um canal de comunicação específico.

### 5.1.3 Identificação dos eventos

Consiste em identificar os eventos e sinalizações existentes no ambiente. Esses acontecimentos são os responsáveis pela execução de mudança de estado dos elementos no ambiente. Alguns eventos são fáceis de se identificar, pois são inerentes aos elementos e suas funções, outros são identificados considerando a finalidade da simulação.

O usuário, João, a princípio, pode levantar, desligar o alarme ou continuar dormindo. Pode também se locomover no ambiente e usar dispositivos. Estas ações podem ser aleatórias ou estabelecidas segundo um controle de tempo. A escolha depende de quais e como as medições estatísticas são aplicadas. De qualquer forma, essas ações constituem eventos e devem ser agendadas no controlador de tempo (TimeController). O rádio-relógio tem um evento agendado para disparar o alarme e deve cessar, na primeira vez, 30 segundos após ser acionado ou até o usuário desligar. Há um evento de agendamento e um de duração, sendo que o de duração pode ser sobreposto por um evento desligar executado pelo usuário. De qualquer maneira, se o usuário não levantar, um evento deve ser agendado para acionar o alarme novamente após cinco minutos. Esse evento, entretanto, somente ocorre se o sensor indicar que o usuário ainda está dormindo. Outro evento é o envio de um comando para o atuador acender as luzes. O sensor responsável por verificar se o usuário levantou deve enviar eventos para outros cômodos da casa com a finalidade de preparar o ambiente tradicional de todas manhãs. Os eventos podem ser comandos para acender as lâmpadas, ligar o computador e preparar o café. A geladeira possui uma interface para acionar um evento para exibir o estoque. Esse evento é originado de uma interface de controle provida pelo dispositivo embutido. O mesmo é válido para o visor de notas, contudo ele pode ser acionado via voz, por exemplo, e executar o evento para pronunciar os compromissos do dia.

### 5.1.4 Preparação, medição e execução da simulação

Consiste em estabelecer como a simulação é executada, quais variáveis são medidas e como preparar o modelo para suportar essas questões. Preparar a simulação é atender os requisitos para completar o modelo. As variáveis são acumuladores estatísticos e fornecem medidas para avaliar o modelo simulado. A execução consiste em estabelecer a ordenação e execução dos eventos.

Após o processo de identificação, preparar a simulação consiste em projetar e inserir os elementos no espaço, definir as variáveis de estado, as características e localização. Também deve ser definido o tempo de início de execução. Podem ser adicionadas limitações aos elementos, tais como alcance, interferências, entre outros. Nesse exemplo, a preparação consiste apenas em inserir os elementos em uma posição no ambiente e iniciar a simulação em um tempo específico. A especificação provê os mecanismos para alcançar essa finalidade.

A medição é baseada segundo o objetivo da simulação. Até o momento não foi estabelecido nenhum problema para ser respondido sobre o cenário. Por exemplo, o cenário acima poderia ser usado para representar um protótipo de cenário real e apenas ser usado para verificar a viabilidade de sua construção. Em outra situação, poderia ser usado para medir a quantidade mínima de sensores para a implementação prática. Ou ainda, para avaliar o tempo de resposta nos canais de comunicação. O modelo não especifica esse tipo de informação, pois ele foi desenvolvido somente para representar um mundo simulado e não para descrever e especificar as variáveis estatísticas.

A execução da simulação é obtida através da execução ordenada dos eventos. As classes **TimeController** e **SituationalController** controlam, respectivamente, eventos baseados em tempo e situações. Outros eventos são originados pelo uso de interfaces dos dispositivos. A mobilidade e autonomia do usuário podem ser definidas por sorteio, isto é, especifica-se a ordem das ações básicas e sorteia-se a marcação de tempo da execução. Uma alternativa é permitir o controle do usuário em tempo de execução da simulação. Desse modo, os eventos dependentes do usuário são originados segundo as expectativas do supervisor da simulação.

## 5.2 Cenário: Escritório

O escritório corresponde a um ambiente com sensores e dispositivos espalhados no espaço, e pessoas portando seus próprios dispositivos. Este cenário apresenta quantidade e variedade de dispositivos, sensores, objetos e pessoas. Como conseqüência, a mobilidade e as comunicações se destacam como fatores complicadores, e os eventos e restrições são mais variados e simultâneos. Tudo isso implica no aumento de complexidade para a modelagem e simulação.

A figura 5.1 ilustra o cenário do escritório evidenciando as classes dos elementos segundo a especificação definida no capítulo 4. Há uma escala para auxiliar na localização dos elementos e sinais indicativos para representar os elementos abstratos, tais como os canais de comunicação, os eventos e as associações.

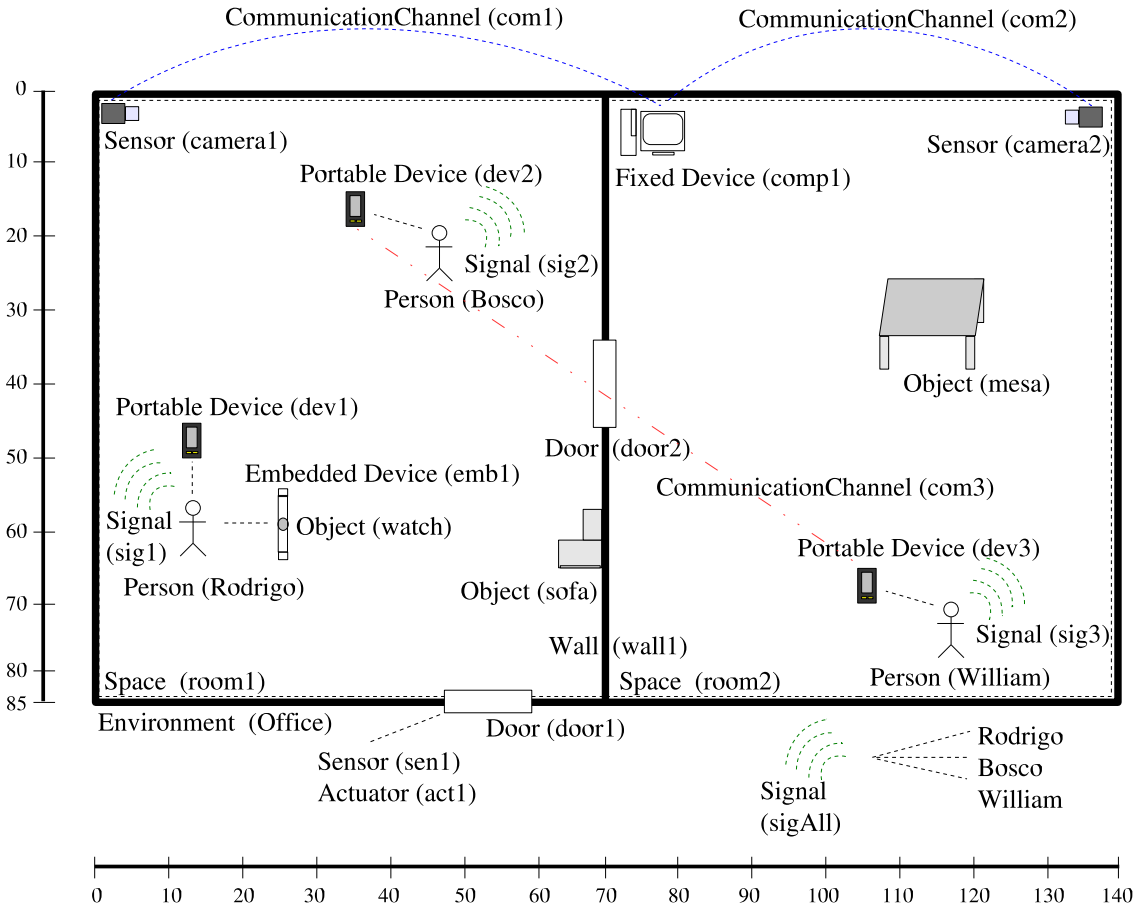


Figura 5.1: Ilustração do cenário escritório

O ambiente é composto por dois espaços com uma abertura entre eles. Há três pessoas com dispositivos portáteis e uma delas possui um dispositivo de GPS

embutido no relógio. Em cada espaço há uma câmera de vídeo para registrar as pessoas que se encontram no ambiente. Toda informação de registro é armazenada em um computador central. Há dois objetos ilustrados no escritório, apesar da existência de outros. A porta de entrada abre e fecha automaticamente ao perceber uma pessoa se aproximando.

A descrição anterior constitui o segundo exemplo de cenário para a análise e discussão. Nesse exemplo, são identificados todos os tipos de elementos definidos na especificação, exceto as janelas. O cenário ilustrado é transposto de acordo com as características especificadas no modelo. A transposição não é um mecanismo existente em Object-Z para verificar validade ou corretismo, mas permite inferir e refletir sobre a especificação e o cenário. Para acompanhar sua aplicação, recorra ao apêndice B.

Comumente ao exemplo da casa inteligente, os elementos, relações e associações são definidos e representados. Entretanto, a metodologia e o resultado obtido são distintos, pois as informações são mais detalhadas e especificadas obedecendo às estruturas e restrições do modelo. O resultado é um conjunto de objetos derivados de um conjunto base de classes. Uma analogia a essa metodologia é o uso de frameworks, onde há uma implementação inacabada provendo a base, cuja implementação e extensão resulta em uma aplicação específica.

Nesta seção é abordada a aplicação, análise e discussão da representação de todos os elementos físicos e abstratos do cenário. Os elementos são identificados, suas características e funções comentadas, enquanto os problemas sobre a adequação do modelo são discutidos. A ordenação da discussão se baseia na especificação do apêndice B.

As pessoas são as primeiras entidades avaliadas. Há três pessoas presentes no ambiente, cada uma portando seu(s) dispositivo(s). Como apresentado em capítulos anteriores, pessoa constitui uma entidade complexa por si só. Para minimizar essa complexidade, o modelo representa as características baseadas em propriedades psicológicas, físicas e posicionais. Não são abordadas somente as questões básicas de localização e associação com dispositivos ou objetos. Há flexibilidade para estender e representar outras propriedades. Baseada nessa flexibilidade, as propriedades extras **patience** e **tolerance_error** são adicionadas ao modelo específico. Infelizmente, o modelo não especifica e não implementa como devem ser usadas, mas viabiliza essa

possibilidade ao projetista.

Na especificação são definidas duas pessoas em movimento, como pode ser observado pela propriedade **speed**. A mobilidade é uma característica obrigatória, mas constitui um problema. Deve-se tratar colisões e estabelecer as rotas para os indivíduos no cenário. As colisões são difíceis de se tratar, pois não há somente elementos fixos no espaço. Estabelecer rotas implica em criar movimentos que se aproximem de condições reais do uso de um cenário. Uma alternativa é dividir o ambiente em regiões quadradas com áreas idênticas, como um tabuleiro de xadrez. Desse modo, os movimentos são coordenados e verificados segundo a ocupação dessas áreas. Esta solução resolve o problema de identificar colisões e minimiza o problema de identificar rotas, visto que um indivíduo pode se mover somente para os quadrados adjacentes. Porém, se as regiões adjacentes estiverem todas ocupadas, ocorre uma situação onde não há caminho acessível a partir de uma posição, conduzindo o indivíduo para um estado de inanição temporário.

