UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

JAIR JONKO ARAUJO

Framework Orientado a Objetos para o Desenvolvimento de Aplicações de Automação Predial e Residencial

Dissertação submetida à avaliação, como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação

Prof. Dr. Carlos Eduardo Pereira
Orientador

Porto Alegre, abril de 2005.

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Araujo, Jair Jonko

Framework Orientado a Objetos para o Desenvolvimento de Aplicações de Automação Predial e Residencial / Jair Jonko Araujo. – Porto Alegre: PPGC da UFRGS, 2005.
115f.:il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul, Programa de Pós-Graduação em Computação, Porto Alegre, RS-BR, 2005. Orientador: Carlos Eduardo Pereira.

1. Automação predial. 2. Automação residencial. 3. Orientação a Objetos. 4. Framework.. I. Pereira, Carlos Eduardo. II. Título.

À minha esposa Valdriane e à minha filha Aline pela aceitação das minhas ausências e por darem um sentido especial a cada etapa de minha vida.

AGRADECIMENTOS

Em cada etapa de nossa vida somos auxiliados por muitas mãos visíveis e invisíveis a quem temos muito a agradecer quando conseguimos concluí-la. Ao nomear algumas pessoas que contribuíram conosco para materializar algo importante, corremos o risco de sermos injustos. Entretanto algumas foram fundamentais nesse caminho que merecem o destaque de um agradecimento especial. Agradeço:

em primeiro lugar à minha esposa Valdriane pela compreensão das minhas ausências e pelo estimulo durante a realização deste trabalho e à minha filha Aline, hoje com quatro anos, que foi o grande farol que me iluminou em cada dificuldade me estimulando a superá-las;

ao professor Carlos Eduardo, meu orientador, não só pela oportunidade de trabalho, pela orientação firme e tranqüila, pelas incontáveis discussões e revisões ao longo deste trabalho, mas também pelo exemplo de dedicação e ética com o trabalho de pesquisa, na certeza de que este é um dos caminhos importante e possível para gerar desenvolvimento ao país;

ao colega Leandro Buss Becker, companheiro de longa data e com quem tive oportunidade de dividir, além da sala de estudos no DELET/UFRGS, as minhas infindáveis dúvidas e questionamentos. Agradeço também sua contribuição na revisão das versões iniciais do texto da dissertação;

ao Instituto de Informática e ao Departamento de Engenharia Elétrica da UFRGS, personificados pelos seus professores e funcionários, pela lição de como pesquisar e promover ciência com dedicação e ética.

E finalmente, deixo um agradecimento a tantos, não citados nominalmente, mas sem cuja colaboração este trabalho não teria sido realizado, especialmente a meus colegas de trabalho no CEFET-Pelotas que criaram meios para que sempre houvesse tempo para realizar as atividades relacionadas com este trabalho.

SUMÁRIO

LISTA DE ABREVIATURAS OU SIGLAS

ANSI	*American National Standards Institute*;
API	*Application Programming Interface*;
ASHRAE	*American Society of Heating, Refrigerating and Air-Conditioning Engineers;*
AV	Áudio e Vídeo;
BACnet	*Building Automation and Control Network;*
BCU	*Bus Coupling Unit*;
BIBBs	*BACnet Interoperability Building Blocks*;
CAL	*Common Application Language;*
CD	*Compact Disk*;
CEBus	*Consumer Electronics Bus*;
CIC	*CEBus Industry Council*;
CORBA	*Common Object Request Broker Architecture*;
CSMA	*Carrier Sense Multiple Access*;
DCOM	*Distributed Component Object Model*;
DHCP	*Dynamic Host Configuration Protocol*;
DSL	*Digital Subscriber Line*;
DVD	*Digital Video Disk*;
EEPROM	*Electrically Erasable Programmable Read-Only Memory;*
EIA	*Electronic Industries Association*;
EIB	*European Installation Bus*;
EIBA	*European Installation Bus Association*;
EIS	EIB *Interworking Standards for Group-Communication-Objects*;
EHS	*European Home Systems*;
ETS	*EIB Tool Software*;
FMS	*Facility Management Systems*;
GENA	*General Event Notification ArchitectureI*;
HomePnP	*Home Plug & Play*;
HTTP	*Hypertext Transfer Protocol*;
HVAC	*Heating, Ventilation and Air Conditioning*;
IEC	*International Electrotechnical Commission*;
IEEE	*Institute of Electrical and Electronics Engineers*;

IP	*Internet Protocol*;
ISO	*International Standard Organization*;
ITU	*International Telecomunication Union*;
LAN	*Local Area Network*;
LON	*Local Operating Network*;
OMG	*Object Managment Group*;
OPC	*OLC (Object Linking and Embedding) for Process Control*;
OSGi	*Open Services Gateway Initiative*;
OSI	*Open System Interconnection*;
PC	*Personal Computer*;
PEI	*Physical External Interface*;
PLC	*Power Line Carrier*;
RAM	*Random Access Memory*;
RM-ODP	*Reference Modelo-Open Distributed Process*;
ROM	*Read-Only Memory*;
RTOS	*Real-Time Operating System*;
SAP	Sistema de Automação Predial;
SCPTs	*Standard Configuration Property Types*;
SNVTs	*Standard Network Variable Types*;
SOAP	*Simple Object Access Protocol*;
SSDP	*Simple Service Discovery Protocol*;
TCP	*Transmission Control Protocol*;
UCPTs	*User-defined Configuration Property Types*;
UDP	*User Datagram Protocol*;
UML	*Unified Modeling Language*;
UNVTS	*User-Defined Network Variable Types;*
UPnP	*Universal Plug & Play*;
URL	*Uniform Resource Locator*;
VCR	*Video Cassette Recorder*;
XML	*Extensible Markup Language*;
XSL	*eXtensible Stylesheet Language*;

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

O crescente aumento pela exigência de funcionalidades na implementação dos atuais sistemas de automação predial, vem provocando um aumento da complexidade de projeto e de gerenciamento desses sistemas. O grande desafio que se apresenta atualmente é como, a partir de dispositivos isolados e subsistemas, conseguir sistemas totalmente integrados, os quais permitam economia no investimento inicial, na operação e na manutenção dos sistemas de automação, garantindo um aumento no desempenho geral da edificação

Acredita-se que uma etapa importante para avaliar a real necessidade da integração seja projetar o sistema de automação sem foco em uma tecnologia específica, o que não ocorre atualmente, uma vez que, pela carência de ferramentas de apoio ao projeto, as etapas de especificação e projeto geralmente já estão focadas em uma tecnologia disponível para implementação. Este trabalho busca preencher a lacuna deixada pela carência dessas ferramentas, tendo por finalidade a especificação de um *framework* orientado a objetos para o desenvolvimento de aplicações de automação predial e residencial que permita modelar estes sistemas de forma independente da tecnologia que ele irá utilizar, possibilitando o mapeamento posterior para a mais adequada ou disponível.

Serviram como base para o *framework* proposto a análise de vários padrões abertos disponíveis para implementação de sistemas de automação predial e a especificação ISO/IEC10746, o modelo de referência para processamento distribuído aberto, usado como suporte a metodologia de projeto proposta. O trabalho também discute o mapeamento dos conceitos definidos para uma arquitetura alvo, apresentado um estudo de caso para validação da metodologia proposta.

Palavras-chave: automação predial, automação residencial, orientação a objetos, *framework.*

Object-Oriented Framework for The Design of Home and Building Automation Applications.

