Álvaro Messias Bigonha Tibiriçá

UMA ARQUITETURA DE *SOFTWARE* NEURO-REATIVA PARA SISTEMAS DE AUTOMAÇÃO DO AMBIENTE CONSTRUÍDO

Tese apresentada à Escola de Engenharia de São Carlos da Universidade de São Paulo, como parte dos requisitos para obtenção do Título de Doutor em Engenharia Mecânica.

Orientador: Prof. Titular Mário Pinotti Jr.

São Carlos, 25 de outubro de 2008.

FOLHA DE JULGAMENTO

Candidato: Engenheiro **ALVARO MESSIAS BIGONHA TIBIRIÇÁ**

Tese defendida e julgada em 11/12/2008 perante a Comissão Julgadora:

Aprovado

Prof. Titular **MARIO PINOTTI JUNIOR (Orientador)**
(Escola de Engenharia de São Carlos/USP)

Fabio Ferraz Júnior — Aprovado

Dr. **FABIO FERRAZ JUNIOR**
(Empresa Sensoft)

Aprovada

Profª. Associada **MARIA DA GRAÇA CAMPOS PIMENTEL**
(Instituto de Ciências Matemáticas e de Computação/USP)

APROVADO

Prof. Dr. **CARLOS MAGNO DE OLIVEIRA VALENTE**
(Centro Universitário de Araraquara/UNIARA)

APROVADO

Prof. Dr. **RAFAEL VIEIRA DE SOUZA**
(Centro Universitário Hermínio Ometto/UNIARARAS)

Prof. Associado **JONAS DE CARVALHO**
Coordenador do Programa de Pós-Graduação em
Engenharia Mecânica

Prof. Associado **GERALDO ROBERTO MARTINS DA COSTA**
Presidente da Comissão da Pós-Graduação da EESC

À Cíntia, aos meus pais e ao meu irmão,

Agradecimentos

São inúmeras as pessoas com quem contamos no dia-a-dia, sem as quais a execução de qualquer trabalho seria muito mais difícil. Deixo aqui, meus agradecimentos a todas elas, e especialmente:

- ao Prof. Mário Pinotti Jr., por sua inestimável amizade, confiança, conselhos e sugestões;
- ao Prof. Luís Carlos Passarini, pela amizade e colaboração para que este trabalho pudesse ser realizado;
- ao Prof. Rafael Vieira de Sousa, pelas conversas e sugestões sobre arquitetura reativas;
- ao Cristiano, pela amizade, pelo companheirismo e pela parceira em muitos momentos desse trabalho;
- à Cíntia, pelo apoio, carinho e compreensão durante todo esse trabalho;
- aos meus pais, pelo incentivo e apoio ao longo de toda minha caminhada;
- aos inúmeros professores e autores de livros e de artigos que nos acompanham durante nossa jornada, muito obrigado.

"*Nunca ande apenas pelos caminhos traçados, pois eles conduzem somente até aonde os outros já foram.*"

Alexander Graham Bell

Resumo

Tibiriçá, A. M. T. **Uma Arquitetura de *Software* Neuro-Reativa para Sistemas de Automação do Ambiente Construído.** 2008, 133p. Tese (Doutorado em Engenharia Mecânica) – Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, SP, 2008.

Esta tese propõe uma arquitetura de *software* neuro-reativa para sistemas de Automação do Ambiente Construído. O objetivo é facilitar o desenvolvimento, a manutenção e a expansão desses sistemas, através de três requisitos norteadores: modularidade, flexibilidade e capacidade de integração das partes. Um modelo baseado em unidades chamadas de “neurônios” e de “glândulas” é proposto. Esses elementos fundamentais têm características reativas e podem ser combinados formando diferentes sistemas de automação. Uma versão da arquitetura proposta é programada na linguagem Java utilizando tecnologias como CORBA e MySQL. Por fim, uma casa fictícia é utilizada como exemplo para demonstrar a aplicação da arquitetura proposta.

Palavras chaves: arquitetura de *software*, automação, sistemas de controle, automação predial, automação residencial.

Abstract

This thesis presents a neuro-reactive *software* architecture applied to building automation systems. The objective is to make development, maintenance and expansion of these systems easier through three main requirements: modularity, flexibility and parts integration capability. A model with units called “neurons” and “glands” is proposed. These fundamental elements have reactive characteristics and are combined to constitute automation systems. A version of proposed architecture is programmed in Java language using technologies like CORBA and MySQL. In the end, a fictitious home automation system is used as example.

Keywords: *software* architecture, automation, control systems, building automation, home automation.

Sumário

Índice de Figuras

1. Introdução

O homem passa mais tempo em casa do que em qualquer outro lugar (INTILLE, 2002). Segundo Kolokotsa *et al.* (2001), as pessoas permanecem 80% de suas vidas dentro das edificações. Promover condições de conforto e eficiência energética nos ambientes construídos tem se tornado mais importante a cada dia. Neste sentido, os avanços tecnológicos vêm permitindo que os computadores "governem" os ambientes buscando melhorar o conforto dos usuários, o consumo de energia e a segurança (CAYCI; CALLAGHAN; CLARKE, 2000).

"*Uma casa é como uma máquina para se viver dentro*" (SHARPLES; CALLAGHAN; CLARKE, 1999). As construções modernas têm fortes similaridades com uma máquina no sentido de conterem uma miríade de dispositivos mecânicos, elétricos, eletrônicos e computacionais (CALLAGHAN *et al.*, 2000). Esses dispositivos microprocessados podem se comunicar uns com os outros e se comportar "inteligentemente" (MOZER, 1998). Esse comportamento "inteligente" dos ambientes é uma meta que envolve uma variedade de disciplinas (COOK; DAS, 2007).

A literatura aponta pelo menos duas grandes vertentes preocupadas com a aplicação de sistemas computacionais nos ambientes construídos, também chamados de Ambientes Inteligentes quando são embarcados com diversos dispositivos computacionais. A primeira vertente tem como foco o controle do ambiente, isto é, sistemas que através de sensores monitoram variáveis ambientais (como luz, temperatura, movimentação, etc.) e atuam no ambiente através de atuadores (como aquecedores, luminárias, janelas operadas eletronicamente, etc.) (HAGRAS *et al.*, 2003). Algoritmos de controle visando eficiência são propostos por diversos autores: Guillemin (2003), Kolokotsa *et al.* (2006) e Dounis (2007). Busca-se aproveitar melhor os

recursos materiais e energéticos disponíveis como o calor e a luz solares, os ventos, etc., e garantir o conforto e a segurança aos usuários. Essa vertente está relacionada com as áreas conhecidas como Automação Predial e Automação Residencial, aqui tratada como Automação do Ambiente Construído.

Cabe neste momento, conceituar o termo Automação do Ambiente Construído que será utilizado ao longo de todo este trabalho. Automação refere-se à idéia de não intervenção humana; ou de sistemas capazes de executar tarefas com certo grau de autonomia. Ambiente Construído refere-se ao espaço físico artificial provido pelo homem, tais como edifícios, casas, dormitórios, cozinhas, banheiros, espaços públicos, etc. onde o homem passa a maior parte da sua existência. Conceitua-se, a partir dessas idéias, Automação do Ambiente Construído como os sistemas que possuem certo grau de autonomia para operação do Ambiente Construído através do controle automático de seus subsistemas. Como conseqüência da automação desses ambientes, novos serviços e novas facilidades são oferecidos aos usuários, provendo conforto, economia e segurança.

A segunda vertente se preocupa em criar ambientes saturados com dispositivos computacionais e com grande habilidade de comunicação (SATYANARAYANAN, 2001). Estes dispositivos devem interagir de forma transparente com as pessoas, isto é, sem sobrecarregar seus sentidos, através de uma interação suave (WEISER; BROWN, 1996). O objetivo é permitir que os computadores participem de atividades que tradicionalmente não envolvem computação, com as quais a interação com sistemas computacionais ocorrerá de forma imperceptível, como se estivesse interagindo com outras pessoas (COEN, 1998). Ao invés de colocar as pessoas no mundo virtual do computador, busca-se colocar o computador no

mundo real das pessoas (BROOKS, 1997). Esta área é conhecida como Computação Ubíqua ou Computação Pervasiva.

Enquanto a Automação do Ambiente Construído tem como preocupação fundamental o ambiente, a Computação Pervasiva preocupa-se fundamentalmente com as interações entre os diversos dispositivos computacionais dispersos no ambiente e as pessoas. No primeiro caso, a computação é um meio, isto é, utiliza-se o *software* para automatizar o ambiente. No segundo caso a computação é o fim, isto é, utiliza o ambiente como interface de comunicação do *software*.

As duas áreas se entrelaçam e possuem características comuns. Entre elas está a importância do reconhecimento de contexto (GUILLEMIN; MOREL, 2001; HAGRAS *et al.*, 2003; HELAL *et al*., 2005; COOK *et al.*, 2003). Dey (2001) define contexto como qualquer informação que pode ser usada para caracterizar a situação de uma entidade. Uma entidade é uma pessoa, um lugar ou um objeto que é considerado relevante para a aplicação. Reconhecer contexto significa poder reconhecer situações importantes para o funcionamento do sistema.

Nas duas áreas, a arquitetura de *software* desempenha um papel crucial, já que é ela que define a estrutura central do sistema, o *software*. A arquitetura de *software* engloba duas características importantes de um programa de computador: a estrutura hierárquica de componentes (módulos) procedimentais, e a estrutura de dados (PRESSMAN, 1995). A facilidade de programação, de reutilização e de expansão do *software* está intimamente ligada à arquitetura de *software* utilizada.

Apesar da importância da arquitetura de *software* nas duas áreas de pesquisa, os trabalhos ligados a Automação do Ambiente Construído praticamente não abordam esse tema. Provavelmente, isso ocorre porque a computação é um meio e não

um fim na Automação do Ambiente Construído, o que também reflete nos perfis dos pesquisadores desta área que na sua grande maioria não são ligados à área de computação. No entanto, a não preocupação com a arquitetura de *software* é um fator limitante na organização e na integração de sistemas em soluções de Automação do Ambiente Construído que se tornam cada vez mais complexas.

Os sistemas de Automação do Ambiente Construído estão sujeitos a constantes expansões e reconfigurações (COEN, 1997). Mudanças nos gostos, hábitos e vontades dos usuários, e a evolução tecnológica contínua são alguns dos fatores que devem ser levados em conta no projeto de um sistema de Automação do Ambiente Construído. Cada solução deve contemplar especificidades dos usuários e do ambiente, o que torna uma solução geral quase impraticável para estes sistemas (MOZER, 1998; DOUNIS *et al.*, 1995). Vale ressaltar ainda o caráter multi-objetivo, baseado em contexto e integrado dessas aplicações, as quais buscam satisfazer os usuários, ser eficientes energeticamente, garantir a segurança, entre outros, de forma integrada e de acordo com contexto (DOUNIS *et al.*, 1995; GUILLEMIN; MOLTENI, 2002; KOLOKOTSA *et al.*, 2002; HAGRAS *et al.*, 2003; HELAL *et al.*, 2005; DEY; ABOWD; SALBER, 1999; GUILLEMIN; MOREL, 2001). Uma arquitetura de *software* para estes sistemas deve ser capaz de lidar com todos esses aspectos.

1.1. Objetivos

O objetivo central deste trabalho é propor uma arquitetura de *software* para Automação do Ambiente Construído que facilite o desenvolvimento de sistemas de Automação do Ambiente Construído. Ao objetivo central se associam os seguintes objetivos:

- investigar soluções de *software* nas áreas de Automação do Ambiente Construído e Computação Pervasiva;
- investigar os sistemas de Automação do Ambiente Construído;
- implementar em *software* a arquitetura proposta; e
- exemplificar o uso da arquitetura proposta.

1.2. Contribuições

Duas contribuições principais se destacam:

- Proposição de uma Arquitetura de *Software* para Automação do Ambiente Construído.
- Programação da arquitetura proposta em *software*.

1.3. Organização da tese

Esta tese está organizada da seguinte maneira:

- **Capítulo 2.** São apresentados os principais paradigmas e tecnologias utilizados em sistemas de Automação do Ambiente Construído. As tecnologias apresentadas neste capítulo são fundamentais para a construção de sistemas de Automação de Ambiente Construído.
- **Capítulo 3.** Através de três estudos bibliográficos, descreve exemplos de arquiteturas de *software* relacionadas a sistemas de Automação do Ambiente Construído.

- **Capítulo 4.** Propõe uma Arquitetura de *Software* para Automação do Ambiente Construído baseada em unidades com características reativas chamadas neurônios e glândulas.

- **Capítulo 5.** Descreve a programação de uma versão da arquitetura proposta no capítulo 4.

- **Capítulo 6.** Ilustra a aplicação da arquitetura proposta numa casa fictícia.

- **Capítulo 7.** São apresentadas conclusões e considerações sobre as principais contribuições, e sugestões para trabalhos futuros.

2. Paradigmas e Tecnologias para Sistemas de Automação do Ambiente Construído

A literatura relacionada à Automação do Ambiente Construído possui uma variedade de soluções para sistemas de Automação do Ambiente Construído baseadas em diferentes paradigmas e tecnologias. Busca-se neste capítulo descrever os paradigmas e as tecnologias usuais nesta área do conhecimento. É importante ressaltar que estes não são concorrentes e sim complementares.

2.1. Paradigmas para Sistemas de Automação do Ambiente Construído

Nesta seção são tratados três paradigmas (Arquitetura Reativa, Redes Neurais Artificiais e Lógica Difusa) utilizados em aplicações de Automação do Ambiente Construídos. As Arquiteturas Reativas e as Redes Neurais Artificiais são base para a arquitetura proposta no capítulo 4. Controladores Difusos são utilizados cada vez mais em sistemas de controle no Ambiente Construído e precisam ser levados em consideração no desenvolvimento de soluções de automação para o ambiente construído.

2.1.1. Arquitetura Reativa

A Arquitetura Reativa foi proposta por Brooks (1986) como uma abordagem alternativa para sistemas de controle em robôs móveis. Nessa abordagem, não há um modelo abstrato interno do ambiente – a atuação é uma resposta direta a um estímulo ambiental – contrapondo-se à Arquitetura Deliberativa ou Clássica, mais antiga. Os mapeamentos entre estímulos e atuação são chamados de comportamentos.

É da combinação dos comportamentos que emerge o comportamento geral do sistema. A construção de sistemas reativos é freqüentemente conhecida como programação por comportamentos, já que o componente fundamental neste tipo de arquitetura é um comportamento (MURPHY, 2000).

Na Arquitetura Deliberativa há um módulo de planejamento que possui um modelo abstrato interno do ambiente. Este módulo processa os estímulos segundo esse modelo interno de ambiente e determina qual ação tomar. O comportamento geral do sistema é determinado por um ciclo contínuo e seqüencial de sensoriar, planejar e atuar (*Sense, Plan, Act*) (MURPHY, 2000). Brooks (1986) classificou esta arquitetura como decomposição vertical, na qual todo o processamento ocorre de maneira seqüencial e linear, em contraposição à decomposição horizontal (sistemas reativos), na qual cada comportamento é uma unidade independe e paralela de processamento. A Figura 1 ilustra esses dois conceitos.

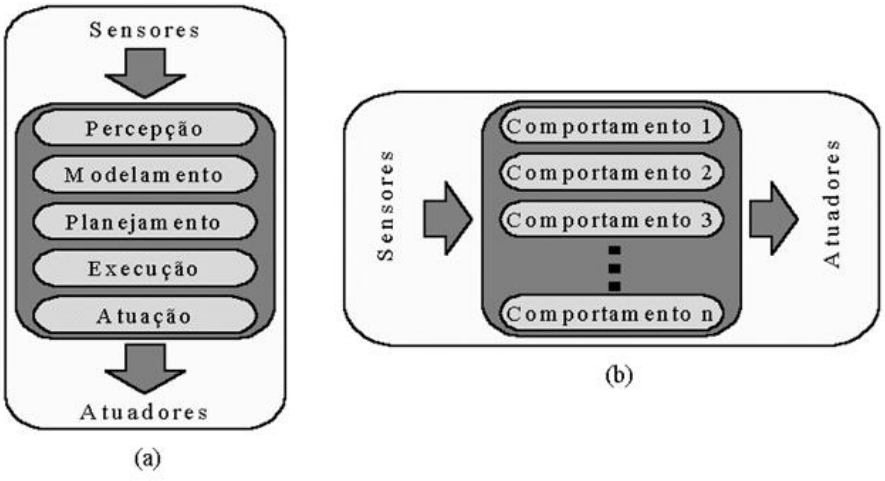

Figura 1 - Sistema de controle por decomposição: (a) vertical (arquitetura deliberativa) e (b) horizontal (arquitetura reativa). Adaptado de Brooks (1986).

A robótica móvel e os sistemas de automação do ambiente construído possuem similaridades. Ambos se relacionam diretamente com o mundo físico (HAGRAS *et al.*, 2003), ambos devem reagir a influências externas as quais estão sujeitos

(KULKARNI, 2002), ambos necessitam processar múltiplos eventos simultaneamente e reagir a mudanças rápidas de contexto (COEN, 1997), ambos capturam informações de uma variedade de sensores e usam inteligência embarcada para determinar as ações de controle (HAGRAS *et al.*, 2003). O ambiente construído é como "*um robô no qual vivemos dentro*" (HAGRAS *et al.*, 2003). A principal vantagem no uso de sistemas reativos é que estes descartam a necessidade de um modelo abstrato, substituindo-o pelo próprio ambiente (CALLAGHAN *et al.*, 2004). Este princípio é sumarizado por Brooks (1991): "**o mundo é ele próprio o melhor modelo**".

Murphy (2000) cita algumas vantagens na utilização da Arquitetura Reativa: a modularidade dos comportamentos e a facilidade de testá-los isoladamente; a facilidade de expansão do sistema aumentando sua capacidade; a possibilidade de surgimento de comportamentos complexos a partir da combinação de comportamentos mais simples; e o suporte às boas práticas de programação como decomposição, modularidade e testes incrementais. Essas características tornam o uso dessa arquitetura interessante para aplicações de Automação do Ambiente Construído.

Hagras *et al.* (2003) e Callaghan *et al.* (2004) utilizam a Arquitetura Reativa como base para construção de um sistema computacional para automação de um quarto, chamado de i.Dorm. Para lidar com o grande número e a diversidade de dados de entrada e de saída, assim como a presença de fatores de imprecisão e de imprevisibilidade (como as pessoas), os autores propõem a divisão do problema em múltiplos comportamentos. Cada um responde a uma situação específica. No sistema proposto, duas variáveis são controladas: a temperatura de aquecimento e o nível de iluminância. Para controlá-las quatro tipos de comportamentos foram propostos: segurança (garante que as condições do ambiente sempre ficarão num patamar seguro), emergência (no caso de alarmes de emergência, como incêndio, as portas

são abertas, e a iluminação e o aquecimento são desligados), economia (garante que energia não será desperdiçada quando o ambiente estiver desocupado) e conforto (conjunto de comportamentos que adaptam o ambiente às preferências de cada usuário). Cada comportamento é um controlador independente para temperatura e para iluminância. De acordo com o contexto, um conjunto de pesos atrelados a cada comportamento é determinado dinamicamente. O comportamento geral do sistema é resultado da resposta de cada comportamento ponderado por seus pesos.

Kulkarni (2002) propõe um sistema baseado em comportamentos reativos para automação de ambientes inteligentes. Conjuntos de reações são agrupadas em comportamentos que correspondem à determinada atividade, como uma reunião, uma apresentação de filme ou a ausência de atividade em uma sala vazia. Um comportamento é ativado quando é detectada a ocorrência da atividade relacionada a ele. Existe um grau de hierarquia entre os comportamentos. Assim quando mais de um comportamento é ativado ao mesmo tempo, o de maior peso na hierarquia suprime ações conflitantes nos comportamentos de menor peso. Um mecanismo de dependência também permite que certos comportamentos sejam atrelados a outros comportamentos hierarquicamente inferiores, e só possam se ativar após a ativação desses últimos.

2.1.2.Redes Neurais Artificiais

Uma rede neural artificial é um sistema paralelo distribuído composto por neurônios artificiais (unidades de processamento simples) dispostos em camadas e interligados por conexões. Na maioria dos casos, essas conexões são associadas a pesos que armazenam o conhecimento adquirido pela rede neural através de um processo de aprendizagem. As capacidades de aprender e generalizar são os prin-

cipais atrativos desses sistemas (BRAGA; CARVALHO; LUDERMIR, 2007; HAYKIN, 1999).

O primeiro modelo artificial de um neurônio foi fruto dos trabalhos de McCulloch e Pitts (BRAGA; CARVALHO; LUDERMIR, 2007). Foi uma simplificação do conhecimento da época sobre o neurônio biológico. Em conjunto, esses neurônios possuem alto poder computacional e podem resolver problemas de elevada complexidade. Matematicamente, o neurônio de McCulloch e Pitts é uma entidade com *m* entradas (x_1,...,x_m) e uma saída y_k. Pesos (w_{k1},...,w_{km}) são acoplados às conexões de entrada e ponderam os valores das mesmas. A saída do neurônio, y_k, é resultado da aplicação de uma função de ativação, *f(.)*, sobre a soma ponderada das entradas pelos seus pesos. A Figura 2 ilustra o modelo de neurônio artificial de McCulloch e Pitts (BRAGA; CARVALHO; LUDERMIR, 2007).

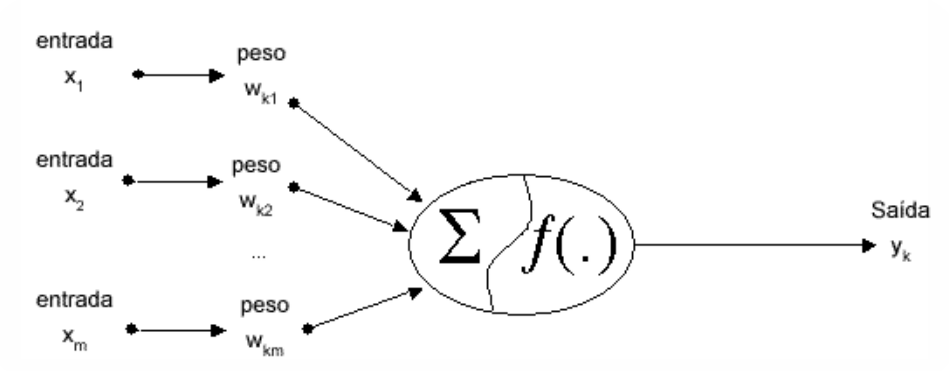

Figura 2 - Modelo de neurônio artificial de McCulloch e Pitts. Adaptado de Braga, Carvalho e Ludermir (2007).

Mais tarde, Frank Rosenblat (BRAGA; CARVALHO; LUDERMIR, 2007) apresentou o *perceptron*, um modelo de rede neural com pesos ajustáveis que através de um algoritmo de aprendizado poderia ser treinado para classificar certos tipos de padrões. O aprendizado é uma das características mais importantes das redes neurais artificiais. Segundo Braga, Carvalho e Ludermir (2007), aprendizado "é o processo pelo qual parâmetros livres de uma rede neural são ajustados por meio de

uma forma continuada de estímulo pelo ambiente externo, sendo o tipo específico de aprendizado definido pela maneira particular como ocorrem os ajustes dos parâmetros livres". Neurônios individuais possuem capacidade computacional limitada. **O poder de uma rede neural artificial está na capacidade de associação e de conexão dos neurônios.**

As aplicações das redes neurais artificiais em problemas de automação do ambiente construído concentram-se em fazer predições. Mozer (1998) propôs um sistema neural para predizer o comportamento e a vontade de habitantes em uma casa, *The Neural Network House*. Uma rede neural com três camadas, 107 entradas, 50 neurônios intermediários e 8 saídas, foi utilizada para predizer a ocupação de (oito) zonas da casa com dois segundos de antecedência. As entradas são alimentadas com sinais de *reed switches*, sensores de movimento e de nível sonoro, e a data. As predições de ocupação das oito zonas da casa são utilizadas para controlar bancos de lâmpadas, aquecimento (ar/água), ventiladores e alto-falantes. Rivera-illingworth, Callaghan e Hagras (2005) utilizaram uma rede neural adaptativa para reconhecer atividades, como dormir, trabalhar no computador ou comer. Guillemin e Morel (2001) usaram uma rede neural artificial para prever radiação solar com seis horas de antecedência em escritórios.

2.1.3.Lógica Difusa

A Lógica Difusa ou Lógica Nebulosa foi inicialmente descrita por Lofti A. Zadeh em um artigo intitulado "Fuzzy Sets" (ZADEH, 1965). O autor propôs uma teoria para o tratamento de conjuntos chamados difusos (*fuzzy*). Os conjuntos difusos ou classes podem conter elementos com graus de pertinência (*membership*) variando de 0 a 1, em contraposição aos conjuntos da teoria clássica de conjuntos que possuem

grau de pertinência 0 ou 1. A possibilidade de trabalhar com graus de pertinência contínuos de 0 a 1 permite modelar aspectos encontrados comumente no mundo real e de difícil modelagem na teoria de conjuntos tradicional. Como exemplo, em uma classe copo cheio, um copo com água até a metade pode facilmente ser modelado como cheio com pertinência de 50% (0,5), e como vazio com pertinência 50% (0,5) (OLIVEIRA JÚNIOR, 1999). A Lógica Difusa oferece um arcabouço para representar conhecimento impreciso e incerto, o que permite lidar com informações vagas e incompletas (HAGRAS *et al.*, 2003), como conforto (DOUNIS *et al.*, 1995; KOLOKOTSA *et al.*, 2001), típicas em aplicações de Automação do Ambiente Construído.

Em aplicações de engenharia de controle, os controladores lógicos difusos (FLC – *Fuzzy Logic Controller*) possibilitam o controle de processos complexos utilizando a experiência humana (ZIMMERMANN, 2001). A idéia básica é incorporar a experiência humana no projeto do controlador. De um conjunto de regras lingüísticas que descrevem a estratégia de controle do operador humano, um algoritmo de controle é construído utilizando palavras (classes) que são definidas como conjuntos difusos (ZIMMERMANN, 2001).

A Figura 3 ilustra um controlador lógico difuso com duas entradas (x1 e x2) e uma saída (s). Duas variáveis reais de entrada (x1 e x2) são relacionadas a quatro classes ou conjuntos difusos ou variáveis lingüísticas (A e C; B e D) através de funções de pertinência (no exemplo, com formato triangulares) – etapa chamada de *fuzzificação*. De acordo com o valor da variável de entrada e a função de pertinência, um grau de pertinência é atribuído. Comumente as funções de pertinência têm formatos triangular, trapezoidal, gaussiana e sigmoidal (OLIVEIRA JÚNIOR, 1999).

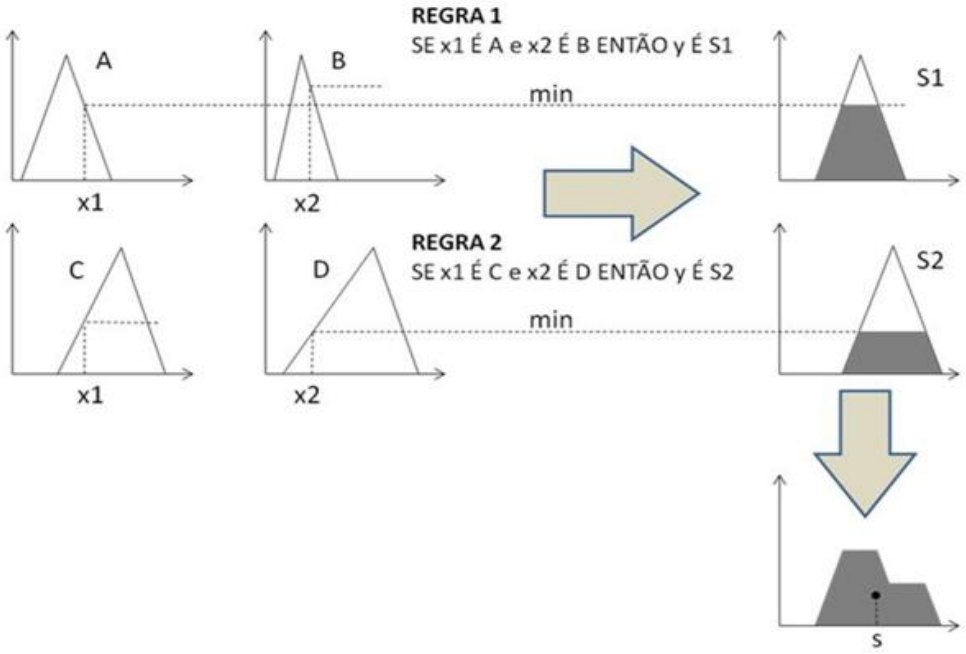

Figura 3 – Exemplo de um controlador difuso com duas variáveis reais de entrada (x1 e x2) e uma variável de saída (s). Há duas variáveis lingüísticas (classes) relacionadas a x1 (A e C) e duas relacionadas x2 (B e D) - *fuzzificação*. A base de regras contém duas regras. Há duas variáveis lingüísticas (classes) relacionadas à saída do sistema (S1 e S2) que são combinadas pelo método do centro de gravidade (COG) e resultam na variável real *s* – *defuzzificação*. Adaptado de Oliveira Junior (1999).

Uma base de regras com duas regras relaciona as classes A e B, e C e D, utilizando o operador lógico E (mínimo), o que resultará em um valor de pertinência para os conjuntos difusos de saída (S1 e S2) – etapa chamada de *inferência difusa*. Os operadores lógicos E e OU são utilizados para combinar conjuntos difusos. O operador E, de interseção, retorna o valor mínimo entre seus operandos. O operador OU, de união, retorna o valor máximo entre seus operandos. Cada regra é formada por uma estrutura condicional do tipo SE...ENTÃO que estabelece a dependência entre uma preposição e uma conseqüência. Assim se uma preposição é avaliada com grau de pertinência de 0,6 (60% verdadeira), a conseqüência terá grau de pertinência 0,6 (60%).

Controladores que relacionam uma preposição com uma conseqüência representada por **conjuntos difusos** são conhecidos como controladores do tipo Mam-

dani. Quando as conseqüências são representadas por **funções reais** que relacionam a saída diretamente com valores reais das entradas, tem-se controladores do tipo Sugeno.

A última etapa do processo é transformar o resultado da inferência difusa em valores reais, etapa conhecida como *defuzzificação*. Alguns métodos são comumente utilizados neste momento: centro de gravidade (COG – *Center Of Gravity*), média dos máximos (MOM – *Mean Of Maxima*), centro de área (COA – *Center Of Area*), esquerda do máximo (LOM – *Left Of Maximum*), direita do máximo (ROM – *Right Of Maximum*) e centro do máximo (COM – *Center Of Maximum*). Todos esses métodos buscam encontrar um valor real a partir do resultado da inferência difusa, isto é, são métodos de ponderação de regras difusa. No exemplo, foi utilizado o método centro de gravidade (COG).

Vários trabalhos utilizam controladores lógicos difusos em sistemas de automação do ambiente construído. Dounis, Lefas e Argiriou, (1995) apontam como dificuldade levantar modelos específicos para cada edificação onde se instalará sistemas de condicionamento de ar. Os autores destacam que controladores lógicos difusos se adéquam naturalmente a esse tipo de problema. Eles propõem um controlador lógico difuso do tipo Mamdani para controlar o ângulo de abertura da janela (AW), a potência de aquecimento (AH) e de refrigeração (AC) de uma sala. O controlador tem como entradas a temperatura ambiente (T_{amb}) e um índice de conforto térmico (PMV – *Predictive Mean Vote*). Variáveis lingüísticas são utilizadas para modelar as variáveis reais do sistema. A base de regras possui 23 regras. Para *defuzzificação* são utilizados dois métodos: média dos máximos (MOM) e centro de área (COA).