Outra influência de pessoas sobre os cenários é quanto ao porte e o uso dos dispositivos. As propriedades **hasObject** e **hasEntity** especificam quais objetos são portados. O uso deve ser controlado através de eventos durante a simulação. Esses eventos utilizam informações de contexto do indivíduo para estabelecer um padrão de uso que se aproxime da realidade. Informações de início de uso, tempo e finalidade são variáveis importantes para a estatística de uma simulação. O modelo não especifica essas variáveis, mas é intuitivo coletá-las.

Há três objetos definidos no cenário, são eles o relógio, o sofá e a mesa. O relógio é portado por uma pessoa, logo possui mobilidade. Os outros dois objetos apenas ocupam espaço e interferem na mobilidade de outros elementos. Podem interferir, em outros casos, nas comunicações e sinalizações, mas suas dimensões praticamente não apresentam obstáculos nesse cenário. O relógio é associado a dois elementos, um dispositivo embutido e uma pessoa. A associação no caso do dispositivo embutido é uma composição para representar um relógio com GPS. Os objetos não possuem nenhum estado específico, pois não há um atuador para alterar os valores. No relógio, em especial, os estados estão associados às características do dispositivo embutido.

Os sensores presentes no ambientes são três: um sensor de detecção de presença na porta e dois sensores câmera posicionados no canto das salas. O sensor de detecção de presença aciona um atuador para abrir a porta quando uma pessoa se

aproxima a dois metros de distância. O comando é enviado ao atuador através de uma conexão por canal de comunicação e protocolo específicos. Na prática, o problema é como e quando deve ser realizada a detecção do sinal, pois não é satisfatório verificar a todo momento se existem elementos na área de percepção. Abstraindo esse problema, ele se resume a determinar a localização de elementos móveis. Os sensores câmera monitoram o ambiente e coletam imagens para identificar pessoas no cenário. Estão conectados a um computador principal através de canais de comunicação e utilizam protocolos idênticos para a transmissão do conteúdo. Na prática, os problemas são a colisão de identificação, isto é, monitorar e registrar consecutivamente a mesma pessoa, tratamento de requisições no serviço de armazenagem, pois há duas câmeras para serem atendidas, e como anteriormente, detectar a presença de um elemento móvel em sua área de percepção.

Na especificação, os sensores são dependentes dos sinais enviados. Há quatro sinais neste cenário, sendo que um deles é compartilhado por três elementos. Os sinais **sig1**, **sig2** e **sig3** representam a imagem capturada pela câmera, neste caso, a face e o corpo. Um cuidado em sua detecção é verificar se a orientação da pessoa está direcionada para câmera. A opção **target** melhora o desempenho na execução de uma simulação, pois se um elemento estiver em um espaço onde não haja sensores que detectem esse sinal, não é necessário se preocupar com a sua propagação constante. O sinal **sigAll** indica presença e tem sua propagação em forma circular com um raio de alcance de 3 metros. Como ele é idêntico para todos os elementos pessoa, é necessária apenas uma instância para representá-lo, portanto a propriedade **source** é identificada como vazia.

O atuador está comprometido a agir sobre o objeto porta, especificado como **door1**. Ele atua sobre o estado da porta indicando se está aberta ou fechada. Recebe o comando diretamente do sensor de detecção de presença. Quando o sensor detecta a presença de um elemento envia o comando abrir (**openCommand**), se não detecta a presença envia o comando fechar (**closeCommand**). Se for muito constante o envio destes comandos pode ocorrer uma sobrecarga no atuador, originando filas contendo um mesmo comando. Uma alternativa é definir uma variável interna ao sensor para indicar o não envio de comandos quando as detecções são consecutivas ou em intervalos próximos. Estatísticas sobre filas de acesso a recursos é uma variável que deve ser avaliada em simulação.

O dispositivo embutido, denominado **emb1**, é um receptor de GPS associado ao objeto relógio. A sua especificação foi baseada em um produto real, com as funcionalidades para informar tempo, localização, distância, velocidade, histórico e navegação em mapa. Possui uma interface com seis botões de controle e uma tela para selecionar opções e exibir as informações. Os botões são representados na especificação como meio para entrada de dados e a tela como a saída de dados. Em um ambiente real, o dispositivo também está recebendo informações de satélites. Um satélite poderia ser especificado como um dispositivo de entrada. Outra alternativa é especificar o satélite como um dispositivo fixo e utilizar as propriedades de comunicação para associá-lo ao dispositivo embutido. As propriedades **entry** e **exit** permitem visualizar as informações de entrada e saída em um instante determinado, por isso, nessa especificação, são apenas um exemplo simbólico. Na prática, deve-se elaborar e utilizar técnicas estatísticas e computacionais para gerar as entradas e as saídas.

As entradas e saídas possuem relação com outras características do dispositivo. Os processos fornecem os recursos para modificar, formatar e criar dados de entrada e saída. Os estados permitem visualizar informações e alterações internas no dispositivo. A memória armazena dados e os eventos causam alterações no estado do dispositivo(s) associado(s). Essas propriedades são especificadas brevemente, pois tentam abstrair a estrutura de uma máquina real, cuja complexidade não pode apenas ser satisfeita por uma especificação sucinta. Entretanto, elas descrevem esses elementos e seus papéis, contribuindo para racionalizar sobre a existência dessas questões durante a construção de um modelo para simulação. Desse modo, em um ambiente de simulação, pode-se simular algoritmos locais ou distribuídos, que fazem o uso de eventos, memória e alteram variáveis de estados. Neste cenário, os processos apenas realizam ações básicas, os estados são utilizados para exibir as informações, a memória armazena o mapa e o histórico, e os eventos são responsáveis pela atualização dos estados ou ativar processos para tratar de entradas.

Uma carência do modelo é não especificar como os processos monitoram a ocorrência de eventos. Na especificação, os eventos estão apenas associados aos dispositivos, quando na verdade quem está interessado em um evento é um processo. Isto não foi abordado para não introduzir um nível mais refinado de especificação. Em um próximo nível de abstração deve ser tratado, assim como diversas outras questões sobre os dispositivos não abordados até o momento. Todas as questões dis-

cutidas sobre o dispositivo embutido são válidas para os outros dispositivos presentes no cenário.

Há três dispositivos portáteis no ambiente: dois palmtops e um celular. A mobilidade e a capacidade de se conectar com outros dispositivos são os fatores que se destacam nestes aparelhos. O dispositivo **dev1** está desligado e não tem nenhuma conexão ativa. Os dispositivos **dev2** e **dev3** tem conexões estabelecidas através de um feixe de infravermelho. Há diversos obstáculos para se estabelecer e manter essa comunicação. Primeiro, como descobrir que um dispositivo está disponibilizando uma porta para conexão? Segundo, como, quando e qual a importância em manter a conexão durante a simulação? São muitas as variáveis envolvidas, como a interferência, o alinhamento, a distância entre os dispositivos e outras conexões existentes. Terceiro, como simular o protocolo de troca de informações? Neste caso, pode ser simples, mas em casos onde há muitas conexões simultâneas ou quando justamente o protocolo de comunicação é o objetivo do modelo simulado, não é trivial. Responder essas questões requer se aprofundar na especificação e ter consciência das técnicas para implementar eficientemente as soluções.

O dispositivo fixo é responsável por registrar os indivíduos no sistema. Como ele representa um computador tradicional, não há preocupações quanto a simulação de suas funções, pois os métodos para esse tipo de simulação são bem fundamentados. O que não é simples é tratar da carga recebida e gerenciar o tempo de atendimento para cada requisição, considerando ao mesmo tempo a simulação do restante do cenário. Um aspecto interessante de se relatar é que o dispositivo fornecedor dos dados de entrada corresponde a uma interface de rede. O canal de comunicação é descrito pelas estruturas **com1** e **com2**. Através dessas estruturas é possível obter informações sobre o meio de comunicação e calcular o tempo de chegada de uma requisição ao computador. O canal **com3** demonstra a inserção de obstruções a serem observadas durante a simulação. As propriedades **range** e **constraints** definem as limitações.

Em geral, nos ambientes de computação ubíqua, os dispositivos fixos provêem um ponto comum de acesso para os dispositivos móveis. Não é o caso nesse cenário, pois o dispositivo é usado apenas para verificar e armazenar registros coletados por sensores. Logo, toda a composição móvel deve ocorrer baseado em redes ad hoc. Em teoria, os dispositivos deveriam estar sempre preparados para tal composição,

entretanto como pode ser observado, o dispositivo **dev1** está desligado e não pode participar se for solicitado. Essa limitação é destacada, pois a realidade em ambientes reais é como descrito, aleatoriamente no tempo um indivíduo pode deixar seu dispositivo desligado ou simplesmente a bateria pode acabar.

Em se tratando de bateria, o dispositivo **dev1** está sendo monitorado pelo controlador de energia **energy_crt**. A precisão do controle é dependente de uma tabela com os percentuais de gastos para cada operação. Em dispositivos reais não é um problema este controle, pois o consumo é proporcional ao uso do dispositivo. Em simulação, apenas uma aproximação é alcançada.

Para simplificar o controle de comunicação e ilustrar o uso do controlador de localização, são definidas propriedades detalhadas de localização para a pessoa **William**. A localização relativa facilita o gerenciamento da direção e distância entre as entidades. Por exemplo, na comunicação por infravermelho contribui para alinhar os dispositivos, controlar a distância e avaliar a existência de objetos no percurso do feixe. No cenário atual temos este caso, onde somente a distância não é satisfeita, como conseqüência, a comunicação está inativa.

Definindo os espaços há os objetos porta e parede, denominados como delimitadores espaciais. O cenário contém sete paredes e duas portas. Não foi definido um teto, pois não interfere nesta modelagem. Poderia ser definido se o cenário fosse composto por diversos andares. Essa possibilidade implica em tratar de outras variáveis, como por exemplo, a comunicação entre elementos em andares distintos e a altitude (eixo z). A principal discussão quanto ao uso das paredes se refere à necessidade de sua aplicação, pois o espaço pode definir seus limites através da propriedade **shape** herdada de **ModelBase**. A escolha em sempre definir paredes é uma opção para padronizar e possibilitar agregar propriedades exclusivas. Um caso excêntrico de aplicação pode ser observado no exemplo a seguir. Suponha uma pessoa interessada em trocar informações de contato somente com pessoas conhecidas de vista. Basicamente, a pessoa localiza visualmente com quem deseja trocar informações, aproxima-se e aciona o dispositivo. A ação de visualizar uma pessoa é totalmente dependente dos obstáculos visuais. Nesta situação, parede desempenha um papel importante sobre o comportamento das entidades em um modelo, ainda mais se as paredes forem, por exemplo, transparentes.