ABSTRACT

The increasing demand on the funcionalities in the implantation of the new automating building systems has caused an increase in the complexity of the project as well as in the management of these systems. The greatest challenge nowadays is how to obtain systems entirely integrated from isolated devices and subsystems, which allow savings in the initial investments as well as in the operations and maintenance of the automation systems in order to guarantee an increase in the general performance of the building.
It is believed that an important stage to assess the actual need of this integration is to project an automating system without a specific technological focus, which does not occur nowadays. Due to the lack of supporting tools for the project, the stages of specification and project are generally focused on the available technology for implementation. This work aims to fill in this gap by specifying an object-oriented framework for the development of applications in the building automation, enabling the modelling of the systems regardless of the technology it will use, leaving this mapping for the last stage of the project
Several open protocols available for implementation of building automation systems and the specification ISO/IEC10746, Reference Model of Open Distributed Processing, were a foundation for the proposed framework. The latter was used as the support to proposed methodology. This work also discusses the mapping of the defined concepts to target architecture and it presents a case study in order to validate the proposed methodology.

Keywords: Building automation, home automation, object-oriented framework

1 INTRODUÇÃO

1.1 Contextualização do Trabalho

Com o crescente aumento da competição global, a produtividade e a eficiência tornam-se imperativos em todos os negócios. Os países mais ricos e desenvolvidos são aqueles que conseguem produzir bens de elevado valor agregado com maior produtividade para atender suas próprias necessidades e ao mercado externo. A automação constitui papel fundamental nesse processo e assim observa-se, desde o surgimento dos microprocessadores a partir de 1970 e com o desenvolvimento da Informática, um enorme avanço na oferta de dispositivos automatizados e a crescente importância de sistemas computacionais nas mais variadas aplicações de automação.

Isto ocorre porque os avanços tecnológicos na área da microeletrônica tornam disponíveis a custos bastante acessíveis, processadores de suficiente desempenho, tamanho reduzido, baixo consumo, etc. Adicionalmente, no campo da Informática, evoluções nas áreas de sistemas distribuídos, linguagens de programação, sistemas operacionais, entre outras permitem a realização de sistemas computacionais fisicamente distribuídos, capazes de serem monitorados, executados, e até mesmo atualizados remotamente.

A automação está relacionada a equipamentos que controlam processos ou plantas. Segundo (MIYAGI, 1996) controle é a aplicação de uma ação pré-planejada para que aquilo que se considera como objeto de controle atinja certo objetivo. Uma instalação predial, em analogia com os sistemas de automação industrial, é basicamente um processo que se deseja controlar.

Genericamente os sistemas de controle são classificados em dois tipos: os sistemas de variáveis contínuas onde o objetivo é igualar o valor de uma variável física, denominada variável de controle, a um valor de referência e os sistemas de eventos discretos nos quais a execução de operações ocorre conforme procedimentos pré-estabelecidos.

No primeiro caso o dispositivo de realização do controle é denominado regulador ou controlador e é responsável por executar um algoritmo de controle para ajustar o sinal recebido do processo a um valor de referência, enquanto no segundo caso tem-se um processador de comando que avalia o estado atual do processo e determina a próxima tarefa a ser executada. Na Figura 1.1, pode-se observar os principais elementos envolvidos em um processo de controle.

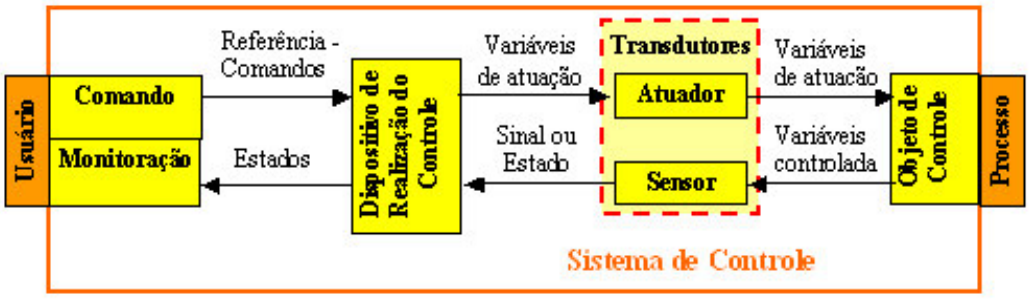


Figura 1.1: Diagrama conceitual básico de um sistema de controle

Pode-se dizer que atualmente existem três grandes áreas para utilização da automação: a automação industrial, a automação predial e a automação residencial.

Num primeiro momento o foco de atenção de fabricantes de dispositivos e prestadores de serviço foi a área industrial. Consolidada a automação industrial ele voltou-se às construções comerciais (hotéis, hospitais, prédios de escritórios, prédio públicos, lojas de departamento, supermercados, entre outros), dando origem à automação comercial e automação predial.

Nessas áreas a automação além de provocar mudanças na metodologia de trabalho permitiu também o controle das variáveis relacionadas ao ambiente de trabalho (iluminação, aquecimento e ventilação, segurança patrimonial, controle de acesso, gerenciamento energético, etc) dando origem ao conceito de prédios inteligentes.

Outra observação é que inicialmente a automação predial e residencial foi realizada com as mesmas tecnologias (equipamentos, barramentos, etc) que eram utilizados na área industrial. A observação, entretanto, que cada uma destas áreas possuem características específicas tem levado ao desenvolvimento de tecnologias que tentam se adequar a necessidades próprias de cada uma delas.

Embora nos primórdios da automação predial o conceito de inteligência estivesse associado apenas à disponibilidade de equipamentos modernos, hoje é consenso que um prédio inteligente deve oferecer mais do que soluções tecnológicas: deve aproveitar ao máximo os recursos naturais e a tecnologia de novos materiais para atingir seus objetivos. Becker define:

> "Prédios inteligentes podem ser definidos como naqueles onde a integração dos sistemas e as novas tecnologias são usadas para maximizar a produtividade de seus ocupantes, permitindo gerenciamento eficiente de recursos e minimização de custos." (BECKER,1995)

Ou seja, o conceito de prédio inteligente não se restringe apenas ao uso de dispositivos eletromecânicos, sensores, atuadores e sistemas computacionais para automação de tarefas e controle de processos. Todavia, no presente trabalho focar-se-á nos conceitos e técnicas que permitam o desenvolvimento de sistemas computacionais usados em automação predial e residencial

Em relação à automação predial o que se observa é que os sistemas necessitam executar um número cada vez maior de funções, entre as quais pode-se citar:

- o controle do sistema de iluminação ambiente, mantendo os níveis especificados conforme a aplicação;
- o controle de acesso a ambientes e equipamentos permitidos apenas aos usuários autorizados;

- o controle de parâmetros relativos a aquecimento, ventilação e condicionamento de ar;
- o controle das condições ambientais que causem ameaça à segurança de pessoas e instalações, relativamente a controle de gases perigosos e fogo;
- o gerenciamento de energia com objetivo de maximizar a produtividade com minimização do consumo de energia;
- o controle dos sistemas de movimentação de pessoas e objetos, tais como elevadores e escadas rolantes.

Já na automação residencial os aspectos importantes a serem considerados são o estilo de vida e preferências de quem vai residir ou reside no local, por isso as soluções tendem a ser muito pessoais e dirigidas. A automação residencial tem que se valer de interfaces amigáveis porque, na maioria das vezes, os usuários são totalmente leigos em computação, sendo avessos a programações complexas. Além disso não é realista considerar-se que cada residência possua um técnico treinado para operar o sistema de automação, algo que pode ser viável no caso de automação de um prédio comercial. Outra peculiaridade refere-se à importância do entretenimento, muito maior em sistemas de automação residencial do que em prédios comerciais.

Logicamente existem características comuns às áreas de automação predial e residencial das quais pode-se destacar: adequada relação custo/benefício, confiabilidade, interatividade, atualização tecnológica (*upgrades*) simples, entre outras. Muitas destas características são também desejáveis em sistemas de automação industrial.

Neste trabalho não se fará distinção entre automação predial e residencial, o termo automação predial será usado indistintamente.