Os controladores convencionais, como PID e ON-OFF, apresentam respostas ineficientes a distúrbios e a modificações no ambiente construído, e dificuldade na determinação de modelos matemáticos para estes ambientes, o que torna os controladores lógicos difusos propícios para aplicações deste tipo (LAH *et al.*, 2006; KOLOKOTSA *et al.*, 2001; KOLOKOTSA *et al.*, 2006). Lah *et al.* (2006) apresentam um controlador lógico difuso do tipo Sugeno com duas entradas (*setpoint* de iluminância e erro de iluminância) e uma saída (ângulo de abertura da persiana), e funções de pertinência dos tipos triangular e trapezoidal, com objetivo de controlar o ângulo de abertura de uma persiana para aproveitamento de luz natural. Kolokotsa *et al.* (2006) propõem um controlador lógico difuso do tipo Mamdani visando garantir o conforto dos usuários. São medidos o conforto térmico (PMV), a qualidade do ar (ppm de CO_2 no ar) e o conforto visual (iluminância), e controlados a potência de aquecimento/refrigeração, o ângulo de abertura de uma janela, a abertura de um dispositivo de sombreamento e a potência da iluminação. Foram utilizadas funções de pertinência dos tipos triangular e trapezoidal, um controlador para cada variável controlada e o método do centro de gravidade (COG) para *defuzzificação*.

Guillemin (2003) utilizou controladores lógicos difusos em um sistema integrado para controle do aquecimento, da iluminação e da abertura de um dispositivo de sombreamento. O autor utilizou um controlador do tipo Mamdani com um conjunto de nove regras e as seguintes entradas: iluminância horizontal externa, estação do ano, altitude solar e azimute solar, para o controle do aquecimento. Para controlar a porcentagem de abertura do dispositivo de sombreamento foi utilizado um controlador Sugeno com um conjunto de 25 regras e as seguintes entradas: conforto atual e conforto previsto para as próximas seis horas, onde o conforto está relacionado a diferença entre o *setpoint* de temperatura e a temperatura interna.

Outros trabalhos também utilizam controladores lógicos difusos para automação de sistemas no ambiente construído. Dounis e Caraiscos (2007) desenvolveram um sistema difuso para coordenar diferentes controladores difusos visando conforto e eficiência energética. Hagras *et al.* (2003) criaram um sistema reativo com comportamentos difusos e um coordenador difuso de comportamentos. Kristl *et al.*(2008) propuseram um sistema de controle difuso para um dispositivo de sombreamento visando conforto térmico e lumínico, e eficiência energética. Alcalá *et al.* (2005) utilizaram um controlador difuso para sistemas de condicionamento de ar.

2.2. Tecnologias para Sistemas de Automação do Ambiente Construído

Esta seção aborda tecnologias de redes de comunicação e de *middleware* utilizadas aplicações de automação do ambiente construído. Essas tecnologias são fundamentais para a construção de sistemas de automação do ambiente construído, e servirão de base para a implementação da arquitetura proposta no capítulo 5.

2.2.1.Redes de Comunicação

As redes de comunicação são sistemas de comunicação formados por nós com capacidade de transmitir e receber informações (COOK; DAS, 2005). Elas desempenham um papel de destaque na automação do ambiente construído, permitindo que dispositivos como sensores, atuadores, controladores e computadores possam se comunicar. Devido a sua usual complexidade, o projeto, a construção e a descrição de uma rede de comunicação são feitos utilizando como referência uma estrutura de camadas padronizada pela ISO (*International Organization for Standardization*) – o modelo de referência OSI (*Open Systems Interconnection*) – que possui sete

camadas descritas a seguir (ZHENG; AKHTAR, 2002; SOARES; LEMOS; COLCHER, 1995):

- **Física.** Responsável pelo formato do canal e sinal de comunicação sem a preocupação com significado ou com a forma de agrupamento dos dados. Incluem-se características mecânicas, elétricas, funcionais e de procedimentos de transmissão de dados pelo meio físico.

- **Enlace.** Responsável pelo acesso e controle do canal de comunicação. Incluem-se protocolos de acesso ao meio, de detecção de erros, de delimitação de quadros, etc..

- **Rede.** Responsável pelo formato individual dos pacotes de dados. Controla o chaveamento e estabelece a rota para conexão e a troca de informação entre nós.

- **Transporte.** Responsável pela entrega de seqüências de pacotes.

- **Sessão.** Responsável pela organização, sincronia e gerência de conexão entre programas.

- **Apresentação.** Responsável por conversões para representação de dados. Incluem-se a compressão de texto, a criptografia, etc..

- **Aplicação.** Responsável por detalhes e características específicas das informações trocadas entre os aplicativos. É a interface entre a rede comunicação e o aplicativo.

Neste modelo, “cada nível (camada) deve ser pensado como um programa ou processo, implementado em *hardware* ou *software*, que se comunica com o proces-

so correspondente em outro nó. As regras que governam a conversação de um nível N qualquer são chamadas de protocolo de comunicação de nível N" (SOARES; LEMOS; COLCHER, 1995). A Figura 4 ilustra o modelo de referência OSI/ISO.

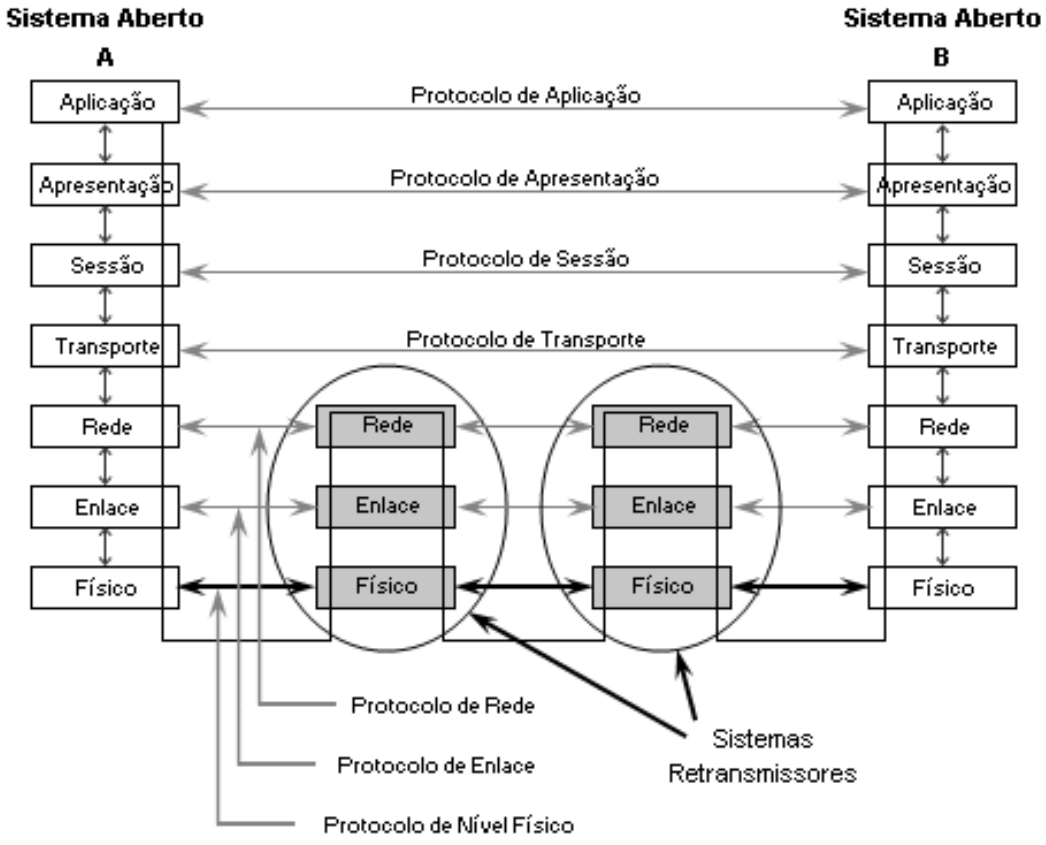

Figura 4 - Modelo de referência OSI/ISO. Fonte: Soares, Lemos e Colcher (1995).

O tamanho (distância de abrangência) da rede de comunicação repercute, em geral, nas tecnologias utilizadas (PETERSON; DAVIE, 2003). Segundo o tamanho, as redes podem ser classificadas em:

- **WAN (*Wide Area Networks*).** Redes de grande abrangência, até mesmo mundial. Cabos, rádio, microondas e satélites são meios físicos utilizados para sistemas de comunicação tipicamente ponto-a-ponto[1] (ZHENG; AKHTAR, 2002).

[1] Transmissão feita através de conexões estabelecidas entre pares de nós.

- **MAN (*Metropolitan Area Networks*).** Interligam dispositivos com abrangência metropolitana. As conexões são tanto ponto-a-ponto como *broadcasting*[2] ou *multcasting*[3] (ZHENG; AKHTAR, 2002).

- **LAN (*Local Area Networks*).** Englobam distâncias que não costumam ultrapassar as dimensões do prédio onde a rede é instalada. As conexões são quase sempre *multicasting* ou *broadcasting* (ZHENG; AKHTAR, 2002).

- **PAN (*Personal Area Networks*)** Terminologia nova para redes de curta distância usadas para interligar dispositivos como PDAs, laptops, computadores, periféricos e sensores dentro de ambientes. Costumam se restringir a um ambiente como uma sala. Foi popularizada com tecnologia sem fio Bluetooth (COOK; DAS, 2005).

- **SAN (*System Area Networks*)**. São redes, usualmente, confinadas em uma única sala e usadas para conectar dispositivos de grandes sistemas computacionais como processadores paralelos e servidores de dados. Por essa última aplicação, essas redes também são referidas como *Storage Area Networks* (PETERSON; DAVIE, 2003).

- **BAN (*Body Area Networks*).** Redes de comunicação utilizadas para conectar dispositivos atrelados ao corpo humano. Relacionam-se com as tecnologias *wearable devices*, nas quais dispositivos computacionais são embarcados na vestimenta e comunicam-se entre si (COOK; DAS, 2005).

A Figura 5 mostra a distância de abrangência típica para cada tipo de rede descrita acima, de acordo com Cook e Das (2005).

[2] Transmissão feita de um nó para vários nós que compartilham o mesmo canal de comunicação.
[3] Transmissão feita de um nó para todos os nós que compartilham o mesmo canal de comunicação.

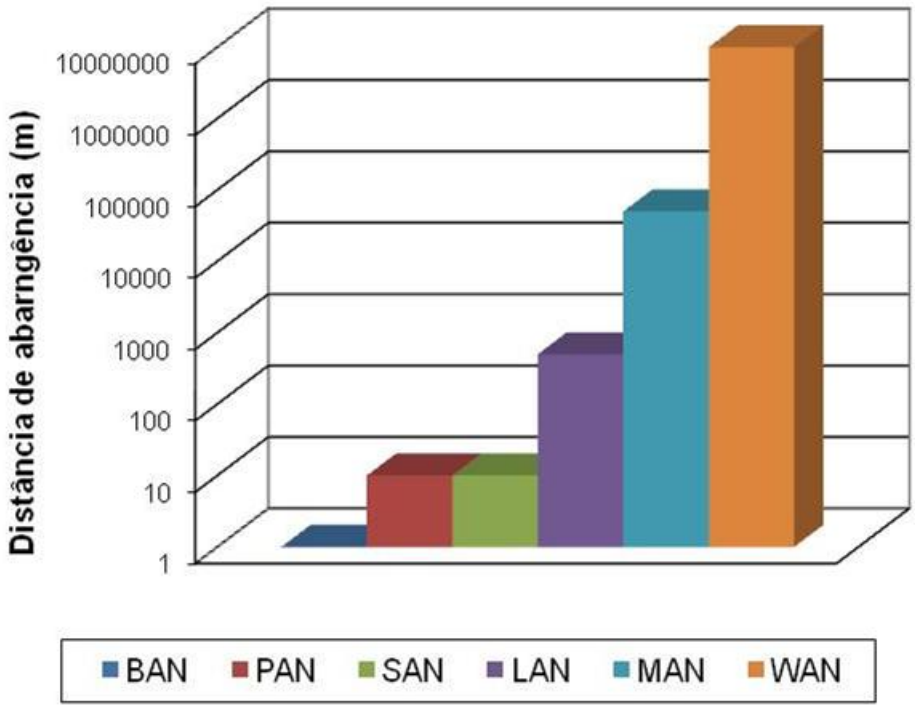

Figura 5 - Classificação das redes pela distância de abrangência. Adaptado de Cook e Das (2005).

A topologia da rede de comunicação, isto é, a maneira como os nós da rede se conectam fisicamente uns aos outros, é um fator importante na escolha da rede para determinada aplicação. A topologia da rede influência na estrutura física de montagem da rede, na maneira como as mensagens são transmitidas, na robustez do canal de comunicação, entre outros. São topologias usuais (Figura 6): estrela (todos os nós se conectam a um nó central), anel (cada nó se interliga a dois vizinhos, formado ao final um anel), em barra (todos os nós se interligam a um canal de comunicação), em árvore (os nós são conectados de forma hierárquica, contendo pais e filhos, como numa árvore), totalmente conectada (todos os nós estão interligados dois a dois) e malha (os nós são interligados em forma de matriz).

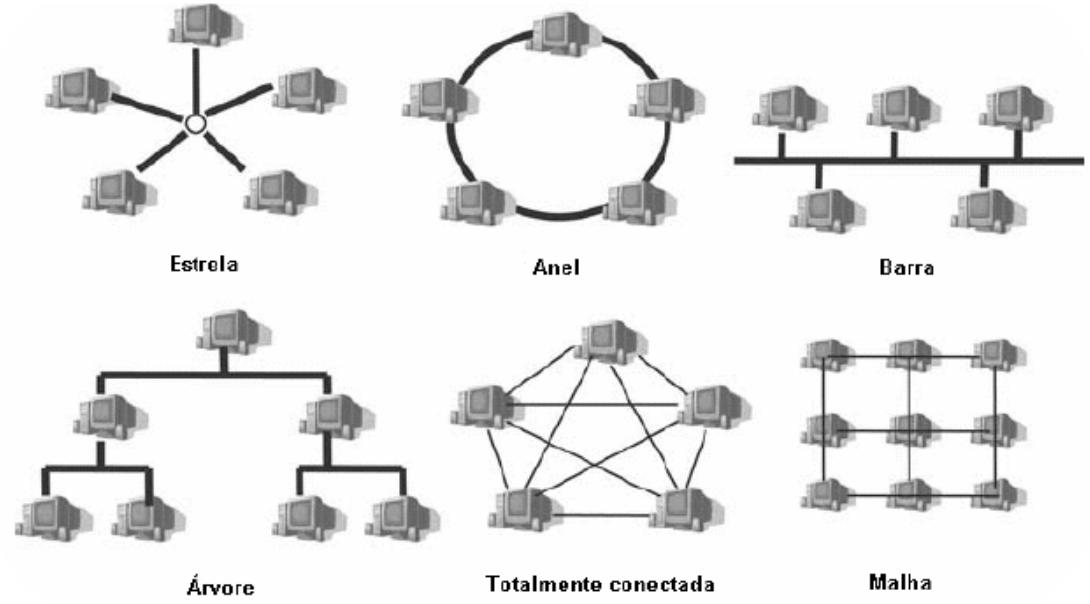

Figura 6 - Topologias usuais de redes de comunicação. Adaptado de Cook e Das (2005).

No contexto da automação do ambiente construído, e de acordo com o objetivo da rede de comunicação, distinguem-se três grupos de redes de comunicação: de computadores, de campo e de sensores. É importante ressaltar que cada grupo é na verdade uma especialização de um grupo mais abrangente. Assim uma rede de sensores é uma especialização de uma rede de campo que é uma especialização de uma rede de computadores que é uma especialização de uma rede de comunicação.

2.2.1.1. Redes de Computadores

Uma Rede de Computadores é uma interconexão de dispositivos (*hardware*) programáveis de propósito geral (não são otimizadas para uma aplicação particular) com *software* relacionado que executa funções de comunicação para diferentes tipos de dados (ZHENG; AKHTAR, 2002; PETERSON; DAVIE, 2003). Entre os protocolos de comunicação existentes para redes de computadores, dois se destacam por serem amplamente utilizados: Ethernet e Internet (TCP/IP).

2.2.1.1.1. Ethernet

A Ethernet é considerada a tecnologia de rede local (LAN) mais amigável e com preço mais acessível disponível atualmente (ZHENG; AKHTAR, 2002). É um subconjunto do padrão IEEE 802.3 que define serviços para as camadas física e de enlace para redes de 10 Mbps (SOARES; LEMOS; COLCHER, 1995). A estrutura física de uma rede Ethernet é composta por cabos, transceptores e adaptadores. O cabeamento mais comumente utilizado é o 10BaseT (par-trançado categoria 5 com segmento máximo de 100 metros). Múltiplos segmentos 10BaseT podem ser interligados por *hubs* (ver Figura 7) (PETERSON; DAVIE, 2003; SOARES; LEMOS; COLCHER, 1995).

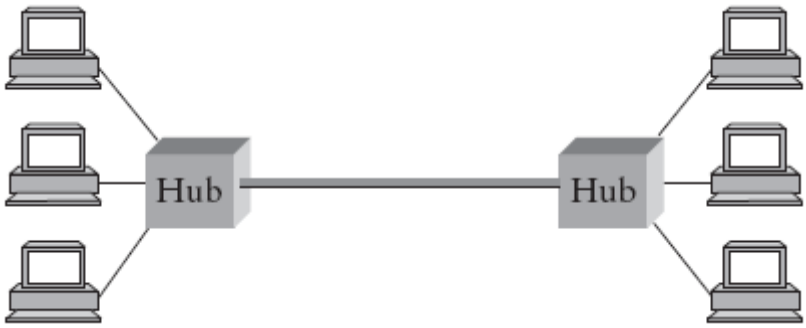

Figura 7 – Rede Ethernet 10BaseT. Fonte: Peterson e Davie (2003).

O protocolo de acesso ao meio nas redes Ethernet é da família CSMA/CD (*Carrier-Sense Multiple Access with Collision Detection*). Cada estação monitora o canal de comunicação e transmite mensagens somente quando não há transmissão pelo canal. Caso duas estações comecem a transmitir ao mesmo tempo, a colisão é detectada e a transmissão é abortada, as estações tentarão retransmitir após um tempo aleatório. Os pacotes Ethernet podem transmitir até 1500 bytes de dados e utilizam um endereçamento único de seis bytes (PETERSON; DAVIE, 2003; SOARES; LEMOS; COLCHER, 1995).

2.2.1.1.2. Internet (TCP/IP)

Os protocolos TCP (*Transmission Control Protocol*) e IP (*Internet Protocol*) permitem a interligação de diferentes tecnologias de rede. As diferentes redes são interligadas através de roteadores (ver Figura 8). “A idéia baseia-se na seguinte constatação: não existe nenhuma tecnologia de rede que atenda aos anseios de toda a comunidade de usuários” (SOARES; LEMOS; COLCHER, 1995). A capacidade de interligar diferentes redes de computadores tornou o TCP e o IP blocos primários da Internet (ZHENG; AKHTAR, 2002).

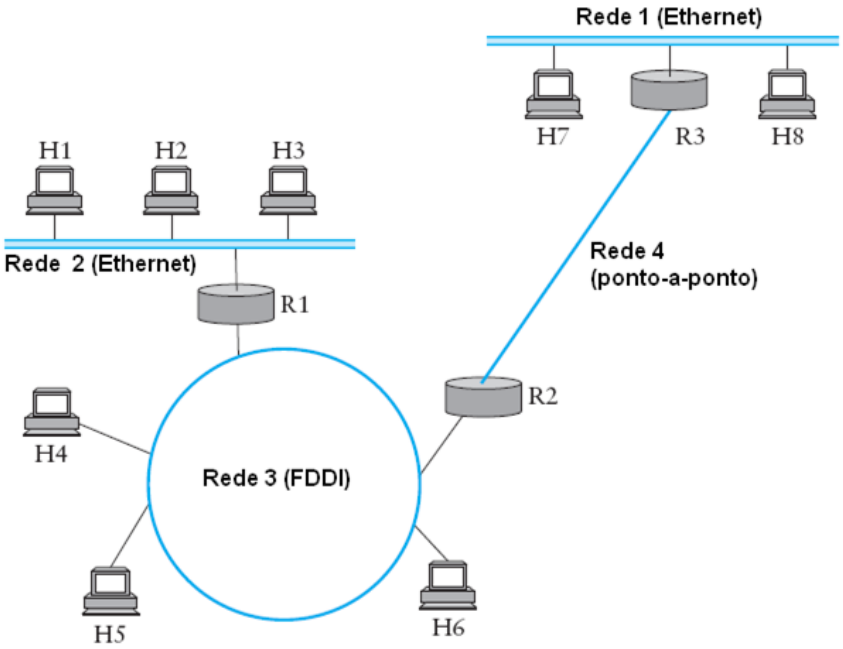

Figura 8 - Interligação de diferentes tipos de rede (*internet*).
H*n* = estação; R*n* = roteador. FDDI = *Fiber Distributed Data Distributed.*
Fonte: Peterson e Davie. (2003).

O protocolo IP é um protocolo de rede, sem conexões, com a função de transferir pacotes de dados denominados datagramas de um nó origem para um nó destino. Os nós são identificados por endereços IP. Também é oferecido um serviço de fragmentação e remontagem de datagramas longos. Os endereços IP são números de 32 bits, geralmente, escritos na forma de quatro octetos (em notação decimal), exemplo 128.6.2.1. Uma parte do endereço identifica a rede, e outra parte uma esta-

ção dentro da rede. Há três classes de endereço IP (A, B e C) com quantidades diferentes de bits reservados para o endereço da rede e da estação, como ilustra a Figura 9. Há ainda uma classe D para comunicação *multcasting* e uma classe E reservada para uso futuro. Para comunicação *broadcasting,* todos os bits do endereço devem receber valor 1 (um) (SOARES; LEMOS; COLCHER, 1995; ZHENG; AKHTAR, 2002).

Figura 9 - Endereços IP: (a) classe A; (b) classe B; (c) classe C.
Fonte: Peterson e Davie (2003)

O protocolo TCP é um protocolo de transporte, orientado a conexão que fornece um serviço confiável de transferência de dados fim a fim, utilizando como base o protocolo IP. Utiliza-se o conceito de porta associado a cada processo TCP em uma estação. A concatenação de um endereço IP e uma porta TCP é definida como soquete (*socket*). Uma conexão é identificada pelo par de soquetes de suas extremidades.

2.2.1.2. Redes de Campo

Uma Rede de Campo é um sistema de comunicação em tempo real que conecta dispositivos de campo como sensores, atuadores e controladores (THOMESSE, 1999). Utilizam, em geral, a topologia em barra (ver Figura 10) e distribuem a tarefa

de controle entre vários nós da rede. Por esta razão são também denominadas Redes de Controle (SCHICKHUBER; MCCARTHY, 1997) ou Sistemas de Controle Distribuído (MAHALIK *et al.,* 2006). Como a tarefa de controle é distribuída entre os vários nós cooperantes, a possibilidade de falha geral dos sistemas de controle distribuído se torna mais remota quando comparado aos sistemas de controle centralizado (MAHALIK; LEE, 2003).

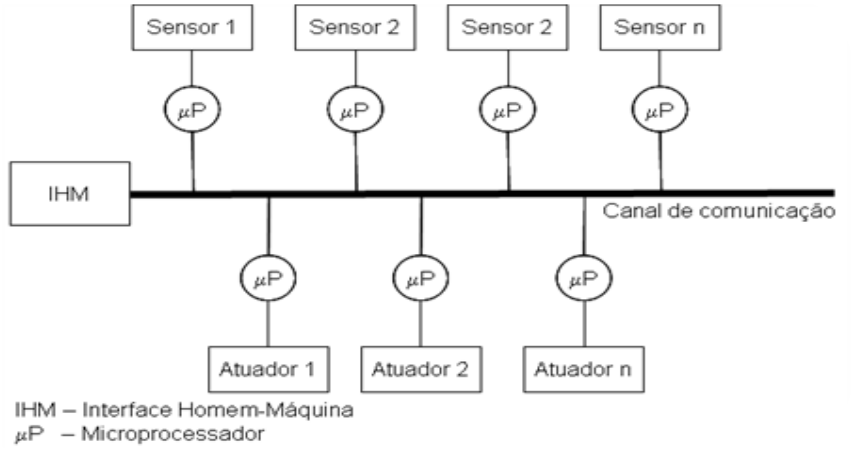

Figura 10 - Rede de campo (rede de controle): topologia em barra. Adaptado de Mahalik e Lee (2003).

Algumas tecnologias de redes de campo são consagradas para aplicações de automação do ambiente construído, entre elas: Lonworks, EIB (*European Instalation Bus*), BACNet[4] (*Building Automation and Control Networks*) e X-10[5].

2.2.1.3. Redes de Sensores

Uma Rede de Sensores é uma rede especializada na aquisição e na distribuição de dados de sensores (com capacidade de processamento e comunicação) monitorada e controlada por um centro de gerenciamento (COOK; DAS, 2005; AKYILDIZ *et al.,* 2002). Algumas redes de sensores comerciais e acadêmicas são utilizadas em problemas na área de automação do ambiente construído. A Figura 11

[4] Bacnet não é propriamente uma rede de campo, mas uma especificação de camada de aplicação para redes de campo.
[5] X-10 é uma tecnologia para comunicação pela rede elétrica da década de 1970. É extremamente limitada em comparação com as outras.

ilustra nós de algumas dessas redes de sensores e seus respectivos desenvolvedores. O Quadro 1 mostra a evolução das redes de sensores.

	Passado (1980)	Presente	Futuro
Tamanho	Maiores que caixa de sapato	Tamanho de um cartão a pequena caixa de sapatos	Partícula de poeira
Peso	Kilogramas	Gramas	Desprezível
Arquitetura dos nós	Sensoriamento, processamento e comunicação separados	Sensoriamento, processamento e comunicação integrados	Sensoriamento, processamento e comunicação integrados
Topologia	Ponto a ponto, estrela	Cliente servidor, ponto a ponto	Ponto a ponto
Duração da alimentação	Grandes baterias, horas, dias	Baterias AA, dias a semanas	Solar, meses a anos
Instalação	Sensores individuais lançadas do ar ou de veículos	Colocação manual	Embutida em objetos, pulverizada

Quadro 1 - Evolução das redes de sensores. Adaptado de Chong e Kumar (2003).

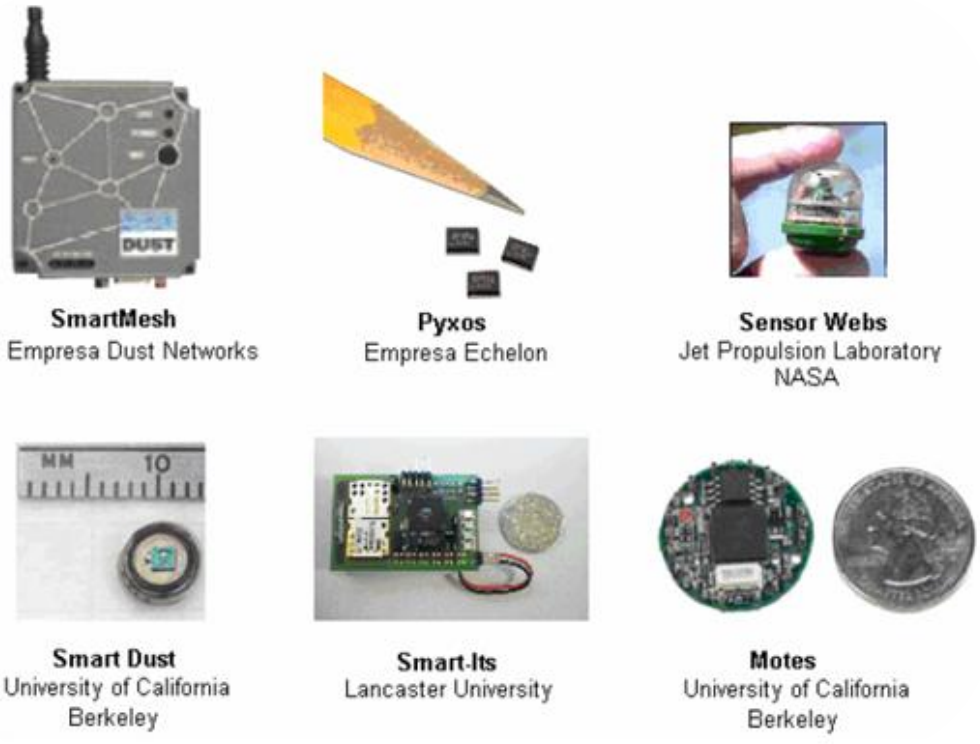

Figura 11 - Exemplos de nós de redes de sensores comerciais e acadêmicas. Fonte: Tibiriçá (2007).

2.2.2. *Middleware*

Middleware é um conceito atrelado à idéia de *software* de conectividade que associa aplicações por meio de mecanismos de comunicação, criando transparência[6], escalabilidade[7] e interoperabilidade[8]. Oferece serviços que tornam transparente a comunicação entre aplicativos, inclusive situados em diferentes computadores. Em geral, é uma camada de *software* situada entre o sistema operacional e os aplicativos que se utilizam do *middleware* (COOK; DAS, 2005). Entre os formatos de *middleware* disponíveis, destacam-se dois por sua ampla utilização: os orientados a objetos e os bancos de dados.

2.2.2.1. Orientação a Objetos

No enfoque orientado a objetos, os átomos do processo de computação são os objetos. Os objetos trocam mensagens entre si ativando os métodos uns dos outros. Cada objeto possui atributos que representam seu estado e são modificados por ações realizadas através dos métodos. Objetos que compartilham a mesma interface (mesmos atributos e métodos) são agrupados em classes. O sistema computacional final é resultado da combinação e interação dos vários objetos que o compõe (COLEMAN *et al.,* 1996). A Figura 12 ilustra um modelo computacional orientado a objetos.

Middleware orientado a objetos ou ORB (*Object-Request Brokers*) estende o paradigma orientado a objetos às tecnologias de *middleware*. Esses sistemas permitem o desenvolvimento de aplicativos orientados a objetos distribuídos, isto é, objetos pertencentes a diferentes processos e em diferentes computadores comunicam-

[6] Qualidade de tornar um sistema computacional mais amigável, poupando o usuário de detalhes técnicos.
[7] Facilidade de expansão.
[8] Capacidade de operar em conjunto.

se de forma transparente (como se pertencessem ao mesmo processo na mesma máquina). O ORB é responsável pela comunicação entre os objetos. São exemplos de ORB: COM/DCOM[9], Java RMI[10] da Sun e CORBA da OMG (COOK; DAS, 2005).

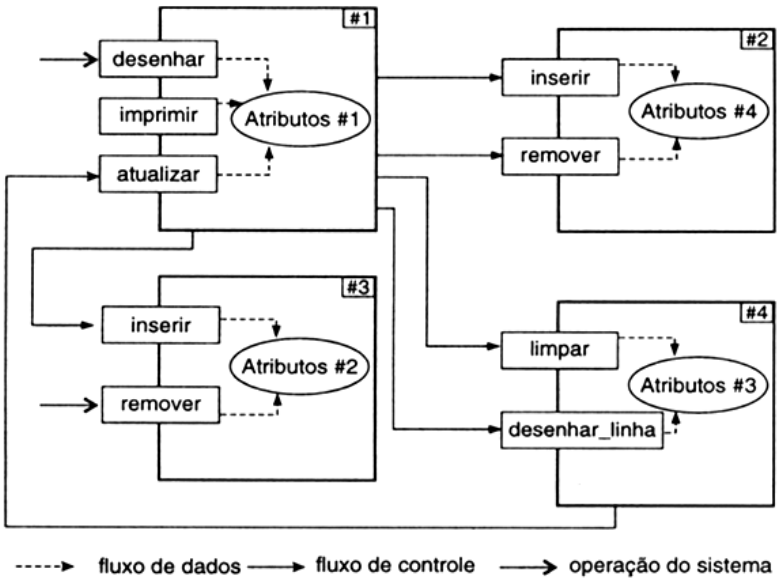

Figura 12 - Exemplo de modelo computacional orientado a objetos.
Fonte: Coleman *et al.* (1996).