As portas, além de entradas e saídas, atuam como gatilhos de eventos sobre

os cenários. Uma entidade ao atravessar uma porta, entra em um novo contexto e diversos controladores devem ser atualizados. Além disso, pode ocasionar uma série de novos eventos nos elementos do espaço recém chegado. Por exemplo, se a pessoa **Rodrigo** ir ao espaço **room2**, o sensor **camera2** vai capturar sua imagem, **dev3** pode estabelecer uma conexão e o seu dispositivo, em uma situação imprevista, pode causar alguma interferência nas comunicações locais. Como porta é um objeto, possui estados e associações. A porta **door1** está associada a um sensor e um atuador, tem seu estado modificado através de ações (**Action**) ativadas por comandos recebidos pelo atuador. A definição desse esquema garante o isolamento e a documentação dos papéis dos elementos no cenário. Por fim, as portas podem ser usadas como entradas e saídas de elementos na simulação. Neste cenário, **door1** não está conectada a dois espaços, logo todo elemento que passar por ela está sendo criado ou eliminado.

Quanto aos espaços e ao ambiente não há muito que comentar, apenas englobam os elementos que compõem o cenário. O ambiente ainda suporta os controladores de tempo, situação, energia, localização e comunicação.

## 5.3 Cenário: Shopping Center

O shopping center corresponde a um cenário composto por uma grande quantidade de pessoas com seus dispositivos e por diversos espaços fechados e abertos (lojas, áreas de exposição, salões, escadas rolantes) bem delimitados fisicamente pela estrutura do shopping. Há sensores e dispositivos distribuídos, monitorando e fornecendo serviços para os indivíduos nos limites internos do ambiente. Os serviços têm como objetivo conquistar e fornecer comodidade aos usuários. Para os serviços não serem intrusivos, isto é, não cometerem abusos, o ambiente dispõe de uma infra-estrutura para proteger a privacidade de seus usuários. Exemplos de serviços abusivos são o envio de propagandas e anúncios para os dispositivos de usuários sem consentimento, coletas de informação sem permissão (ex.: dados pessoais), rastreamento e identificação sem autorização.

A estrutura física do ambiente corresponde à estrutura real de um shopping center. Os dispositivos e sensores podem estar localizados no espaço de lojas. Eles podem se comunicar internamente na loja, isto é, o dispositivo de um cliente é

detectado dentro dos limites da loja, ou ainda se comunicar a uma determinada distância externa à loja. O mesmo princípio é válido para os sensores. Além disso, sensores e dispositivos podem ser de propriedade do shopping e estarem distribuídos em outros pontos, sendo compartilhados por diversas lojas.

Com o objetivo de fornecer privacidade aos clientes, o shopping tem sensores nas portas de entrada/saída para atribuir um pseudônimo de identificação e registrar as políticas de privacidade definida pelos clientes. As políticas restringem ou liberam as informações que podem ser enviadas ou coletadas dos dispositivos. São armazenadas em um servidor central, acessado por todos os sensores e dispositivos via uma conexão segura por cabo, pois assim evitam protocolos desgastantes de verificação e economizam a bateria do dispositivo do cliente. Para garantir o anonimato do cliente, são inseridas zonas de mixagem (mix zones) (BERESFORD; STAJANO, 2003) nos ambientes movimentados. Nessas zonas o usuário não pode ser rastreado e seu pseudônimo é misturado e alterado, garantindo anonimato e evitando que dispositivos rastreiem sua identificação anterior. A figura 5.2 ilustra a estrutura e as zonas de mixagem do shopping.

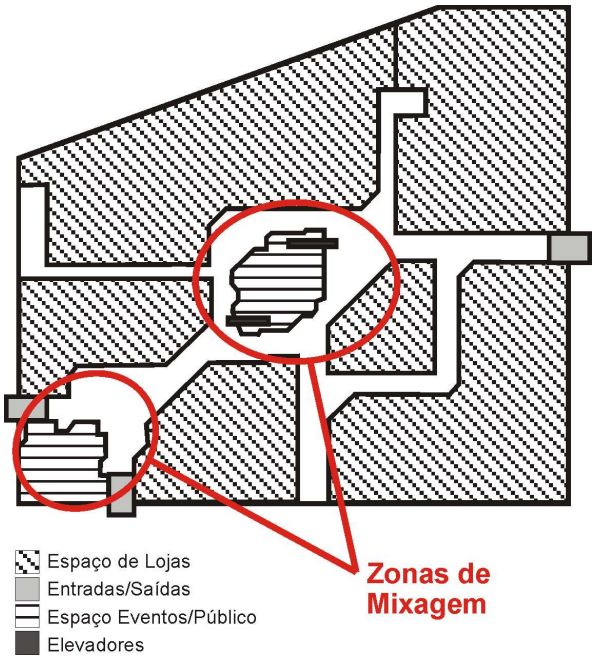


Figura 5.2: Ilustração do cenário shopping

Diferentemente dos outros cenários, há um problema definido para utilizar simulação e avaliar as perspectivas da solução. O problema é garantir a privacidade no ambiente. A solução é utilizar políticas de privacidade e zonas de mixagem, intro-

duzindo mais sensores e dispositivos ao cenário e criando protocolos para viabilizar a proposta.

Sob esta óptica, nesta seção são discutidas algumas questões pertinentes para a implantação de um modelo real do ambiente. Pretende-se evidenciar os benefícios do uso de modelagem e simulação através da apresentação e discussão das diferentes preocupações, com o objetivo de encontrar respostas que atendam aos requisitos do problema e da solução.

### 5.3.1 Modelagem

A primeira providência para modelar o cenário é definir e posicionar os sensores e dispositivos. Na definição, as características definidas pelas classes **Sensor** e **Device** norteiam o projetista, facilitando a especificação das propriedades. Obviamente, o projetista deve ter o conhecimento sobre as características reais dessas entidades. Entretanto, as classes concentram a atenção quanto à associação entre sensores de um mesmo tipo, associação com outros dispositivos, formas e distância de percepção do sinal, modificação de estados internos, eventos, entradas e saídas dos dispositivos, entre outros. No posicionamento, os sensores e dispositivos são inseridos obedecendo as restrições de suas características. Logo, pode-se observar a interceção de áreas de cobertura e percepção, concentração insuficiente ou excessiva em determinadas regiões, e adequação não satisfatória em determinadas posições. Com a simulação, são observados a interferência ou ineficiência para a coleta de informação e comunicação com os clientes, a interferência nas áreas isoladas, por exemplo, nas zonas de mixagem, medição de variáveis estatísticas, entre outros. Através dos resultados, pode-se ainda modificar facilmente a posição dos sensores e dispositivos para avaliar se a posição dos sensores de coleta de política de privacidade é a mais adequada, onde e quantas zonas de mixagem são necessárias, quantos e quais sensores podem ser compartilhados para diminuir os gastos individuais das lojas.

### 5.3.2 Desempenho

A distribuição dos sensores e dispositivos influencia no desempenho das entidades e comunicações, mas há outros fatores. Esses fatores são provenientes das estruturas de dados, protocolos, políticas e meios de comunicação utilizados. A

modelagem permite representar esses fatores e a simulação permite testar e avaliar individualmente ou conjuntamente as implicações sobre o desempenho no cenário.

A preocupação com as estruturas tem origem nas características inerentes aos dispositivos, principalmente nos embutidos e portáteis. Devido às restrições de memória e energia, a definição de estruturas de dados otimizadas é de crucial importância para o sucesso de um ambiente de computação ubíqua. No cenário, há duas estruturas importantes a serem abordadas. Primeiro, a estrutura que define a política de privacidade do usuário. Esta política é definida pelo próprio usuário e transmitida ao sistema pelo seu aparelho. Com o auxílio da modelagem e simulação, pode-se testar diferentes estruturas para averiguar o desempenho. Além das estruturas, pode-se averiguar e definir o melhor dispositivo para armazenar as informações, por exemplo, celular, cartão, chaveiro, entre outros. Segundo, a estrutura para armazenar e transmitir as políticas coletadas para as entidades distribuídas no shopping. A princípio, são armazenadas em um servidor central através de uma conexão por cabo e segura. Pode-se testar outras opções, como por exemplo, servidores distribuídos.

Os protocolos de comunicação são responsáveis por intermediar a comunicação entre os dispositivos do shopping e os dispositivos dos clientes. Por conseqüência, estão sujeitos às restrições de memória, energia e disponibilidade de comunicação. Construir protocolos que exijam minimamente a interação com os dispositivos requer simular diferentes implementações e situações no cenário. O primeiro protocolo ativado ao entrar no cenário é o envio das políticas e atribuição do pseudônimo. Se for executado por uma conexão sem fio, a infra-estrutura deve ser elaborada para que todos os usuários sejam atendidos. Na zona de mixagem, deve-se estabelecer um protocolo capaz de misturar os pseudônimos considerando diferentes números de pessoas. Na simulação, pode-se avaliar a taxa efetiva, a freqüência e as colisões de alteração, intervalo de tempo entre as mixagens e se o anonimato é garantido. Também por simulação, os protocolos de envio de propagandas e anúncios, coletas de informação e identificação podem ser avaliados para averiguar o desempenho em diferentes situações.

As políticas definem diversos aspectos sobre o funcionamento dos sensores e dispositivos no ambiente. São responsáveis por estabelecer a ordem e evitar que o excesso de comunicações prejudique ou interfira no funcionamento geral do ambiente.

Por meio da modelagem e simulação pode-se definir as melhores políticas e averiguar as implicações sobre o cenário. Por exemplo, pode-se definir e simular diferentes freqüências de envio de propagandas para obter mínimas colisões e perda de pacotes e maximizar o número de clientes atingidos.

Os meios de comunicação fornecem os canais e conexões para a interação entre os dispositivos no ambiente. A principal preocupação é com as interferências. Com a aplicação da modelagem e simulação é possível averiguar alguns dos problemas e tentar recriar um cenário aproximado do real, pois muitas suposições sobre simulação em redes sem fio não abrangem todos os aspectos reais. De qualquer forma, problemas e estatísticas básicas sobre a comunicação podem ser inferidas.

### 5.3.3 Escalabilidade

Lidar com escalabilidade dos componentes em ambientes de computação ubíqua é um dos grandes desafios para tornar real a visão proposta por Mark Weiser. Em simulação também é um problema, pois é difícil simular um ambiente com muitas variáveis influenciadas por elevado e constante nível de interação. Neste cenário, em especial, ocorre esta situação. O espaço físico e a quantidade de pessoas com dispositivos são grandes, logo exige avaliar a escalabilidade dos elementos no cenário.

Supondo a ausência de problemas de escalabilidade para a simulação, ou seja, um número expressivo de elementos pode ser adicionado e simulado em um cenário. Por meio de proposições e variações da quantidade desses elementos é plausível avaliar as implicações sobre o funcionamento real do cenário. Desse modo, é viável constatar a quantidade mínima ou máxima de pessoas que as entidades em uma região podem cobrir, a quantidade de dispositivos que podem ser atendidos em um determinado instante e situação, os diferentes comportamentos e decisões das pessoas mediante as situações, as modificações necessárias para solucionar problemas, entre outros. Pode-se ainda concluir se a solução trouxe benefícios ou prejuízos ao cenário. Para tal, compara-se os índices estatísticos medidos sem e com a aplicação da solução, considerando o benefício da garantia de privacidade.