A Figura 1.2 mostra uma visão parcial de um sistema de automação predial e exemplifica algumas funções de controle que podem fazer parte um projeto.

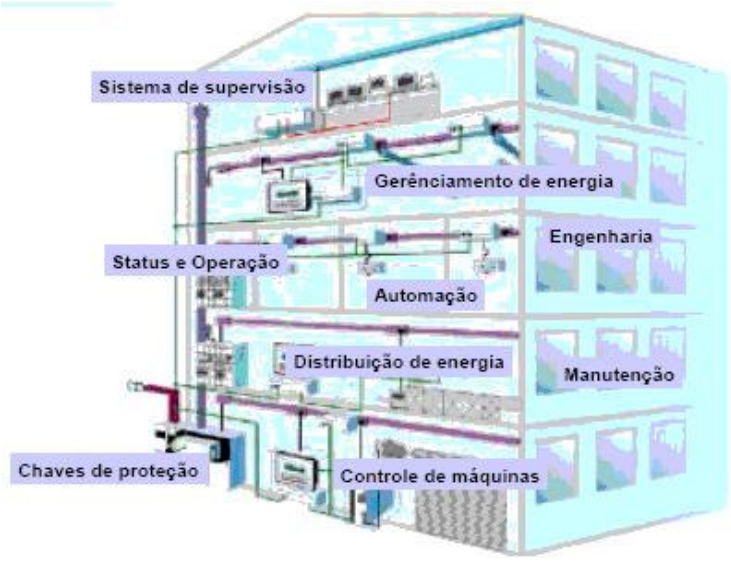


Figura 1.2 : Visão parcial de funções de um sistema de automação predial

Como ocorre em sistemas de automação industrial, em sistemas de automação predial, cada vez mais os dispositivos (sensores, atuadores, equipamentos autônomos) são dotados de "inteligência" e existe uma clara tendência à descentralização do controle entre os dispositivos, conectados através de uma arquitetura de rede adequada. Isto permite que a troca de informações seja realizada

diretamente entre eles para a execução das tarefas programadas, aumentando a eficiência e a confiabilidade dos sistemas instalados.

Em contrapartida há um aumento da complexidade de projeto e de gerenciamento desses sistemas: os desafios da comunicação e da integração ainda persistem e diferentes soluções estão sendo propostas, em geral por diferentes grupos de fabricantes de dispositivos, para tentar resolvê-los.

Em trabalhos anteriores - vide (ARAUJO, 2002), (ARAUJO, 2003) e (ARAUJO, 2004), concluiu-se que os padrões mais recentes estão se concentrando em definir especificações para as últimas camadas do modelo OSI, em especial a camada de aplicação (TANENBAUM, 1997), a fim de que a integração entre diferentes equipamentos seja realizada de maneira mais rápida e simples. Muitos destes protocolos utilizam o paradigma da orientação a objetos para sua especificação e padronizam vários perfis de objetos para áreas especificas da automação predial citadas acima.

Ao mesmo tempo, vários trabalhos têm discutido estratégias para implementação de *middlewares* para solução do problema da interoperabilidade, tal como pode ser observado em (WILS, 2002), (CHO, 2002), (KU, 2002) e (MOON, 2002).

Uma outra linha de pesquisa investiga a utilização de inteligência artificial em sistemas de automação predial a fim de facilitar-se a operação pelos usuários. Nesta linha de pode-se citar o projeto EMBASSI (EMBASSI, 2004) que busca o desenvolvimento de novos paradigmas e arquiteturas para aumentar a interação intuitiva com diferentes sistemas, entre eles os de automação predial, fornecendo assistência inteligente, interação multímodo e interfaces de usuário antropomórficas dentro de um *framework*, o qual forma uma camada acima da arquitetura física e lineariza a interação com o usuário, independente da tecnologia que esteja sendo empregada para implementação.

Ao nível de projeto observa-se, entretanto, a carência de ferramentas que permitam ao projetista modelar o sistema baseado nas funcionalidades necessárias, de forma independente da tecnologia que ele irá utilizar, possibilitando o mapeamento posterior para a tecnologia mais adequada ou disponível. Atualmente o projeto da instalação deve ser realizado utilizando a ferramenta disponível para o protocolo, utilizando os conceitos que ela disponibiliza, de acordo com as suas características e com os conceitos implementados por esse protocolo especificamente.

1.2 Objetivos do Trabalho

O crescente aumento pela exigência de funcionalidades na implementação dos atuais sistemas de automação predial, vem provocando um aumento da complexidade de projeto e de gerenciamento desses sistemas: os desafios da comunicação e da integração, ponto chave para interoperabilidade entre os dispositivos e subsistemas, ainda permanecem sem solução.

O grande desafio que se apresenta atualmente é como a partir de dispositivos e subsistemas isolados conseguir-se sistemas totalmente integrados, os quais permitam economia no investimento e maior facilidade na operação e na manutenção dos sistemas de automação, permitindo assim um aumento no desempenho geral da edificação (REIS, 2002).

Acredita-se que uma etapa importante para avaliar a real necessidade da integração seja projetar o sistema de automação sem foco em uma tecnologia específica,

o que não ocorre atualmente. Esta constatação é decorrente da carência de ferramentas de apoio ao projeto, uma vez que as etapas de especificação e implementação geralmente já estão focadas em uma tecnologia pré-definida.

Este trabalho busca preencher a lacuna deixada pela carência dessas ferramentas, tendo por finalidade a especificação de um *framework* orientado a objetos para o desenvolvimento de aplicações de automação predial, permitindo modelar estes sistemas de forma independente da tecnologia que ele irá utilizar, possibilitando o mapeamento posterior para a mais adequada ou disponível, conforme será abordado nos próximos capítulos.

1.3 Organização do Texto

O texto desta dissertação está organizado como segue: o próximo capítulo apresenta o estado da arte em sistemas de automação predial, onde são apresentados as arquiteturas usuais nesses sistemas, os principais padrões abertos disponíveis, alguns *frameworks*, uma breve descrição do modelo de referência para processamento distribuído aberto e uma avaliação de alguns trabalhos relacionados com os conceitos de visão e com este modelo de referência.

O capítulo 3 define todas as etapas de projeto do arcabouço proposto. O capítulo 4 apresenta a metodologia para o projeto de sistemas de automação predial, a qual é suportada por este *framework*.

O Capítulo 5 apresenta a validação do modelo apresentado nos dois capítulos anteriores: os conceitos apresentados são mapeados para uma arquitetura alvo e é apresentado um estudo de caso com o objetivo de validar a metodologia proposta. Finalmente o último capítulo apresenta as conclusões obtidas e aponta para os trabalhos futuros.

No capítulo de anexos pode ser encontrada a descrição detalhada sobre o modelo, com a descrição de todas as classes e dos subsistemas especificados, o detalhamento da arquitetura alvo e do mapeamento tecnológico para esta arquitetura e o detalhamento de parte do estudo de caso não apresentado no corpo do texto.

2 ANÁLISE DO ESTADO DA ARTE EM AUTOMAÇÃO PREDIAL

Conforme já abordado no Capítulo 1, as primeiras soluções de automação em sistemas prediais implementavam as mesmas tecnologias utilizadas em automação industrial. Entretanto, a observação de que estes sistemas apresentam diferentes requisitos, levou ao desenvolvimento de soluções específicas para esta área nos últimos anos.

Este capítulo tem por objetivo fornecer uma breve descrição do estado da arte em sistemas de automação predial. Inicialmente serão apresentadas as arquiteturas usais em sistemas de automação predial e posteriormente serão abordadas algumas propostas de protocolos de comunicação disponíveis para implementação desses sistemas e seus modelos. Na seqüência serão analisados alguns *frameworks* e no final do capítulo é incluída uma breve descrição do modelo de referência para sistemas distribuídos abertos, o qual serviu de base para o modelo de objetos e a metodologia de projeto proposta neste trabalho.