2.2.2.1.1. CORBA

CORBA (*Common Object Request Broker Architecture*) é um *middleware* padronizado pelo OMG (*Object Management Group*) com o objetivo produzir uma infraestrutura completa para computação distribuída, reunindo duas tecnologias: a orientação a objetos (OO) e a chamada de procedimento remota[11] (RPC – *Remote Procedure Call*). Essa associação permite que objetos distribuídos em diferentes computadores possam se comunicar de forma transparente, como se estivessem na mesma máquina. Operações remotas são agrupadas em interfaces cujas instâncias

[9] DCOM (*Distributed Component Object Model*) é uma tecnologia de *middleware* da Microsoft para comunicação entre objetos em sistemas distribuídos, por meio de redes de computadores de abrangência local até mundial. É uma extensão do COM (*Component Object Model*), tecnologia que permite que componentes de software (objetos) se comuniquem (COOK; DAS, 2005).
[10] Java RMI (*Remote Method Invocation*) é um *middleware* que permite a comunicação entre objetos programados na linguagem Java em diferentes computadores com máquinas virtuais Java (COOK; DAS, 2005).
[11] Tecnologia que permite a chamada de funções remotas de forma transparente, como se a chamada fosse local.

são os objetos CORBA (BOLTON, 2002; BROSE; VOGEL; DUDDY, 2001). A Figura 13 ilustra a invocação de operações em objetos CORBA.

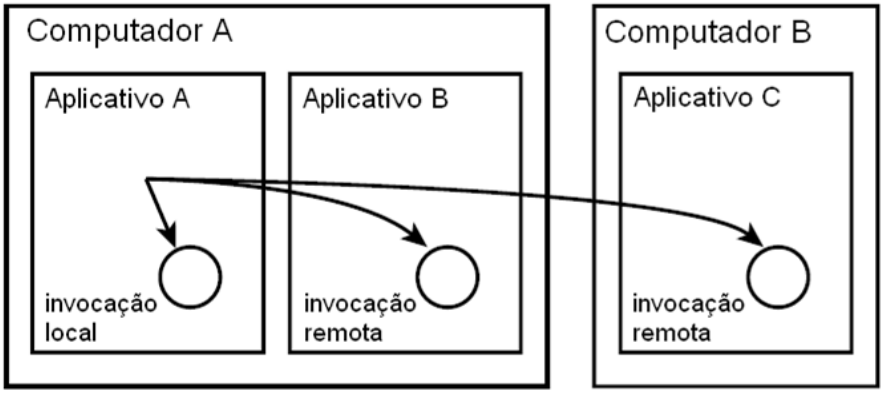

Figura 13 – Invocação de operações em objetos CORBA. Adaptado de Bolton (2002).

Há uma separação clara entre a implementação dos objetos CORBA e suas interfaces que são definidas através de uma Linguagem para Definição de Interface (*IDL – Interface Definition Language*). Um compilador IDL mapeia a interface IDL para uma linguagem de programação como Java, C, C++, COBOL, Ada, Smaltak ou Lisp, na qual a implementação será escrita (BOLTON, 2002).

Toda comunicação entre objetos CORBA transcorre mediante um ORB que age como um canal de comunicação entre objetos que podem estar locados em qualquer computador de uma rede, ser implementados em diferentes linguagens de programação e ser executados em diferentes plataformas de sistema operacional e de *hardware*. Vários fabricantes disponibilizam ORBs para CORBA. Um protocolo, IIOP (*Internet Inter-ORB Protocol*), define a comunicação entre diferentes ORBs sobre a camada de transporte TCP/IP (BROSE; VOGEL; DUDDY, 2001).

2.2.2.2. Banco de Dados

Bancos de dados são coleções de dados relacionados. Permitem o armazenamento e a consulta de informações através de uma linguagem padrão, SQL (*Structured Query Language*). Interfaces padrões (API – *Application Programming Interface*),

como ODBC (*Open DataBase Connectivity*) e JDBC (*Java DataBase Connectivity*), permitem a conexão entre aplicativos e bancos de dados, tornando os bancos de dados canais de comunicação entre diversos aplicativos (COOK; DAS, 2005; ELMASRI; NAVATHE, 2005).

Um sistema de gerenciamento de banco de dados (SGBD) é um conjunto de programas que permitem aos usuários criar e manter banco de dados. Entre os SGBD, o MySQL se destaca por ser um software de licença livre, portável e compatível com diversas linguagens de programação que utiliza a linguagem SQL como interface. Sua interface permite manipular dados através de funções como de recuperação de dados específicos, criação e atualização de tabelas, entre outras. O compartilhamento permite aos múltiplos usuários e programas acessar, de forma concorrente, o banco de dados (ELMASRI; NAVATHE, 2005).

3. Automação do Ambiente Construído

Este capítulo descreve, através de três estudos bibliográficos, exemplos de arquiteturas de *software* relacionadas a sistemas de Automação do Ambiente Construído. Estes três projetos abrangem o tema de forma geral, sistêmica e integrada, pois relatam aspectos de comunicação, de controle e de *software*. Os dois primeiros estão relacionados a grupos de pesquisa em Computação Pervasiva e tem como pontos fortes o reconhecimento de contexto e a arquitetura de software. O último está relacionado a um grupo de pesquisa em Automação do Ambiente Construído, resume como o assunto Automação do Ambiente Construído tem sido abordado pelos pesquisadores da área e como esses sistemas têm sido construídos. Em cada um dos trabalhos, cinco aspectos de cada projeto são abordados:

- **Reconhecimento de Contexto**. Um ponto importante na Automação do Ambiente Construído é a capacidade do sistema em reconhecer contextos. Este item aborda algoritmos para reconhecimento de contexto e contextos reconhecidos em cada projeto.

- **Ações de Controle**. Os sistemas de automação têm como base malhas de controle que objetivam fazer com que o ponto de operação de um sistema fique num patamar determinado (*setpoint*). Este item descreve as malhas de controle de cada projeto.

- **Arquitetura Física**. Um sistema de Automação do Ambiente Construído possui elementos físicos como sensores, atuadores, computadores, redes de comunicação, entre outros. Este item descreve quais elementos físicos compõem cada projeto e como se interligam fisicamente.

- **Arquitetura Lógica**. Um mesmo problema pode ser resolvido de diversas formas, isto é, usando diferentes formas de organização lógica. Este item descreve a concepção lógica de cada projeto.

- **Arquitetura de *Software***. A partir da concepção lógica, um problema computacional precisa ser programado. Este item aborda a estrutura de *software* utilizada.

3.1. MavHome

Mavhome (*Managing and Adaptive Versatile Home*) é uma casa utilizada para estudar as características de Ambientes Inteligentes criados na Universidade do Texas em Arlington. Estes ambientes são definidos como aqueles capazes de adquirir e aplicar conhecimento sobre o ambiente e seus habitantes, visando a melhorar suas experiências no ambiente. A operação do ambiente é feita através de um ciclo onde se percebe o estado do ambiente, infere-se a respeito do estado, metas e resultados de possíveis ações, e age-se sobre o ambiente mudando seu estado (COOK; DAS, 2007). O objetivo é criar uma casa que atue como um agente racional com percepção através de sensores e ações através de atuadores. Procura-se maximizar o conforto e a produtividade dos habitantes, e minimizar custos de operação (DAS *et al.,* 2002). A planta baixa da casa pode ser vista na Figura 14.

A predição de mobilidade e de atividades são características apontadas como fundamentais na MavHome. O roteamento de mensagens e de informação multimídia e a automatização de ações antes efetuadas de forma manual são feitos a partir das predições. Técnicas de aprendizagem de máquina são propostas para predizer padrões de movimentação, tarefas e interações típicas com os ambientes. Essas informações são posteriormente utilizadas na tomada de decisão (DAS *et al.;* 2002).

Busca-se desenvolver e integrar componentes que tornarão possível a inteligência dos ambientes no futuro (YOUNGBLOOD; HOLDER; COOK, 2005).

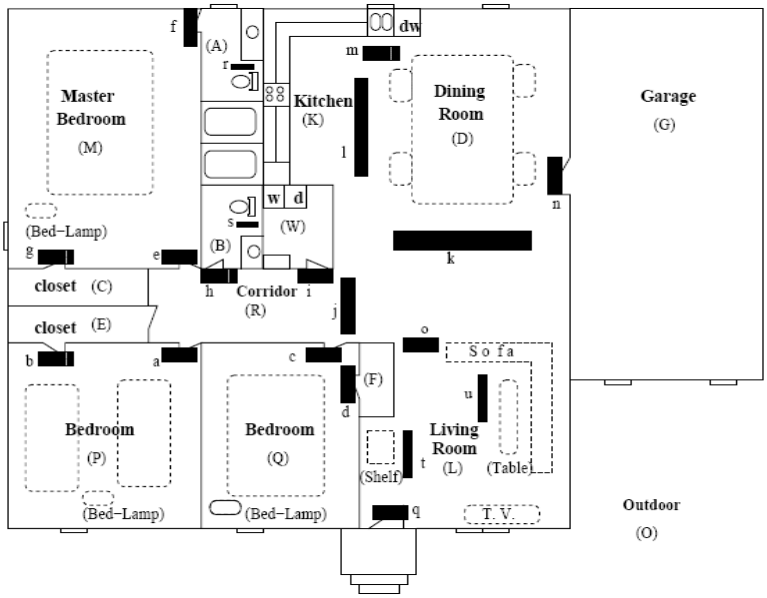

Figura 14 – Planta baixa da Mavhome. Fonte: Roy *et al.* (2003).

3.1.1.Reconhecimento de Contexto

Localização de pessoas e reconhecimento de atividades fazem parte do sistema de reconhecimento de contexto da Mavhome. A movimentação do habitante é considerada meramente um reflexo de padrões de ações cotidianas do usuário que podem ser aprendidos. Divide-se a casa em zonas ou setores, que são representados por caracteres distintos (ver Figura 15). A partir do monitoramento de movimentação do usuário, modelos de mobilidade são obtidos. Esses modelos são trajetórias do usuário através dos cômodos, representadas por cadeias de caracteres (*strings*). Utiliza-se o algoritmo LeZi-Update, baseado no algoritmo de compressão de dados LZ78 para codificação dos dados em um dicionário. De acordo com a freqüência de cada “palavra” no dicionário, é feita a predição de movimentos. É importante ressaltar que o sistema é proposto somente para o acompanhamento de um usuário por vez (COOK; DAS, 2007; DAS *et al.,* 2002; ROY *et al.;* 2003).

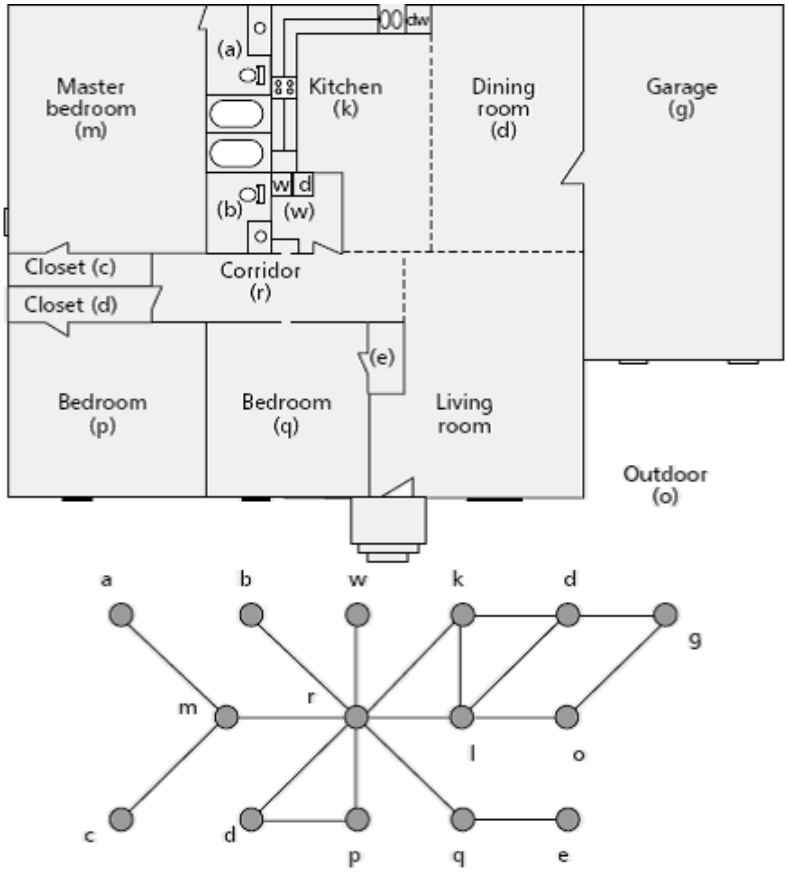

Figura 15 – Planta da MavHome e modelo gráfico da disposição dos setores. Fonte: Das *et al.* (2002).

O reconhecimento de atividades utiliza o histórico de eventos dos usuários para predição de atividades e diferentes algoritmos para análise dos dados. O primeiro algoritmo, SHIP (*Smart Home Inhabitant Prediction*), combina a seqüência mais recente de eventos com seqüências históricas. Exemplos de eventos incluem ligar luzes, tocar música, ligar o irrigador, etc. Para fazer predições, o SHIP procura a seqüência história mais adequada à última ação do usuário. É considerado o tamanho das seqüências que terminam com a mesma ação e suas freqüências históricas. Podem-se aplicar fatores de decaimento para diminuir pesos de seqüências muito antigas, e *thresholds* de inexatidão para a comparação entre seqüências com ligeiras diferenças. Acertos da ordem de 53% em média e de 80% com as melhores três combinações foram obtidos com dados reais pelo SHIP (COOK *et al.,* 2003; DAS *et al.;* 2002).

ALZ (*Active LeZi*) é um algoritmo para predição de atividades na MavHome baseado na técnica de compressão de dados LZ78 e com características funcionais semelhantes ao LeZi-Update descrito anteriormente. As interações usuários-dispositivos são representadas por caracteres e um conjunto de interações é modelado como uma seqüência de caracteres (COOK *et al.,* 2003; YOUNGBLOOD; HOLDER; COOK, 2005).

Um terceiro algoritmo, ED – *Episode Discovery*, baseado em técnicas de *data mining* também é utilizado. Ele objetiva reduzir a quantidade de estados possíveis por meio de um conjunto de episódios centrados em tarefas de habitantes. Procura-se descobrir grupamentos de interações estreitamente relacionadas no tempo, os episódios. A freqüência de ocorrência, o tamanho e a regularidade dos grupamentos de interações são levados em consideração na busca de episódios, sendo estes diretamente relacionados a tarefas dos habitantes. São exemplos de episódios detectados na MavHome: Ligar/Desligar Aquecedor, Ligar/Desligar Despertador, Ligar Luzes do Quarto, Ligar Cafeteira, Ligar Luzes do Banheiro, Ligar Vídeo do Banheiro e Ligar Chuveiro com freqüências diárias; e Ligar Irrigador com freqüência semanal (COOK *et al.,* 2003; DAS *et al.,* 2002; YOUNGBLOOD; HOLDER; COOK, 2005).

3.1.2. Ações de Controle

A MavHome possui interfaces padrões e remotas para que seus usuário possam comandar diretamente os atuadores e sistemas de controle simples como o de um motor de passo para ajustar quatro mini-persianas. Sensores de movimento, umidade, temperatura, fumaça e luz são utilizados (YOUNGBLOOD; HOLDER; COOK, 2005). No entanto, nenhuma ação de controle relacionada diretamente a esses sensores foi descrita nos trabalhos revisados sobre a MavHome. As ações de

controle relacionam-se principalmente às ações de atuação ordenadas em função do reconhecimento de contexto (COOK *et al.,* 2003; COOK; DAS, 2007; DAS *et al.,* 2002; ROY *et al.,*2003; YOUNGBLOOD; HOLDER; COOK, 2005).

3.1.3.Arquitetura Física

Sensores de movimento, umidade, temperatura, fumaça e luz, e controladores de lâmpadas, persianas, aquecedores, cafeteira, irrigadores e chuveiros são os elementos de interface entre o mundo físico e o sistema de automação da Mavhome. Os sensores de movimento destacam-se por sua importância no sistema de rastreamento de pessoas. Utilizam tecnologia de infravermelho passivo e um tubo para sombreamento do campo de visão, que visa a diminuir a área de cobertura de cada sensor.

Uma rede Argus, baseada em microcontroladores PIC16F877 interliga os sensores. Há um nó mestre conectado a um computador através da porta serial. Até cem nós escravos podem ser interligados a um mestre, e até sessenta e quatro sensores podem ser conectados a cada nó escravo. Mestres para aplicações especiais também foram desenvolvidos, como um para o controle de quatro mini-persianas. Dispositivos baseados na tecnologia X-10 são utilizados para acionar lâmpadas. Há também sensores, atuadores e PDAs (usados como interface com os usuários) que utilizam a tecnologia Bluetooth para comunicação. Todo processamento é feito por aplicativos instalados em computadores interligados por uma rede local (ROY *et al.,* 2003; YOUNGBLOOD; HOLDER; COOK, 2005).

3.1.4. Arquitetura Lógica

Os componentes físicos e de *software* são estruturados em quatro camadas abstratas na MavHome (COOK *et al.,* 2003; COOK; DAS, 2007; DAS *et al.,* 2002; YOUNGBLOOD; HOLDER; COOK, 2005), a saber:

- **Física.** Monitora o ambiente através dos sensores e muda o estado do ambiente através dos atuadores (COOK; DAS, 2007). Contém o *hardware* espalhado pela casa, o que inclui sensores, atuadores, computadores e equipamentos de rede (COOK *et al.,* 2003; DAS *et al.,* 2002; YOUNGBLOOD; HOLDER; COOK, 2005).

- **Comunicação.** Facilita a comunicação entre componentes de todas as camadas, a mobilidade de processos e o descobrimento de serviços. Inclui o sistema operacional, drivers de dispositivos, interfaces com dispositivos de baixo nível e *middleware* (YOUNGBLOOD; HOLDER; COOK, 2005). É também a interface de comunicação entre a casa e os usuários (DAS *et al.,* 2002).

- **Informação.** Colhe, guarda e gera conhecimento útil para camada de decisão (COOK *et al.,* 2003; DAS *et al.,* 2002). Inclui componentes de predição, bancos de dados, interfaces para usuários, componentes de *data mining*, sintetizador de voz e componentes de reconhecimento (YOUNGBLOOD; HOLDER; COOK, 2005). Transforma dados em informações úteis, como modelos de ação, padrões, etc. (COOK; DAS, 2007).

- **Decisão.** Seleciona as ações a serem executadas baseando-se nos dados das outras camadas enviados pela camada de Informação (COOK *et al.,*

2003; DAS *et al.,* 2002). Trabalha a informação, aprende com informações armazenadas, toma decisões através de ações, visando a automatizar o ambiente, e monitora e desenvolve políticas para manter a segurança dos usuários (YOUNGBLOOD; HOLDER; COOK, 2005).

A percepção é um processo visto “de baixo para cima” (*bottom-up process*). Sensores monitoram o ambiente e, se necessário, transmitem informação através da camada de Comunicação. Bancos de dados armazenam a informação na camada de Informação, transformam-na em informação útil, atualizam abstrações e predições aprendidas e alertam a camada de Decisão sobre novos dados. A camada de Decisão seleciona uma ação, verifica sua segurança e relata sua decisão à camada de Informação para armazenamento, e à camada de Comunicação para o envio do comando de ação ao atuador correto. A camada Física executa a ação (COOK *et al.,* 2003; COOK; DAS, 2007; DAS *et al.,* 2002; YOUNGBLOOD; HOLDER; COOK, 2005).

3.1.5.Arquitetura de *Software*

A programação do sistema de automação da MavHome é dividido em seis camadas de *software* (YOUNGBLOOD; HOLDER; COOK, 2005):

- **Componentes Físicos**. Consiste de todos os dispositivos reais utilizados pelo sistema. Inclui interfaces de comunicação via rede elétrica, telas de toque, dispositivos de leitura de gestos, câmeras, etc.

- **Interface de Computador**. Contém as interfaces com os dispositivos físicos. Inclui interfaces PCI, USB, *firewire*, drivers de dispositivos, sistema operacional do computador, interfaces de *software*, serviços para acesso aos dis-

positivos físicos, os computadores e a infra-estrutura de rede. São utilizados computadores PCs com sistema operacional Linux.

- **Interface Lógica**. Cria uma série de serviços atômicos em torno de cada sensor e atuador do sistema, *proxies* lógicos. Os *proxies* para componentes das redes Argus e X-10 são programados em C++ e utilizam memória compartilhada para comunicação local e soquetes para comunicação remota. Eles mantêm o estado atual de cada dispositivo e mecanismos de leitura e escrita (quando aplicável). A interligação entre os dispositivos é feita através da tecnologia ZeroConf.

- ***Middleware***. Sua função é facilitar a comunicação, a mobilidade de processos e o descobrimento de serviços/nomes. Através da tecnologia ZeroConf, os *proxies* dos dispositivos físicos nas redes Argus e X-10 são encontrados pelos demais componentes do sistema. Os componentes de alto nível podem ser programados em C++ e Java e se comunicam através do *middleware* CORBA (OmniORB).

- **Serviços**. Disponibiliza uma série de funções de suporte como armazenamento de informações (banco de dados PostgreSQL), geração de conhecimento, agregação de componentes dos níveis inferiores e interfaces para os usuários através de controles remotos, *PDAs,* páginas de internet, reconhecimento e síntese de voz. Utiliza o *middleware* para capturar dados nas camadas inferiores e fornecer informações aos aplicativos.

- **Aplicativos**. Utilizam as demais camadas para desenvolver funções de aprendizado e tomada de decisão.

3.2. Gator Tech Smart House

A Gator Tech Smart Home (GTSH) é uma iniciativa do Laboratório de Computação Pervasiva e Móvel da Universidade da Flórida. Trata-se de uma casa-laboratório onde são desenvolvidas tecnologias para Espaços Pervasivos Programáveis. Estes espaços se monitoram e monitoram seus usuários continuamente, interligando o mundo físico aos sistemas de monitoramento remoto e aos serviços (*software*) que podem intervir diretamente no funcionamento da GTSH. O objetivo é a criação de uma plataforma para programação destes espaços através de aplicativos formados por composição de serviços (HELAL *et al.;* 2005). A Figura 16 mostra a planta da casa.

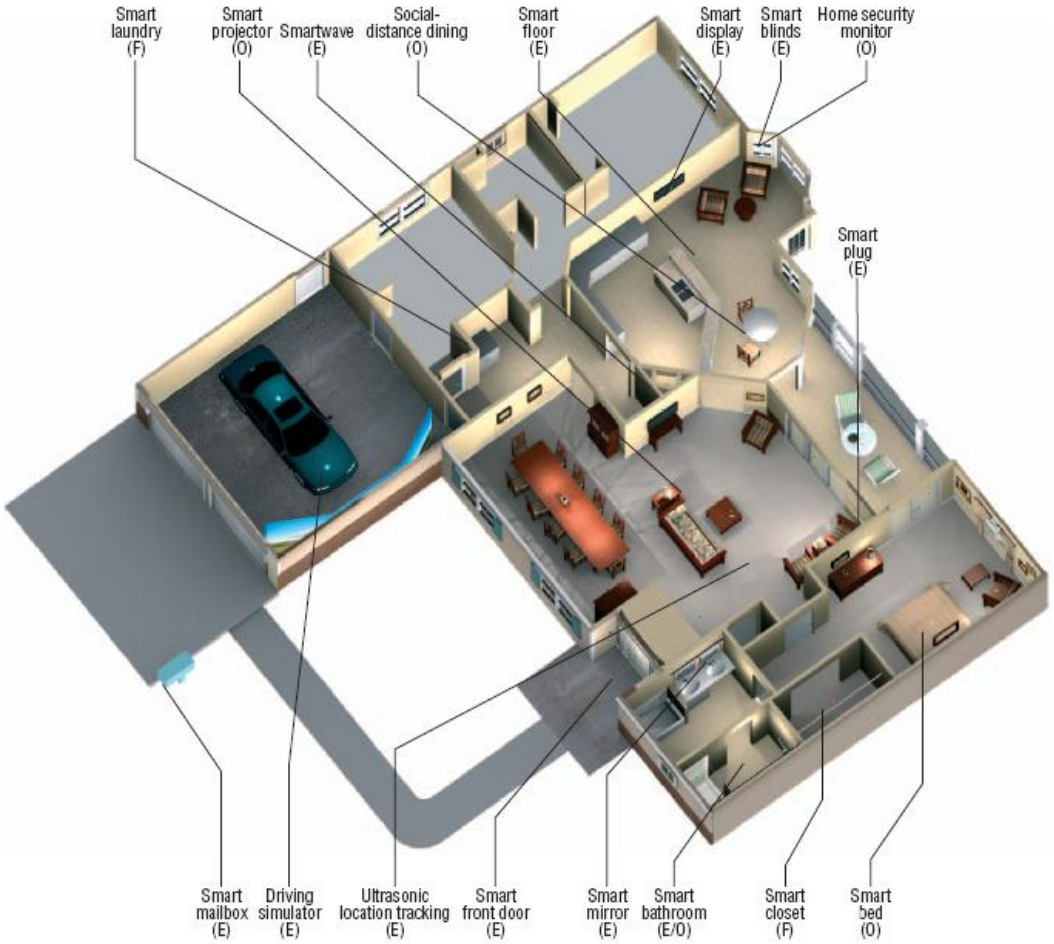

Figura 16 – Gator Tech Smart House. Fonte: Helal *et al.* (2005).

A plataforma proposta é programada em um conjunto de computadores e de microcontroladores. Sensores, atuadores e outros dispositivos de automação da casa são acoplados aos microcontroladores. Estes últimos comunicam entre si e com os computadores através de diferentes redes de comunicação (KING *et al.,* 2006; HELAL *et al.,* 2005).

O reconhecimento de contexto e a localização de pessoas e objetos são algumas das aplicações encontradas na GTSH. Há também uma série de dispositivos e tecnologias que fazem parte da GTSH, como: uma caixa de correio inteligente que detecta a chegada de correspondência e notifica o ocupante; uma porta de entrada inteligente que possui identificação por RFID (*Radio Frequency Identification*) usado para o acesso de pessoas autorizadas, um microfone, uma câmera, um *display* de texto, um sistema automático para abertura de porta, uma fechadura elétrica e alto-falantes para comunicação entre os ocupantes e os visitantes; uma cama inteligente que permite o monitoramento dos padrões de sono dos ocupantes; lavanderia inteligente que utiliza tecnologia RFID para identificação de roupas e avisa os residentes quando e como separar as roupas para lavar; entre outros (HELAL *et al.,* 2005).

3.2.1. Reconhecimento de Contexto

O monitoramento de variáveis dos ambientes e a localização, a movimentação e a orientação de pessoas compõem o sistema de reconhecimento de contexto da GTSH. A localização, a movimentação e a orientação de pessoas são feitas por meio de um sistema ultra-sônico. Emissores são colocados nos ombros das pessoas e receptores são instalados nos ambientes. Através de técnicas de triangulação são detectadas a localização, o movimento e a orientação da pessoa portadora dos emissores (HELAL *et al.,* 2005). Um segundo sistema, menos intrusivo, para detectar

localização e movimentação através de sensores de pressão instalados abaixo do piso também foi proposto. Há um sensor de pressão para cada placa de piso. Assim, faz-se um mapeamento entre cada placa e sua posição relativa a uma origem (Figura 17). O movimento e a localização são percebidos através das mudanças de pressão causadas nos diferentes blocos de piso (KADDOURA; KING; HELAL, 2005; HELAL *et al.,* 2005).

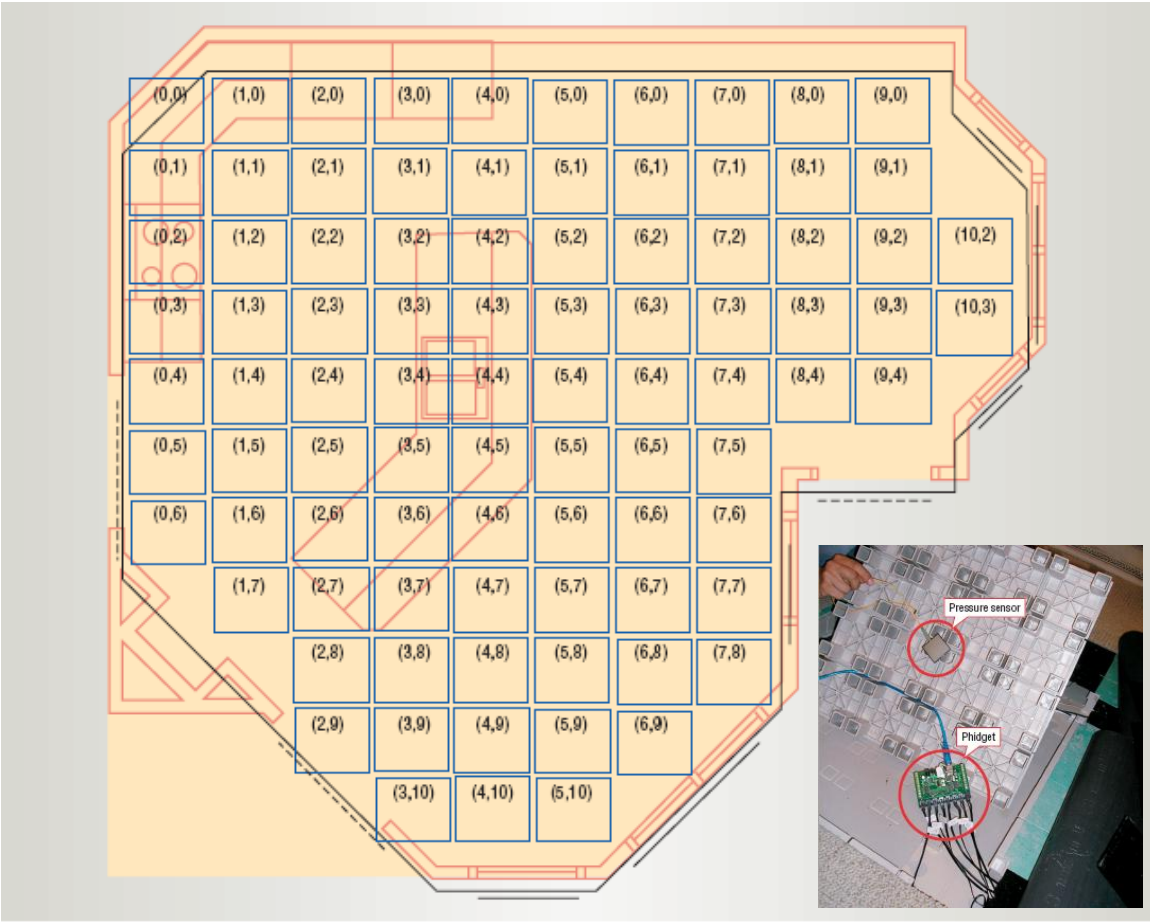

Figura 17 – Mapeamento do piso inteligente e placa de piso. Adaptado de Helal *et al.* (2005).