## 5.4 Discussão dos trabalhos relacionados

Na seção 3.5 foram apresentados dois trabalhos correlatos que acreditam nos benefícios da simulação, assim como esta dissertação. Ubiwise é o simulador resultante do projeto desenvolvido por Barton e UbisWorld é o resultado do projeto desenvolvido por Heckmann.

Ubiwise apesar de oferecer uma visão interessante, combinando a simulação em duas visões, uma em 2D e outra em 3D, tem muitas limitações para simular cenários complexos. As suas características são direcionadas para simular a interação entre pessoas e dispositivos que possuem algum tipo de interface gráfica, como por exemplo, câmeras, palmtops e celulares. Através da interface os usuários podem acionar os eventos e interagir com o mundo simulado. A principal contribuição realmente está em representar o modelo simulado em três dimensões. A desvantagem mais marcante é a não coleta de variáveis estatísticas sobre a simulação, como por exemplo, o tempo.

UbisWorld oferece uma rica ontologia para representar pessoas, objetos, localização, tempo, eventos e suas propriedades e características. A ontologia de representação do usuário é a mais detalhada, pois o projeto tem por objetivo prover uma ontologia para modelagem do usuário em ambientes de computação ubíqua. Apesar dessa especificidade, o resultado foi uma ontologia que representa diversas partes do mundo real e simula diversos cenários por regras de inferência e eventos. A principal contribuição é utilizar a ontologia como base para modelagem e simulação de cenários. A desvantagem é a não visualização das interações e não tratamento de protocolos de comunicação.

A especificação desta dissertação combina algumas características de ambas as abordagens. A relação com o Ubiwise é na representação dos dispositivos, processos e simulação de protocolos. A relação com o UbisWorld é a modelagem dos elementos, também evidenciando a importância do usuário nos cenários. Todos os três trabalhos têm por finalidade desenvolver peças de simulação para solucionar os problemas enfrentados para modelar, testar e avaliar cenários reais. No presente trabalho, um diferencial é a tentativa de aproximar a representação do modelo com a simulação, considerando as transições de estados e as variáveis específicas de simulação.

O Ubiwise é um projeto finalizado e inacabado enquanto o Ubisworld tem grande

parte completada e ainda continua em desenvolvimento. O Ubiwise utiliza XML para descrever e relacionar os elementos, UbisWorld utiliza ontologias e este trabalho utiliza uma linguagem de especificação formal (Object-Z). Não há como comparar diretamente as formas de representação, entretanto as ontologias parecem suprir melhor as necessidades para modelagem que as outras notações. A flexibilidade e representatividade das linguagens específicas de ontologias permitem expressar e associar os elementos mais facilmente comparadas com as lógicas matemáticas clássicas. Por outro lado, as lógicas são mais rigorosas para definir restrições, transições de estado e apresentar provas de corretismo.

# 6 Conclusões e Trabalhos Futuros

A dificuldade para modelar e simular ambientes de computação ubíqua conduziu a investigação dos diversos aspectos presentes nestes ambientes. Por meio da especificação dos requisitos foi projetado um modelo que considera as propriedades necessárias para suprir a carência em processos de modelagem e simulação dos cenários baseados nessa filosofia. A especificação abstraiu as questões mais relevantes e definiu as estruturas para agregar e representar computacionalmente os elementos presentes em cenários reais, além de fornecer estruturas para tratar de questões inerentes à simulação. O resultado final é um modelo que define a maioria das estruturas para a composição, modelagem e discussão de uma diversidade de ambientes de computação ubíqua.

Todos os elementos básicos necessários para a composição de cenários foram contemplados ou podem ser derivados da especificação. Algumas características foram especificadas mais detalhadamente, outras descritas sucintamente. Independentemente dessa questão, a reflexão sobre as características permite projetar modelos de simulação mais facilmente e padronizados, pois definem nomenclatura, representação, relacionamentos e problemas nos cenários. Também contribuem para o projeto de modelos mais completos. Como as características e problemas básicos são fornecidos, os projetistas são obrigados a vislumbrar novas características e novos problemas.

Através da especificação do modelo, os requisitos e os obstáculos para a aplicação prática de modelagem e simulação em ambientes de computação ubíqua foram apresentados, discutidos e analisados. Desse modo, pode-se utilizar essas informações para tornar realidade o projeto de um simulador que atenda as expectativas para modelar, testar e avaliar os elementos, interações e implicações nos cenários. Entre-

tanto, deve-se ressaltar que há um longo caminho para se percorrer antes de tornar a especificação um modelo de implementação.

O uso de uma linguagem de especificação formal, Object-Z, trouxe diversos benefícios para tratar as questões de representação, associação e relação entre os elementos. No entanto, em outros aspectos apresentou-se ineficiente ou complexa para representar conceitos relacionados com o tempo e comunicação. De qualquer forma, o foco do trabalho é discutir e analisar os aspectos e não somente se concentrar na modelagem dos cenários. Conseqüentemente, as lógicas matemáticas, a sintaxe e a semântica definidas por Object-Z são suficientes.

Além de uma especificação de requisitos, a especificação pode ser comparada com uma ontologia ou com um framework. Ontologia é uma especificação formal explícita de como representar objetos, conceitos, entidades e os relacionamentos entre eles, no escopo de um domínio específico. Como o presente trabalho define estruturas e relacionamentos entre elas em um domínio específico através de uma linguagem formal, a especificação pode ser considerada uma ontologia. Por outro lado, frameworks são estruturas de classes que constituem implementações incompletas que, estendidas, permitem produzir diferentes artefatos de software (SILVA, 2000). Na especificação foram definidas apenas classes inacabadas, isto é, as classes apresentam as características e os comportamentos básicos. Logo, para uma simulação específica, devem ser estendidas e modeladas segundo o problema abordado. Baseado nessa observação, a especificação pode ser considerada um framework.

A aplicação do modelo permitiu observar suas vantagens e desvantagens para modelagem e futura simulação de ambientes de computação ubíqua. Como vantagens destacam-se a capacidade de representar e distribuir os elementos nos cenários, a possibilidade de fazer inferências sobre problemas e soluções no modelo e a avaliação da viabilidade de simulação. Como desvantagens destacam-se a complexidade de representar eventos, interfaces e protocolos nos cenários. Entretanto, esta complexidade de representação pode ser amenizada através da construção de um software de modelagem para automatizar a especificação.

Por fim, pode-se confirmar que computação ubíqua é um paradigma promissor, porém ainda enfrenta muitas limitações para se tornar onipresente na vida e ambientes das pessoas. Não foram estabelecidos os limites para seu crescimento e desenvolvimento, portanto pode-se afirmar que está em evolução. Talvez, a dificul-

dade de se estabelecer estes limites, deve-se ao fato de sua constante e gradativa presença em cenários atuais e a quantidade de idéias interessantes que surgem diariamente. Inevitavelmente, não é possível distinguir o que é, não é, ou será real nos anos que se procedem.

A discussão dos aspectos de modelagem e simulação de ambientes de computação ubíqua permitiu vivenciar a realidade atual e a realidade proporcionada por ambientes impregnados de dispositivos ubíquos. Para motivar as pesquisas futuras, as conclusões são finalizadas com uma citação de Trovato (AHMED; THOMPSON, 2004, p. 96), no Percom2004[1], na qual ele compara o paradigma de computação ubíqua com o início da Internet dizendo: "*muitas idéias interessantes, mas nenhum caminho definido*".

## 6.1 Perspectivas Futuras

O presente trabalho procurou contribuir para a modelagem e simulação de ambientes de computação ubíqua. Entretanto, o assunto abordado, devido a sua complexidade e diversidade, necessita de pesquisas futuras e aperfeiçoamentos. A seguir, são apresentadas possíveis ramificações para continuar o tema versado nesta dissertação.

1. Aplicar outros métodos de modelagem formal para ressaltar as características não endereçadas por Object-Z. Em específico, questões referentes ao tempo e comunicação. Apesar de Object-Z permitir a especificação dessas características, há outros formalismos capazes de expressar mais claramente situações dependentes do tempo e comunicação. A partir da modelagem dessas características, pode-se analisar novos cenários destacando a simulação de situações em que elas estão sob avaliação, e verificar se elas realmente suprem as necessidades quando comparadas com outras modelagens.

2. Especificar mais detalhadamente todos os elementos que compõem os cenários e construir uma pequena biblioteca com alguns objetos pré-especificados. A especificação atual fornece apenas uma visão de cada elemento e introduz idéias para sua expansão. Expandir os elementos na especificação depende do

[1] www.percom.org

estudo aprofundado dos elementos reais correspondentes. Novas características ou subgrupos de objetos com características comuns podem ser encontrados.

3. A especificação apenas descreve os requisitos e os relacionamentos. Para implementar o modelo em um meio computacional é necessário um modelo de implementação. Para tal, diagramas UML (*Unified Modeling Language*) podem ser projetados baseados na especificação. Como conseqüência, teremos uma descrição mais detalhada das estruturas para modelar e simular os cenários de computação ubíqua.

4. Implementar um simulador ou algumas de suas partes baseadas nas estruturas definidas pela especificação. Alcançar este objetivo requer conhecimentos de técnicas e métodos de simulação, além de sobrepor os desafios para representar computacionalmente a complexidade dos cenários de computação ubíqua. Questões de desempenho e qualidade das simulações são os principais obstáculos. A princípio, a especificação é concentrada nas interações, logo uma interface gráfica é uma das exigências básicas no processo de construção da ferramenta de modelagem e simulação.

5. A ontologia apresentada no UbisWorld é mais completa que esta especificação para conceituar as entidades nos ambientes de computação ubíqua. Entretanto, não relaciona nenhuma questão quanto à implementação de simulação. Adaptar a ontologia a esta especificação permite a elaboração de uma especificação detalhada sobre o processo de modelagem e simulação. A ontologia pode ajudar a preencher possíveis lacunas e a padronizar algumas inconsistências no modelo. Posteriormente, entidades descritas com a ontologia podem ser transpostas para um simulador baseado na especificação.

# *Referências*

ABOWD, G. D. et al. Context-aware computing. *IEEE Pervasive Computing*, v. 1, n. 3, p. 22–23, July-September 2002.

AHMED, T.; THOMPSON, M. Percom 2004: Prommoting the pervasive computing paradigm. *IEEE Pervasive Computing*, v. 3, n. 3, p. 94–96, July-September 2004.

BARTON, J. J.; VIJAYRAGHAVAN, V. *UBIWISE, A Ubiquitous Wireless Infrastructure Simulation Environment*. Palo Alto, November 2002.