2.1 Arquiteturas em Sistemas de Automação Predial.

Quanto à arquitetura física dos componentes de um sistema de automação predial, pode-se ter três soluções distintas. Na solução mais simples um dispositivo central de controle executa todos os algoritmos de controle, sendo responsável pelo gerenciamento total do sistema. As características desse tipo de sistema estão amplamente discutidas na literatura de redes de computadores e sistemas operacionais (TANENBAUM, 2003).

As evoluções de setores como microeletrônica e da informática deram origem ao segundo tipo de arquitetura no qual a implementação do sistema de automação predial está distribuída entre dispositivos autônomos (sensores, atuadores, interfaces com usuário, etc) que são conectados através de uma arquitetura de rede adequada e distribuem entre si a tarefa de controle, com a troca de informações diretamente entre eles para a execução das tarefas programadas, sem a necessidade de uma unidade de controle centralizada. Uma discussão das características dessa arquitetura pode ser encontrada na mesma referência citada anteriormente.

O terceiro tipo de solução é uma solução intermediária entre os sistemas centralizados e os totalmente distribuídos. Nesta solução existem dispositivos autônomos que funcionam com um conjunto de funcionalidades que independem de uma unidade central, porém determinadas funcionalidades só são possíveis pelo gerenciamento da unidade de controle central.

Um sistema de Automação Predial pode ser subdividido em diversas partes de acordo com a funcionalidade que se deseja controlar. Cada uma dessas partes, posteriormente denominada subsistema no âmbito deste trabalho, é responsável pelo controle de um conjunto de funcionalidades afins, das quais pode-se destacar:

- **aquecimento, ventilação e ar condicionado:** permite o controle da temperatura, da ventilação e da umidade ambiental com parâmetros selecionados de acordo com a estação do ano, período do dia, ocupação, dia da semana (domingo, feriado), etc., a fim de aumentar o conforto e a produtividade das pessoas usuárias do prédio e minimizar o consumo de energia;
- **iluminação:** possibilita o controle de iluminação de acordo com os parâmetros tais como horário, ocupação do ambiente, entre outros, garantindo níveis de iluminação adequados nos ambientes internos e externos;
- **o controle de acesso**: permite o acesso apenas a usuários autorizados às instalações internas (salas, elevadores, estacionamento, etc) e aos equipamentos;
- **segurança**: mantém as condições de segurança das instalações em caso de problemas elétricos ou mecânicos, fumaça e outros gases, incêndio, vazamentos de água, entre outros, devendo ser capaz de acionar as equipes de emergência, fechar automaticamente portas contra fogo, acionar sistemas de emergência, etc.
- **gerenciamento de energia:** permite o monitoramento e o gerenciamento do consumo de energia e de outros recursos tais como gás, água fria e quente, fornecimento e consumo de energia elétrica (controle da demanda, a programação horária de cargas, a supervisão de geradores de emergência e de sistemas de alimentação permanentes,etc.) entre outros;
- **movimentação:** tem como objetivo o gerenciamento dos recursos de transporte tais como elevadores e escadas rolantes, com algoritmos selecionados conforme horário, dia da semana, fluxo de pessoas, etc.;
- **entretenimento:** possibilita o controle de *HomeTeather* (DVD, CD, TV, VCR), som ambiente, vídeo e áudio sob demanda, internet rápida, jogos em rede, etc.

2.2 Protocolos de Comunicação

Para implementar estas tarefas de controle um sistema de automação predial pode utilizar uma solução proprietária ou soluções abertas. No primeiro caso um determinado fabricante oferece uma solução, a qual pode ser para um subsistema específico ou para um sistema de automação completo que envolva todos os subsistemas sendo que apenas dispositivos fornecidos por este fabricante podem operar corretamente no sistema. A interoperabilidade com subsistemas de outros fabricantes é uma questão que depende de negociações comerciais e da existência de *gateways* que permitam a conversão de protocolos para permitir a comunicação.

A outra possibilidade é utilizar soluções abertas. Neste caso equipamentos fornecidos por diferentes fabricantes podem interoperar desde que tenham sido fabricados segundo um mesmo padrão. Numa analise da situação atual de mercado, o que se observa são fabricantes trabalhando em conjunto na especificação de padrões que são administrados por associações encarregadas da manutenção e da atualização do padrão e pela certificação de produtos. Novamente a comunicação entre dispositivos de diferentes padrões exige a utilização de *gateways*.

Um padrão define uma arquitetura para interligação dos dispositivos e o protocolo que será implementado para permitir a comunicação entre eles. Alguns dos padrões abertos mais utilizados serão discutidos a seguir, sendo que uma análise mais detalhada dos mesmos pode ser encontrada em (ARAUJO, 2002), (ARAUJO, 2003) e (ARAUJO, 2004).

2.2.1 X10

O sistema X-10 PLC (Power Line Carrier) (X10,2003) é um dos mais antigos protocolos de comunicação para sistemas de automação residencial, tendo sido originalmente desenvolvido nos anos 70 e tendo os primeiros produtos disponíveis a partir de 1979. Atualmente uma grande gama de produtos X-10 encontra-se disponível (vide referência anterior).

O protocolo X-10 transmite dados modulados sobre a rede elétrica sendo que uma informação binária é transmitida sempre que o sinal senoidal de tensão elétrica passa pelo zero. E em função disto podem ser utilizados em qualquer residência utilizando-se a infra-estrutura da rede elétrica já existente.

Na Figura 2.1 pode-se observar o formato do frame para transmissão de mensagens entre um transmissor e um receptor. Um endereço de dispositivo é composto por um código de ambiente (*house code*) mais um código de unidade (*key/number code*), permitindo endereçar 256 pontos diferentes. Uma mensagem básica em X-10 usa 13 bits. Os primeiros 4 bits são um código de entrada (*start code*), os 4 seguintes um código de ambiente (*house code*), os próximos 4 são um código de função (*function code*) ou de unidade (k*ey* ou *number code*) e o último bit indica se os 4 anteriores devem ser interpretados como função ou como unidade.

Código transmitido quando o número do receptor é selecionado

Start Code	*House Code*	*Number Code*	*Start Code*	*House Code*	*Number Code*

Código transmitido quando a função é selecionada

Start Code	*House Code*	*Function Code*	*Start Code*	*House Code*	*Function Code*

Figura 2.1 : Esquema de transmissão de mensagem no protocolo X10

As funções disponíveis são basicamente para ligar, desligar e realizar configurações básicas em dispositivos de automação predial. Para acionar um equipamento X-10 são necessários dois conjuntos de 13 bits, um para transmitir o endereço e outro para transmitir o comando em si. A transmissão é em broadcast e todo o comando é repetido duas vezes, assim um comando completo ocupa necessita aproximadamente 0,8s. A rede elétrica pode ocasionar alguns comportamentos erráticos dos componentes tais como falta de energia ou descargas eletromagnéticas.

Pode-se utilizar o X-10 em diversas aplicações, tais como acionamento remoto de lâmpadas, eletrodomésticos e portas. No entanto, como sua confiabilidade é limitada, não se recomenda seu uso em aplicações criticas, ligadas à segurança doméstica, já que os dispositivos não implementam sistemas de monitoramento para avaliar o *status* de um equipamento.

2.2.2 LONWorks

O padrão LonWorks (ECHELON, 1999-a) é uma tecnologia proposta pela empresa Echelon em 1990 e abrange toda a infra-estrutura necessária (hardware e

software) para a operação da rede local denominada LON (*Local Operating Network*). É um sistema aberto e de controle distribuído, com o funcionamento baseado num protocolo de comunicação denominado LonTalk. Em outubro de 1999, LonTalk tornou-se o padrão ANSI 709.1. Uma associação de usuários denominada *LonMark Interoperability Association* estabelece padrões e certifica dispositivos de controle interoperáveis.