A GTSH transforma dados “brutos”, como valores de temperatura, umidade e iluminância, em estados como quente, frio, úmido, claro, escuro, etc. que são classificados em estados desejáveis, transitórios e não-permitidos. A partir destes, são construídos grafos de contexto contendo todos os possíveis estados de interesse. O

objetivo é promover ações que coloquem o ambiente sempre em estados desejáveis e que evitem os estados não-permitidos (HELAL *et al.,* 2005; HELAL *et al.,* 2007).

3.2.2.Ações de Controle

Nenhuma descrição específica de malhas de controle existentes na GTSH foi encontrada nos trabalhos revisados. As ações de controle baseiam-se em mapeamentos entre estados e ações. Cada atuador é associado a um efeito intencional. Por exemplo: o aquecedor tem como efeito intencional aquecer. Se o ambiente está frio, um gerenciador de contextos liga o aquecedor e verifica se o ambiente não entrará em estados não-permitidos (HELAL *et al.,* 2005; HELAL *et al.,* 2007).

3.2.3.Arquitetura Física

Sensores, atuadores, dispositivos domésticos inteligentes (descritos anteriormente), microcontroladores, computadores e redes de comunicação são componentes físicos da GTSH. Uma plataforma de comunicação e programação, Atlas, foi desenvolvida para interconectar os diversos componentes físicos e lógicos na GTSH. Sensores e atuadores são conectados a microcontroladores Atmel ATmega128L. Diversos tipos de redes de comunicação são utilizados para interligar microcontroladores e computadores: 10BaseT Ethernet, 802.11b WiFi, Bluetooth, USB e ZigBee (KING *et al.,* 2006).

3.2.4. Arquitetura Lógica

O sistema de automação da GTSH é dividido em seis camadas lógicas, ilustradas na Figura 18, com particularidades funcionais e estruturais (HELAL *et al.,* 2005):

- **Física**. São os componentes físicos existentes na casa: sensores, atuadores e dispositivos inteligentes. Envia e recebe sinais representando grandezas físicas como temperatura, umidade, ligar, desligar, porcentagem de atuação, etc..

- **Plataforma de Sensores**. Representa os sensores/atuadores da camada física através de serviços (*software*). Desacopla particularidades dos componentes físicos das camadas superiores.

- **Serviço**. Conjunto de serviços (*software*) com funcionalidades do sistema. Permite a criação de novas funcionalidades através da composição de serviços.

- **Conhecimento**. Contém a ontologia de serviços, aplicativos e dispositivos que compõem o sistema.

- **Gerência de contextos**. Conjunto de contextos de interesse representados por grafos que interligam sensores e estados. Um contexto pode definir ou restringir a ativação de serviços.

- **Aplicação**. Consiste de um gerente de aplicações que ativa e desativa serviços e um ambiente de desenvolvimento integrado (IDE – *Inegrated Development Environment)* gráfico para o gerenciamento das camadas anteriores.

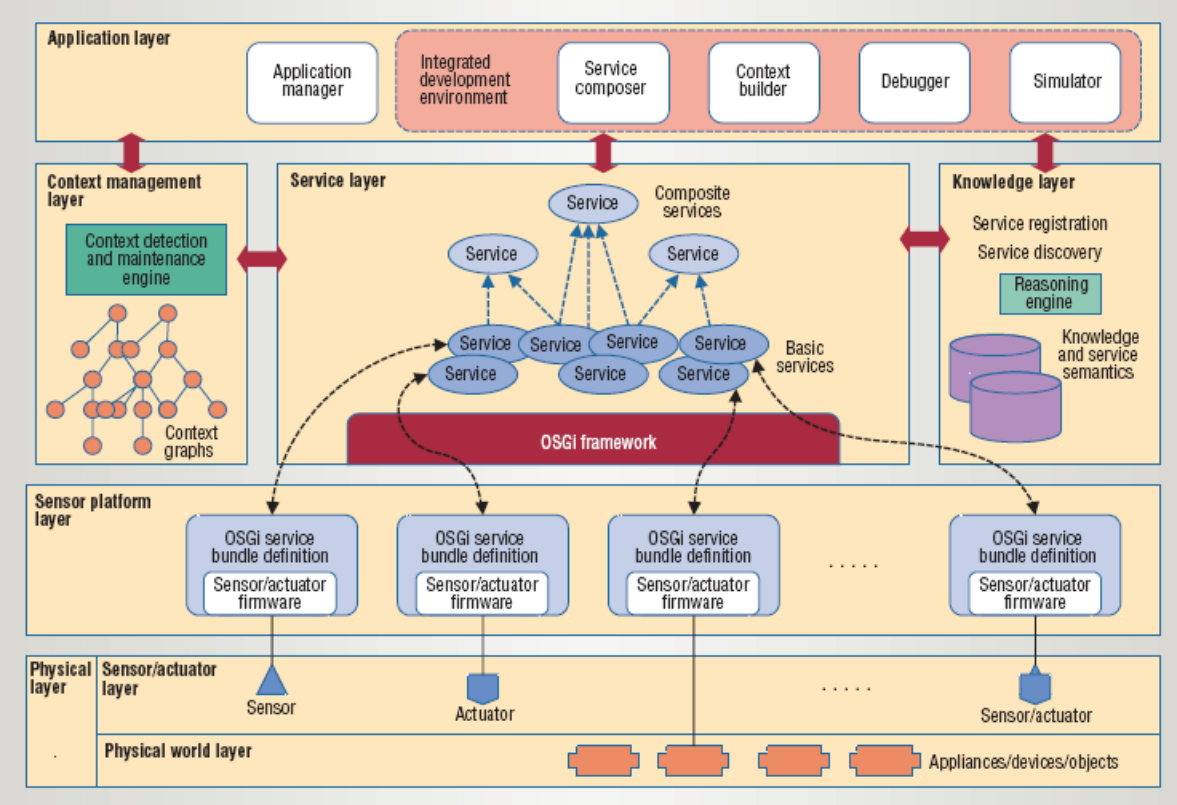

Figura 18 – Arquitetura lógica da GTSH. Fonte: Helal *et al.* (2005).

3.2.5. Arquitetura *Software*

A arquitetura de *software* é orientada a serviços (SOA – *Service-Oriented Architecture*), baseada na plataforma OSGi e associada à arquitetura lógica descrita na seção 0. Os componentes de *software*, chamados de *OSGi bundles* (ou simplesmente *bundles*) na plataforma OSGi, podem dinamicamente descobrir uns aos outros e colaborarem. Os componentes físicos (Plataforma de Sensores), os serviços e as aplicações são representados por *bundles* (ver seção 0 e Figura 18) (HELAL *et al.,* 2007; KING *et al.,* 2006).

Há uma série de aplicações utilizadas para administrar a plataforma de *software* utilizada na GTSH como: Gerenciador de Rede, Gerenciador de Configuração, *Bunddle Repository*, Gerenciador de Contextos e Monitor de Segurança e Módulos de Comunicação (HELAL *et al.,* 2007).

3.3. EDIFICIO – sistema de controle integrado e adaptável às vontades dos usuários

EDIFICIO (*Efficient Design Incorporating Fundamental Improvements for Control and Integrated Optimization*) é um projeto de pesquisa com objetivo de desenvolver um sistema integrado de controle para aquecimento, sombreamento e iluminação artificial visando o desempenho energético e o conforto dos usuários. Um controlador foi proposto para escritórios do LESO-PB (*Solar Energy and Building Physics Laboratory*) da Escola Politécnica Federal de Lausanne, na Suíça (Figura 19). A descrição apresentada nesta seção se baseia no trabalho de doutorado de Antoine Guillemin (GUILLEMIN, 2003).

Guillemin e Morel (2001) propõem um controlador para regular a abertura de um dispositivo de sombreamento externo, a iluminação artificial e o sistema de aquecimento em ambientes do tipo escritório para um único usuário visando a melhorar a eficiência energética. Valores padrões de conforto para temperatura e iluminância são adotados como *setpoints* iniciais do sistema. A posição do dispositivo de sombreamento também é controlada de modo a não causar ofuscamento. Algoritmos baseados em lógica difusa, redes neurais artificiais e algoritmos genéticos fazem predições e controlam o ambiente de forma eficiente (energeticamente) e, ao mesmo tempo, satisfazem as vontades do usuário de cada escritório investigado (GUILLEMIN, 2003).

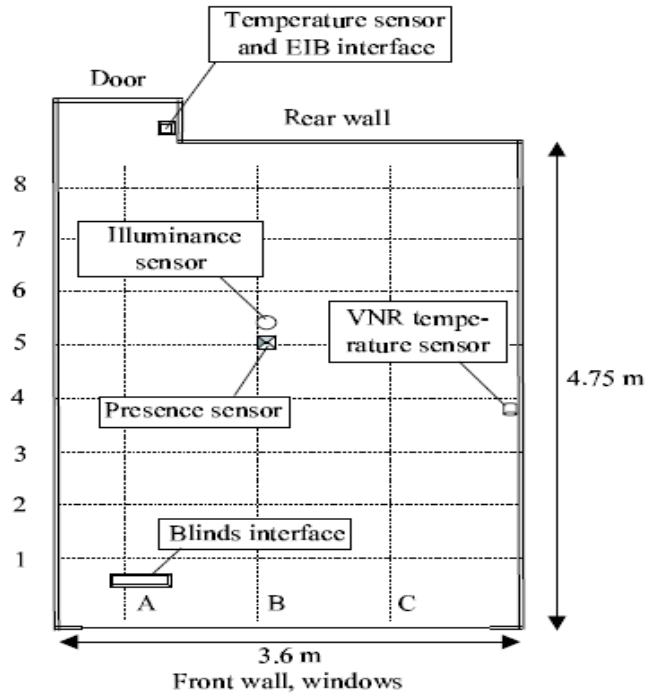

Figura 19 – Escritório típico do LESO-PB.
Fonte: Guillemin (2003).

3.3.1. Reconhecimento de Contexto

O controlador desenvolvido trabalha com dois contextos principais detectados através de sensores de movimento: ausência de usuário e presença de usuário. Esses contextos são utilizados para mudar os modos de operar do controlador entre modo de otimização energética e modo de otimização visual. Um terceiro contexto, chamado de *sleep mode*, permite o ajuste dos *setpoints* para tarefas não corriqueiras (ex. um apresentação de *slides*), desativando o controle automático e tornando manual a operação do sistema até que seja detectada ausência novamente (GUILLEMIN, 2003; GUILLEMIN; MOREL, 2001).

Quando o usuário do escritório está presente (modo de otimização visual), algoritmos de aprendizagem adaptam os pontos de funcionamento do sistema (*setpoints*) aos desejos reportados pelo usuário através de interfaces de comandos instaladas em cada escritório. Há três algoritmos específicos, que procuram equilibrar a vontade do usuário e a eficiência energética, para adaptação de cada um dos três

parâmetros do sistema (temperatura interna, iluminância interna e posição do dispositivo de sombreamento) aos desejos do usuário, diminuindo, desse modo, as possibilidades de rejeição ao sistema (GUILLEMIN, 2003; GUILLEMIN; MOLTENI, 2002).

3.3.2. Ações de Controle

As ações de controle visam, de forma integrada, garantir o *setpoint* de temperatura e iluminância do ambiente, aproveitando ao máximo a iluminação e o calor provenientes dos raios solares e tomando o cuidado de evitar o ofuscamento. São variáveis controladas a intensidade da iluminação artificial e do sistema de aquecimento, e a abertura do dispositivo de sombreamento instalado exteriormente à janela (GUILLEMIN, 2003).

São monitoradas duas variáveis externas ao prédio dos escritórios: a irradiância global horizontal e a temperatura, a partir das quais, e de acordo com a data, a hora e a posição de cada escritório são calculadas as seguintes variáveis: estação (média de temperatura das últimas 24 horas), azimute solar, altitude solar, iluminância vertical direta, iluminância global vertical e irradiância global vertical. A iluminância interna, utilizada para o controle da iluminação artificial, é calculada com base na iluminância vertical externa e a abertura do dispositivo de sombreamento. A medição de iluminância na área de trabalho é utilizada para calibrar o modelo que relaciona a iluminância vertical externa com a iluminância interna. Detalhes sobre o tratamento matemático dessas variáveis podem ser encontrados em Guillemin (GUILLEMIN, 2003). A Figura 20 ilustra o esquema do pré-processamento de dados.

Três controladores trabalham em conjunto: um para dispositivo de sombreamento, um para iluminação artificial e um para o sistema de aquecimento. Cabe res-

saltar que o segundo opera como suplemento do primeiro em momentos onde a luz solar é insuficiente.

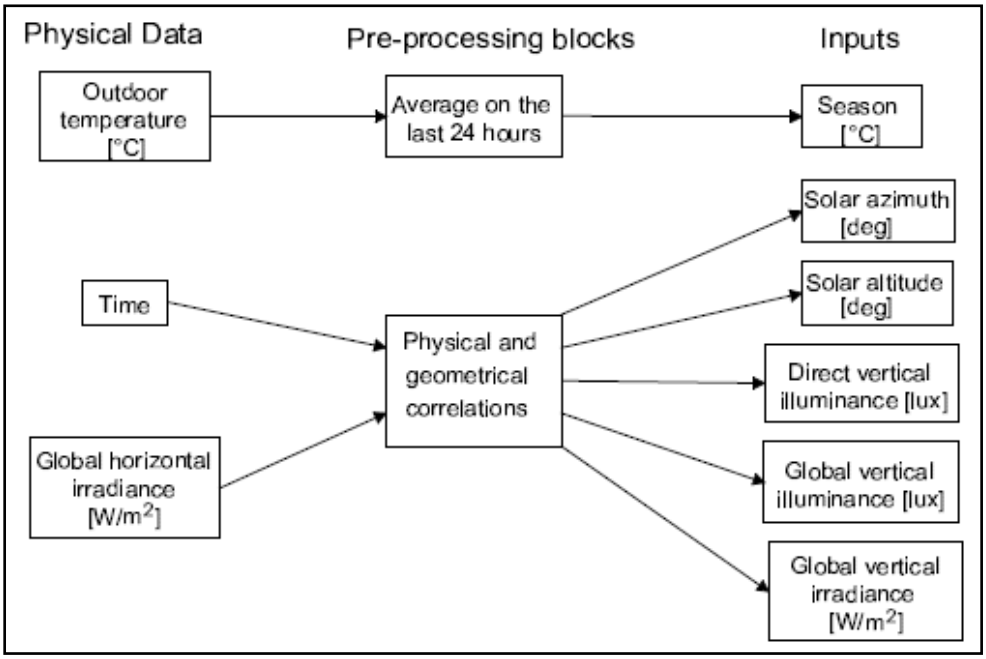

Figura 20 – Pré-processamento dos dados. Fonte: Guillemin (2003).

O controlador do dispositivo de sombreamento é difuso e possui dois modos de operação: ausência de usuário e presença de usuário. No primeiro modo (ausência de usuário), o controlador regula a abertura do dispositivo de sombreamento e tem como variáveis de entrada: a estação (média de temperatura externa das últimas 24 horas), a irradiância solar global horizontal e a diferença entre as temperaturas interna e externa (GUILLEMIN, 2003).

No segundo modo (presença de usuário) são utilizados dois grupos de regras para controlar o dispositivo de sombreamento. O primeiro grupo determina a abertura máxima do dispositivo de sombreamento para evitar o ofuscamento, e tem como variáveis de entrada: a iluminância vertical direta, a altitude solar e o azimute solar. O segundo determina a abertura do dispositivo de sombreamento para o aproveitamento da luz solar, baseando-se na iluminância global vertical e na estação (média de temperatura externa das últimas 24 horas). O mínimo entre os dois valores de

abertura do dispositivo de sombreamento será o valor final do controlador. Posteriormente, um filtro de movimento evita que o novo valor de abertura seja propagado ao atuador caso a variação seja pequena, ou em um intervalo curto de tempo em relação à última movimentação (GUILLEMIN, 2003).

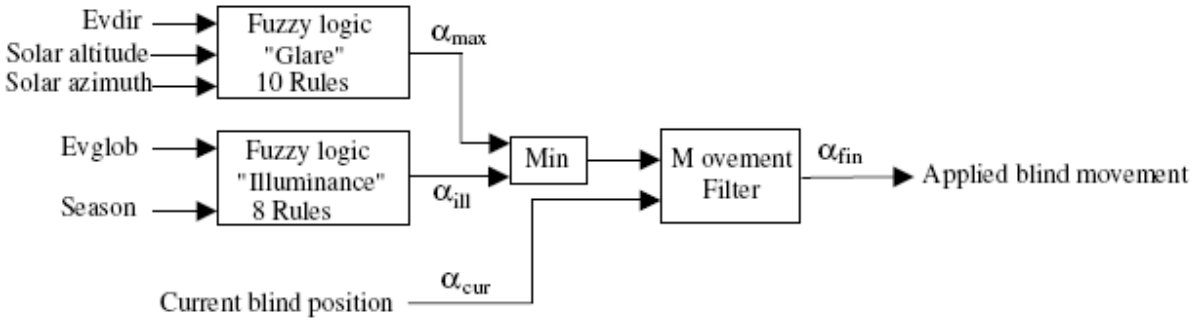

Figura 21 – Diagrama do controlador de posição do dispositivo de sombreamento quando há presença de usuários. Fonte: Guillemin (2003).

O controlador do sistema de aquecimento controla a fração de potência do sistema de aquecimento a partir das seguintes variáveis de entrada: *setpoint* especificado, temperatura interna, irradiância solar média prevista para as próximas seis horas e máxima irradiância teoricamente possível para as próximas seis horas. A previsão de irradiância solar média prevista para as próximas seis horas é feita por uma rede neural artificial que recebe como dados de entrada as irradiâncias corrente, de uma hora atrás, de dezoito horas atrás e a máxima já computada para as próximas seis horas. O controlador assume uma previsão de ocupação das 8h às 18h horas durante os dias da semana (GUILLEMIN, 2003).

3.3.3. Arquitetura Física

O sistema de controle utilizado no LESO-PB utiliza três computadores interligados por uma rede Ethernet: um executa os algoritmos de controle, um faz interface com uma rede EIB (*European Installation Bus*), onde estão ligados sensores e atua-

dores, e um último faz a aquisição de dados através de um sistema legado, VNR, existente no prédio desde a década de 1980 (ver Figura 22).

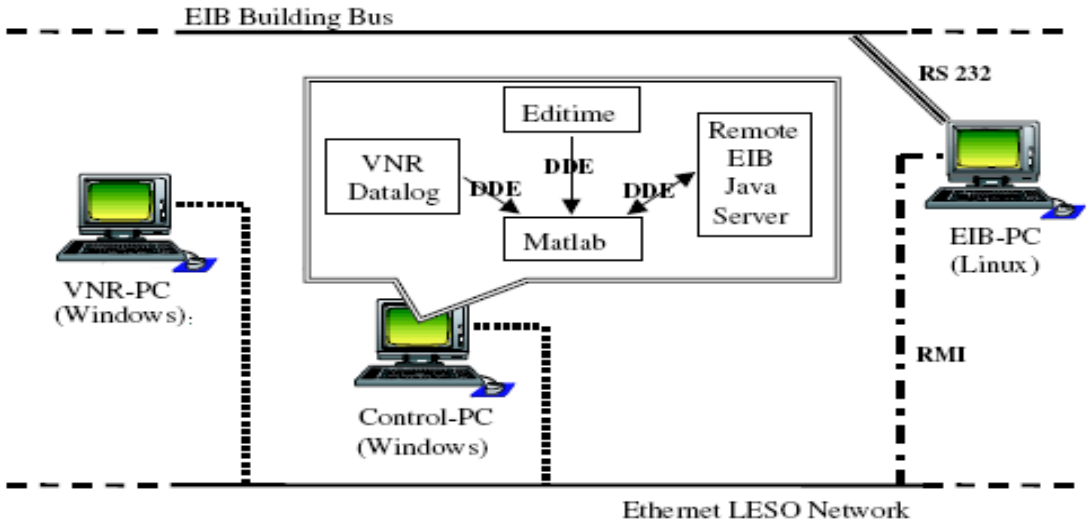

Figura 22 – Arquitetura do sistema de controle do LESO-PB. Fonte: Guillemin (2003).

Os atuadores de cada escritório (iluminação artificial, sistema de aquecimento e dispositivo de sombreamento) estão conectados à rede EIB (ver Figura 23). Os sensores estão interligados à rede EIB ou ao sistema de aquisição de dados VNR, alguns são comuns a todo o prédio (globais) e outros são individuais por escritório. O Quadro 2 descreve os sensores instalados no LESO-PB, seu tipo e a qual rede de comunicação (EIB/VNR) o sensor está conectado (GUILLEMIN, 2003).

	Sensor (grandeza)	Tipo de sensor	EIB / VNR
Individual (escritório)	Temperatura do ar (controle)	PT-100	EIB
	Temperatura do ar (monitoramento)	PT-100	VNR
	“Presença” (movimento)	Infravermelho passivo	EIB
	Iluminância na área de trabalho	Luxímetro	EIB
	Abertura da janela (0/1)	Magnético	EIB
Global (prédio)	Temperatura do ar externa	PT-100	VNR
	Irradiância horizontal global	Piranômetro	VNR
	Velocidade do vento	Anemômetro	VNR
	Direção do vento	“Biruta”	VNR

Quadro 2 – Sensores disponíveis no LESO-PB. Fonte: Guillemin (2003).

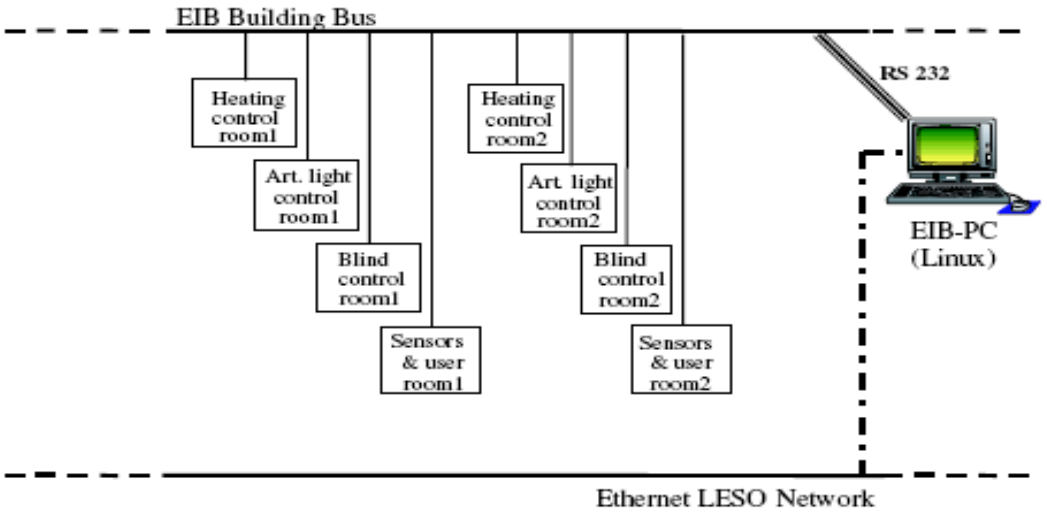

Figura 23 – Rede EIB e atuadores e sensores. Fonte: Guillemin (2003).

Há dois módulos de interface para ajuste dos parâmetros do sistema em cada escritório. Um está relacionado à iluminação artificial e ao sistema de aquecimento. O outro está relacionado ao ajuste da posição do dispositivo de sombreamento externo.

3.3.4.Arquitetura Lógica

O sistema de automação no LESO-PB é dividido em três níveis de controle (ver Figura 24). O primeiro nível transforma grandezas físicas em sinais elétricos e vice-versa. Possui a malha de controle primária do sistema. É totalmente dependente do ambiente onde o sistema é instalado. O segundo nível possui o conhecimento, na forma de regras difusas, que gera os parâmetros de controle (*setpoints*) para o primeiro nível. O terceiro nível é responsável pela adaptação de longa duração do segundo nível e leva em consideração mudanças nas características do prédio, dos dispositivos, dos desejos dos usuários e da eficiência energética. Os dois últimos níveis são facilmente ajustáveis para diferentes situações (GUILLEMIN, 2003).

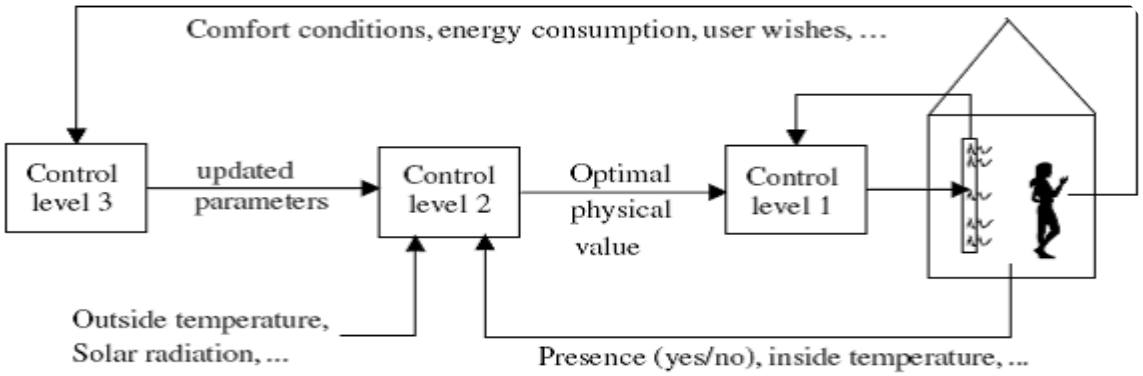

Figura 24 – Arquitetura de controle de três níveis do LESO-PB. Fonte: Guillemin (2003).

3.3.5. Arquitetura *Software*

Dois computadores são responsáveis pela aquisição de dados da rede. O primeiro está conectado ao sistema legado VNR (PC-VNR). A cada dois segundos são lidos dados gerados no sistema VNR e gravados em um arquivo texto. Este arquivo é usado como interface para comunicação de dados. O segundo computador está conectado à rede EIB através da sua porta serial. Um servidor EIB é encarregado de ler e escrever dados na rede EIB e disponibilizá-los através de protocolos tipo RMI[12] (GUILLEMIN, 2003).

Um terceiro computador é responsável pela execução dos algoritmos de controle e utiliza para tal tarefa quatro aplicativos principais (GUILLEMIN, 2003):

- **MATLAB**. É a parte central do sistema. Comunica-se com os outros módulos do sistema via protocolo DDE, calcula as saídas dos controladores e propaga seus valores ao servidor EIB via protocolo DDE. Também é o *software* responsável pelos algoritmos de adaptação.

- **EIB Java Server**. Comunica-se com o servidor EIB no PC-EIB via RMI. Fornece dados da rede EIB através do protocolo DDE. Dados podem ser requi-

[12] *Remote Method Invocation*. Chamada de método remota.

sitados pelo cliente (*cold link*) ou propagados automaticamente a cada atualização de dados (*hot link*).

- **VNR Datalog Server**. Lê a cada segundo o arquivo texto gerado no PC-VNR. Disponibiliza os dados via protocolo DDE.

- **EDITIME**. Servidor DDE responsável por gerar base tempo (data e horário) do sistema. Envia a cada minuto o horário e a data correntes.

3.4. Considerações finais

Os três trabalhos descritos resumem, de forma geral, as soluções encontradas na bibliografia para arquiteturas de software aplicadas a sistemas computacionais em ambientes construídos. É consenso a necessidade de uma camada física responsável pela comunicação com dispositivos físicos. O processamento dos dados é organizado de modo particular em cada solução, o que implica em diferentes abordagens de programação.

A arquitetura de software proposta para a Mavhome baseia-se em um modelo de fluxo de dados, no qual a comunicação com os dispositivos físicos ocorre através de um processo visto “de baixo para cima” (*bottom-up process*). Informações provenientes de dispositivos são processadas através de camadas com diferentes funções: comunicação, tratamento de informação e decisão. Não há uma separação clara de partes responsáveis por tarefas como reconhecimento de contexto e/ou execução de algoritmos de controle. Nenhum tipo de abstração é descrito, diferentemente do que ocorre na *Gator Tech Smart House*. Nesta última, abstrações como a representação de dispositivos físicos e de funcionalidades em forma de serviços, e a

representação de contextos através de grafos interligando sensores e estados, orientam o desenvolvimento de aplicativos.

A *Gator Tech Smart House* apresenta uma arquitetura de software com camadas específicas para lidar com os diferentes aspectos de um software pervasivo. Os aplicativos são construídos a partir de um conjunto de módulos que são abstrações de diferentes aspectos sistema. Contextos são programados através de grafos em uma camada específica, as funcionalidades do sistema através serviços em outra camada, a ontologia do sistema em uma camada de conhecimento, e os dispositivos físicos são associados aos serviços através de uma plataforma de sensores. As informações provenientes dos dispositivos físicos são representadas por serviços básicos que interagem com outros serviços responsáveis por processar as informações. Estes últimos interagem com as camadas de contexto, de conhecimento e de aplicação, executando funcionalidades programadas nestas camadas.

O último trabalho, EDIFICIO, é um exemplo típico de sistema de automação do ambiente construído. Como em outros trabalhos da área de automação do ambiente construído, não se observa preocupação com a estruturação do software usado para automação. Em comum, esses aplicativos se comunicam com dispositivos físicos e processam a informação utilizando um software mais conveniente ao programador. Não é raro encontrar soluções nas quais a comunicação entre aplicativos ocorre através da leitura e da gravação de dados em um arquivo texto. Essa falta de preocupação com a estruturação do software dificulta a expansão, a reutilização e a manutenção desses sistemas. Em geral, essas soluções são restritas ao ambiente de laboratório e a problemas específicos.

4. Uma Arquitetura de *Software* Neuro Reativa para Sistemas de Automação do Ambiente Construído

Este capítulo propõe e descreve uma arquitetura de *software* para Automação do Ambiente Construído baseada em unidade funcionais básicas chamadas neurônios e glândulas. A primeira parte coloca os requisitos necessários a um sistema de Automação do Ambiente Construído. Em seguida é descrita uma proposta de Arquitetura de *Software* para Automação do Ambiente Construído. Por fim, são feitas considerações sobre a arquitetura proposta e os requisitos do sistema.

4.1. Requisitos

Nesta seção são identificadas particularidades dos ambientes construídos e três requisitos principais que uma arquitetura de *software* para automação do ambiente construído deve atender: modularidade, flexibilidade e capacidade de integração das partes. O objetivo é uma arquitetura de *software* que torne fácil o desenvolvimento, a manutenção e a expansão de sistemas de Automação do Ambiente Construído.

4.1.1. Particularidades dos Ambientes Construídos

Um sistema de automação do ambiente construído é um espaço físico, onde um sistema computacional interage com três entidades relevantes: usuários, sensores e atuadores (JANSEN *et al.,* 2005). Os usuários interagem com o ambiente buscando satisfazer suas necessidades e desejos. Os sensores capturam informações sobre o ambiente. Os atuadores agem sobre o ambiente, modificando-o.

O ambiente construído é uma entidade dinâmica sujeito a variações relacionadas a diversas variáveis. Cabe ao projetista do sistema de automação determinar quais variáveis, contextos e comportamentos são relevantes ao sistema projetado. Os contextos são as informações, em geral, colhidas pelos sensores que caracterizam a situação do ambiente ou da entidade em questão (DEY, 2001). Os comportamentos são conjuntos de ações executadas, em geral, pelos atuadores visando a manter o ambiente em um estado desejado (HAGRAS *et al.,* 2003; MURPHY, 2000).