BERESFORD, A. R.; STAJANO, F. Location privacy in pervasive computing. *IEEE Pervasive Computing*, v. 2, n. 2, p. 46–55, April-June 2003.

BILLINGHURST, M. Special issue on wearable computing. *Personal and Ubiquitious Computing*, v. 6, n. 1, p. 1–2, February 2002.

CAMPBELL, R. et al. Towards security and privacy for pervasive computing. In: *International Symposium on Software Security*. Tokyo, Japan: [s.n.], 2002.

CARRUTHERS, J. Encyclopedia of telecommunications. In: _____. New York: Wiley, 2002. cap. Wireless Infrared Communications.

CULLER, D. E.; HONG, W. Wireless sensor networks. *Communications of the ACM*, v. 47, n. 6, p. 30–33, June 2004.

DAVIES, N.; GELLERSEN, H.-W. Beyond prototypes challenges in deploying ubiquitous systems. *IEEE Pervasive Computing*, v. 1, n. 1, p. 26–35, January-March 2002.

DRYER, D. C.; EISBACH, C.; ARK, W. S. At what cost pervasive? a social computing view of mobile computing systems. *IBM Systems Journal*, v. 38, n. 4, p. 652–676, 1999.

EBLING, M. R.; HUNT, G. D.; LEI, H. Issues for context services for pervasive computing. In: *Workshop on Middleware for Mobile Computer*. Heidelberg, Germany: [s.n.], 2001.

EDWARDS, W. K. *Core Jini*. 1. ed. [S.l.]: Pearson Education, 1999. (Java Series).

EKLUND, C. et al. Ieee standard 802.16: A technical overview of the wirelessman air interface for broadband wireless access. *IEEE Communications Magazine*, v. 40, n. 6, p. 98–107, June 2002.

ERICKSON, T. Some problems with the notion of context-aware computing. *Communications of ACM*, v. 45, n. 2, p. 102–104, February 2002.

FERREIRA, A. B. de H. *Novo Dicionário Aurélio da Língua Portuguesa*. 3. ed. [S.l.]: Positivo, 2004.

FILHO, P. J. de F. *Introdução à Modelagem e Simulação de Sistemas*. [S.l.]: Visual Books, 2001.

GARLAN, D. et al. Project aura: Toward distraction-free pervasive computing. *IEEE Pervasive Computing*, v. 1, n. 2, p. 22–31, April-June 2002.

GOLDBERG, L. R. The development of markers for the big-five factor structure. *Psychological Assessment*, v. 4, n. 1, p. 26–42, March 1992.

GORDON, G. *System Simulation*. [S.l.]: Prentice-Hall, 1969.

GRIMM, R. et al. System support for pervasive applications. *ACM Transactions on Computer Systems*, v. 22, n. 4, p. 421–486, November 2004.

HANSMANN, U. et al. *Pervasive Computing*. 2. ed. [S.l.]: Springer, 2003.

HECKMANN, D. Ubiquitous user modeling for situated interaction. In: *8th International Conference User Modeling*. Sonthofen, Germany: [s.n.], 2001. p. 280–282.

HECKMANN, D. *UbisWorld - Research on User Modeling and Ubiquitous Computing*. January 2005. Page: http://www.u2m.org/. PHD Thesis.

HIGHTOWER, J.; BORRIELLO, G. Location systems for ubiquitous computing. *IEEE Computer*, v. 34, n. 8, p. 57–66, August 2001.

HILL, J. et al. The platforms enabling wireless sensor networks. *Communications of the ACM*, v. 47, n. 6, p. 41–46, June 2004.

KAHN, J. M.; BARRY, J. R. Wireless infrared communications. In: *IEEE*. [S.l.: s.n.], 1997. v. 85, n. 2, p. 265–298.

LEICK, A. *GPS Satellite Surveying*. 2. ed. New York: Wiley, 1995.

LYYTINEN, K.; YOO, Y. Issues and challenges in ubiquitous computing. *Communications of the ACM*, v. 45, n. 12, p. 62–65, December 2002.

MANN, S. Wearable computing as means for personal empowerment. In: *International Conference on Wearable Computing*. Fairfax VA: [s.n.], 1998.

MIT, L. for C. S. *MIT Project Oxygen*. November 2004. Page: http://oxygen.lcs.mit.edu/.

MORLA, R.; DAVIES, N. Evaluating a location-based application: A hybrid test and simulation environment. *IEEE Pervasive Computing*, v. 3, n. 3, p. 48–56, July-September 2004.

MOTOROLA. *Motorola Automotive*. November 2004. Page: http://www.motorola.com/automotive/.

PEGDEN, C. D. et al. *Introduction to Simulation Using Siman*. 2. ed. [S.l.]: McGraw-Hill Companies, 1995.

SATYANARAYANAN, M. Pervasive computing: Vision and challenges. *IEEE Personal Communications*, v. 8, n. 4, p. 10–17, August 2001.

SENGERS, P. et al. Culturally embedded computing. *IEEE Pervasive Computing*, v. 3, n. 1, p. 14–21, January-March 2004.

SILVA, R. P. e. *Suporte ao desenvolvimento e uso de frameworks e componentes*. Tese (Doutorado) — Programa de Pós-Graduação em Computação - UFRGS, Porto Alegre, Março 2000.

SMITH, G. P. *An Object-oriented Approach to Formal Specification*. Tese (Doutorado) — University of Queensland, Australia, October 1992.

SOUZA, J. P.; GARLAN, D. Aura: An architectural framework for user mobility in ubiquitous computing environments. In: *Software Architecture: System Design, Development, and Maintenance Proceedings of the 3rd Working IEEE/IFIP Conference on Software Architecture*. [S.l.: s.n.], 2002. p. 29–43.

SOUZA, J. P.; GARLAN, D. *The Aura Software Arquitecture: An Infrastructure for Ubiquitous Computing*. [S.l.], August 2003. Disponível em: <http://reports-archive.adm.cs.cmu.edu/anon/2003/CMU-CS-03-183.pdf>.

SPIVEY, J. M. *The Z Notation : A Reference Manual*. [S.l.]: Prentice Hall, 1989.

STAJANO, F. *Security for Ubiquitous Computing*. 1. ed. [S.l.]: John Wiley & Sons, 2002.

STAJANO, F. Security for whom? the shifting security assumptions of pervasive computing. In: *International Security Symposium*. [S.l.]: Springer-Verlag GmbH, 2002. (Lectures Notes in Computer Science, v. 2609), p. 16–27.

STARNER, T. et al. Augmented reality through wearable computing. *Presence*, v. 6, n. 4, p. 386–398, 1997.

STARNER, T. E. Wearable computers: No longer science fiction. *IEEE Pervasive Computing*, v. 1, n. 1, p. 86–88, January-March 2002.

SUN. *Jini Architecture Specification*. 2003. Page: http://java.sun.com/products/jini/.

TANENBAUM, A. S. *Redes de Computadores*. 4. ed. [S.l.]: Editora Campus, 2003.

TENNENHOUSE, D. Proactive computing. *Communications of the ACM*, v. 43, n. 5, p. 43–50, May 2000.

VASCONCELOS, W. W. M. P. de. *O Tempo como Modelo: A Aplicação de Lógicas Temporais na Especificação Formal de Sistemas Distribuídos*. Dissertação (Mestrado) — Engenharia de Sistemas e Computação - COPPE - UFRJ, Rio de Janeiro, 1989.

WANT, R. et al. The active badge location system. *ACM Transactions on Information Systems*, v. 10, n. 1, p. 1, January 1992.

WARD, A.; JONES, A.; HOPPER, A. A new location technique for the active office. *IEEE Personal Communications*, v. 4, n. 5, p. 42–47, October 1997.

WEISER, M. The computer for the twenty-first century. *Scientific American*, p. 94–104, September 1991.

WEISER, M. Some computer science issues in ubiquitous computing. *Communications of the ACM*, v. 36, n. 7, p. 75–84, July 1993.

WEISER, M.; GOLD, R.; BROWN, J. S. The origins of ubiquitous computing research at parc in the late 1980s. *IBM Systems Journal*, v. 38, n. 4, p. 693–696, 1999.

WU, J.; STOJMENOVIC, I. Ad hoc networks. *IEEE Computer*, v. 37, n. 2, p. 29–31, February 2004.

ZIMMERMAN, T. G. *Personal Area Networks(PAN): Near-Field Intra-Body Communication*. Dissertação (Mestrado) — MIT Media Laboratory, 1995.

ZIMMERMAN, T. G. Wireless networked digital devices: A new paradigm for computing and communication. *IBM Systems Journal*, v. 38, n. 4, p. 566–574, 1999.

# *APÊNDICE A – Especificação do Modelo em Object-Z*

Neste apêndice é apresentada a especificação do modelo em Object-Z. As classes especificam as características e restrições dos elementos para modelar e simular ambientes de computação ubíqua, além de algumas funções básicas para gerenciar seus atributos e suas associações. Há classes que não apresentam os métodos básicos, como por exemplo, métodos de adição, acesso e remoção, para não tornar a especificação extensa e excessivamente carregada de funções não tão relevantes.

$$[Text, Data, Shape, Time, Date, Action, State]$$
$$Properties : Text \pfun Data$$
$$Direction : \{North, South, East, West, Northeast, Northwest, Southeast, Southwest\}$$

**ModelBase**

$$ClassType : \{Person, Object, Entity, Space\}$$

$$identification : \mathbb{N}$$
$$class : ClassType$$
$$description : Text$$
$$position : \mathbb{N} \times \mathbb{N} \times \mathbb{N}$$
$$mass : \mathbb{R}_1$$
$$width : \mathbb{N}$$
$$height : \mathbb{N}$$
$$shape : \mathbb{P}(\mathbb{R} \times \mathbb{R} \times \mathbb{R})$$
$$emittedSignal : \mathbb{F}\ Signal$$
$$location : Location$$
$$created : TimeStamp$$

---

$$\forall x, y : identification \bullet x \neq y$$
$$\forall x, y : identification;\ p1, p2 : position \mid x \neq y \bullet p1 \neq p2$$

**SetPosition**

$$\Delta(position)$$
$$p? : \mathbb{N} \times \mathbb{N} \times \mathbb{N}$$

---

$$\exists_1 s : Space \mid s \in Environment.space \bullet p? \in s.shape$$
$$position' = p?$$

*Person*

*ModelBase*

$[Positional, Physical, Psychological]$

$Personality : \{Extraversion, Agreeableness, Conscientiousness, EmotionalStability, Openness\}$

$$
\begin{array}{l}
positional : \mathbb{P}\, Positional \\
physical : \mathbb{P}\, Physical \\
psychological : \mathbb{P}\, Psychological \\
\\
speed : \mathbb{R} \\
direction : Direction \\
orientation : Direction \\
\\
gender : \{male, female\} \\
age : \mathbb{N} \\
\\
personality : Personality \rightarrow \mathbb{N} \\
\\
hasObject : \mathbb{F} \downarrow Object \\
hasEntity : \mathbb{F} \downarrow Entity \\
\hline
speed \in positional \\
direction \in positional \\
orientation \in positional \\
gender \in physical \\
age \in physical \\
personality \in psychological \\
\mathrm{ran}\, personality \leq 100
\end{array}
$$