O controle da rede é distribuído nos dispositivos os quais são denominados nós. Cada nó tem seu próprio programa aplicativo, de acordo com a sua função no processo, e é capaz de comunicar-se com os outros nós usando o protocolo de comunicação. É possível construir uma rede com até 32385 dispositivos, os quais podem ser configurados em várias topologias usando-se diversos meios físicos.

É um protocolo dividido em camadas que implementa todas as camadas mo modelo OSI, baseado em pacotes, com comunicação *peer-to-peer* e utiliza o algoritmo CSMA p-persistente preditivo para acesso ao meio físico. Um esquema de prioridades garante o acesso preferencial ao meio para pacotes com prioridade alta.

Diversos esquemas de endereçamento são possíveis: os pacotes podem ser endereçados, a partir do nó de origem, para um nó específico (*physical adress* ou *device adress*), para um grupos de nós(*group adress*) ou para todos os dispositivos(*broadcat adress*). Todo o pacote transmitido contém o endereço do dispositivo de origem (*source adress*) e do dispositivo de destino (*destination adress*), que pode ser qualquer um dos citados acima.

Um programa de aplicação consiste em um ou mais objetos LonWorks. Cada objeto é baseado na definição de um perfil funcional (*functional profile*) – vide Figura 2.2, o qual define uma camada de interface, incluindo as variáveis de rede, propriedades de configuração e os relacionamentos requeridos pelo dispositivo para executar as suas funções de controle (ECHELON, 2002-a).

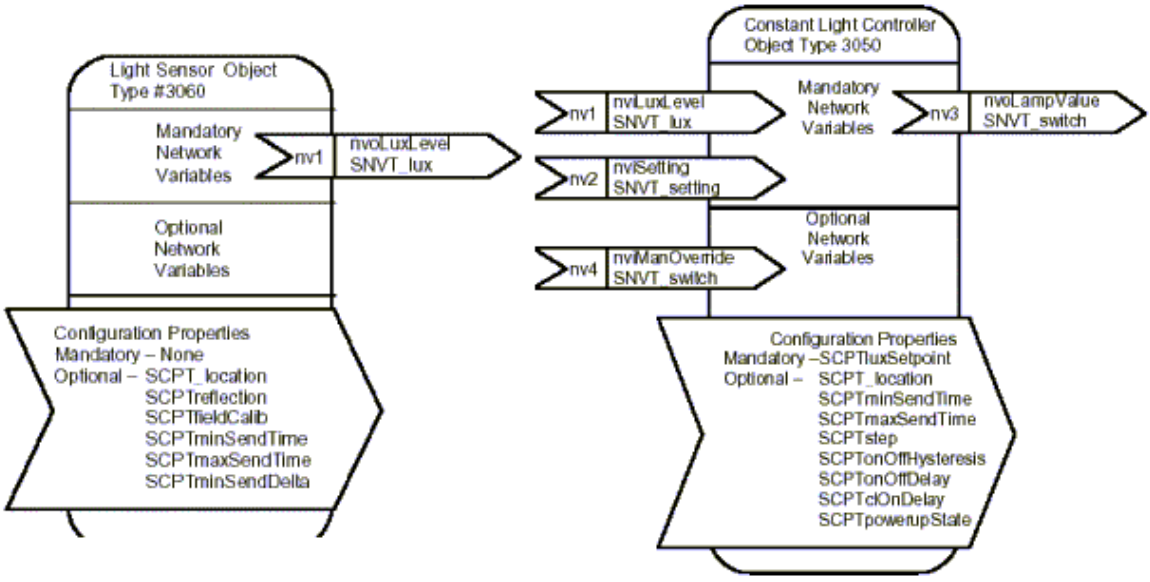


Figura 2.2 : Perfil funcional de um sensor e de um controlador de luminosidade

Os perfis padronizam as funções de um objeto, fornecendo uma descrição comum do comportamento funcional do objeto especificado. Existem perfis padronizados para as áreas de HVAC, gerenciamento de energia, iluminação, acesso, intrusão e monitoração, controle de motores, sensores, refrigeração, detecção de incêndio, transporte de cargas, entre outros.

A interoperabilidade de dispositivos de diferentes fabricantes na rede é garantida por um conjunto padrão de variáveis de rede (*Standard Network Variable Types* – SNVTs). São 166 variáveis padronizadas atualmente (ECHELON, 2002-b), sendo ainda possível que fabricantes definam variáveis de rede não padronizada (*User-Defined Network Variable Types* – UNVTs), que só serão acessíveis a dispositivos que as conheçam previamente.

O programa da aplicação não precisa saber de onde vem uma variável de rede ou para que dispositivo ela irá. Outras camadas do hardware/software são configuradas, durante o projeto da rede, para saber o endereço lógico dos dispositivos que esperam aquela variável. Esse processo de configuração é denominado ligação (*binding)* e cria conexões lógicas entre os nós da rede.

2.2.3 EIB

EIB (*European Installation Bus*) é um sistema aberto, cuja padronização abrange dispositivos, gerenciamento de rede, interfaces para criação de ferramentas e esquemas de certificação de produtos. A tecnologia é administrada por uma associação chamada EIBA, a qual publica o padrão do EIB, realiza certificações e fornece a ferramenta de desenvolvimento denominada ETS (*EIB Tool Software*) a qual é usada para preparar e configurar a instalação, fazer manutenção e diagnósticos.

O sistema é, geralmente, implementado de maneira descentralizada e diferentes meios físicos são disponíveis para construção do barramento (par trançado, fiação elétrica existente (*powerline*), rádio freqüência, etc.). Com o objetivo de reduzir o tráfego na rede, os dispositivos são divididos em linhas e a rede pode ter até 61455 dispositivos (EIB, 1999-a). Uma unidade de acesso ao barramento padronizada (BCU – *Bus Coupling Unit*) permite que os dispositivos sejam fornecidos por diferentes fabricantes.

Um conjunto padrão de objetos é definido a partir dos quais as aplicações são construídas (EIB, 1999-c), conforme pode ser na observado na Tabela 2.1. Para cada um desses objetos está definido um conjunto de propriedades, algumas das quais são obrigatórias e outras opcionais. Objetos padrões também foram definidos para indicação tipos de dados das propriedades, IDs e tipos de objetos (*object-types*).

Tabela 2.1 : Objetos EIB Padronizados

Objeto	Finalidade
Device Object	informações sobre o dispositivo
Addresstable Object	inclui o endereço físico e o endereço de grupo do dispositivo. Fornece operações de gerenciamento para *downloading*
Associationtable Object	conecta *Groupobjects* em endereços de grupo
Applicationprogram Object	contém informações globais sobre o programa de aplicação do usuário
Interfaceprogram Object	especifica um tipo especial de programa de aplicação que gerencia a PEI
EIB-Object Associationtable Object	definição dos objetos escravos de um dispositivo
Router Filtertable Object	inclui informações sobre a *Filtertable* para um roteador
Poling Master EIB Object	Descreve um mestre para um grupo de *polling* específico
File Object	descreve arquivo de dados em um dispositivo
Analog Input/Output/Value-Object	define um objeto cujas propriedades representam as características visíveis de um dispositivo físico ou *hardware* de entrada/saída analógico
Binary Input/Output/Value-Object	define um objeto cujas propriedades representam as características visíveis de um dispositivo físico ou *hardware* de entrada/saída que pode ter apenas dois distintos estados

Objeto	Finalidade
Counter Object	usado para dispositivos de hardware contadores
Loop Object	permite representar *loops* de controle

Para permitir a interoperabilidade entre dispositivos de diferentes fabricantes, estão padronizados os valores e a interpretação de dados incluídos na comunicação entre objetos, os quais serão usados pelos objetos de comunicação. Estas funções padronizadas, são denominadas EIS (EIB *Interworking Standards for Group-Communication-Objects*) (EIB, 1999-b). De uma maneira geral são definidas a descrição e a finalidade de cada função, a faixa de valores admissíveis e a definição da função.