Tradicionalmente, os ambientes construídos são divididos em ambientes menores, como uma casa que é dividida em quartos, banheiros, sala e cozinha, onde são desenvolvidas diferentes atividades ou tarefas (RUTISHAUSER; SCHÄFER, 2002). É natural que a granularidade de um sistema computacional para automação do ambiente construído seja baseada nessas unidades onde são desenvolvidas funções específicas (HAGRAS *et al.,* 2003). Desse modo, pode-se tratar cada uma dessas unidades como uma entidade autônoma. Essas, por sua vez, podem se combinar formando unidades maiores e mais complexas. É possível, assim, construir o sistema de automação de uma casa, por exemplo, tratando-a como um conjunto de unidades autônomas menores como quartos, banheiros, sala e cozinha. Estas últimas ainda podem ser dividas em partes menores como, por exemplo, a pia da cozinha onde tarefas específicas são desenvolvidas. Cada uma dessas unidades (grão) possui *setpoints*, contextos e comportamentos específicos.

4.1.2. Modularidade

A modularidade é o único atributo de *software* que permite que um programa seja intelectualmente administrável (PRESSMAN, 1995). Uma característica interessante, descoberta por meio da experimentação relativa à solução de problemas hu-

manos, é que a complexidade para solução de um problema que seja combinação de outros problemas é maior que a soma das complexidades individuais para solução de cada problema combinado (PRESSMAN, 1995), isto é,

$$C\left(\sum_{i=2}^{n} p_i\right) > \sum_{i=2}^{n} C(p_i),$$

onde *C(p)* é uma função que define a complexidade de um problema *p*. Vale ressaltar que o aumento indiscriminado do número de módulos pode gerar um novo problema associado ao gerenciamento dos módulos (PRESSMAN, 1995).

Em aplicações de automação do ambiente construído, um controlador monolítico torna difícil a reconfiguração dinâmica do ambiente ou a utilização de partes do sistema independentemente das outras (COEN, 1997). Sistemas modulares facilitam a manutenção e a realocação quando comparados a sistemas monolíticos (YOUNGBLOOD; HOLDER; COOK, 2005). Vale ainda ressaltar as facilidades de teste, de expansão e de combinação que arquiteturas modulares possuem (MURPHY, 2000).

Uma arquitetura de *software* para Automação do Ambiente Construído deve ser modular o suficiente para permitir que módulos possam ser combinados de maneira particular em cada aplicação, e para facilitar testes, manutenções e expansões.

4.1.3. Flexibilidade

Flexibilidade é a característica relacionada inversamente ao esforço exigido para modificação do *software*, isto é, a facilidade de alteração do *software* para sua adequação a um novo modo de operação (PRESSMAN, 1995). Dentro do contexto da Automação do Ambiente Construído, a flexibilidade de um sistema relaciona-se à facilidade de modificação do *software* para sua adaptação a uma nova solução parti-

cular. O primeiro passo para tornar uma arquitetura de *software* flexível é a modularidade. No entanto, a modularidade não garante que os módulos terão flexibilidade suficiente para se associarem de maneira particular de acordo com a necessidade de cada novo problema.

Visando à flexibilidade, uma arquitetura de *software* para Automação do Ambiente Construído deve definir uma interface de comunicação para os módulos, facilitando combinações diferentes em cada solução particular. Cada ambiente construído possui particularidades que o tornam único, o que implica em soluções únicas.

4.1.4. Capacidade de Integração das Partes

As aplicações de Automação do Ambiente Construído podem contemplar vários subsistemas, como iluminação, condicionamento de ar, controle de acesso, incêndio e segurança. Guillemin e Morel. (2001) e Kolokotsa *et al.* (2002) apontam a importância de uma abordagem integrada. É importante que todas as partes do sistema possam responder de forma cooperativa e coordenada aos diferentes contextos de uma aplicação.

Uma arquitetura de *software* para Automação do Ambiente Construído deve possuir mecanismos que torne possível a integração entre as partes, isto é, as partes componentes devem responder de forma cooperativa e coordenada aos diferentes contextos da aplicação.

4.2. Proposta

Visando a satisfazer os requisitos expostos anteriormente, é proposta aqui, uma arquitetura de *software* para Automação do Ambiente Construído baseada em unidades básicas chamadas neurônios e glândulas, e composta por duas camadas

de *software*: camada Reativa e camada Ontológica. Essas duas camadas possuem interface com mais duas camadas, Física e de Aplicação, formando um modelo de quatro camadas (ver Figura 25):

- **Física.** É a interface entre o mundo físico e o sistema de automação. Sensores, atuadores e redes de campo são componentes básicos desta camada.

- **Reativa.** É a camada responsável por processar toda informação enviada pelos sensores e reagir comandando os atuadores. Toda ação do sistema é o resultado da reação dos dois elementos constituintes desta camada: os neurônios e as glândulas.

- **Ontológica.** É a camada que guarda informações sobre a configuração e a estrutura da camada Reativa. É formada por um conjunto de tabelas inter-relacionadas contendo os parâmetros de configuração da camada Reativa.

- de **Aplicação.** É a camada entre o sistema de Automação do Ambiente Construído e os aplicativos que o utilizam.

As camadas Física e de Aplicação são, respectivamente, camadas de interface com o mundo físico e com os aplicativos que se comunicam com as camadas Reativa e Ontológica, núcleo da arquitetura proposta. Conjuntos de neurônios e glândulas, na camada Reativa, representam as malhas de controle, os contextos e os comportamentos do sistema de automação do ambiente construído.

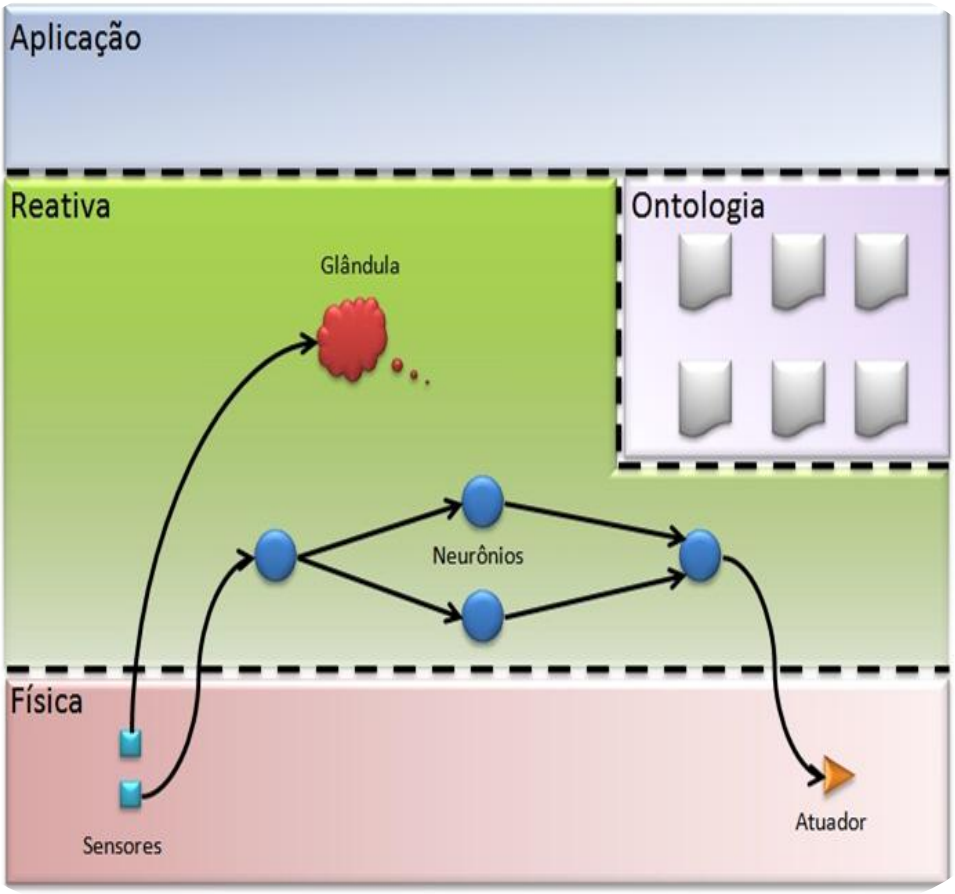

Figura 25 - Camadas da arquitetura proposta.

4.2.1. Camada Física

A camada Física engloba sensores, atuadores e toda a infra-estrutura de rede de campo e *hardware* necessários para a leitura de dados dos sensores e para comandar ações no ambiente através dos atuadores. Sua função é fazer a interface entre o mundo físico e a camada Reativa, onde todo o processamento de controle é feito. A infra-estrutura de rede de computadores também compõe esta camada quando computadores interligados em rede são utilizados.

4.2.2. Camada Reativa

A camada Reativa é responsável por toda ação do sistema de automação. Ela monitora continuamente os sensores e reage comandando os atuadores. Para atingir os requisitos de modularidade, flexibilidade e capacidade de integração entre as

partes, esta camada deve ser formada por elementos modulares e flexíveis, e ao mesmo tempo capazes de se integrarem com os objetivos gerais do sistema. A questão fundamental é: quais tipos de elementos podem ser utilizados para essa tarefa?

Para solucionar esta questão, buscou-se inspiração em um sistema capaz de executar tarefas extremamente complexas e integradas: o sistema de controle do corpo humano. O homem é capaz de executar ao mesmo tempo inúmeras tarefas de controle com objetivos diferentes: manter sua temperatura corpórea, digerir, ficar alerta a sons e movimentos, movimentar pernas e braços, dentre outras. Todo controle é resultado de associações de unidades funcionalmente simples, como os neurônios e as glândulas que compõem os dois sistemas de controle fundamentais do ser humano: o sistema nervoso e o sistema endócrino (GUYTON; HALL, 2006).

O sistema nervoso humano é responsável pela execução de funções sensório-motoras e autônomas. Ele possui em torno de 100 bilhões de neurônios (sua célula fundamental) que processam e se comunicam continuamente e paralelamente (BRAGA; CARVALHO; LUDERMIR, 2007). A cada minuto milhões de bits de informação de diferentes nervos e órgãos sensoriais são recebidos e processados, resultando em ordens executadas pelo corpo humano (GUYTON; HALL, 2006).

Pode-se dividir o sistema nervoso em três grandes camadas com características funcionais específicas: a corda espinhal, o baixo cérebro e o alto cérebro. A corda espinhal possui vários centros de controle para controlar movimentos e reflexos de órgãos espalhados por todo o corpo. Os níveis superiores do cérebro enviam sinais aos centros de controle da corda espinhal para que estes desempenhem suas funções. O baixo cérebro controla funções inconscientes e vitais como a respiração,

a pressão arterial, entre outras. O alto cérebro é onde reside a consciência, o processo de pensamento e a memória. Seu funcionamento é sempre combinado com centros de controle dos níveis inferiores (GUYTON; HALL, 2006).

Além do sistema nervoso, o sistema endócrino desempenha papel importante nas funções de controle do ser humano. Múltiplas atividades do corpo humano são coordenadas por diversos tipos de mensageiros químicos, entre os quais se destacam os hormônios. Esses últimos são secretados por glândulas na corrente sanguínea em resposta a estímulos neurais ou de outros hormônios, e influenciam o funcionamento de células em outras partes do corpo. Alguns afetam células em todo o corpo, outros afetam apenas tecidos específicos (GUYTON; HALL, 2006). Boa parte do comportamento humano é resultado dos níveis de hormônio na corrente sanguínea.

A camada Reativa inspira-se no funcionamento dessas unidades de controle fundamentais do corpo humano. Analogamente ao sistema de controle do corpo humano, neurônios e glândulas “artificiais” são os elementos básicos desta camada. Os neurônios em conjunto formam malhas de controle e associam-se transformando a informação como, por exemplo, combinando informações de diferentes sensores para caracterizar um contexto. As glândulas representam comportamentos e estados (*setpoints*) do sistema. Enviam mensagens (“secretam hormônios”) que alteram o modo de operar de diversas partes do sistema. A abrangência de ação de uma glândula é determinada pelo domínio a que ela pertence, isto é, somente neurônios e glândulas pertencentes a este domínio é que recebem suas mensagens (“hormônios”).

Os neurônios da camada Reativa realizam funções análogas às exercidas na corda espinhal e no baixo cérebro. São especializadas, possuem característica fortemente reativa, não possuem consciência do todo, não possuem qualquer processo de decisão ou de aprendizagem. As características não encontradas nos neurônios aqui propostos podem ser fornecidas por aplicativos desenvolvidos para a camada de aplicação. Nas próximas seções descrevem-se os elementos fundamentais da camada Reativa: neurônios, glândulas e domínios.

4.2.2.1. Neurônios

O neurônio é formado por um conjunto de entradas contínuas, uma única saída contínua, que representa seu estado e um algoritmo que mapeia as entradas e a saída. Ele representa uma pequena função do sistema, como o sinal de um sensor, a transformação de um sinal em outro tipo de sinal, um controlador, um atuador, uma função lógica ou uma função combinatória que combina sinais provenientes de vários neurônios. O neurônio tem seu comportamento influenciado pelas mensagens enviadas (“hormônios secretados”) pelas glândulas que podem desativá-lo ou ativá-lo ou ainda deixá-lo em um estado de funcionamento intermediário.

A associação de neurônios é feita através da interconexão deles. A saída de um neurônio pode ser conectada às entradas de vários neurônios. Cada conexão propaga um valor numérico real (v) e um peso (ou prioridade - w). Os pesos podem ser utilizados ou desprezados pelo neurônio receptor. Eles ponderam (ou priorizam) os valores das entradas. Enquanto o valor reflete o estado do neurônio, os pesos são características de cada conexão. A Figura 26 ilustra o modelo de neurônio proposto para a camada Reativa.

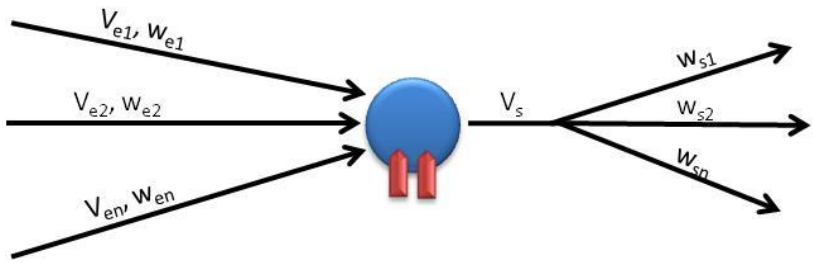

Figura 26 – Modelo de neurônio. As entradas possuem valores (v_e) e pesos (w_e) distintos. A saída propaga o mesmo valor (v_s) para diferentes neurônios através de conexões com pesos (w_s) distintos. Há também receptores para as mensagens enviadas por glândulas (em vermelho).

4.2.2.2. Glândulas

As glândulas são responsáveis por enviar mensagens (“secretar hormônios”) que representam estados (*setpoints*) ou comportamentos dos domínios aos quais elas pertencem, de acordo com o contexto. Cada mensagem (“hormônio”) possui um *status* (s) (ou valor) e um tipo (t). O tipo caracteriza a mensagem, o que permite que o receptor diferencie mensagens (“hormônios”) provenientes de diferentes glândulas. Já o status representa o nível ou valor que determinada mensagem (“hormônio”) carrega. As glândulas são capazes de detectar contextos através de suas conexões com neurônios e através do recebimento de mensagens (“hormônios”) de outras glândulas. Como os neurônios, as mensagens recebidas (“hormônios secretados”) por outras glândulas podem ativar, desativar ou deixar em nível intermediário o estado de funcionamento de uma glândula. A Figura 27 ilustra o modelo de glândula proposto para camada Reativa.

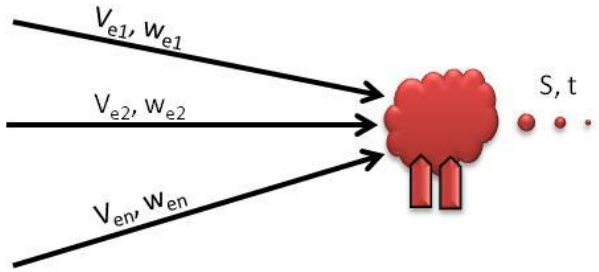

Figura 27 – Modelo de glândula. As entradas possuem valores (v_e) e pesos (w_e) distintos. A saída propaga o mesmo *status* (S) e tipo (t) para todos os neurônios e as glândulas pertencentes ao seu domínio. Há também receptores para as mensagens enviadas por outras glândulas.

4.2.2.3. Domínio

O domínio representa a zona de ação das glândulas pertencentes a ele. É uma entidade abstrata da camada Reativa: não executa nenhum tipo de tarefa. Mas é de fundamental importância, pois determina para quais entidades uma glândula deve enviar uma mensagem (“secretar hormônio”). Os domínios podem se relacionar hierarquicamente: um domínio pode pertencer a outro. Desse modo, as mensagens enviadas (“hormônios secretados”) por glândulas de um domínio “pai” são recebidas por todas as unidades de domínios “filhos”.

Um domínio caracterizará, em geral, um espaço dentro do ambiente construído com comportamentos e estados (*setpoints*) particulares. As glândulas caracterizarão os estados e os comportamentos destes domínios. E os neurônios caracterizarão sensores, atuadores, controladores, etc.

Os domínios desempenham outro papel importante de organização da camada Reativa. Eles podem ser utilizados para agrupar neurônios e glândulas pertencentes a determinado sistema ou que possuam características comuns, tornando mais fácil a localização de grupamentos com funcionalidades específicas.

4.2.3. Camada Ontológica

A camada Ontológica descreve em forma de tabelas relacionais os componentes da camada Reativa e suas relações. Ela guarda todas as informações de configuração da camada Reativa, o que engloba quais neurônios, glândulas e domínios compõem o sistema, os parâmetros de configuração de neurônios e de glândulas, as conexões entre neurônios e seus parâmetros de configuração.

Há uma íntima relação entre as camadas Reativa e Ontológica. Os elementos básicos da camada Reativa, neurônios e glândulas, utilizam a camada Ontológica para obter os parâmetros necessários para comunicação com outros neurônios e glândulas, e para sua própria configuração. Isto desacopla a configuração da programação. Programam-se os tipos básicos ou as classes de neurônios e glândulas, e todo o resto do sistema torna-se uma especificação de parâmetros na camada Ontológica.

4.2.4. Camada de Aplicação

A camada de Aplicação oferece aos aplicativos uma interface de comunicação com as camadas Reativa e Ontológica. Algoritmos de aprendizagem, de IHM (Interface Homem-Máquina) e programas de computação pervasiva são exemplos de aplicativos que podem ser desenvolvidos para a camada de Aplicação.

4.3. Considerações finais

A arquitetura proposta possui características reativas, isto é, é um sistema que monitora o ambiente e reage aos seus estímulos através de unidades fundamentais chamadas de neurônios e de glândulas. Os neurônios representam informações de controle relevantes, enquanto que as glândulas representam estados (*setpoints*) e comportamentos esperados. O sistema final é o resultado da associação dessas unidades fundamentais simples, ou módulos básicos, o que torna o sistema modular.

Os neurônios e as glândulas se associam através de conexões que possuem características comuns, isto é, um neurônio ou uma glândula pode propagar informação para qualquer outro neurônio ou glândula. Cada neurônio ou glândula receptor interpreta a mensagem recebida segundo os parâmetros da conexão. Desse modo um neurônio pode, a qualquer momento, receber novas conexões de novos neu-

rônios, criando uma nova conexão (externa ao neurônio), ou pode receber mensagens (“hormônios”) de novas glândulas criando um receptor para o novo tipo de mensagem (“hormônio”, também externo ao neurônio). O mesmo vale para as glândulas. Todas essas modificações de associação entre neurônios e glândulas não requerem modificação na programação dessas unidades básicas, o que permite que os mesmos módulos básicos se associem de inúmeras maneiras diferentes, tornado a arquitetura proposta flexível.

As glândulas propagam mensagens (“hormônios”) que representam estados e comportamentos de um domínio. Todas as entidades do domínio recebem essas mensagens (“hormônios”), e algumas (aquelas com receptores) reagem mudando seu ponto ou modo de operação. Dessa forma, estados e comportamentos são compartilhados entre unidades pertencentes ao mesmo domínio, o que permite que o sistema final responda a diferentes contextos de forma integrada.

Os domínios permitem agrupar neurônios e glândulas. Eles também podem guardar relações de hierarquia entre si, isto é, um domínio pode pertencer a outro. Usualmente, um domínio representa um espaço físico com determinada funcionalidade ou onde é realizada determinada tarefa, o que permite associar ambientes construídos reais com um conjunto de domínios programados na arquitetura proposta. Uma casa, por exemplo, será um domínio contendo outros domínios menores da casa como quartos, banheiros, salas, etc..

A arquitetura proposta trata diretamente de problemas ligados a automação do Ambiente Construído. A camada Reativa reage aos estímulos do ambiente, e praticamente todas as variáveis de controle importantes no sistema são representadas por neurônios e glândulas nesta camada. A camada Ontológica guarda os parâme-

tros de configuração do sistema de forma relacionada e estruturada. Essas características (das camadas Reativa e Ontológica) resultam em um sistema modular e flexível, e facilitam a programação, a manutenção e a expansão. O desenvolvedor pode manter o foco na aplicação, preocupando-se pouco com detalhes de código. Dados publicados na camada Reativa e parâmetros de configuração guardados na camada Ontológica podem ser acessados facilmente. Os parâmetros de configuração podem ainda ser modificados, assim como novos neurônios e glândulas também podem ser instanciados por aplicativos da camada de Aplicação, o que facilita a programação de aplicativos pervasivos e de aprendizado na camada de Aplicação.

As informações provenientes da camada Física são processadas na camada Reativa de acordo com configuração da camada Ontológica. A forma como neurônios e glândulas são associados determina a resposta do sistema aos estímulos do ambiente. Malhas de controle são programadas como associação de neurônios que representam elementos de um sistema controle como sensores, controladores, atuadores ou partes de controladores mais complexos, como funções de pertinência ou regras difusas em um controlador difuso. Neurônios também são associados para formar grafos para o reconhecimento de contexto. Comportamentos desejados para o sistema são representados por glândulas que “liberam hormônios”. Estes últimos influenciam o comportamento particular de cada neurônio ou glândula, segundo configuração dos mesmos.

5. Implementação da Arquitetura Proposta

Para tornar a arquitetura proposta factível é apresentada aqui uma implementação das camadas Ontológica e Reativa desenvolvida na linguagem de programação Java, baseada no *middleware* CORBA e no sistema de banco de dados MySQL. A camada Física deve comandar e coletar informações de dispositivos físicos no ambiente através de um sistema de comunicação. Diversas redes de campo e de sensores (como Lonworks, Pyxos, EIB, BACnet, X-10) estão disponíveis para essa tarefa. Para sistemas com mais de um computador, redes locais como a Ethernet e redes de grande abrangência como Internet (TCP/IP) são opções usuais de redes de computadores.

5.1. Camada Ontológica

A camada Ontológica da arquitetura proposta é formada por um conjunto de tabelas (um banco de dados) contendo as informações de configuração necessárias para o funcionamento da camada Reativa. Duas tabelas formam o núcleo da camada Ontológica. A primeira armazena as características de cada entidade (neurônios, glândulas, *setpoints* e comportamentos) que compõe a camada Reativa. A segunda lista todas as conexões entre neurônios e seus respectivos parâmetros de configuração como o peso e o tipo. Tabelas contendo parâmetros de configuração e lista de comportamentos (hormônios) que afetam determinado neurônio ou glândula também compõem esta camada. Cada neurônio ou glândula que possui parâmetros de configuração são relacionados a uma tabela de parâmetros com uma lista de parâmetros específica para cada classe de neurônio ou glândula. Da mesma, forma cada neurô-

nio ou glândula que é afetado por “hormônios” (comportamentos) do domínio a que pertencem é relacionado a uma tabela de comportamentos.

O Sistema de Gerenciamento de Banco de Dados (SGBD) MySQL foi utilizado para gerenciamento das tabelas da camada Ontológica. A comunicação entre o SGBD e a camada Reativa é feita através da interface JDBC (*Java Database Connectivity*) que permite a comunicação entre aplicativos escritos em Java com o MySQL. A linguagem SQL é utilizada para programação das tabelas da camada Ontológica.

5.1.1.A tabela de entidades: *entities*

A tabela de entidades (*entities*) lista as entidades que compõem a camada Reativa e possui os seguintes campos:

- **nome (*name*)**: o nome da entidade. O nome da entidade é uma composição entre um nome de identificação e a hierarquia de domínios à qual a entidade pertence. Desse modo um sensor de movimento localizado no quarto de uma casa terá o seguinte nome: casa.quarto.sensorDeMovimento, isto é, a entidade sensor de movimento pertence ao domínio quarto que pertence ao domínio casa;

- **entidade (*entity*)**: o tipo de entidade, neurônio (*neuron*), glândula (*gland*), *setpoint* ou comportamento (*behavior*);

- **classe (*class*)**: classe específica do neurônio ou da glândula. Quando a entidade é um *setpoint* ou um comportamento (*behavior*), este campo deve conter o nome da glândula associada a este *setpoint* ou comportamento (*behavior*);

- **tabela de comportamentos (*behaviorTable*)**: nome da tabela com lista de comportamentos a que o neurônio ou a glândula respondem. Caso o neurônio ou a glândula não responda a nenhum comportamento, este campo deve ser configurado como *null*;

- **tabela de parâmetros (*parameterTable*)**: nome da tabela com lista de parâmetros do neurônio ou da glândula. Caso o neurônio ou a glândula não utilize parâmetros de configuração específicos este campo deve ser configurado como *null*;

- **método (*method*)**: método utilizado para combinar o comportamento do neurônio ou da glândula quando mais de um hormônio com a mesma prioridade está ativo.

5.1.2. A tabela de conexões: *neurotransmiters*

A tabela de conexões contém as conexões neurônio-neurônio e neurônio-glândula e suas características. Como a comunicação entre neurônios é feita por intermédio de neurotransmissores (GUYTON; HALL, 2006), esta tabela é chamada aqui de neurotransmissores (*neurotransmiters*). Ela segue a descrição dos campos da tabela:

- **identificador de conexão (*bindingID*)**: número único gerado automaticamente que identifica cada conexão existente na camada Reativa;

- **fonte (*source*)**: nome do neurônio fonte (que propaga a informação) da conexão;

- **destino (*destination*)**: nome do neurônio ou da glândula destino (que recebe a informação) da conexão;

- **peso (*weight*)**: peso da conexão (parâmetro de uso opcional);

- **valor padrão (*defaultValue*)**: valor padrão utilizado pela entidade destinado caso a entidade fonte esteja inativa.

5.1.3. Tabelas de comportamento

As tabelas de comportamentos são associadas a neurônios e glândulas específicos, e descrevem como o neurônio ou a glândula deve responder aos comportamentos listados. Os seguintes campos compõem as tabelas de comportamento:

- **hormônio (*hormone*)**: nome do hormônio (comportamento);

- **tipo (*type*)**: binário (*binary*) ou linear. O tipo linear interpreta valores contínuos entre 0 e 1 (*unsigned*) ou -1 e 1 (*signed*). O tipo binário interpreta apenas dois valores 0 e 1 (*unsigned*) ou -1 e 1 (*signed*). Neste último caso, o valor zero também pode ocorrer, e como nos demais casos significa ausência do hormônio (comportamento desativado);

- **sinal (*signal*)**: sem sinal (*unsigned*) ou com sinal (*signed*). Os valores devem ser limitados entre 0 e 1 (*unsigned*) ou entre -1 e 1 (*signed*);

- **inversão (*inversion*)**: inverso (*inverse*) ou direto (*direct*). No primeiro caso, o valor recebido é invertido, multiplicado por -1 (*unsigned*) ou calculado o complemento 1 (*signed*).

- **prioridade (*priority*)**: prioridade do comportamento. A prioridade 1 é a maior e a prioridade decresce com o aumento da numeração. Comportamentos sem prioridade são representados pela prioridade 0.

5.1.4. Tabelas de parâmetro

As tabelas de parâmetro são associadas a neurônios e a glândulas específicos e descrevem parâmetros de configuração do neurônio ou da glândula. Os *setpoints* utilizados pelo neurônio ou pela glândula são configurados nesta tabela. Os seguintes campos compõem as tabelas de parâmetro:

- **identificador (ID)**: número único gerado automaticamente que identifica o parâmetro dentro da tabela;

- **parâmetro (*parameter*)**: nome do parâmetro. Este nome é conhecido no código de programação da classe do neurônio ou da glândula;

- **tipo (*type*)**: tipo de parâmetro (*setpoint*, número, texto ou valor booleano). Este campo especifica como o valor do parâmetro deve ser interpretado;

- **valor (*value*)**: valor do parâmetro. No caso de *setpoint*, o valor é o nome do(s) *setpoint*(*s*) ao(s) qual(is) o neurônio ou a glândula responde.

5.2. Camada Reativa

A camada Reativa da arquitetura proposta deve ter dois tipos de unidades básicas, os neurônios e as glândulas, que são unidades de processamento paralelo de informação e autônomas. Autonomia, neste contexto, relaciona-se à independência de código de cada unidade, isto é, as diferentes unidades são processos computacionais independentes que podem ser processados, inclusive, em diferentes unida-

des computacionais (computadores). Essas características possibilitam a distribuição da tarefa de processamento e facilitam a programação do paralelismo necessário aos neurônios e às glândulas.

A programação da camada Reativa é baseada em CORBA e na linguagem Java. Esta linguagem é orientada a objetos e possui recursos para programação de aplicativos em rede, para acesso a banco de dados, para programação de *threads*[13] e um pacote para programação de aplicativos CORBA (bibliotecas, compilador IDL e ORB). O objetivo é a construção de uma plataforma que facilite a criação de classes de neurônios e de glândulas, deixando transparente ao desenvolvedor os detalhes de implementação da camada Reativa. O foco do desenvolvedor deve ser a aplicação e não a programação dos mecanismos de comunicação e gerenciamento de neurônios e de glândulas.

Nas próximas seções é descrito o arcabouço utilizado para a programação da camada Reativa.

5.2.1. Comunicação CORBA

Neurônios e glândulas se comunicam através de uma interface bem definida composta por dois métodos: *neurotransmiter* e *hormone.* O primeiro recebe mensagens de neurônios enquanto o segundo recebe mensagens de glândulas.

O método *neurotransmiter* possui três parâmetros:

- **ID**, número de identificação da conexão na tabela *neurotransmiters* (na camada Ontológica);

[13] *Thread* é uma seção de código executada independentemente de outras partes de código dentro um mesmo programa (OAKS; WONG, 1997). Permite a criação em um mesmo programa de processos independentes e paralelos.

- **value**, valor da conexão;

- **weight**, peso da conexão;

Em geral, o parâmetro peso (*weight*) propagado por um neurônio está especificado na tabela *neurotransmiters* da camada Ontológica.

O método *hormone* possui dois parâmetros:

- **type**, tipo ou nome do comportamento ou *setpoint* (hormônio);

- **value**, nível ou valor do comportamento ou *setpoint* (hormônio).

Um compilador IDL-Java transforma o código IDL da interface CORBA em uma série de arquivos (classes) Java utilizados como base para a comunicação entre objetos CORBA (nesse caso os objetos *Neuron* e *Gland*). Para tornar transparentes os mecanismos específicos do CORBA, duas classes, uma relacionada aos neurônios e outra relacionada às glândulas, foram criadas: *NeuronCORBA* e *GlandCORBA*. Essas duas classes foram programadas como *threads*, isto é, processos independentes. Quando um objeto dessas classes é instanciado, eles se tornam uma nova unidade processamento. A Figura 28 e a Figura 29 mostram diagramas com os principais métodos das classes *NeuronCORBA* e *GlandCORBA*, respectivamente. As duas classes possuem um construtor no qual o nome do neurônio ou da glândula deve ser passado, assim como argumentos de inicialização CORBA (como nome do servidor ORB e número da porta de comunicação). As classes NeuronCORBA e GlandCORBA possuem métodos (*propagate* e *secrete*, respectivamente) para enviar dados a outros neurônios e glândulas.