*Init*

$$
\begin{array}{l}
class = Person
\end{array}
$$

$\Phi SelectEntity$

$$
\begin{array}{l}
\Delta(hasEntity) \\
e?, e' : \downarrow Entity \\
\hline
e? \in hasEntity \Rightarrow e' = e?
\end{array}
$$

$\Phi SelectObject$

$$
\begin{array}{l}
\Delta(hasObject) \\
o?, o' : \downarrow Object \\
\hline
o? \in hasObject \Rightarrow o' = o?
\end{array}
$$

*TakeObject*

$\Delta(hasObject)$
$o? : \downarrow Object$

---

$hasObject' = hasObject \cup \{o?\}$

*TakeEntity*

$\Delta(hasEntity)$
$e? : \downarrow Entity$

---

$hasEntity' = hasEntity \cup \{e?\}$

*PutObject*

$\Delta(hasObject)$
$o? : \downarrow Object$

---

$o? \in hasObject$
$hasObject' = hasObject \setminus \{o?\}$
$o?.SetPosition$

*PutEntity*

$\Delta(hasEntity)$
$e? : \downarrow Entity$

---

$e? \in hasEntity$
$hasEntity' = hasEntity \setminus \{e?\}$
$e?.SetPosition$

*Move*

$\Delta(speed, direction)$
$s? : \mathbb{R}$
$d? : Direction$

---

$d? \neq direction \Rightarrow direction' = d?$
$s? \neq speed \Rightarrow speed' = s?$
$speed' \neq 0 \Rightarrow SetPosition$

---

$\Box(speed \neq 0) \Rightarrow \Box(Move$ **occurs**$)$

*Object*

*ModelBase*

$$
\begin{array}{l}
initialState : \mathbb{P}\, State \\
possibleState : \mathbb{P}\, State \\
changeState : Action \pfun (State \rel State) \\
associations : \mathbb{F} \downarrow ModelBase \\
\hline
initialState \subseteq possibleState
\end{array}
$$

*INIT*

$$
class = Object
$$

*AddAssociation*

$$
\begin{array}{l}
\Delta(associations) \\
m? : \downarrow ModelBase \\
\hline
associations' = associations \cup \{m?\}
\end{array}
$$

*RemoveAssociation*

$$
\begin{array}{l}
\Delta(associations) \\
m? : \downarrow ModelBase \\
\hline
associations' = associations \setminus \{m?\}
\end{array}
$$

*ReceiveAction*

$$
\begin{array}{l}
\Delta(initialState) \\
a? : Action \\
\hline
a? \in \mathrm{dom}\, changeState \Rightarrow \\
\quad initialState' = (\ initialState \setminus \\
\quad \{first(changeState(a?))\}\ ) \\
\quad \cup \{second(changeState(a?))\}
\end{array}
$$

*Entity*

*ModelBase*

[*Process*, *Protocol*]

$enabled : \mathbb{B}$

$connections : \text{seq} \downarrow Entity$
$communication : \text{seq}\ Protocol$
$channel : \text{seq}\ CommunicationChannel$

---

$\#connections = \#communication = \#channel$
$\exists f : \text{ran}\ connections \times \text{ran}\ communication \times \text{ran}\ channel$
$\quad | \forall i : 1..\#f \bullet$
$\quad f(i) = (connections(i), communication(i), channel(i))$

*INIT*

$class = Entity$

*TurnOn*

$\Delta(enabled)$

---

$enabled' = \text{true}$

*TurnOff*

$\Delta(enabled)$

---

$enabled' = \text{false}$

*AddConnection*

$\Delta(connections, communication, channel)$
$e? : \downarrow Entity$
$p? : Protocol$
$c? : CommunicationChannel$

---

$connections' = connections^\frown < e? >$
$communication' = communication^\frown < p? >$
$channel' = channel^\frown < c? >$

*RemoveConnection*

$\Delta(connections, communication, channel)$
$index? : \mathbb{N}$

---

$connections' = squash(\{index?\} \lhd connections)$
$communication' = squash(\{index?\} \lhd communication)$
$channel' = squash(\{index?\} \lhd channel)$

*SendNotify*

$t! : \downarrow Trigger$

---

$connections \neq \langle\rangle$

*ReceiveNotify*

$t? : \downarrow Trigger$

---

$\Box\Diamond(SendNotify\ \textbf{enabled}) \Rightarrow$
$\Box\Diamond(\forall ent : \text{ran}\ connections \bullet ent.receiveNotify\ \textbf{occurs})$

*Sensor*

*Entity*

$type : \mathbb{N}$
$perceptionShape : Shape$
$perceptionDistance : \mathbb{N}$
$knownSignal : SignalType$
$internalState : \mathbb{P}\, State$
$stateTransition : Signal \pfun (State \rel State)$

---

$\#knownSignal = 1$
$\mathrm{dom}\ \mathrm{ran}\ stateTransition \subset internalState$
$\mathrm{ran}\ \mathrm{ran}\ stateTransiction \subset internalState$

*PerceiveSignal*

$\Delta(internalState)$
$s? : Signal$

---

$knownSignal = s?.knownSignal \Rightarrow$
$\quad internalState' = (\ internalState \setminus$
$\quad \{first(stateTransition(s?))\}\ )$
$\quad \cup \{second(stateTransition(s?))\}$

*SendNotify*

$e? : Event$
$c? : Command$

---

$\forall\, d : Device \mid d \in \mathrm{ran}\ connections \bullet t! = e?$
$\forall\, a : Actuator \mid a \in \mathrm{ran}\ connections \bullet t! = c?$

---

$\Box\Diamond(PerceiveSignal\ \mathbf{enabled}) \Rightarrow \Box\Diamond(SendNotify\ \mathbf{occurs})$

*Actuator*

*Entity*

$object : \mathbb{P} \downarrow Object$
$action : \mathbb{P}\, Action$
$actuation : Command \pfun (Object \rel Action)$

---

$\mathrm{dom}\ \mathrm{ran}\ actuation \subseteq object$
$\mathrm{ran}\ \mathrm{ran}\ actuation \subseteq action$

*ReceiveNotify*

$c! : Command$

---

$c! = t?$

*SendAction*

$c? : Command$
$a! : Action$
$o! : Object$

---

$o! = first(actuation(c?))$
$a! = second(actuation(c?))$

---

$\Box\Diamond(ReceiveNotify\ \mathbf{enabled}) \Rightarrow \Box\Diamond(SendAction\ \mathbf{occurs})$

*Trigger*

$identification : \mathbb{N}$
$contents : \mathbb{P}\ Properties$

$\forall\, x, y : identification \bullet x \neq y$

*Disparate*

[abstract operation to run a event]

*Stop*

[abstract operation to cancel a event]

*SignalType*

$identification : \mathbb{N}$
$description : Text$

*EventType*

$identification : \mathbb{N}$
$description : Text$

*Signal*

*Trigger*

[*SignalConstraints*]

$type : SignalType$
$source : \downarrow ModelBase$
$target : \mathbb{P}\ Sensor$
$propagation : Shape$
$range : \mathbb{N}$
$constraints : SignalConstraints$

*Event*

*Trigger*

$type : EventType$
$source : \downarrow Device$
$target : \mathbb{P} \downarrow Device$

*Command*

*Trigger*

$$
\begin{array}{l}
source : \downarrow Entity \\
target : \mathbb{P}\, Actuator \\
sintaxe : Data \\
parameters : \mathbb{P}\, Data \\
\hline
sintaxe \in \operatorname{ran} contents \\
parameters \subset \operatorname{ran} contents
\end{array}
$$

*Device*

*Entity*

$[InputDevice, OutputDevice]$
$pid, mid : \mathbb{N}$

---

$model : Text$

$input : \mathbb{P}\, InputDevice$
$output : \mathbb{P}\, OutputDevice$
$entry : InputDevice \pfun \mathbb{P}\, Data$
$exit : OutputDevice \pfun \mathbb{P}\, Data$

$process : pid \pfun Process$
$aliveProcess : \mathbb{P}(Process)$

$state : \mathbb{P}\, State$
$stateTransition : Event \pfun (State \rel State)$

$memory : \mathbb{P}\, Data$
$addressMemory : mid \pfun Data$

$event : \mathbb{P}\, Event$
$eventMap : Event \pfun \mathbb{P}\, Device$

---

$input = \mathrm{dom}\, entry$
$output = \mathrm{dom}\, exit$
$aliveProcess = \mathrm{ran}\, process$
$\mathrm{dom}\ \mathrm{ran}\, stateTransition \subseteq state$
$\mathrm{ran}\ \mathrm{ran}\, stateTransition \subseteq state$
$memory = \mathrm{ran}\, addressMemory$
$event = \mathrm{dom}\, eventMap$

---

*Register*

$\Delta(eventMap)$
$e? : Event$
$d? : Device$

---

$e? \in Event$
$eventMap'(e?) = eventMap(e?) \cup \{d?\}$

---

*Unregister*

$\Delta(eventMap)$
$e? : Event$
$d? : Device$

---

$e? \in Event$
$eventMap'(e?) = eventMap(e?) \setminus \{d?\}$

$\Phi SelectInput$

$\Delta(input)$
$i?, i' : InputDevice$

---

$i? \in InputDevice \Rightarrow i' = i?$

$\Phi SelectOutput$

$\Delta(output)$
$o?, o' : OutputDevice$

---

$o? \in OutputDevice \Rightarrow o' = o?$

*SendNotify*

$e! : Event$

---

$e! \in event$

*ReceiveNotify*

$e? : Event$

---

$e? = t?$
[event implications]

---

$\Box\Diamond(SendNotify\ \mathbf{enabled}) \Rightarrow$
$\Box\Diamond(\forall\, d : Device \mid d \in eventMap(e!) \bullet d.ReceiveNotify\ \mathbf{occurs})$

*EmbeddedDevice*

*Device*

---

$embeddedIn : Object$

*PortableDevice*

*Device*

---

$owner : Person$

INIT

$owner = \varnothing$

---

$\Box(owner \neq \varnothing) \Rightarrow \Box(position = owner.position)$

*FixedDevice*

*Device*

---

$space : Space$

*Location*

$$symLocation : SymbolicLocation$$
$$relLocation : \text{seq}\ RelativeLocation$$

*SymbolicLocation*

$$place : Space$$
$$point : \mathbb{N} \times \mathbb{N} \times \mathbb{N}$$

*RelativeLocation*

$$toObject : \downarrow ModelBase$$
$$orientation : Direction$$
$$distance : \mathbb{N}$$

*Wall*

*Object*

$$visible : \mathbb{B}$$
$$opaque : \mathbb{B}$$
---
$$\#shape = 2$$

*Window*

*Object*

$$stateOpened : \mathbb{B}$$
$$opaque : \mathbb{B}$$
---
$$stateOpened \in possibleState$$