2.2.4 BACNet

BACnet (*Building Automation and Control Network*) é um protocolo aberto de comunicação de dados, desenvolvido por iniciativa da ASHRAE (*American Society of Heating, Refrigerating and Air-Conditioning Engineers*) e adotado pela ANSI. O grupo de trabalho que elaborou o protocolo levou aproximadamente nove anos para definir o padrão. A padronização pela ASHRAE e pela ANSI ocorreu em 1995.

As funcionalidades das camadas de apresentação, sessão e transporte do modelo OSI são implementadas na camada de aplicação ou são eliminadas. Para meio físico foram adotados alguns padrões de LAN já estabelecidos no mercado.

O modelo definido para o BACnet está fundamentado nos seguintes aspectos (SWAN, 2003):

- objetos para representar o sistema de informação e a base de dados com propriedades padronizadas;
- serviços que permitem a um objeto acessar informações de outro objeto;
- regras de interoperabilidade que fornecem mecanismos para que sistemas com componentes de diferentes fornecedores operem corretamente em conjunto, usando requisições padrão de serviço.

Os objetos podem representar uma característica física ou um grupo de características que realizam uma determinada função. Podem representar também conceitos abstratos como um programa, horários e dados históricos. Cada objeto possui um conjunto de propriedades que permite o seu controle. A Tabela 2.2 permite a visualização dos objetos BACnet padronizados.

Tabela 2.2 : Objetos BACnet padronizados

OBJETO	EXEMPLO DE USO	OBJETO	EXEMPLO DE USO
Analog Input	Entrada de um sensor	***Event Enrollment***	Descreve um evento que pode ser uma condição de erro ou um alarme que outro dispositivo de saber. Ele pode chamar um dispositivo ou usando um objeto *Notification Class* chamar múltiplos dispositivos.
Analog Output	Saída de um controlador	***File***	Permite acesso para ler e escrever arquivos de dados suportados pelo dispositivo.
Analog Value	*Setpoint* ou outro parâmetro de um controle analógico	***Group***	Fornece acesso para múltiplas propriedades de múltiplos objetos em uma operação simples de leitura.
Binary Input	Entrada de um dispositivo tipo ON/OFF	***Loop***	Fornece acesso padronizado para um "*control loop*".
Binary Output	Saída à relé	***Multi-state Input***	Representa o estado de um processo de múltiplos estado.
Binary Value	Parâmetro de controle binário (digital)	***Multi-state Output***	Representa o estado desejado de um processo de múltiplos estado.
Calendar	Define uma lista de datas para agendamento.	***Notification Class***	Contém a lista de dispositivos que devem ser informados por um objeto *Event Enrollment* sobre uma mensagem de alarme.

OBJETO	EXEMPLO DE USO	OBJETO	EXEMPLO DE USO
Command	Escreve múltiplos valores para múltiplos objetos em múltiplos dispositivos para atender um propósito especifico (como um modo de emergência).	***Program***	Permite que um programa possa ser iniciado, parado, carregado e descarregado e informa a situação atual do programa.
Device	Propriedades que informam que objetos e serviços o dispositivo suporta e outras informações específicas do dispositivo como fornecedor, revisão de firmware, etc.	***Schedule***	Define um agendamento semanal de operações. Pode usar o objeto *Calendar* para tratar as exceções.

Foram definidos e padronizados dezoito tipos de objetos e 123 propriedades, algumas das quais são obrigatórias e outras opcionais, dependendo do objeto. As propriedades presentes em cada objeto irão depender do tipo de dispositivo no qual o objeto reside e das próprias características do objeto que está sendo implementado.

Os dispositivos comunicam-se através de serviços. Cada requisição de serviço (*service request*) enviada e cada confirmação de serviço (*service acknowledgment*) recebido torna-se um pacote de mensagem que é transferido através da rede, entre os dispositivos. Cada dispositivo implementa os serviços adequados a sua função e a sua complexidade. O serviço *ReadProperty*, por exemplo, deve estar disponível em todos os dispositivos.

Cada programa de aplicação, que é o software que desempenha a operação do dispositivo, recebe as requisições de serviço e as processa. BACnet define 32 serviços e os classifica em cinco categorias, que são: a*larm and event*, *file access*, *object access*, *remote device management*, *virtual terminal services*.

Para garantir interoperabilidade a idéia é dividir o problema da interoperabilidade em conjuntos comuns de funções e definir requisições BACnet para cada um desses conjuntos. Foram definidas cinco áreas áreas de interoperabilidade para classificar os BIBBs (*BACnet Interoperability Building Blocks*) . Em cada área há uma série de serviços que serão executados pelos BIBBs (BUSHBY, 1999): são quatorze para *data sharing*, oito para *alarms and events*, dois para *scheduling*, quatro para *trending* e vinte e sete para *network management*.

Um BIBB pode ser implementado por um ou mais serviços e na execução cada serviço há sempre duas entidades envolvidas: um cliente (dispositivo que pergunta) e um servidor (dispositivo que responde). O servidor deve ser capaz de receber uma requisição de serviço, executá-la e retornar o resultado e um cliente deve ser capaz de iniciar uma requisição de serviço e processar a resposta quando ela chegar.

2.2.5 HomePnP

HomePnP (CIC, 1998), busca estabelecer um padrão dentro do nível de aplicação, definindo o conteúdo das mensagens de controle que são trocadas entre dispositivos e controladores, descrevendo como diferentes produtos podem cooperar entre si. O protocolo fornece todos os detalhes necessários para construir uma mensagem para efetuar uma operação específica em um dispositivo ou subsistema.

O padrão é baseado na CAL (*Common Application Language*) e no nível de aplicação do protocolo CEBus (*Consumer Electronics Bus*). Este protocolo é um padrão EIA (*Electronic Industries Association*) desde 95, EIA 600, após onze anos de trabalho do Comitê CEBus que contou com o envolvimento de mais de 400 empresas de diferentes áreas. A fim de permitir seu uso por diferentes fornecedores de dispositivos, independente do protocolo de comunicação utilizado, a CAL (CIC, 1996), acabou sendo publicada em 1997 como um padrão separado do EIA 600, sob a denominação de EIA 721.

O protocolo HomePnP padroniza a estrutura das mensagens e os códigos de controle usados nas mensagens. O protocolo de aplicação fornece todos os detalhes necessários para construir a mensagem para efetuar uma operação específica num dispositivo ou subsistema, independente do protocolo usado na camada de transporte.

Para garantir que os produtos interoperem entre si, eles devem ser projetados usando o padrão CAL e utilizando os contextos, objetos e variáveis de instância contidos na especificação HomePnP.

Um objeto CAL define uma função de controle simples dentro de um contexto e é implementado por um conjunto de variáveis, chamando variáveis de instância (IVs - *Instance Variables*). A especificação define um conjunto de 22 objetos, a partir dos quais os dispositivos e sistemas são construídos. Um contexto é um grupo de objetos que representa uma função comum. Um dispositivo pode implementar vários contextos podendo ser um hardware com finalidades específicas ou um software em um PC, por exemplo

Estão especificados vários grupos de contexto (*General*, *Audio/Video*, *Lighting*, *Communication*, HVAC, *Security*, *Utility/Energy Management*, *Convenience*, *Computer/Home Office* e *Appliance*). Cada grupo contém um certo número de contextos chamado Classes de Contexto. No grupo *Lighting* há seis classes (*Light Sensor*, *Light*, *Light Scene*, *Light Sensor Status*, *Lighting Scene Request* e *Lighting Scene Status*), no grupo *Environmental* há oito (*Environmental Zone*, *Environmental Sensor*, *Environmental Sensor Status*, etc) e assim por diante.