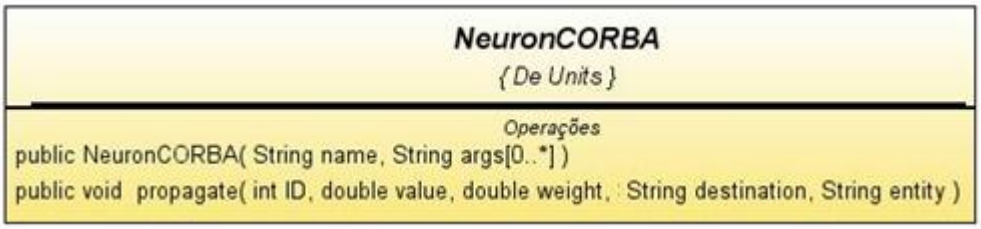

Figura 28 - Diagrama da classe NeuronCORBA.

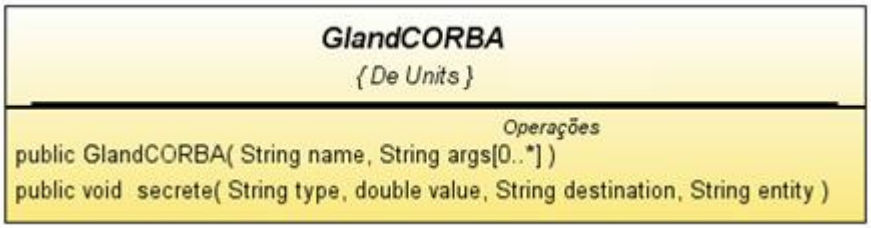

Figura 29 - Diagrama da classe GlandCORBA.

As classes acima são responsáveis pela comunicação, utilizando CORBA, entre as entidades da camada reativa. Dessa forma, as funcionalidades de comunicação são separadas das demais funcionalidades do *software*, o que permite, no futuro, a mudança dos mecanismos de comunicação sem a alteração do restante do sistema.

5.2.2. Gerenciamento das funcionalidades de neurônios e de glândulas

A utilização direta dos métodos *propagate* e *secrete* (das classes NeuronCORBA e GlandCORBA) necessita que sejam passados como parâmetros, informações como nome do destino, tipo de entidade (glândula ou neurônio) e identificador de conexão (somente o método *propagate*). Para que essas informações fiquem transparentes ao desenvolvedor, duas classes, uma para os neurônios (*NeuronClass*) e outra para as glândulas (*GlandClass*), foram criadas. O objetivo dessas classes é gerenciar todos os aspectos envolvidos com o processamento e com a troca de informações entre as entidades da camada Reativa, assim como carregar todos os parâmetros armazenados na camada Ontológica necessários para o correto funcio-

namento do neurônio ou da glândula. Essas classes são filhas das classes *Neuron-CORBA* e *GlandCORBA*, e por isso herdam todas suas funcionalidades.

Cada neurônio e cada glândula recebe informações de outros neurônios e glândulas, seja através de conexões (neurotransmissores) de entrada ou através de comportamentos e *setpoints* (hormônios), aos quais o neurônio ou a glândula responde. A partir das informações recebidas, o estado do neurônio ou da glândula é determinado em função dos dados de entrada. A cada novo valor de entrada, o neurônio ou a glândula recalcula seu estado e propaga-o, o que torna a comunicação na camada Reativa assíncrona.

Para que o estado do neurônio ou da glândula possa ser calculado a qualquer momento, listas contendo os valores de cada conexão (neurotransmissores) de entrada, *setpoints* e comportamentos (hormônios), e parâmetros de configuração devem estar disponíveis. As seguintes classes utilitárias foram construídas para armazenar e gerenciar esses dados:

- **InBindings.** Armazena lista com parâmetros e valores de cada conexão de entrada.

- **InHormones.** Armazena lista de comportamentos aos quais o neurônio ou a glândula responde. Calcula um fator de ponderação, baseado nos valores recebidos pelos comportamentos ativos (com valores diferentes de zero) e suas prioridades, que pode ser usado pelo neurônio ou pela glândula para ponderar seu estado. Cinco métodos foram implementados para combinar comportamentos ativos com o mesmo grau de prioridade: *last* (utiliza o último valor recebido), *first* (utiliza o primeiro valor recebido), *mean* (utiliza a média de valores recebidos), *higher* (utiliza o maior valor recebido) e *smaller*

(utiliza o menor valor recebido). A maneira como cada comportamento deve ser interpretado é determinada pelos parâmetros de configuração contidos na tabela de comportamento, do neurônio ou da glândula, localizada na camada Ontológica.

- **Parameters.** Armazena lista de parâmetros de configuração e a lista de *set-points* aos quais o neurônio ou a glândula responde.

A cada atualização do estado do neurônio ou da glândula, o novo valor deve ser propagado. Para gerenciar as conexões (neurotransmissor) de saída dos neurônios e a saída (hormônio) das glândulas, foram criadas respectivamente duas classes utilitárias:

- **OutBindings.** Armazena lista com parâmetros, valor e entidade destino de cada conexão de saída. Esta classe é utilizada somente por neurônios.

- **OutHormones.** Armazena lista com nome de todas as entidades pertencentes ao mesmo domínio que a glândula. Esta classe é utilizada somente por glândulas.

As classes NeuronClass e GlandClass, utilizando as classes descritas acima, são responsáveis por todo processo de comunicação da camada Reativa em consonância com os parâmetros contidos na Camada Ontológica. Essas duas classes são abstratas, isto é, não podem ser utilizadas diretamente ou não podem ter objetos instanciados. Elas são as bases para que classes específicas de neurônios e de glândulas possam ser criadas através do processo de herança. Quatro métodos abstratos devem ser implementados nas classes filhas:

- **value( ).** Este método deve retornar o valor que será propagado pelo neurônio ou pela glândula. O desenvolvedor deve utilizar os dados de entrada para calcular o valor que será propagado. Toda vez que um novo valor for recebido, este método é chamado e o valor calculado é propagado.

- **neurotransmiter( ).** Este método é executado toda vez que o neurônio ou a glândula recebe valores em suas conexões de entrada. Pode ser utilizado para determinar a execução de alguma tarefa quando o neurônio ou glândula recebe valores pelas conexões de entrada.

- **hormone( ).** Este método é chamado toda vez que o neurônio ou a glândula recebe hormônios (comportamentos ou *setpoints*). Pode ser utilizado para determinar a execução de alguma tarefa quando o neurônio ou glândula recebe hormônios (comportamentos ou *setpoints*).

- **sent( ).** Este método é chamado toda vez que o neurônio ou a glândula propagou alguma informação. Pode ser utilizado para determinar a execução de alguma tarefa quando o neurônio ou glândula propaga alguma informação.

A Figura 30 e a Figura 31 mostram diagramas com os métodos das classes *NeuronClass* e *GlandClass*, respectivamente descritos acima. As duas classes possuem um construtor no qual o nome do neurônio ou da glândula deve ser passado, assim como argumentos de inicialização CORBA (como nome do servidor ORB e número da porta de comunicação) e de acesso a camada Ontológica.

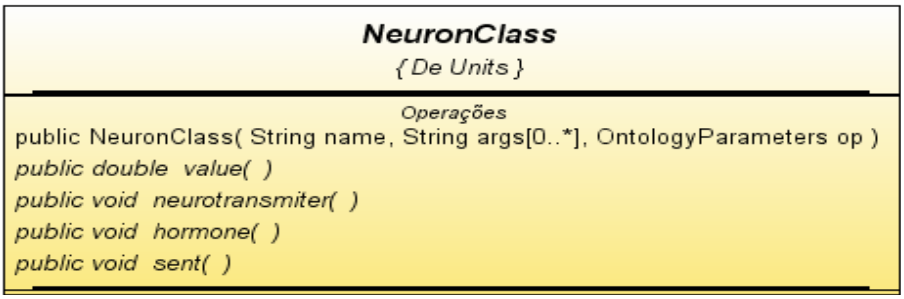

Figura 30 - Diagrama da classe NeuronClass com os métodos abstratos.

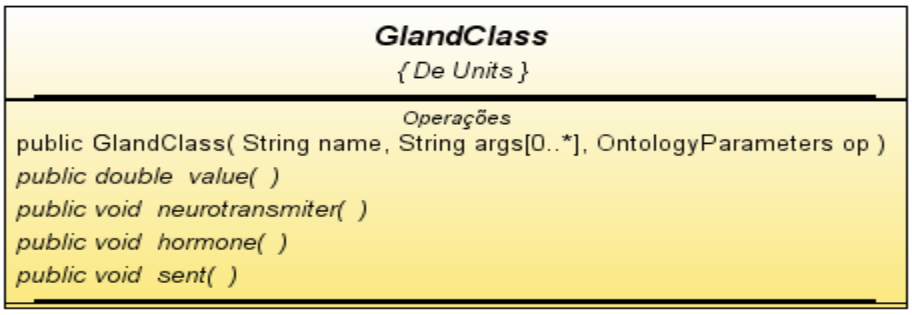

Figura 31 - Diagrama da classe GlandClass com os métodos abstratos.

Além dos métodos abstratos já descritos, as classes *NeuronClass* e *Gland-Class* possuem um conjunto de métodos utilitários que podem ser utilizados pelo desenvolvedor para criação de novas classes de neurônios ou de glândulas:

- **getInBindings( ).** Retorna a lista com as conexões de entrada.
- **getInHormones( ).** Retorna a lista com os hormônios recebidos.
- **getParameters( ).** Retorna a lista com os parâmetros.
- **getInBindingLastValue( ).** Retorna o último valor recebido pelo neurônio ou pela glândula através das conexões (neurotransmissores) de entrada.
- **getInBindingLastWeight( ).** Retorna o último peso recebido pelo neurônio ou pela glândula através das conexões (neurotransmissores) de entrada.
- **getInBindingHigherValue( ).** Retorna o maior valor entre as conexões (neu-rotransmissores) de entrada.

- **getInBindingSmallerValue( ).** Retorna o menor valor entre as conexões (neurotransmissores) de entrada.

- **getWeight( ).** Retorna o fator de ponderação dos hormônios ativados.

- **getSetpoint(setpointName).** Retorna o último valor de *setpoint* recebido.

- **getDoubleParameter(parameterName).** Retorna valor numérico do parâmetro de configuração com o nome *parameterName*.

- **getStringParameter(parameterName).** Retorna valor tipo texto do parâmetro de configuração com o nome *parameterName*.

- **canNotPropagate( ).** Impede que o próximo valor seja propagado.

A Figura 32 e a Figura 33 mostram diagramas com os métodos utilitários descritos acima das classes *NeuronClass* e *GlandClass*, respectivamente.

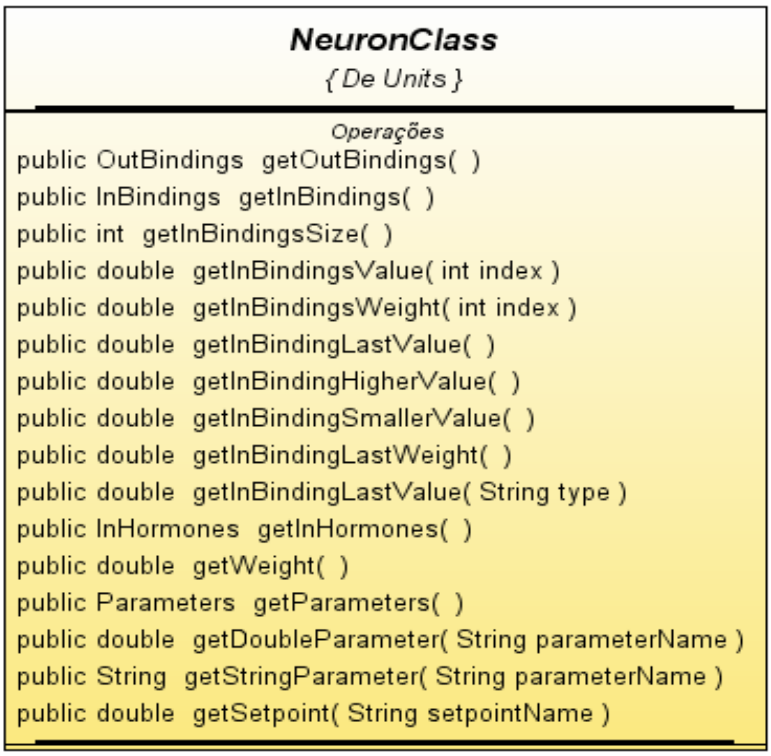

Figura 32 - Diagrama da classe NeuronClass com os métodos utilitários.

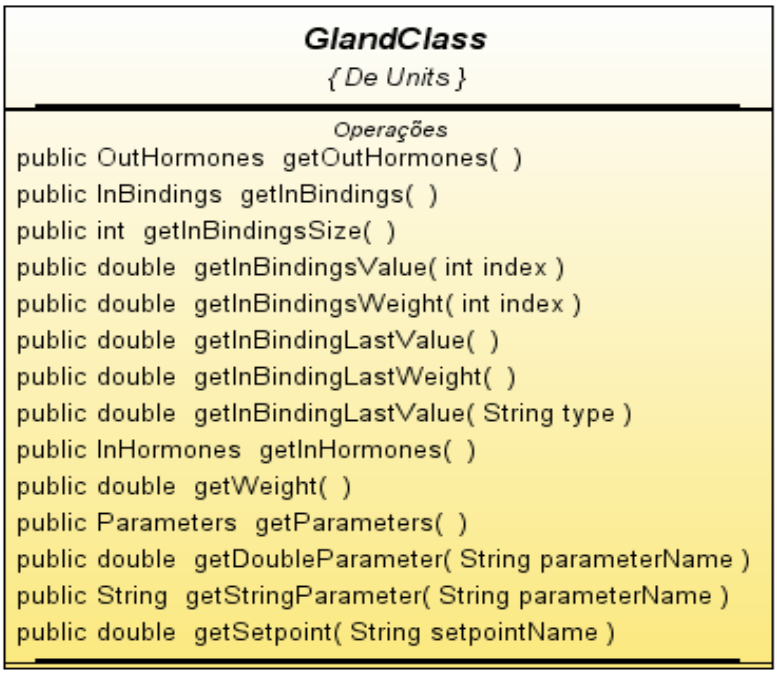

Figura 33 - Diagrama da classe GlandClass com os métodos utilitários.

5.2.3. Classes de Neurônios e de Glândulas

Todo o arcabouço descrito nas seções anteriores tem por objetivo simplificar o desenvolvimento de novas classes de neurônios e de glândulas. Os mecanismos relacionados com a comunicação e consulta à camada Ontológica foram programados nas classes descritas anteriormente. Dois *templates* (modelos), um de neurônio e outro de glândula, foram criados. Os dois são bastante similares e por isso será descrito aqui o *template* de neurônio. Segue abaixo o código Java inicial do *template* de neurônio.

```
public class Neuron extends NeuronClass implements NeuronInterface {

   public Neuron(String name,
            String args[],
            OntologyParameters op) throws SQLException {

      super(name, args, op);
   }
   @Override
   public double value() {
      // calculate the state value of neuron and return it
      return (getInBindingLastValue()*getInBindingLastWeight()*getWeight());
   }
```

```
    @Override
    public void neurotransmiter() {
        // insert here the code to be executed
        // when a neurotransmiter is received
    }
    @Override
    public void hormone() {
        // insert here the code to be executed
        // when a hormone is received
    }
    @Override
    public void sent() {
        // insert here the code to be executed
        // when a value is propagated
    }
}
```

A classe de neurônio criada acima tem o nome Neuron e é filha da classe NeuronClass, isto é, herda todas as suas funcionalidades. Quatro métodos da classe NeuronClass devem obrigatoriamente fazer parte da nova classe de neurônio. Um deles é fundamental, o método **value( )**. O valor calculado neste método será aquele propagado pelo neurônio. No exemplo, o neurônio propagará o produto entre o último valor recebido (getInBindingLastValue( )), o último peso recebido (getInBindingLastWeight( )) e o fator de ponderação dos hormônios ativados (getWeight( )).

Os demais métodos devem ser utilizados em casos especiais onde é necessário que o neurônio execute ações no recebimento de um novo valor em uma conexão de entrada (*neurotransmiter( )*) ou um novo valor de hormônio (*hormone( )*) ou após a propagação de um novo valor (*sent( )*).

Um neurônio, instância da classe *Neuron* exemplificada acima, propagará o valor calculado em *value( )* sempre que um novo valor (neurotransmissor) ou hormônio (*setpoint* ou comportamento) for recebido.

5.3. Considerações finais

A implementação da arquitetura proposta descrita neste capítulo buscou construir um arcabouço que permitisse o desenvolvimento de sistemas de automação do ambiente construído baseados em neurônios e em glândulas. Estes últimos foram programados na linguagem Java como *threads* (processos computacionais independentes) e utilizam o middleware CORBA como plataforma de comunicação. Eles são os componentes básicos da camada Reativa na arquitetura proposta.

A forma de comunicação, os parâmetros de configuração e as relações entre os comportamentos de neurônios e de glândulas e o comportamento geral do sistema são descritas em forma de tabelas relacionais partes de um banco de dados. O Sistema de Gerenciamento de Banco de Dados (SGBD) MySQL foi utilizado na implementação apresentada. Esse conjunto de tabelas forma a camada Ontológica da arquitetura proposta.

Classes de neurônios e de glândulas são criadas a partir de *templates* (modelos) Java (descrito na seção 5.2.3). A partir de um conjunto de classes, um aplicativo de automação do ambiente construído é programado através da configuração de um conjunto de tabelas na camada Ontológica que descrevem a estrutura, as conexões e o comportamento das unidades básicas da arquitetura, neurônios e glândulas. O desenvolvedor mantém o foco na aplicação, pois os mecanismos de funcionamento interno e de comunicação de neurônios e de glândulas ficam transparentes durante a programação do sistema.

6. Aplicação da Arquitetura Proposta

Este capítulo ilustra o desenvolvimento de um aplicativo para automação do ambiente construído orientado a neurônios e a glândulas. Um sistema de automação de uma casa fictícia é desenvolvido em oito etapas. A comunicação com dispositivos físicos em um ambiente construído real também é descrita. O capítulo é encerrado com considerações sobre a aplicação da arquitetura proposta.

6.1. Construindo o Sistema de Automação de uma Casa

Para ilustrar a aplicação da arquitetura proposta na automação de um ambiente construído, o sistema de automação de uma casa fictícia é modelado como um conjunto de neurônios e de glândulas. O objetivo é demonstrar como as características de modularidade, de flexibilidade e de capacidade de integração entre partes são alcançadas com a utilização da arquitetura proposta. São exemplificados sistemas de controle e de reconhecimento de contexto de um quarto (foco principal) e de um banheiro da casa. A modelagem do sistema é feita em partes, simulando a implantação de um sistema de automação construído em etapas distintas, e que deverá trabalhar de forma integrada. São oito etapas independentes de implantação descritas a seguir.

6.1.1. Etapa 1 – Sistema de Iluminação do Quarto

O sistema de iluminação do quarto deve regular a intensidade da iluminação fornecida pela luminária de acordo com o *setpoint* escolhido pelo usuário através de um controle manual deslizante. O usuário deve ser capaz de desligar a iluminação quando desejar através de um interruptor. Na ausência de pessoas, a iluminação

deve ser desativada para evitar desperdício de energia elétrica. Cinco elementos de interface entre o sistema e o mundo físico são identificados na descrição anterior:

- Um **controle manual deslizante**. Informa ao sistema o valor do *setpoint* desejado.
- Um **sensor de movimento**. Informa ao sistema a ausência de movimento.
- Um **interruptor de luz**. Informa ao sistema quando o usuário deseja que a iluminação seja desligada ou ligada.
- Um **sensor de luz**. Captura a iluminância do quarto.
- Uma **luminária com intensidade regulável**. Ilumina o quarto com intensidade de luz controlada entre 0 e 100%.

Uma malha de controle com um controlador PID (Proporcional-Integral-Derivaitvo) é escolhida para regular a iluminância do quarto (ver Figura 34).

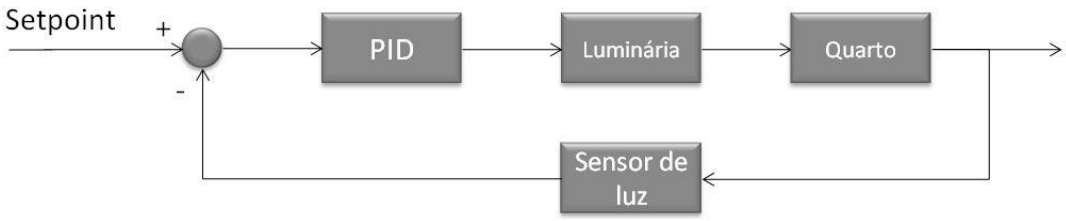

Figura 34 - Malha de controle da iluminação do quarto.

Dois comportamentos são identificados a partir da descrição inicial:

- **Ausência**. Comportamento é ativado quando há ausência de pessoas no quarto.
- **Desativação da Iluminação**. Comportamento é ativado quando o usuário desativa a iluminação.

Cada um dos comportamentos deve ser ativado de acordo com contexto. O passo seguinte é determinar quais contextos estão relacionados a quais comportamentos e como estes contextos são detectados:

- **Ausência**. A ausência de pessoas no quarto é detectada através de um sensor de movimento. Habitualmente é necessária a utilização de um tempo de retardo. Esse tempo é medido entre a detecção de um sinal de "não movimento" e a propagação de um sinal de ausência, já que a não movimentação de uma pessoa por um período curto de tempo não significa a sua ausência (ver Figura 35).

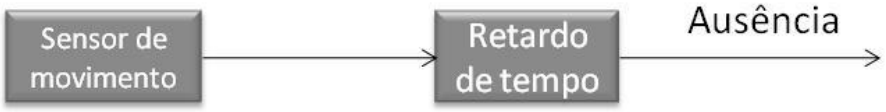

Figura 35 - Detecção de ausência

- **Desativação da Iluminação**. A desativação da iluminação é detectada quando o usuário aciona o interruptor de luz na posição "desligado".

A partir da descrição e da análise anteriores, é possível modelar o sistema como um conjunto de neurônios e de glândulas. Os elementos funcionais (sensores, controladores, atuadores, etc.) são modelados como neurônios. Os comportamentos e os *setpoints* são modelados como glândulas. A Figura 36 mostra o diagrama de relacionamento com os neurônios e as glândulas que compõem o sistema. O *setpoint* de Luz foi modelado como uma glândula que recebe informação diretamente do controle manual deslizante de iluminação. É importante notar que o neurônio PID Iluminação foi modelado com três receptores de hormônio que recebem hormônios secretados pelas glândulas Setpoint de Luz, Ausência e Desativação da Iluminação.

Desta forma este neurônio responde aos comportamentos “Ausência” e “Desativação da Iluminação”.

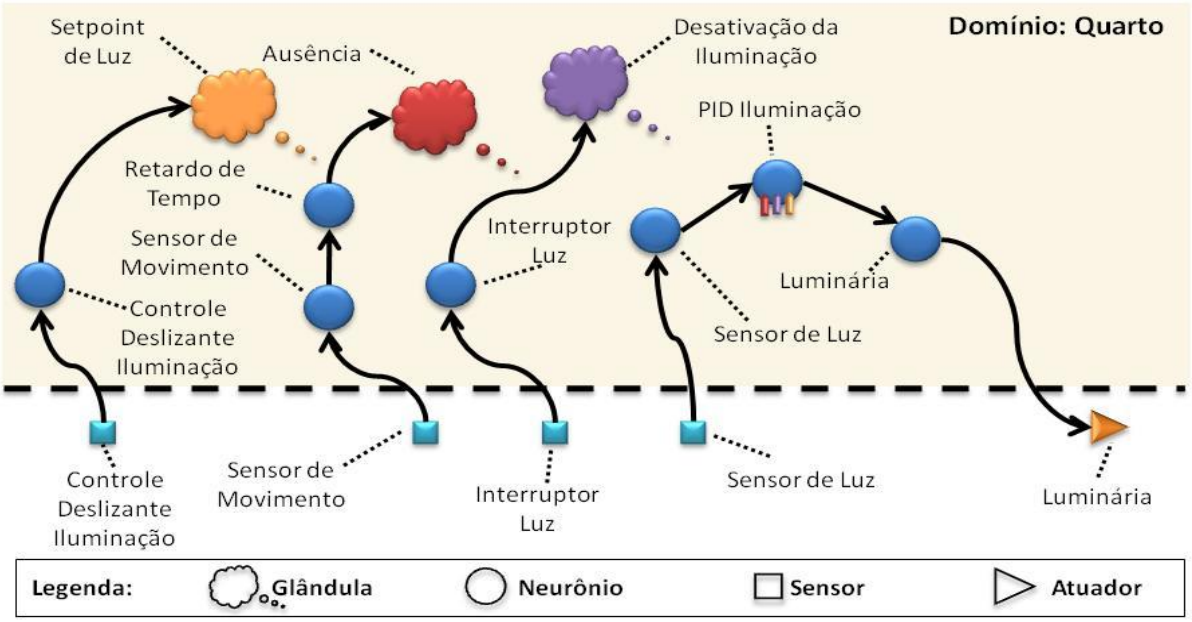

Figura 36 – Etapa 1: Diagrama de Relacionamento do Sistema de Iluminação do Quarto.

Neste momento cabe definir como serão configurados os parâmetros de configuração de cada neurônio/glândula, os pesos de cada ligação e o mecanismo de combinação de comportamentos em cada neurônio/glândula. No sistema modelado, não há necessidade de ponderação ou de prioridade nas ligações entre neurônio/glândulas, portanto todos os pesos são configurados com valor 1 (um). O mecanismo de combinação de comportamentos que define como o neurônio/glândula deve reagir quando comportamentos com mesma prioridade estão ativos será configurado, para todos os neurônios/glândulas do sistema, de tal maneira que a prioridade será dada ao comportamento com maior prioridade ativado por último. Os únicos dois neurônios que possuem parâmetros de configuração são os neurônios Retardo de Tempo e PID Iluminação. A Figura 37 mostra os parâmetros de configuração para estes dois neurônios.

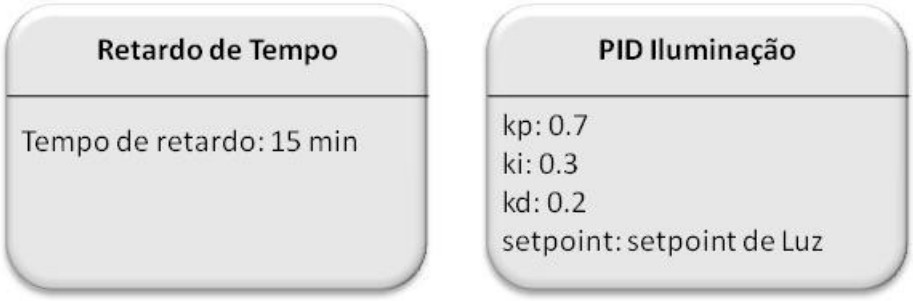

Figura 37 - Parâmetros de configuração: Reatrdo de Tempo e PID Iluminação.

O passo final é a programação do sistema modelado em software utilizando a implementação da arquitetura proposta descrita no capítulo anterior. Tal tarefa requer que todos os neurônios e todas as glândulas, seus parâmetros de configuração e as ligações entre neurônios/glândulas sejam descritos nas tabelas da camada Ontológica. Abaixo seguem as tabelas da camada Ontológica preenchidas com os parâmetros descritos anteriormente. Todos os neurônios e as glândulas do sistema pertencem ao domínio Quarto que por sua vez pertence ao domínio Casa. Os nomes de neurônios/glândulas utilizados nas tabelas são os mesmos utilizados nos diagramas anteriores com justaposição sem preposições para os nomes compostos e sem acentuação. O Quadro 3 ilustra a tabela "*entities*" com os neurônios, as glândulas, os setpoints e os comportamentos do sistema. O Quadro 4 contém a lista de ligações entre neurônios/glândulas, a tabela "*neurotransmiters*". Os Quadro 5 e Quadro 7 contêm as tabelas com os parâmetros de configuração dos neurônios PID de Iluminação e Retardo de Tempo, respectivamente. O Quadro 6 contém a tabela de comportamentos do neurônio PID de Iluminação.

Na implementação descrita no capítulo anterior, as tabelas da camada Ontológica fazem parte do banco de dados MySQL. Elas devem ser criadas e preenchidas utilizando a linguagem SQL ou alguma ferramenta equivalente. Assume-se que as classes de neurônios/glândulas utilizadas para esta aplicação já estão programadas.

Caso novas classes de neurônios/glândulas necessitem ser criadas, um *template* (modelo) de neurônio/glândula em Java deve ser utilizado (descrito na seção 5.2.3).

Name	Entity	Class	BehaviorTable	ParameterTable	Method
Casa.Quarto.SetpointLuz	Gland	Gland	Null	Null	Last
hormonioSetpointLuz	Setpoint	Casa.Quarto.SetpointLuz	Null	Null	Last
Casa.Quarto.Ausencia	Gland	Gland	Null	Null	Last
hormonioAusencia	Behavior	Casa.Quarto.Ausencia	Null	Null	Last
Casa.Quarto.DesativacaoIluminacao	Gland	Gland	Null	Null	Last
hormonioDesativacaoIluminacao	Behavior	Casa.Quarto.Desativacao Iluminacao	Null	Null	Last
Casa.Quarto.ControleDeslizanteIluminacao	Neuron	Neuron	Null	Null	Last
Casa.Quarto.SensorMovimento	Neuron	Neuron	Null	Null	Last
Casa.Quarto.RetardoTempo	Neuron	Neuron	Null	ParamRetardoTempo	Last
Casa.Quarto.InterruptorLuz	Neuron	Neuron	Null	Null	Last
Casa.Quarto.SensorLuz	Neuron	Neuron	Null	Null	Last
Casa.Quarto.PIDIluminacao	Neuron	PID	CompPIDIlum	ParamPIDIlum	Last
Casa.Quarto.Luminaria	Neuron	Neuron	Null	Null	Last

Quadro 3- Tabela "*entities*"

bindindID	source	destination	weight	default
1	Casa.Quarto.ControleDeslizanteIluminacao	Casa.Quarto.SetpointLuz	1	0
2	Casa.Quarto.SensorMovimento	Casa.Quarto.RetardoTempo	1	0
3	Casa.Quarto.InterruptorLuz	Casa.Quarto.DesativacaoIluminacao	1	0
4	Casa.Quarto.SensorLuz	Casa.Quarto.PIDIluminacao	1	0
5	Casa.Quarto.PIDIluminacao	Casa.Quarto.Luminaria	1	0

Quadro 4 – Tabela "*neurotransmiters*"

Parâmetro	Tipo	Valor
kp	double	0.7
ki	double	0.3
kd	double	0.2
setpoint	setpoint	setpointLuz

Quadro 5 – Tabela ParamPIDIlum

Hormone	Type	Signal	Inversion	Priority
hormonioAusencia	binary	unsigned	inverse	1
hormonioDesativacaoIluminacao	binary	unsigned	inverse	1

Quadro 6 – Tabela CompPIDIlum

Parâmetro	Tipo	Valor
delay	double	15

Quadro 7 - Tabela ParamRetardoTempo

6.1.2. Etapa 2 – Sistema de HVAC do Quarto

O sistema de HVAC (*Heating, Ventilation and Air Conditioning* – Aquecimento, Ventilação e Ar-condicionado) do quarto deve regular a temperatura do quarto de acordo com o *setpoint* escolhido pelo usuário através de um controle manual deslizante. O usuário deve ser capaz de desligar o sistema de HVAC quando desejar através de um interruptor. Na ausência de pessoas, o sistema também deverá ser desligado para evitar desperdício de energia elétrica.