*Door*

*Object*

$$alwaysOpened : \mathbb{B}$$
$$stateOpened : \mathbb{B}$$
$$opaque : \mathbb{B}$$
---
$$stateOpened \in possibleState$$
$$\#shape = 2$$

*Space*

*ModelBase*

$$
\begin{array}{l}
name : Text \\
delimiters : \mathbb{F} \downarrow Object \\
persons : \mathbb{F}\ Person \\
objects : \mathbb{F}\ Object \\
entities : \mathbb{F} \downarrow Entity
\end{array}
$$

*INIT*

$$
\begin{array}{l}
class = Space \\
\forall\, d : \downarrow Object \mid d \in delimiters \bullet d.INIT \\
\forall\, p : Persons \mid p \in persons \bullet p.INIT \\
\forall\, o : Objects \mid o \in objects \bullet o.INIT \\
\forall\, e : Entity \mid e \in entities \bullet e.INIT
\end{array}
$$

$\Phi SelectPerson$

$$
\begin{array}{l}
\Delta(persons) \\
p?, p' : Person \\
\hline
p? \in persons \Rightarrow p' = p?
\end{array}
$$

$\Phi SelectObject$

$$
\begin{array}{l}
\Delta(objects) \\
o?, o' : Object \\
\hline
o? \in objects \Rightarrow o' = o?
\end{array}
$$

$\Phi SelectEntity$

$$
\begin{array}{l}
\Delta(entities) \\
e?, e' : \downarrow Entity \\
\hline
e? \in entities \Rightarrow e' = e?
\end{array}
$$

$\Phi SelectDelimiter$

$$
\begin{array}{l}
\Delta(delimiters) \\
d?, d' : \downarrow Object \\
\hline
d? \in delimiters \Rightarrow d' = d?
\end{array}
$$

*Environment*

$$
\begin{array}{l}
name : Text \\
space : \mathbb{F}\, Space \\
timeController : TimeController \\
situationalController : SituationalController \\
locationController : LocationController \\
energyController : EnergyController \\
communicationController : CommunicationController \\
\hline
\#space > 0
\end{array}
$$

*INIT*

$$
\begin{array}{l}
\forall\, s : Space \mid s \in space \bullet s.INIT \\
locationController.INIT \\
energyController.INIT \\
communicationController.INIT \\
situationalController.INIT \\
timeController.INIT
\end{array}
$$

*ΦSelectSpace*

$$
\begin{array}{l}
\Delta(space) \\
s?, s' : Space \\
\hline
s? \in space \Rightarrow s' = s?
\end{array}
$$

*CommunicationChannel*

$$
\begin{array}{l}
CommunicationConstraints : \{Interference, Delay, Obstruction, \\
\quad Noise, DataLost, Synchronization, DeniedAccess, ...\} \\
\\
physicalChannel : \{infrared, radio, opticalfiber, satellite, ...\} \\
source : \downarrow Entity \\
target : \mathbb{P} \downarrow Entity \\
propagation : Shape \\
range : \mathbb{N} \\
transmissionRate : \mathbb{R}_1 \\
frequency : \mathbb{R}_1 \\
constraints : \mathbb{P}\, CommunicationConstraints
\end{array}
$$

*TimeStamp*

$$date : Date$$
$$time : Time$$

*SetTime*

$$\Delta(date, time)$$
$$time? : Time$$
$$date? : Date$$

---

$$time' = time?$$
$$date' = date?$$

*TimeController*

$$clock : TimeStamp$$
$$scheduler : TimeStamp \leftrightarrow \downarrow Trigger$$
$$duration : TimeStamp \leftrightarrow \downarrow Trigger$$
$$frequency : TimeStamp \leftrightarrow \downarrow Trigger$$

---

$$\forall\, t : TimeStamp \mid (t \in \operatorname{dom} scheduler \land t = clock) \bullet$$
$$\quad scheduler(t).Disparate$$
$$\forall\, event : \downarrow Trigger \mid event \in \operatorname{ran} duration \bullet$$
$$\quad \exists\, t : TimeStamp \mid$$
$$\quad (t \in \operatorname{dom} scheduler \land scheduler(t) = event) \bullet$$
$$\quad clock = (t + duration^{\sim}(event)) \Rightarrow event.Stop$$
$$\forall\, event : \downarrow Trigger \mid event \in \operatorname{ran} frequency \bullet$$
$$\quad \exists_1\, t : TimeStamp \mid$$
$$\quad (t \in \operatorname{dom} scheduler \land scheduler(t) = event) \bullet$$
$$\quad \forall\, i : \mathbb{N}_1 \bullet$$
$$\quad clock = (t + i * frequency^{\sim}(event)) \Rightarrow event.Disparate$$

*INIT*

[add events]

$$clock.SetTime$$

*SituationalController*

$$situation : \mathbb{F}\, State \nrightarrow \downarrow Trigger$$

---

$$y : \mathbb{F}\, State \mid y \in \operatorname{dom} situation \bullet$$
$$(\forall\, x : State \mid (x \in y) \land (x = \text{true}) \bullet situation(y).Disparate)$$

*INIT*

[add situations]

*LocationController*

$$registeredElement : \mathbb{P} \downarrow ModelBase$$
$$location : \downarrow ModelBase \nrightarrow Location$$
$$registeredElement = \operatorname{dom} location$$

*Init*

[add elements and locations]

*EnergyController*

$$device : \mathbb{P} \downarrow Entity$$
$$energy : \downarrow Entity \nrightarrow \mathbb{R}$$
$$device = \operatorname{dom} energy$$
$$\forall x : \downarrow Entity \mid x \in device \land energy(x) = 0 \bullet x.TurnOff$$

*Init*

[add monitored devices]

*CommunicationController*

$$activeEntities : \mathbb{P} \downarrow Entity$$
$$activeChannels : \mathbb{P}\ CommunicationChannel$$
$$connections : CommunicationChannel \times \downarrow Entity$$
$$area : \mathbb{P}\ Shape$$
$$restrictArea : Shape \nrightarrow CommunicationChannel.Constraints$$
$$activeEntities = \operatorname{ran} connections$$
$$activeChannels = \operatorname{dom} connections$$
$$area = \operatorname{dom} restrictArea$$

*Init*

$$activeEntities = \varnothing$$
$$activeChannels = \varnothing$$
$$connections = \varnothing$$

[add restrict areas]

# *APÊNDICE B – Aplicação da Especificação*

Neste apêndice, a especificação do modelo apresentada no capítulo 4 e apêndice A é aplicada a um cenário de computação ubíqua. O cenário é o escritório descrito na seção 5.2.

A seguir, algumas considerações importantes para entender a aplicação da especificação:

- Em determinados pontos a linguagem textual é empregada para descrever propriedades complexas, como por exemplo, a propriedade **shape**.
- Propriedades compostas por muitos elementos são resumidas com a notação reticência (...).
- Elementos com propriedades semelhantes são especificados em uma mesma estrutura de classe e distinguidos pela notação ponto-e-vírgula (;).
- Alguns objetos são apresentados resumidos, explicitando somente as propriedades relevantes para discussão.
- Alguns objetos não foram especificados (ex.: eventos), apesar da existência de referência.

*Person* : *Rodrigo*

$identification = 1$
$class = Person$
$description =$ office employee
$position = (10, 63, 0)$
$mass = 70Kg$
$width = 40cm$
$height = 190cm$
$shape = cylinder$
$emittedSignal = \{sigAll, sig1\}$

$positional = \{position, speed, direction, orientation\}$
$physical = \{gender, age\}$
$psychological = \{personality, patience, tolerance\_error\}$

$speed = 0$
$direction = North$
$orientation = North$

$gender = male$
$age = 23$

$personality = \{(1, 70), (2, 80), (3, 90), (4, 50), (5, 70)\}$
$patience = 80\%$
$tolerance\_error = 60\%$

$hasObject = \{watch\}$
$hasEntity = \{dev1\}$

*Person* : *Bosco*

$identification = 2$
$class = Person$
$description =$ office boss
$position = (45, 27, 0)$
$mass = 80Kg$
$width = 50cm$
$height = 172cm$
$shape = cylinder$
$emittedSignal = \{sigAll, sig2\}$

$positional = \{position, speed, direction, orientation\}$
$physical = \{gender, age\}$
$psychological = \{personality\}$

$speed = 0,3m/s$
$direction = West$
$orientation = West$

$gender = male$
$age = 55$

$personality = \{(1, 95), (2, 80), (3, 80), (4, 70), (5, 70)\}$

$hasObject = \{\}$
$hasEntity = \{dev2\}$

*Person* : *William*

$identification = 3$
$class = Person$
$description =$ office employee under stress
$position = (115, 78, 0)$
$mass = 90Kg$
$width = 45cm$
$height = 185cm$
$shape = cylinder$
$emittedSignal = \{sigAll, sig3\}$
$location = william\_location$
$positional = \{position, speed, direction, orientation\}$
$physical = \{gender, age\}$
$psychological = \{personality, patience\}$
$speed = 0,4m/s$
$direction = Northwest$
$orientation = Northwest$
$gender = male$
$age = 21$
$personality = \{(1, 98), (2, 90), (3, 70), (4, 90), (5, 80)\}$
$patience = 50\%$
$hasObject = \{\}$
$hasEntity = \{dev3\}$

*Object* : *sofa*

$identification = 4$
$class = Object$
$description =$ furniture
$position = (63, 65, 0)$
$mass = 10Kg$
$width = 70cm$
$height = 90cm$
$shape = cube$
$emittedSignal = \{\}$

*Object* : *table*

$identification = 28$
$class = Object$
$description =$ furniture
$position = (108, 38, 0)$
$mass = 15Kg$
$width = 70cm$
$height = 100cm$
$shape = parallelogram$
$emittedSignal = \{\}$

*Object* : *watch*

$identification = 5$
$class = Object$
$description =$ watch with GPS
$position = (10, 63, 12)$
$mass = 0,1Kg$
$initialState = \{10h20m\}$
$possibleState = \{...\}$
$changeState = \{update\_time \mapsto (old\_time, new\_time)\}$
$associations = \{emb1, Rodrigo\}$

*Sensor* : *sen*1

$identification = 6$
$class = Entity$
$description =$ presence detector sensor
$position = (50, 85, 18)$
$enabled = true$
$connections = \langle act1 \rangle$
$communication = \langle prot4 \rangle$
$channel = \langle com4 \rangle$
$type = 101$
$perceptionShape = circle$
$perceptionDistance = 2m$
$knownSignal = presence$
$internalState = \{detect[false]\}$
$stateTransition = \{sigAll \mapsto (detect[false], detect[true])\}$

*Sensor* : *camera*1, *camera*2

$identification = 7;\ 8$
$class = Entity$
$description =$ cameras to monitoring the environment
$position = (0, 5, 3);\ (133, 5, 3)$
$enabled = true$
$connections = \langle comp1 \rangle$
$communication = \langle prot1 \rangle;\ \langle prot2 \rangle$
$channel = \langle com1 \rangle;\ \langle com2 \rangle$
$type = 100$
$perceptionShape = 120graus$
$perceptionDistance = 5m$
$knownSignal = video$
$internalState = \{\}$
$stateTransition = \{\}$