Cada classe de contexto é composta pelos objetos de acordo com suas características. A classe *Light* por exemplo é composta pelos objetos *Light Level Control*, *Light Level Setting*, *Feature Select* entre outros. Também está especificado um conjunto de métodos que podem executar em vários objetos, tais como: *setOn*, *setOff*, *setValue*, *getValue*, *setArray*, *getArray*, etc.

2.2.6 UPnP

Universal Plug&Play (UPnP) (Microsoft, 2000-a), é uma arquitetura de software aberta e distribuída que permite às aplicações dos dispositivos conectados em rede trocarem informações e dados de forma transparente para o usuário final, sem necessidade de configurar a rede, os dispositivos e o sistema operacional. A descrição dos serviços e dispositivos (*Device and Service Descriptions*) é mantida pelo *Universal Plug and Play Forum*, uma associação composta, atualmente, por 539 membros das áreas de dispositivos eletrônicos, computação, automação residencial, impressão, fotografia, redes de computadores, computação móvel, etc e o seu desenvolvimento está direcionado às redes domésticas e às redes de pequeno porte (*proximity networks*) em pequenas empresas e pequenos prédios comerciais.

Uma das principais características do UPnP é que ele foi construído usando toda uma infraestrutura de protocolos já padronizados e amplamente utilizado pelo mercado para prover as suas diferentes características, tais como TCP, UDP, IP, SOAP, GENA, entre outros. No nível mais alto as mensagens contêm apenas informações específicas do fabricante sobre o seu dispositivo, as quais são complementadas por informações definidas pelo *UPnP Forum*.

Os elementos básicos do UPnP são os dispositivos (*devices*), os serviços (*services*) e os controladores (*control points*), cujas informações são estruturadas utilizando XML (*Extensible Markup Language*).

Um dispositivo (lógico) é um contêiner de serviços e de dispositivos aninhados. Um dispositivo raiz (físico), o qual possui um endereço IP na rede, pode conter outros dispositivos (lógicos) e serviços. Por este motivo o conjunto de serviços que um determinado dispositivo deve prover e as suas propriedades devem ser padronizadas, para garantir interoperabilidade entre diferentes fabricantes de dispositivos.

Um controlador é um componente lógico capaz de descobrir e controlar dispositivos. Após a descoberta ele deve ser capaz de recuperar a descrição do dispositivo e obter a lista de serviços, recuperar a descrição de um determinado serviço, invocar ações para o controlador do serviço e assinar o servidor de evento.

A descrição de um serviço define as ações e seus argumentos, as variáveis de estado com seus tipos, faixa de valores e eventos característicos. Um serviço pode ter zero ou mais ações, as quais podem ter argumentos de entrada e de saída.

Os serviços publicam atualizações quando suas variáveis são modificadas e um controlador pode se inscrever para receber estas informações. Estas atualizações são enviadas através de mensagens de eventos, as quais contém o nome e o estado atual de uma ou mais variáveis de estado. Estas mensagens também são formatadas usando GENA (*General Event Notification Architecture*).

Quando um dispositivo tem uma URL para apresentação, o controlador pode recuperar a página, carregá-la em um *browser* e, dependendo da capacidade da página, permitir ao usuário o controle e/ou a visualização do estado do dispositivo, conforme capacidades especifica ou da página e do dispositivo

A *UPnP Device Architecture* (Microsoft, 2000-b) define o modelo genérico para criação de dispositivos e para descrição de serviços para qualquer dispositivo e tipo de serviço. Grupos de trabalho organizados em comitês trabalham para padronizar os diferentes dispositivos e serviços, criando os modelos adequados para representá-los e os submetendo para padronização. Existem comitês formados para as áreas de A/V, eletrodoméstico, Automação Residencial e Segurança, Imagem e *Gateways* Internet, entretanto ainda existem poucos dispositivos padronizados disponíveis comercialmente.

2.2.7 KNX

O padrão KNX (KNX, 2003) é um padrão europeu que começou a ser especificado em 2000 e tem como objetivo a convergência dos padrões BatiBUS (BatiBUS, 2004), *European Instalation* Bus (EIB) e *European Home Systems* (EHS, 2004). Ele é administrado pela *Konnex Association* e tem como premissa o controle distribuído entre os dispositivos que compõem a rede.

Os principais conceitos do padrão - vide Figura 2.3, são:

- um modelo para construção de aplicações distribuídas para permitir a distribuição das tarefas entre diferentes dispositivos;
- esquemas de configuração e gerenciamentos que permite construir conexões lógicas entre as partes de uma aplicação distribuída que executa em diferentes nós;
- um sistema de comunicação que suporta a configuração, o gerenciamento e o funcionamento da rede, o qual é suportado pela camada denominada *Common Kernel*;
- modelos para construção de dispositivos físicos, organizado em perfis, os quais padronizam a estrutura básica dos mesmos.

Uma das características que o diferencia de todos os demais padrões existentes é que ele utiliza três diferentes mecanismos de configuração:

- *System-mode* (S-mode): neste modo, herdado do padrão EIB, a configuração da rede e a instalação de novos dispositivos utiliza software de configuração, o ETS (EIB Tool Software), permitindo a configuração plena de todos os dispositivos da rede.
- *Easy-mode* (E-mode): herdado do padrão Batibus, neste modo não é necessário ferramenta de configuração, cada componente possui alguns parâmetros pré-definidos porém é possível reconfigurar alguns parâmetros, principalmente *setups* e conexões lógicas.

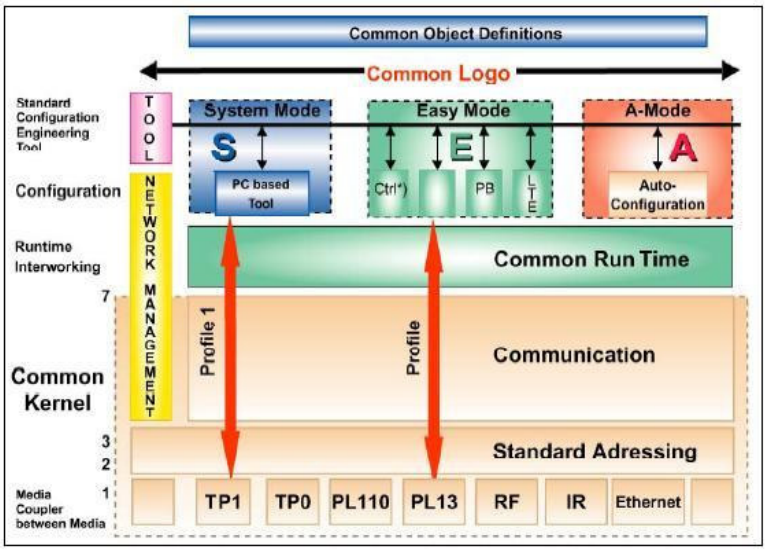


Figura 2.3 : Arquitetura do padrão KNX (KNX, 2003).

- *Automatic-mode* (A-mode): herdado do padrão EHS, permite que ao conectar o dispositivo na rede ele se auto configure (*plug and play*) seguindo um modelo de aplicação pré-definido.

O conceito principal de uma aplicação KNX é o de *data-point*. Eles representam as variáveis do sistema, podendo ser uma entrada, uma saída, um parâmetro, o dado de um diagnóstico, etc. A fim de garantir a interoperabilidade está especificado um conjunto padrão denominado *Standardized Data-point Types* os quais podem ser agrupados formando blocos funcionais (FB – *functional blocks*). Estes blocos funcionais estão relacionados com os campos de aplicação, embora alguns sejam de uso geral, tais como data e tempo. A Figura 2.4 ilustra estes conceitos.