O novo sistema tem requisitos similares ao do sistema de iluminação descrito na etapa anterior. A diferença está na variável controlada, temperatura ao invés de iluminância. O mesmo sistema de detecção de ausência será utilizado pelos dois sistemas. Os mesmos tipos de neurônios/glândulas utilizados na etapa anterior podem agora ser adaptados para o sistema de HVAC. A Figura 38 ilustra um diagrama

com neurônios/glândulas do sistema de HVAC do quarto. Os neurônios/glândulas do sistema de detecção de ausência da Figura 37 foram repetidos na Figura 38.

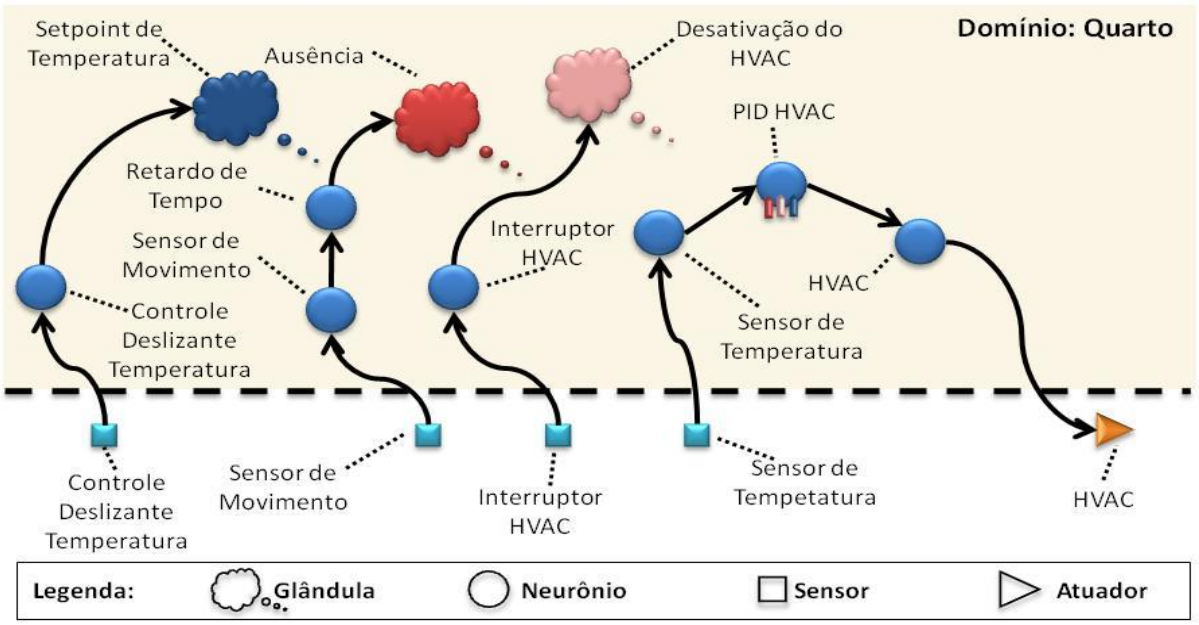

Figura 38 - Etapa 2: Diagrama de Relacionamento do Sistema de HVAC do Quarto.

A programação dos novos neurônios/glândulas deve ser feita acrescentando-se às tabelas da camada Ontológica descritas na etapa anterior, os novos neurônios/glândulas, seus parâmetros de configuração e as ligações entre neurônios/glândulas.

6.1.3. Etapa 3 – Detecção de Ausência no Quarto

O sistema de detecção de ausência usado anteriormente utiliza um sensor de movimento para detectar a ausência. Para melhorar a qualidade da detecção de ausência, um novo sensor é acrescentado ao sistema. Os dois sensores são combinados através de um neurônio do tipo “OU” que propagará o sinal do sensor com maior valor, isto é, o do sensor que estiver detectando movimento dentro do intervalo determinado pelo neurônio “Retardo de tempo”. A Figura 39 ilustra o acréscimo do sensor ao sistema de detecção de ausência.

Outros dois sensores também podem colaborar para melhorar a detecção de ausência no quarto: um sensor de variação de pressão instalado na cama e outro instalado em um tapete localizado em regiões onde há pouca movimentação como numa escrivaninha ou numa cômoda. O acréscimo de mais dois sensores é feito com a inserção de mais dois neurônios que representam os sensores, e mais dois neurônios “Retardo de Tempo” como no caso anterior. A Figura 40 ilustra o acréscimo dos novos sensores ao sistema de detecção de ausência.

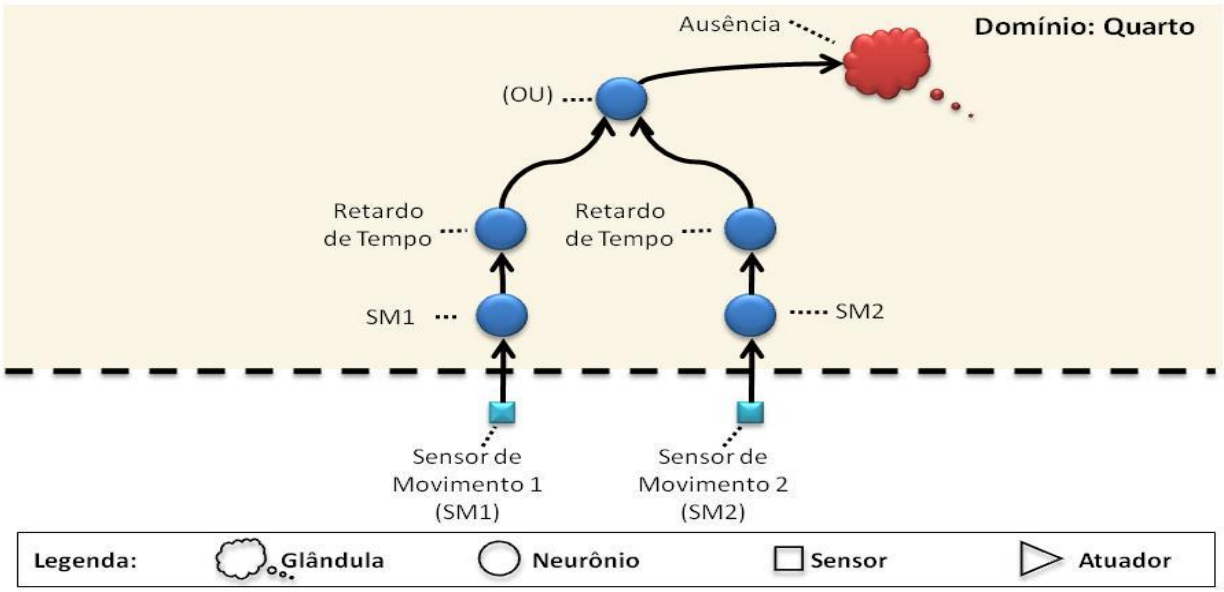

Figura 39 – Etapa 3: Diagrama de Relacionamento do Sistema de Detecção de Ausência.

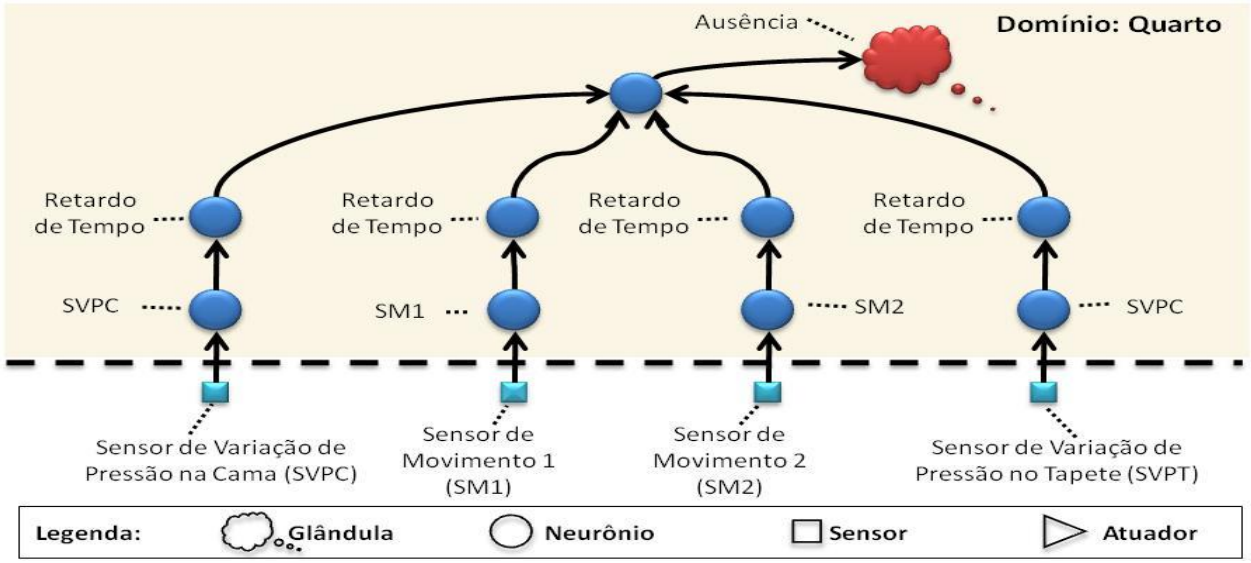

Figura 40 – Etapa 3: Diagrama de Relacionamento do Sistema de Detecção de Ausência com acréscimo de mais dois sensores de variação de pressão.

6.1.4. Etapa 4 – Sistema de *Setpoint* Personalizado

O sistema de reconhecimento de usuários deve reconhecer através de um receptor de RFID (*Radio Frequency Identification*) o usuário A e o usuário B no quarto. Os usuários devem portar um transmissor (*tag*) RFID para serem identificados. A identificação do usuário deve ser usada para personalizar os *setpoints* de temperatura e de luz.

Um receptor de RFID e um neurônio representando o sensor são acrescentados ao sistema. Para detectar o usuário A e o usuário B, dois neurônios que comparam o número de identificação enviado pelo receptor com o peso da conexão (neurotransmissor) são utilizados. A entrada do Usuário A e do Usuário B no quarto deve alterar o comportamento de alguns subsistemas do quarto, por isso são modelados como glândulas.

Um neurônio "*Setpoint* de Iluminação" é acrescentado entre o neurônio "Controle Deslizante de Iluminação" e a glândula "*Setpoint* de Luz". Este último possui receptores para os "hormônios" "Usuário A", "Usuário B" e "Ausência". Na tabela de tabela de parâmetros devem ser registrados "hormônios" "Usuários A" e "Usuário B" com seus respectivos valores de *setpoint*. O valor propagado pelo "Controle Deslizante de Iluminação" possui prioridade maior que os parâmetros tabelados, uma vez modificado o primeiro, ele tem precedência sobre os últimos até que seja detectada "Ausência". Este último "hormônio" deve ser configurado para tal na tabela de parâmetros. O mesmo acréscimo deve ser feito com o *setpoint* de temperatura. A Figura 41 ilustra um diagrama do Sistema de *Setpoint* Personalizado.

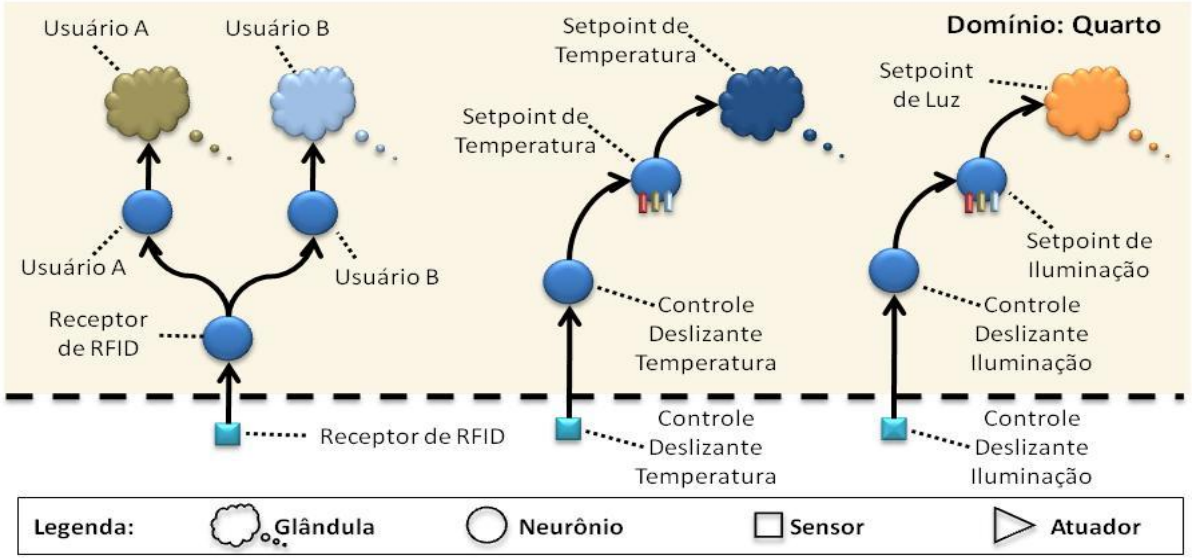

Figura 41 – Etapa 4: Diagrama de Relacionamento do Sistema de *Setpoint* Personalizado.

6.1.5. Etapa 5 – Sistema para Aproveitamento da Luz Solar

Para melhorar a utilização da luz solar através da janela, um aparato de luz deve ser instalado externamente a janela. A abertura do aparato deve ser constantemente ajustada procurando aproveitar a luz solar sem deixar que a carga térmica aumente demasiadamente. Para tal tarefa recomenda-se a adaptação do controlador difuso usado por Guillemin (2003). O controlador necessita de dados ambientais externos à casa: a iluminância vertical e a média das temperaturas das últimas 24 horas. O ângulo de abertura do aparato de luz é calculado em função calculado em função destas variáveis. A temperatura média é "*fuzzificada*" em dois conjuntos difusos: verão e inverno. E a iluminância vertical em quatro conjuntos difusos: noite, baixa, média, alta. Oito regras difusas combinam a temperatura média e a iluminância vertical. Para cada regra há um valor particular de abertura. As oito regras são combinadas ao final através do método centro de gravidade (COG – *Center Of Gravity*).

Três elementos físicos são necessários para implantação do sistema descrito acima: um sensor de iluminância vertical externa, um sensor de temperatura externa e um aparato de luz controlado eletronicamente. Três neurônios representam cada

um destes três elementos na camada Reativa. Como a temperatura e a iluminância não são grandezas particulares do quarto, os neurônios relacionados a estas grandezas pertencem ao domínio “casa”. A temperatura média das últimas 24 horas é calculada por um neurônio (específico para cálculo de médias). Dois domínios especiais foram criados para agrupar os conjuntos difusos relacionados à média de temperatura e a iluminância, Estação e Iluminância, respectivamente. A Figura 42 ilustra um diagrama do Sistema para Aproveitamento da Luz Solar.

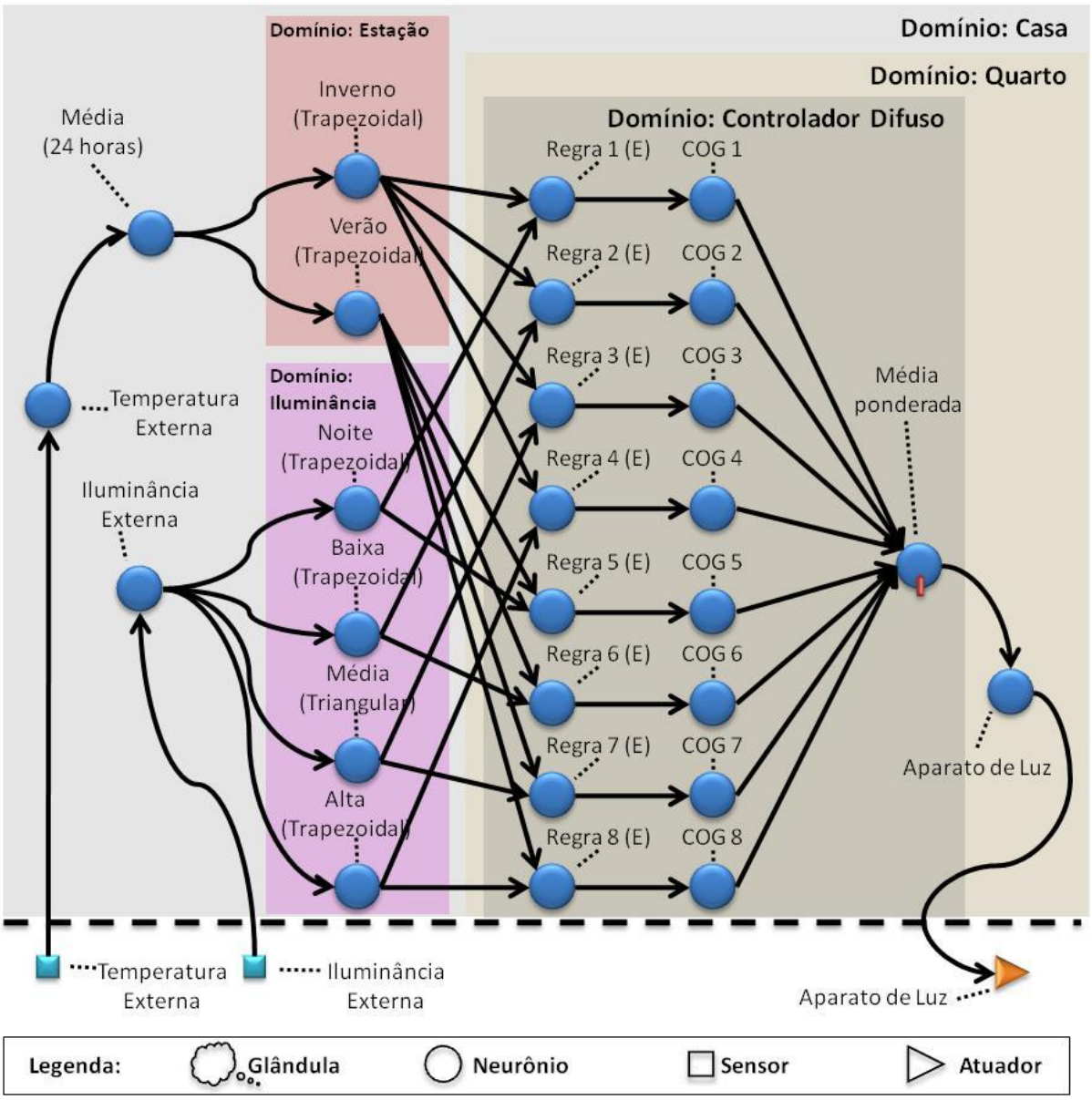

Figura 42 – Etapa 5: Diagrama de Relacionamento do Sistema para Aproveitamento da Luz Solar.

Para o controlador difuso foi também criado um terceiro domínio. Este domínio contém um conjunto de oito neurônios representando as oito regras difusas, um segundo conjunto difuso de oito neurônios *defuzificam* a saída de cada regra usando o método COG (o peso corresponde a área abaixo do grau de pertinência recebido da regra difusa) e um neurônio que calcula a média ponderada pelos pesos das oito saídas *defuzzificadas*. Este último neurônio propaga o valor de abertura para o aparato de luz.

6.1.6.Etapa 6 – Sistema de Controle para o Chuveiro

O sistema de controle para o chuveiro deve ajustar a temperatura da água do banho de acordo com a época do ano ou de acordo com o desejo do usuário reportado através de um controle deslizante.

Para ajustar a temperatura de acordo com a estação do ano, os conjuntos difusos “Estação” (Inverno e Verão) utilizados na etapa anterior (Sistema para Aproveitamento da Luz Solar) serão reutilizados. Os neurônios “Inverno” e “Verão” alimentarão duas glândulas “Inverno” e “Verão”.

Um controle deslizante para o chuveiro, um sensor de temperatura da água e um chuveiro com controle de temperatura são instalados no banheiro e são representados por três neurônios na camada Reativa: “Controle Deslizante Chuveiro”, “Sensor de Temperatura da Água” e “Chuveiro”, respectivamente.

Um neurônio “*Setpoint* de Temperatura da Água” calcula o *setpoint* para a água do banho e propaga-o para a glândula “*Setpoint* de Temperatura da Água”. O neurônio possui receptores para os “hormônios” “Inverno”, “Verão” e “Ausência”. Na tabela de parâmetros devem ser registrados os “hormônios” “Inverno” e “Verão” com seus

respectivos valores de *setpoint*. O valor propagado pelo “Controle Deslizante Chuveiro” possui prioridade sobre os parâmetros tabelados; uma vez modificado o primeiro tem precedência sobre os últimos até que seja detectada “Ausência”. Este último “hormônio” deve ser configurado para tal na tabela de parâmetros.

Similarmente aos sistemas de Iluminação e de HVAC, o sistema de controle do chuveiro utiliza um neurônio PID para controlar a temperatura da água. A Figura 43 ilustra um diagrama do Sistema de Controle para o Chuveiro.

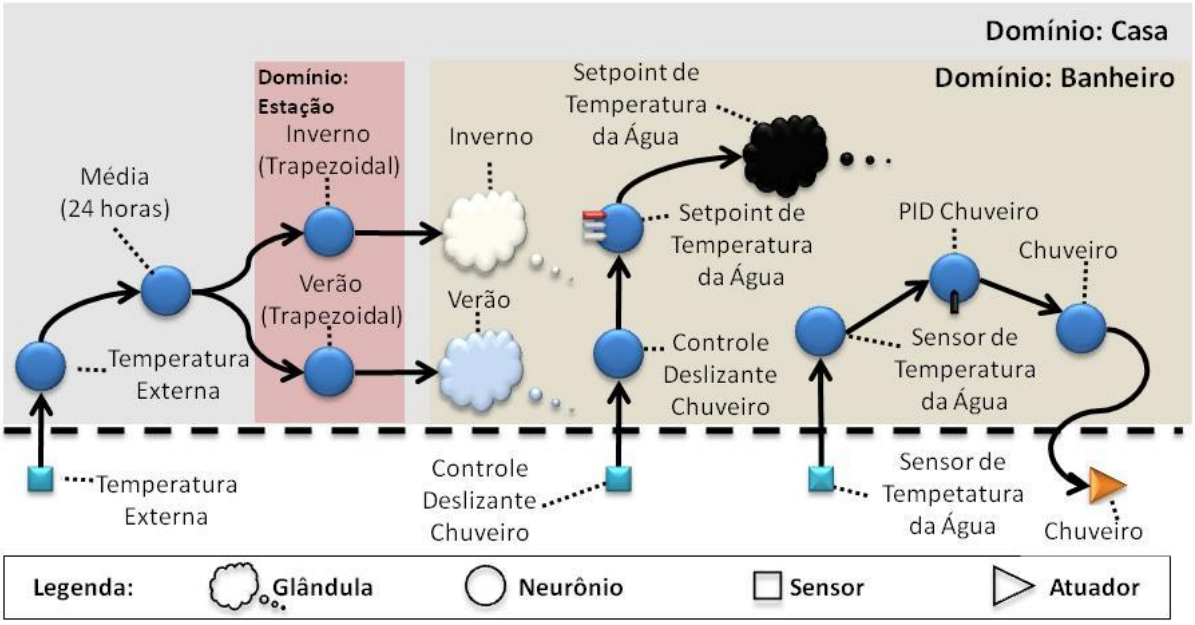

Figura 43 – Etapa 6: Diagrama de Relacionamento do Sistema de Controle para o Chuveiro.

6.1.7.Etapa 7 – Mais comportamentos para o quarto

Duas novas funcionalidades deverão ser acrescentadas ao sistema de automação do quarto: a televisão deve ser programada para que seu volume seja diminuído quando o telefone estiver em uso, evitando assim que o som da televisão interfira na conversa ao telefone; o telefone não deverá tocar e a televisão deverá ser desligada quando o usuário estiver dormindo durante a noite.

Para que as duas funcionalidades acima possam ser desenvolvidas é necessário que a televisão e o telefone troquem informações através da camada Reativa. A

televisão é um aplicativo que contém um neurônio, “TV”, que recebe informações trocadas na camada Reativa. O mesmo acontece com o telefone. Esses dois neurônios propagam o estado da televisão (ligada ou não) e o estado do telefone (em uso ou não) para duas glândulas, “TV Ligada” e “Telefone em Uso”, respectivamente.

O comportamento “Dormindo” será a composição de duas informações, está escuro (é noite) e há pessoas na cama. Esses dados são capturados por um sensor de luz (o mesmo utilizado no controle de iluminância do quarto) e um sensor de variação de pressão na cama (o mesmo utilizado para detectar ausência). Um neurônio, “Escuro”, determina se está escuro ou não (função difusa trapezoidal). Essas duas informações são combinadas através de um neurônio “E”, que propaga o maior valor recebido em suas conexões de entrada para a glândula “Dormindo”.

O aplicativo TV responderá aos “hormônios” “Telefone em Uso” e “Dormindo” diminuindo o som da televisão e desligando-a respectivamente. O aplicativo Telefone responderá ao “hormônio” “Dormindo” não permitindo que o telefone toque. A Figura 45 ilustra um diagrama dos comportamentos descritos acima.

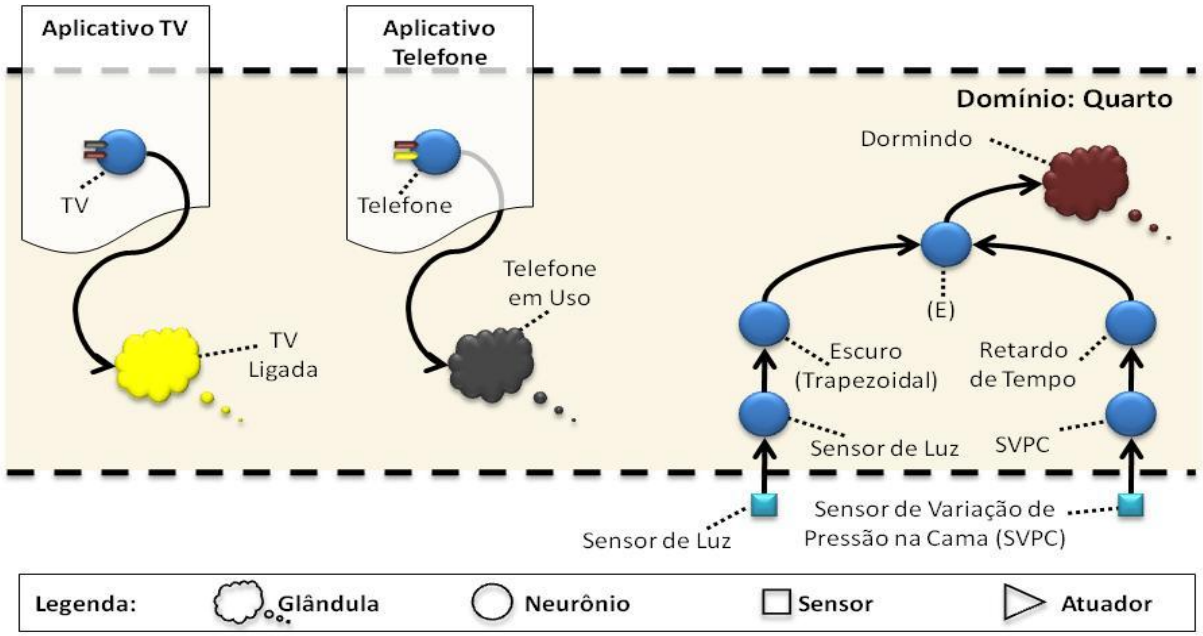

Figura 44 – Etapa 7: Diagrama de Relacionamento de mais Comportamentos para o Quarto.

6.1.8. Etapa 8 – Mais Comportamentos para a Casa

Comportamentos mais genéricos que afetam a casa inteira também podem ser acrescentados, como por exemplo:

- Bloqueio por senha de acesso. O comportamento bloqueio é ativado quando um teclado de acesso não é desativado através de uma senha de acesso.
- Vazamento de gás. Este comportamento é ativado quando é detectado vazamento de gás na cozinha.
- Correspondência. Este comportamento é ativado quando há correspondência na caixa de correio.

A Figura 45 ilustra um diagrama dos comportamentos descritos acima.

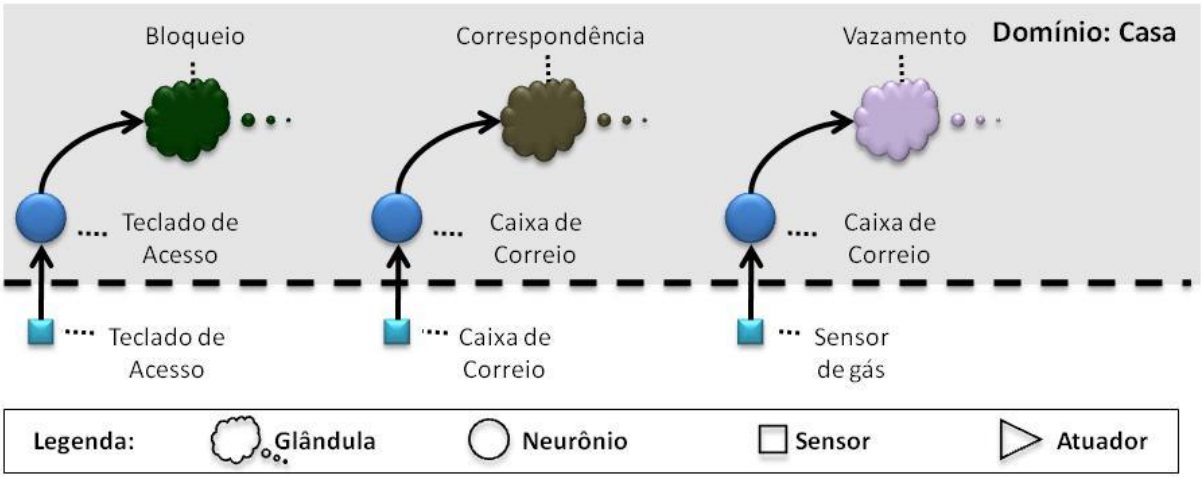

Figura 45 – Etapa 8: Diagrama de Relacionamento de mais Comportamentos para a Casa.

6.2. Comunicação com Dispositivos Físicos

Durante um mês, um aplicativo baseado na implementação da arquitetura proposta foi posto para funcionar em conjunto com uma rede de sensores existente. Esta última interliga sensores localizados em dois quartos em um ambiente residencial. Um conjunto de dispositivos físicos como sensores de movimento, de temperatura, de luz (iluminância) e de pressão, *reed switches* (chaves magnéticas), um re-

ceptor de RFID, uma barreira ótica, conjuntos de *LEDs* para iluminação, entre outros estão interligados em rede. A rede de sensores utilizada foi desenvolvida por Tibiriçá (2007), especialmente para ser um meio de comunicação de baixo custo entre dispositivos físicos utilizados em ambientes construídos. Um aplicativo permite o acesso aos dados da rede através de uma conexão TCP/IP tipo soquete. O Quadro 8 resume algumas características da rede de sensores.