*Actuator* : $act1$

$identification = 9$
$class = Entity$
$description =$ open and close the door
$position = (50, 85, 13)$
$enabled = true$
$connections = \langle sen1 \rangle$
$communication = \langle prot4 \rangle$
$channel = \langle com4 \rangle$
$object = \{door1\}$
$action = \{open, close\}$
$actuation = \{openCommand \mapsto (door1, open),$
$closeCommand \mapsto (door1, close)\}$

*SignalType* : *video*

$identification = 100$
$description =$ video image in a specific time

*SignalType* : *presence*

$identification = 101$
$description =$ presence signal of a person

*Signal* : $sig1, sig2, sig3$

$identification = 201;\ 202;\ 203$
$contents = \{(face, face\_image), (body, body\_image)\}$
$type = video$
$source = Rodrigo;\ Bosco;\ William$
$target = \{camera1, camera2\}$
$propagation = frontal$
$range =$
$constraints = \{object\ obstruction\}$

*Signal* : *sigAll*

$identification = 200$
$contents = \{(presence, true)\}$
$type = presence$
$source = \{\}$
$target = \{sen1\}$
$propagation = circle$
$range = 3m$
$constraints = \{\}$

*Command* : *openCommand*

$identification = 503$
$contents = \{(sintaxe, open)\}$
$source = sen1$
$target = \{act1\}$
$sintaxe = open$
$parameters = \{\}$

*Command* : *closeCommand*

$identification = 504$
$contents = \{(sintaxe, close)\}$
$source = sen1$
$target = \{act1\}$
$sintaxe = close$
$parameters = \{\}$

*EmbeddedDevice* : *emb*1

$identification = 10$
$class = Entity$
$description =$ GPS
$position =$
$enabled = true$
$connections = \langle\,\rangle$
$communication = \langle\,\rangle$
$channel = \langle\,\rangle$
$model =$ Garmin Forerunner 201 GPS Watch
$input = \{power\_button, mode\_button, enter\_button,$
$leftarrow\_button, rightarrow\_button\}$
$output = \{display\}$
$entry = \{(power\_button, \{turn\_on, turn\_off, turn\_on, ...\}), ...\}$
$exit = \{(display, \{10:20, 10graus33m20s(N)20graus(W), ...\}), ...\}$
$process = \{(1, calculate\_position), (2, store\_distance),$
$(3, control\_time), (4, count\_pace), ...\}$
$aliveProcess = \{store\_distance, control\_time\}$
$state = \{display\_time, display\_distance, display\_pace, display\_position\}$
$stateTransition = \{(ev1, (display\_time.value, display\_time.new\_value), ...\}$
$memory = \{map, distance\_history\}$
$addressMemory = \{(1A, map), (2B, distance\_history)\}$
$event = \{ev1, ...\}$
$eventMap = \{(ev1, emb1), ...\}$
$embeddedIn = watch$

*EventType* : *change_time*

$identification = 1$
$description =$ change the current time

*Event* : $ev1$

$identification = 600$
$contents = \{(time, value)\}$
$type = change\_time$
$source = emb1$
$target = \{emb1\}$

*PortableDevice* : $dev1$

$identification = 11$
$class = Entity$
$description =$ palmtop
$position = (12, 63, 12)$
$mass = 0.109Kg$
$width = 11cm$
$height = 7cm$
$enabled = false$
$connections = \langle\,\rangle$
$communication = \langle\,\rangle$
$channel = \langle\,\rangle$
$model =$ Palm Zire 21
$input = \{datebook\_button, addressbook\_button, power\_button,$
$up\_button, down\_button, display\}$
$output = \{display\}$
...
$owner = Rodrigo$

*PortableDevice* : $dev2$

$identification = 26$
$class = Entity$
$description =$ mobile phone
$position = (46, 27, 12)$
$mass = 0.250Kg$
$width = 9cm$
$height = 4cm$
$enabled = true$
$connections = \langle dev3\rangle$
$communication = \langle prot3\rangle$
$channel = \langle com3\rangle$
$model =$ Nokia 7200
$input = \{number\_buttons, arrows\_buttons, menu\_button,$
$enter\_button\}$
$output = \{display\}$
$entry = ...$
$exit = ...$
...
$owner = Bosco$

*PortableDevice* : $dev3$

$identification = 27$
$class = Entity$
$description =$ palmtop
$position = (116, 78, 12)$
$mass = 0.109Kg$
$width = 11cm$
$height = 7cm$
$enabled = true$
$connections = \langle dev2 \rangle$
$communication = \langle prot3 \rangle$
$channel = \langle com3 \rangle$
$model = PalmZire21$
$input = \{datebook\_button, addressbook\_button, power\_button,$
$up\_button, down\_button, display\}$
$output = \{display\}$
$entry = ...$
$exit = ...$
$...$
$owner = William$

*FixedDevice* : $comp1$

$identification = 12$
$class = Entity$
$description =$ computer to register camera data
$position = (73, 10, 10)$
$mass = 5Kg$
$width = 30cm$
$height = 40cm$
$shape = cube$
$enabled = true$
$connections = \langle camera1, camera2 \rangle$
$communication = \langle prot1, prot2 \rangle$
$channel = \langle com1, com2 \rangle$
$model =$ Dell Dimension
$input = \{keyboard, network\}$
$output = \{monitor\}$
$entry = ...$
$exit = ...$
$process = \{(1, registra\_cam1), (2, registra\_cam2),$
$(3, identifica\_pessoa), ...\}$
$aliveProcess = \{registra_cam1, registra_cam2\}$
$state = \{\}$
$stateTransition = \{\}$
$memory = \{person\_info, image\}$
$addressMemory = \{(1A, person\_info), (4D, image)\}$
$event = \{\}$
$eventMap = \{\}$
$space = room2$

*Location* : *william_location*

$symLocation = william\_symbolic$
$relLocation = william\_relative$

*SymbolicLocation* : *william_symbolic*

$place = room2$
$point = (45, 78, 0)$

*RelativeLocation* : *william_relative*

$toObject = Bosco$
$orientation = Northwest$
$distance = 70$

*LocationController* : *location_ctr*

$registeredElement = \{William\}$
$location = \{(William, william\_location)\}$

*EnergyController* : *energy_ctr*

$device = \{dev1\}$
$energy = \{(dev1, 100)\}$

*TimeController* : *time_ctr*

$clock = 0h23m30s40 - 01/01/2005$
$scheduler = \{(0h24m05s - 01/01/2005, e[1]),$
$(0h24m10s - 01/01/2005, e[2]),$
$(0h50m00s - 01/01/2005, f[1]),$
$(1h00m00s - 01/01/2005, d[1]), ...\}$
$duration = \{(10m - 0h0m0s, d[1]), ...\}$
$frequency = \{(50m - 0h0m0s, f[1]), ...\}$

*SituationalController* : *situational_ctr*

$situation = \{(\{door1.stateOpened[true]\}, s[1]), ...\}$

*CommunicationController* : *communication_ctr*

$activeEntities = \{dev2, dev3\}$
$activeChannels = \{com3\}$
$connections = \{(com3, dev2), (com3, dev3)\}$
$area : \{SquareArea[(50, 80)(60, 90)]\}$
$restrictArea : \{(SquareArea[(50, 80)(60, 90)], Interference)\}$

*CommunicationChannel* : $com1, com2$

$physicalChannel =$ cabe
$source = camera1;\ camera2$
$target = \{comp1\}$
$propagation =$
$range =$
$transmissionRate = 10Mbps$
$frequency = 350MHz$
$constraints = \{\}$

*CommunicationChannel* : $com3$

$physicalChannel = infrared$
$source = dev3$
$target = \{dev2\}$
$propagation = line$
$range = 5m$
$transmissionRate = 100kbps$
$frequency =$
$constraints = \{object\ obstruction, signal\ lost, lighting\}$

*Wall* : $wall1$

$identification = 15$
$class = Object$
$description =$ separate the rooms
$position = (70, 0, 0)$
$width = 80$
$height = 3$
$shape = \{(70, 0, 0), (71, 85, 3)\}$
$visible = true$
$opaque = true$

*Wall* : $wall2$

$identification = 16$
$shape = \{(0, 0, 0), (70, 1, 3)\}$

*Wall* : $wall3$

$identification = 17$
$shape = \{(70, 0, 0), (140, 1, 3)\}$

*Wall* : $wall4$

$identification = 18$
$shape = \{(0, 0, 0), (1, 85, 3)\}$

*Wall* : $wall5$

$identification = 19$
$shape = \{(140, 0, 0), (139, 85, 3)\}$

*Wall* : *wall*6

$identification = 20$
$shape = \{(0, 85, 0), (70, 84, 3)\}$

*Wall* : *wall*7

$identification = 21$
$shape = \{(70, 85, 0), (140, 84, 3)\}$

*Door* : *door*1

$identification = 22$
$class = Object$
$description =$ main door
$position = (48, 85, 0)$
$width = 10$
$height = 2$
$shape = \{(48, 85, 0), (58, 84, 2)\}$
$initialState = \{stateOpened.false\}$
$possibleState = \{stateOpened.false, stateOpened.true\}$
$changeState = \{(open, (stateOpened.false, stateOpened.true)),$
$(close, (stateOpened.true, stateOpened.false))\}$
$associations = \{sen1, act1\}$
$opaque = true$
$alwaysOpened = false$
$stateOpened = false$

*Door* : *door*2

$identification = 23$
$class = Object$
$description =$ door between rooms
$position = (70, 35, 0)$
$width = 10$
$height = 2$
$shape = \{(70, 35, 0), (71, 45, 2)\}$
$opaque = true$
$alwaysOpened = true$
$stateOpened = true$

*Space* : *room*1

$identification = 24$
$class = Space$
$description =$ wait room
$position = (0, 0, 0)$
$width = 70$
$height = 3$
$shape = \{(0, 0, 0), (70, 85, 3)\}$
$name = room1$
$delimiters = \{wall1, wall2, wall4, wall6, door1, door2\}$
$persons = \{Rodrigo, Bosco\}$
$objects = \{sofa, watch\}$
$entities = \{dev1, emb1, dev2, camera1, sen1, act1\}$

*Space* : *room*2

$identification = 25$
$class = Space$
$description =$ internal room
$position = (70, 0, 0)$
$width = 70$
$height = 3$
$shape = \{(70, 0, 0), (140, 85, 3)\}$
$name = room2$
$delimiters = \{wall1, wall3, wall5, wall7, door2\}$
$persons = \{William\}$
$objects = \{table\}$
$entities = \{comp1, dev3, camera2\}$

*Environment* : *office*

$name = office$
$space = \{room1, room2\}$
$timeController = time\_ctr$
$situationalController = situational\_ctr$
$locationController = location\_ctr$
$energyController = energy\_ctr$
$communicationController = communication\_ctr$