Topologia	• Em barra para longas distâncias • Livre, para curtas distâncias
Serviços do Protocolo	• Checagem de erros • Endereçamento • Tamanho de pacote variável • Mensagem com ou sem reconhecimento
Máxima taxa de transmissão	250 kb/s
Meio físico	Par trançado
Máximo comprimento do canal	1000 m
Máximo número de nós por canal	256
Máximo número de entradas por nó	128
Consumo típico dos nós	10 mW

Quadro 8 – Características da rede de sensores. Fonte: Tibiriçá (2007).

A Figura 46 mostra a planta dos dois quartos e o posicionamento dos sensores (descritos no Quadro 9). Durante o período de observação o sistema ficou em funcionamento no período diurno, cinco dias por semana. Os neurônios de interface (localizados no computador do Quarto 2) se comunicam com o aplicativo da rede de sensores (localizado no computador do Quarto 1) através de uma conexão TCP/IP tipo soquete. A movimentação dos usuários durante a execução de atividades diárias estimulava os sensores instalados nos dois quartos. *LEDs* para iluminação foram programados para responder diferentemente às diferentes ações dos usuários. Durante o período de observação não foram notados atrasos perceptíveis (maiores

que um segundo) entre a mudança de estado provocada em um sensor, o recebimento do sinal pelos neurônios e a atuação do sistema através dos atuadores.

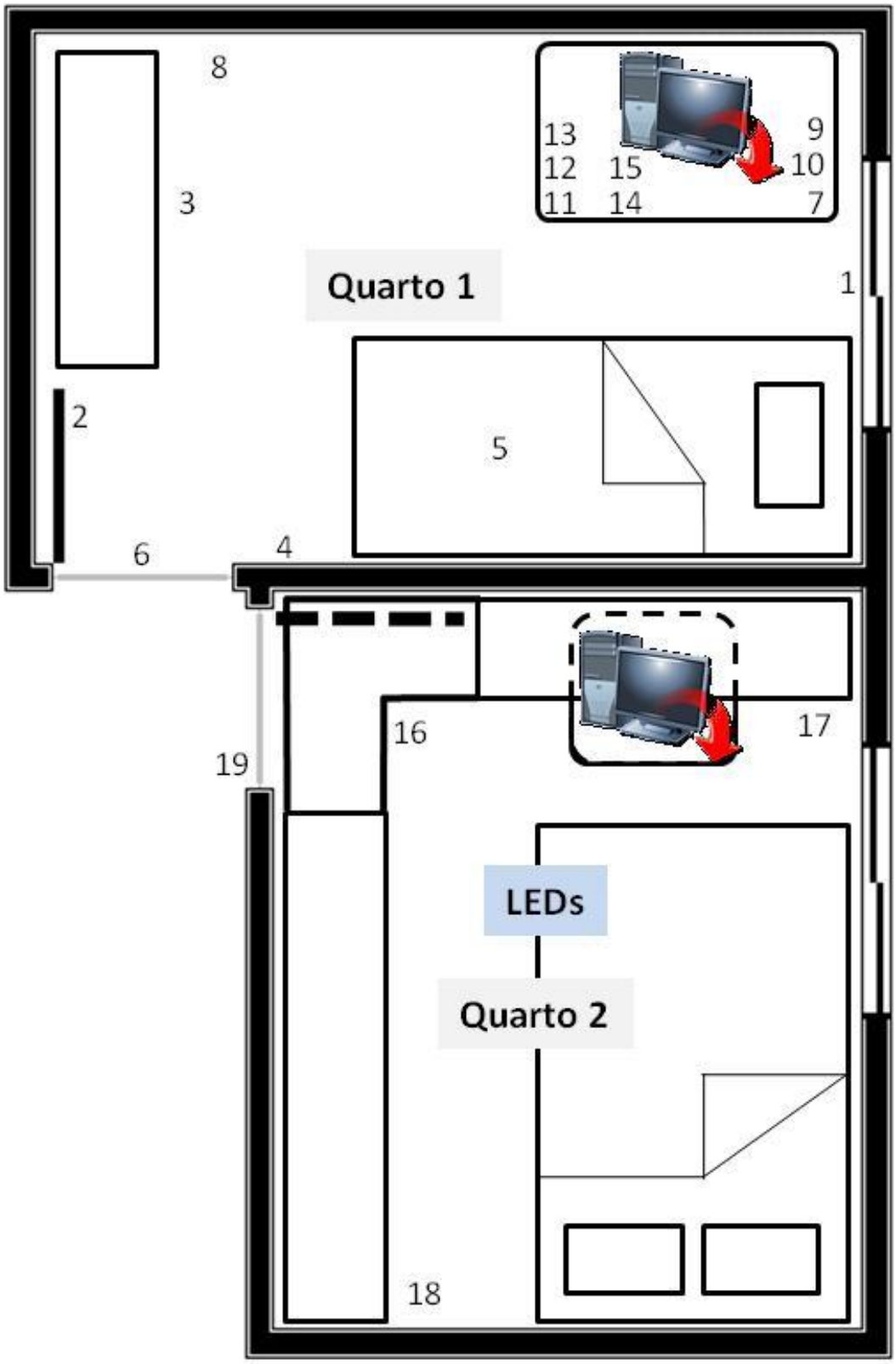

Figura 46 – Planta dos quartos com a posição dos sensores instalados.

Número	Tipo de Sensor	Função
1	Reed switch	Janela aberta/fechada
2	Reed switch	Porta aberta/fechada
3	Reed switch	Armário fechado/aberto
4	Bobina Indutiva	Lâmpada ligada/desligada
5	Pressão	Cama ocupada/desocupada
6	Pressão	Barra de ginástica ocupada/desocupada
7	Pressão	Mouse em uso ou não
8	Movimento	Movimento no Quarto 1
9	Interruptor	Botão + (aumento da intensidade da iluminação)
10	Interruptor	Botão - (diminuição da intensidade da iluminação)
11	Receptor controle remoto	Identificação do código apertado no controle remoto
12	Temperatura	Temperatura
13	Umidade	Umidade
14	Luz	Iluminância
15	Leitora de RFID	Identidade e presença de usuário
16	Movimento	Movimento no Quarto 2
17	Movimento	Movimento no Quarto 2
18	Movimento	Movimento no Quarto 2
19	Barreira Ótica	Passagem ou não de usuários pela porta

Quadro 9 – Descrição dos sensores apontados na Figura 46.

6.3. Considerações finais

O sistema de automação descrito na seção 6.1 é resultado da associação de módulos simples e bem determinados. Com poucas classes de neurônios e de glândulas foi possível criar uma diversidade de sistemas automáticos para o ambiente construído. Com praticamente os mesmos módulos básicos, diferentes sistemas de automação foram programados. Os sistemas de Iluminância e de HVAC no quarto, e o sistema de Controle do Chuveiro no banheiro utilizam a mesma estrutura básica de módulos. Os neurônios Verão e Inverno compõem tanto o sistema de Aproveitamento de Luz Solar no quarto quanto o sistema de Controle do Chuveiro no banhei-

ro. Um conjunto de módulos básicos foi combinado para formar diferentes sistemas de automação, em etapas distintas de programação da casa descrita anteriormente. Cumpre-se assim um dos requisitos colocados inicialmente para a arquitetura proposta, a modularidade.

A associação de novos módulos aos já existentes é facilitada pela interface de comunicação comum de neurônios e de glândulas. Na casa fictícia, o sistema de Detecção de Ausência era inicialmente baseado em um sensor de movimento. Outros sensores foram acrescidos formando novas associações de neurônios e trazendo novas funcionalidades ao sistema. O mesmo é observado na implantação do sistema de *Setpoint* Personalizado. Os neurônios "*Setpoint* de Iluminação" e "*Setpoint* de Temperatura" se associaram às glândulas "*Setpoint* de Luz" e "*Setpoint* de Temperatura", respectivamente, aumentando as funcionalidades anteriores. A interface comum entre os módulos básicos, neurônios e glândulas, permite que estes sejam combinados de inúmeras maneiras, tornado a arquitetura proposta flexível.

O mecanismo de funcionamento das glândulas permite que os comportamentos de neurônios/glândulas pertencentes a diferentes sistemas sejam integrados. A configuração de um novo receptor de hormônio em um neurônio/glândula, o deixa suscetível a ação de uma glândula. Na casa fictícia, a glândula "Ausência" afeta diretamente comportamentos dos sistemas de Iluminação, de HVAC e de Aproveitamento de Luz Solar no quarto. Neurônios em toda a casa também poderiam ser configurados a se comportarem de maneira particular quando a glândula "Bloqueio" ou a glândula "Vazamento" estivessem ativas. Este último exemplo demonstra como a inserção de um novo comportamento pode ser configurada para afetar várias partes do sistema. O sistema final é, desta forma, capaz de integrar as diversas partes que o compõe.

A expansão do sistema acontece de forma natural. É possível desenvolver uma solução de automação em etapas distintas. Funcionalidades não precisam ser previstas desde o início do processo de desenvolvimento. A interface simples e comum entre as diferentes classes de neurônios e de glândulas facilita a associação de novos neurônios e novas glândulas aos já existentes.

Um sistema completo de automação pode ser desenvolvido a partir de classes de neurônios e de glândulas pré-existentes. Todo o sistema se torna fruto da configuração de parâmetros da camada Ontológica. Essas características tornam a arquitetura proposta modular, flexível e capaz de integrar de forma simples os diversos sistemas automáticos que coexistem nos ambientes construídos.

7. Conclusão

Arquitetura de *software* para Automação do Ambiente Construído é um tema ainda pouco explorado. Na literatura, é extensa a lista de trabalhos voltados à aprimoração de sistemas automáticos aplicados aos ambientes construídos, mas muito pouco é encontrado sobre temas relacionados ao desenvolvimento e à estruturação do *software* utilizado nestes sistemas. No entanto, a arquitetura de *software* é a base para construção desses sistemas. À medida que a complexidade do sistema de automação aumenta, maior é a importância da estruturação do *software*.

Como inicialmente estabelecido, este trabalho de pesquisa consistiu na proposição de uma arquitetura de *software* neuro-reativa para sistemas de Automação do Ambiente Construído. Estes últimos são construídos a partir de associações de neurônios e de glândulas “artificiais”, elementos básicos da arquitetura com características fortemente reativas. Um modelo de quatro camadas norteia o desenvolvimento de aplicações. A camada Física é responsável por capturar informações e enviar comandos aos componentes físicos do sistema, como sensores e atuadores. Duas camadas, Reativa e Ontológica, compõem o núcleo da arquitetura. Toda a dinâmica do sistema é resultado das interações entre neurônios e glândulas na camada Reativa. Por outro lado, o modo de interação e de ação de cada componente é descrito na camada Ontológica. As três camadas anteriores servem como plataforma para o desenvolvimento de aplicativos na camada de Aplicação. Esses aplicativos podem modificar dinamicamente os parâmetros de configuração e as conexões descritas na camada Ontológica, e instanciar novos neurônios e glândulas na camada Reativa.

O projeto de *software* pode ser desenvolvido em etapas distintas com utilização de modelos com neurônios, glândulas e domínios que representarão os diversos

elementos de automação do sistema. Diagramas de Relacionamento ajudam a visualizar o sistema em desenvolvimento. Conjuntos de tabelas descrevem as características funcionais de cada neurônio e glândula. Esses diagramas e tabelas são elementos importantes de documentação no projeto, e a sua tradução para um *software* ocorre de maneira natural com o uso da arquitetura proposta.

Requisitos como modularidade, flexibilidade e capacidade de integração das partes foram obtidos. Os elementos básicos da arquitetura proposta, neurônios e glândulas, são estruturalmente simples e com interface também simples, o que torna a solução final modular e flexível. As glândulas representam comportamentos e *set-points*, e propagam essas informações a todas as entidades de um domínio, integrando dessa forma comportamentos individuais em um domínio. O resultado foi uma estrutura final que facilita o desenvolvimento, a manutenção e a expansão do *software* para Automação do Ambiente Construído.

7.1. Principais contribuições

As principais contribuições alcançadas com esta tese foram:

- **Proposição de uma arquitetura de *software* para Automação do Ambiente Construído.** A arquitetura proposta coloca-se como uma opção para o desenvolvimento de *software* para sistemas de Automação do Ambiente Construído. A literatura ainda trata de forma incipiente este assunto e praticamente são inexistentes referências específicas para estruturação de *software* para construção de sistemas de Automação do Ambiente Construído. Nesse contexto, a arquitetura proposta é uma base de referência para futuros trabalhos nesta área.

- **Implementação do arcabouço da arquitetura proposta.** O arcabouço da arquitetura proposta foi programado na linguagem Java utilizando como plataforma as tecnologias CORBA e MySQL. Foram tomados cuidados para que o arcabouço deixasse detalhes de funcionamento da arquitetura transparente aos usuários. A estrutura utilizada para implementação da arquitetura proposta foi descrita. Futuros trabalhos poderão utilizá-la como referência.

- **Modelos baseados em neurônios e em glândulas.** Os neurônios e as glândulas podem ser utilizados para descrever o funcionamento de elementos básicos de sistemas de Automação do Ambiente Construído. Desta forma um sistema de Automação do Ambiente Construído pode ser modelado como uma associação de neurônios e de glândulas. Independentemente da implementação em *software*, o modelo construído colabora para a visualização e o entendimento da dinâmica de funcionamento do sistema.

7.2. Trabalhos futuros

A seguir são apresentadas sugestões para futuros trabalhos:

- **Desenvolvimento de ferramentas visuais para configuração de neurônios e de glândulas.** A configuração de neurônios e de glândulas na implementação apresentada é feita através de códigos SQL. Um ambiente de desenvolvimento com recursos visuais, no qual o usuário trabalhe de maneira mais intuitiva, facilitará bastante o desenvolvimento de novas aplicações de Automação do Ambiente Construído.

- **Métodos para combinação de comportamentos**. Um neurônio ou uma glândula pode estar sujeito a ação de vários hormônios (comportamentos).

Na implementação apresentada, uma hierarquia de prioridades determina qual hormônio ativo tem maior prioridade. Hormônios ativos com a mesma prioridade podem ser combinados através de cinco métodos: *last*, *first*, *mean*, *higher* e *smaller*. No entanto, o gerenciamento de comportamentos é um campo importante dentro das arquiteturas reativas; outros métodos de combinação de comportamento devem merecem ser estudados e adaptados à arquitetura proposta.

- **Implementação da arquitetura proposta em "*hardware*"**. A arquitetura proposta foi implementada completamente na linguagem Java em computadores PC. Parte da arquitetura poderia ser implementada em nós de redes de campo ou de sensores, distribuindo o processamento do sistema.

- **Criação de versões do arcabouço proposto em outras linguagens de programação**. O arcabouço da arquitetura proposta foi programado na linguagem Java. Implementações em outras linguagens também poderiam ser feitas. Outras opções de *middleware* de comunicação e de banco de dados também devem ser exploradas.

- **Estudo e otimização dos mecanismos de comunicação de neurônios e de glândulas.** Neurônios e glândulas foram implementados como processos de *software* independentes, que se comunicam de forma assíncrona e sem preocupação com reconhecimento de mensagens. Futuras versões poderiam ter mecanismos de reconhecimento de mensagem e mecanismos de comunicação síncrona.

- **Programação de classes de neurônios.** Novas classes de neurônios e de glândulas poderiam ser programadas com funcionalidades úteis aos siste-

mas de Automação do Ambiente Construído. Bibliotecas de classes facilitarão a construção desses sistemas.

- **Módulos de aprendizagem**. Módulos de aprendizagem com diferentes algoritmos poderiam ser criados para a camada de Aplicação. Eles poderiam fazer configuração dinâmica de parâmetros na camada Ontológica e instanciações de novas entidades na camada Reativa.

Referências

AKYILDIZ, I. F.; SU, W.; SANKARASUBRAMANIAM, Y.; CAYRICI, E. **A survey on Sensor Networks.** IEEE Communications Magazine, pp. 102-114, 2002.

ALCALÁ, R.; CASILLAS, J.; CORDÓN, O.; GONZÁLEZ, A.; HERRERA, F. **A genetic rule weighting and selection process for fuzzy control of heating, ventilating and air conditioning systems.** Engineering Applications of Artificial Intelligence, 18, pp. 279-296, 2005.

BOLTON, F. **Pure CORBA.** Sams Publishing, 2002.

BRAGA, A.; CARVALHO, A. C.; LUDERMIR, T. B. **Redes Neurais Artificiais: teoria e aplicações.** Rio de Janeiro: LTC, 2007.

BROOKS, R. A. **A robust layered control system for a mobile robot.** IEEE Journal of Robotics and Automation, 1 (1), pp. 1-10, 1986.

BROOKS, R. A. **Intelligence without representation.** Artificial Inteligence, 47, pp. 139-159, 1991.

BROOKS, R. A. **The Intelligent Room Project.** Proceedings of the 2nd International Conference on Cognitive Technology, 1997.

BROSE, G.; VOGEL, A.; DUDDY, K. **Java Programming with CORBA: Advanced Techniques for Building Distributed Applications.** John Wiley & Sons, 2001.

CALLAGHAN, V.; CLARKE, G.; COLLEY, M.; HAGRAS, H.; CHIN, J. S.; DOCTOR, F. **Inhabited intelligent environments.** BT Technology Journal, 22 (3), 2004.

CALLAGHAN, V.; CLARKE, G.; POUNDS-CORNISH, A.; SHARPLES, S. **Buildings as Intelligent Autonomous Systems: A Model for Integrating Personal and Buildings Agents.** 6th International Conference on Intelligent Autonomous Systems, 2000.

CAYCI, F.; CALLAGHAN, V.; CLARKE, G. **A Distributed Intelligent Building Agent Language (DIBAL).** 6th International Conference on Information System Analysis and Synthesis, 2000.

CHONG, C.-Y.; KUMAR, S. P. **Sensor networks: evolution, opportunities and challenges.** 1247-1256, Ed. Proceedings of the IEEE, 91 (8), 2003.

COEN, M. H. **Building brains for rooms: designing distributed software agents.** Nineth Conference on Innovative Applications of Artificial Inteligence (IAA' 99), 1997.

COEN, M. H. **Design Principles for Intelligent Environments.** Proceedings of the fifteenth national/tenth conference on Artificial intelligence/Innovative applications of artificial intelligence, 1998.

COLEMAN, D.; ARNOLD, P.; BODOFF, S.; DOLLIN, C.; GILCHRIST, H.; HAYES, F.; JEREMAES, P. **Desenvolvimento orientado a objetos: o método Fusion.** Rio de Janeiro: Campus, 1996.

COOK, D. J.; DAS, S. K. **How smart are our environments? An updated look at the state of the art.** Pervasive and Mobile Computing, 3, pp. 53-73, 2007.

COOK, D. J.; DAS, S. K. **Smart Environments - Technology, Protocols and Applications.** New Jersey: John Wiley & Sons, 2005.

COOK, D. J.; YOUNGBLOOD, M.; HEIERMAN III, E. O.; GOPALRATNAM, K.; RAO, S.; LITVIN, A.; KHAWAJA, F. **MavHome: An Agent-Based Smart Home.** Proceedings of the First IEEE International Conference on Pervasive Computing and Communications, 2003.

DAS, S. K.; COOK, D. J.; BHATTACHARYA, A.; HEIERMAN III, E. O.; LIN, T.-Y. **The role of prediction algorithms in the Mavhome Smart Home Architecture.** IEEE Wireless Communications, December, 2002.

DEY, A. K. **Understanding and using context.** Personal and Ubiquitous Computing, 5, pp. 4-7, 2001.

DEY, A. K.; ABOWD, G. D.; SALBER, D. **A context-based indrastructure for Smart Environments.** Proceedings of the 1st International Workshop on Managing Interactions in Smart Environments (MANSE ´99), pp. 114-128, 1999.

DOUNIS, A. I.; CARAISCOS, C. **Intelligent Coordinator of Fuzzy Controller-Agent for Indoor Environment Control in Buildings using 3-D Fuzzy Comfort Set.** IEEE International Fuzzy Systems Conference, pp. 1-6, 2007.

DOUNIS, A. I.; LEFAS, C. C.; ARGIRIOU, A. **Knowledge-based versus classical control for solar-building designs.** Applied Energy, 50, pp. 281-292, 1995.

DOUNIS, A. I.; SANTAMOURIS, M. J.; LEFAS, C. C.; ARGIRIOU, A. **Design of a fuzzy set environment comfort system.** Energy and Buildings, 22, pp. 81-87, 1995.

ELMASRI, R.; NAVATHE, S. B. **Sistemas de banco de dados.** São Paulo: Pearson Addison Wesley, 2005.

GUILLEMIN, A. **Using Genetic Algorithms to into Account User Wishes in an Advanced Building Control System.** Lausanne, Suiça: Tese de doutorado da Escola Politécnica de Lausanne, 2003.

GUILLEMIN, A.; MOLTENI, S. **An energy-efficient controller for shading devices self-adapting to the user wishes.** Building and Environment, pp. 1091-1097, 2002.

GUILLEMIN, A.; MOREL, N. **An innovative lighting controller integrated in a self-adaptive building control system.** Energy and Buildings, 33, pp. 477-487, 2001.

GUILLEMIN, A.; MOREL, N. **Experimental results of a self-adaptive integrated control system in buildings: a pilot study.** Solar Energy, 72, pp. 397-403, 2002.

GUYTON, C. A.; HALL, J. E. **Textbook of Medical Physiology.** 11th edition ed., Elsevier Saunders, 2006.

HAGRAS, H.; CALLAGHAN, V.; COLLEY, M.; CLARKE, G. **A hierarchical fuzzy-genetic multi-agent architecture for intelligent buildings online learning, adaptation and control.** Information Sciences, 150, pp. 33-57, 2003.

HAYKIN, S. **Neural networks: a comprehensive foundation.** Upper Saddle River: Prentice-Hall, 1999.

HELAL, A. S.; YANG, H.-I.; KING, J.; BOSE, R. **Atlas - Architecture for Sensor Network Based Intelligent Environments.** ACM Transactions on Sensor Networks, 2007.

HELAL, S.; MANN, W.; EL-ZABADANI, H.; KING, J.; KADDOURA, Y.; JANSEN, E. **The Gator Tech Smart House: A Programmable Pervasive Space.** IEEE Computer magazine, pp. 64-74, 2005.

HELAL, S.; WINKLER, B.; LEE, C.; KADDOURAH, Y.; RAN, L.; GIRALDO, C.; MANN, W. **Enabling location-aware pervasive computing applications for the eldery.** Proceedings of the First IEEE Pervasive Computing Conference, 2003.

INTILLE, S. S. **Design a home of the future.** Pervasive Computing , April-June, 2002.

JANSEN, E.; ABDULRAZAK, B.; YANG, H.; KING, J.; HELAL, S. **A Programming Model for Pervasive Spaces.** Submitted to the 3rd International Conference on Service Oriented Computing, 2005.

KADDOURA, Y.; KING, J.; HELAL, A. S. **Cost-precision tradeoffs in unencumbered floor-based indoor location tracking.** Proceedings of third International Conference on Smart Homes and Health Telematic (ICOST), 2005.

KASTNER, W.; NEUGSCHWANDTNER, G.; SOUCEK, S.; NEWMANN, H. M. **Communication Systems for Building Automation Control.** Proceedings of IEEE, 93 (6), pp. 1178-1203, 2005.

KING, J.; BOSE, R.; YANG, H.-I.; PICKLES, S.; HELAL, A. **Atlas: A Service-Oriented Sensor Platform.** Proceedings of the first IEEE International Workshop on Practical Issues in Building Sensor Network Applications (SenseApp 2006), 2006.

KOLOKOTSA, D.; SARIDAKIS, G.; POULIEZOS, A.; STAVRAKAKIS, G. S. **Design and installation of an advanced EIB fuzzy indoor comfort controller using Matlab.** Energy and Buildings, 38, pp. 1084-1092, 2006.

KOLOKOTSA, D.; STAVRAKAKIS, G. S.; KALAITZAKIS, K.; AGORIS, D. **Genetic algorithms optimized fuzzy controller for the indoor environmental management in buildings implemented using PLC and local operating networks.** Engineering Applications of Artificial Intelligence , 15, pp. 417-428, 2002.

KOLOKOTSA, D.; TSIAVOS, D.; STAVRAKAKIS, G. S.; KALAITZAKIS, K.; ANTONIDAKIS, E. **Advanced fuzzy logic controllers design and evaluation for**

buildings' occupants thermal-visual comfort and indoor air quality satisfaction. Energy and Buildings, 33, pp. 531-543, 2001.

KRISTL, Z.; KOSIR, M.; LAH, M. T.; KRAINER, A. **Fuzzy control system for thermal and visual comfort in building.** Renewable Energy, 33, pp. 694-702, 2008.

KULKARNI, A. A. **A reactive behavioral system for the Intelligent Room.** Master Engineering of Electrical Engineering and Computer Science Thesis, 2002.

LAH, M. T.; ZUPANCIC, B.; PETERNELJ, J.; KRAINER, A. **Daylight illuminance control with fuzzy logic.** Solar Energy, 80, pp. 307-321, 2006.

MAHALIK, N. G.; LEE, S. K. **Design and development of system level software tool for DCS simulation.** Advances in Engineering Software, 34, pp. 451-465, 2003.

MAHALIK, N.; XIE, C.; PU, J.; MOORE, P. R. **Virtual distributed control systems: a components-based design method for mechatronic systems.** Assembly Automation, 26 (1), pp. 44-53, 2006.

MINSKY, M. **The society of mind.** New York: Simon & Schuster, 1986.

MOZER, M. C. **The Neural Network House: an environment that adapts to its inhabitants.** Proceedings of the American Association for Artificial Inteligence Spring Symposium on Intelligent Environments, pp. 110-114, 1998.

MURPHY, R. R. **Introduction to AI Robotics.** Cambridge, Massachusetts: A Bradford Book, 2000.

OAKS, S.; WONG, H. **Java threads.** O 'Reilly, 1997.

OLIVEIRA JÚNIOR, H. A. **Lógica Difusa: aspectos práticos e aplicações.** Rio de Janeiro: Interciência, 1999.

PETERSON, L. L.; DAVIE, B. S. **Computer Networks - A Systems Approach.** 3a Edição ed., San Francisco: Morgan Kaufmann Publishers, 2003.

PRESSMAN, R. S. **Engenharia de Software.** São Paulo: Pearson Makron Books, 1995.

RIVERA-ILLINGWORTH, F.; CALLAGHAN, V.; HAGRAS, H. **A Neural Agent Based Approach to Activity Detection in AmI Environments.** Proceedings of IEE International Workshop on Intelligent Environments, 2005.

ROY, A.; BHAUMIK, S. K.; BHATTACHARYA, A.; BASU, K.; COOK, D. J.; DAS, S. K. **Location Aware Resource Management in Smart Homes.** Proceedings of the First IEEE International Conference on Pervasive Computing and Communications (PerCom 2003), pp. 481- 488, 2003.

RUTISHAUSER, U.; SCHÄFER, A. **Adaptive Building Automation - A multi-Agent approach.** Projeto de Pesquisa, Universidade de Ciências Aplicadas de Rapperswil e Instituto de Neuroinformática da Universidade de Zurique, Departamento de Ciência da Computação, 2002.

SATYANARAYANAN, M. **Pervasive Computing: vision and challenges.** IEEE Personal Communications, 2001.

SCHICKHUBER, G.; MCCARTHY, O. **Distributed fieldbus and control network systems.** IEEE Computing & Control Engineering Journal, 8 (1), pp. 21-32, 1997.

SHARPLES, S.; CALLAGHAN, V.; CLARKE, G. **A multi-agent architecture for intelligent building sensing and control.** International Sensor Review Journal, 1999.

SOARES, L. F.; LEMOS, G.; COLCHER, S. **Redes de computadores - das LANs, MANs e WANs às Redes ATM.** Rio de Janeiro: Campus, 1995.

TANDLER, P.; STREITZ, N.; PRANTE, T. **Roomware - Moving toward Ubiquitous Computers.** IEEE Micro, pp. 36-47, 2002.

THOMESSE, J. P. **Fieldbuses and interoperability.** Control Engineering Practice, 7, pp. 81-94, 1999.

TIBIRIÇÁ, A. M. B. **Um estudo sobre os sistemas de iluminação automáticos e os sistemas de controle distribuído para automação predial.** São Carlos, SP: Dissertação de Mestrado, EESC-USP, 2004.

TIBIRIÇÁ, C. B. **Detecção de usuários e suas interações com o ambiente utilizando rede de sensores.** São Carlos: Dissetação de mestrado da EScola de Engenharia de São Carlos da Universidade de São Paulo, 2007.

TOSHIBA. **Neuron chip data book.**

WEISER, M.; BROWN, J. S. **The coming age of calm technology.** Disponível em: <http://www.ubiq.com/hypertext/weiser/acmfuture2endnote.htm>. Acessado em 23 de outubro de 2008.

YOUNGBLOOD, G. M.; HOLDER, L. B.; COOK, D. J. **Managing Adaptive Versatile Environments.** Proceedings of the 3rd IEEE International Conference on Pervasive Computing and Communications, 2005.

ZADEH, L. A. **Fuzzy Sets.** Information and Control, 8, pp. 338-353, 1965.

ZHENG, Y.; AKHTAR, S. **Networks for computers scientists and engineers.** New York: Oxford University Press, 2002.

ZIMMERMANN, H.-J. **Fuzzy set theory and its applications.** 4a Edição ed., Kluwer Academic Publishers, 2001.

Glossário

Bluetooth. Protocolo de comunicação sem-fio para transmissão de dados em curtas distâncias entre dispositivos fixos e móveis, criando uma rede de comunicação pessoal (PAN).

Cluster. Conjunto de computadores que trabalham em conjunto.

Data Mining. Processo de explorar grandes quantidades de dados à procura de padrões consistentes, como regras de associação ou seqüências temporais.

Firewire. Interface serial para computadores pessoais e aparelhos digitais de áudio e vídeo que oferece comunicações de alta velocidade e serviços de dados em tempo real.

Grafo. Representação gráfica das relações existentes entre elementos de dados. Pode ser descrito num espaço euclidiano de *n* dimensões como sendo um conjunto *V* de vértices e um conjunto *A* de curvas contínuas (arestas).

OmniORB. ORB CORBA de código aberto para as linguagens C++ e Python.

Ontologia. Modelo de dados que representa um conjunto de conceitos dentro de um domínio e os relacionamentos entre estes.

OSGI (*Open Services Gateway Initiative*). Plataforma de serviços Java que especifica um modelo de gerenciamento de ciclo de vida de aplicativos, um registro de serviços, um ambiente de execução e módulos.

PCI (Peripheral Component Interconnect - Interconector de Componentes Periféricos). Barramento para conectar periféricos em computadores baseados na arquitetura IBM PC.

PDA (Personal Digital Assistant – Assistente Pessoal Digital). Computador de dimensões reduzidas da ordem de grandeza da mão ou de um papel A6.

PostgreSQL. Sistema gerenciador de banco de dados objeto relacional desenvolvido como projeto de *software* livre.

Proxy. Tipo de servidor que atende a requisições repassando os dados a outros servidores.

SOA (*Service-Oriented Architecture*) ou Arquitetura Orientada a Serviço. Arquitetura de *software* cujo princípio fundamental é disponibilização, na forma de serviços, as funcionalidades implementadas pelas aplicações.

USB (Universal Serial Bus). Barramento serial de conexão rápida para acoplamento de periféricos ao computador.

Wi-Fi. Tecnologia de redes sem fios (WLAN) baseada no padrão IEEE 802.11.

Zeroconf (Zero Configuration Networking). Conjunto de técnicas que criam de forma automática uma rede IP sem necessitar de configuração ou servidores.

ZigBee. Tecnologia de redes sem fios baseada no padrão IEEE 802.15.4 que realiza a interligação de pequenas unidades de comunicações de dados em pequenas áreas.