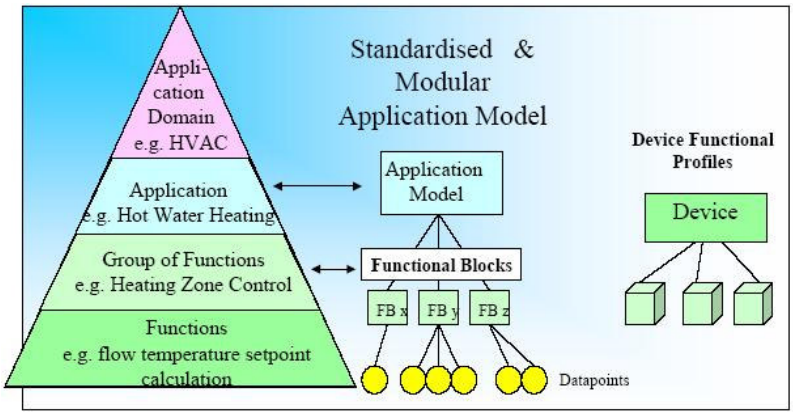


Figura 2.4 : Estrutura do modelo de objetos(KNX, 2003).

É desejável que todo o processo de comunicação esteja baseado no acesso a *data-points* e não a dispositivos, o que garante o acoplamento fraco (independência de aspectos de implementação), desejável em sistemas distribuídos. Uma aplicação basicamente corresponde a um conjunto de envio e recebimento de *data-points*. Todos os dispositivos que mapearam um determinado *data-point* na sua tabela de endereços receberão uma atualização seu valor, sempre que uma mensagem deste tipo for colocada na rede.

Quanto à topologia e endereçamento o padrão segue o modelo proposto pelo protocolo EIB e prevê a utilização de três diferentes meios físicos: o par trançado (TP), a rede elétrica (PL) e radio freqüência (RF). Uma camada, denominada *Common Kernel*, representa o controle da sobre as diferentes camadas da rede.

Na conexão lógica entre *data-points* podem ser usados três esquemas:

- o primeiro é denominado *free binding*: não há definição *a priori* de quais *data-points* devem ser conectados entre si, sendo permitido qualquer configuração das funcionalidades dos dispositivos;
- o segundo é denominado *structured binding*: o modelo da aplicação estabelece um modelo preciso para conexão de um conjunto de *data-points*, os quais correspondem a determinados blocos funcionais ou canais. *Controller* (CTRL), *push-button* (PB) e A-Modes seguem este conceito;
- o terceiro é denominado *tagged binding*. A conexão lógica também é pré-definida no modelo da aplicação, porém o valor de endereço nunca é alterado, uma vez que parte deste valor implica diretamente no *data-point* alvo no dispositivo parceiro na comunicação. O modelo LTE (*Logical Tag Extended mode*) usa este princípio.

2.2.8 Comparação entre os Protocolos

Uma das primeiras limitações que se observa nos protocolos apresentados é o suporte a fluxo contínuo de dados entre dispositivos, característicos de áudio e vídeo: LonWorks, EIB, KNX e BACnet são desenvolvidos para dispositivos de controle e não suportam a distribuição de áudio ou vídeo analógico ou dados digitais, além disso as taxas de transmissão disponíveis atualmente não suportam estas aplicações. HomePnP embora especifique Classes de Contexto para Áudio/Vídeo até o momento ainda não especificou os padrões e os microprocessadores disponíveis no mercado não oferecem este suporte.

O protocolo KNX é um padrão recente em fase final de especificação e como padrão de convergência busca inserir os pontos fortes de cada um dos padrões em que se baseia, embora seja fortemente baseado no padrão EIB, adotando seus principais conceitos. Seu grande diferencial é a possibilidade dos dispositivos oferecerem diferentes modos de configuração que vão desde plug and play, passando por algumas configurações básicas no próprio dispositivo e chegando na configuração plena de todas as funcionalidade utilizando ferramenta de configuração.

Uma característica comum entre diferentes protocolos é a classificação das aplicações na tentativa de padronizar uma estrutura comum a ser utilizada nos dispositivos, a fim de garantir a interoperabilidade entre produtos de diferentes fornecedores. Assim têm-se os perfis nos protocolos LonWorks e EIB, os contextos no HomePnP e os grupos no protocolo UPnP, entre outros.

A Tabela 2.3 apresenta, em ordem alfabética, a classificação adotada pelos protocolos LonWorks, HomePnP e UPnP, onde pode ser observado que existe um conjunto de classes comuns.

Tabela 2.3 : Classificação de dispositivos por diferentes padrões

Perfis Funcionais LONMark	Contextos HomePnP	Grupos UPnP
Access/Intrusion/Monitoring	Appliance	Appliance
Energy Management	Audio/video	Audio/video
Fire & Smoke Device	Communications	Home Automation & Security
HVAC	Computer/Home Office	Image
Industrial	Convenience	Internet Gateways
Input/Output	HVAC	
Lighting	Lighting	
Perfis Funcionais LONMark	**Contextos HomePnP**	**Grupos UPnP**
Motor Control	Security	
Refrigeration	Utility	
Sensor		
Vertical/Conveyor		
Transportation		

A padronização de objetos é muito bem estruturada nos protocolos BACNet, EIB e HomePnP. A Tabela 2.4 apresenta os objetos comuns entre estes protocolos

Tabela 2.4 : Objetos comuns entre alguns protocolos

EIB-Type_Id (dec) EIB Object types	BACnet ID(dec) BACnet Object types	HomePnP ID(hex) HomePnP Object types
0 - Device Objec	8 - Device Object	01 - Node Control
1 - Addresstable Object		
2 - Associationtable Object		
		02 - Context Control
3 – Application program Object	16 - Program Object	
4 - Interface Program		
5 - EIB-Object-Associationtable-Object		
6 - Router Filtertable		
10 - Pollingmaster		
11 - File	10 -File	16 - Data Memory
100 - Analogue-Input	0 - Analogue-Input	08 - Analog Sensor
101 - Analogue-Output	1 - Analogue-Output	07 - Analog Control
102 - Analogue-Value	2 - Analogue-Value	
103 - Binary-Input	3 - Binary-Input	06 - Binary Sensor
104 - Binary-Output	4 - Binary-Output	05 - Binary Switch
105 - Binary-Value	5 - Binary-Value	
106 - Counter		1C - Counter/Timer
107 - Loop	12 - Loop	0A - Matrix Switch
108 - Multistate-Input	13 - Multistate-Input	10 - Multi-position Sensor
109 - Multistate-Output	14 - Multistate-Output	09 - Multi-position Switch
	6 - Calendar	
	7 - Command	
	9 - Event Enrollment	
	11 - Group	
	15 - Notification Class	
	17 - Schedule	
		15 - List Memory
		03 - Data Ch. RCVR
		04 - Data Ch. Trans
		0F - Metter
		10 - Display
		11 - Medium Transport
		13 - Dialer
		14 - Keypad

EIB-Type_Id (dec) EIB Object types	BACnet ID(dec) BACnet Object types	HomePnP ID(hex) HomePnP Object types
		17 - Motor
		19 - Synth/Tuner
		1A - Tone Generator
		1D - Clock

Finalmente o que se observa é que não existe claramente um protocolo superior aos outros: os protocolos mais recentes tentam preencher as lacunas deixadas por limitações dos protocolos mais antigos, visando atingir uma faixa de mercado que eles não atendem bem.

2.3 Frameworks

2.3.1 OSGI

OSGi (*Open Services Gateway Initiative*) (OSGi, 2001) é uma associação fundada em 1999 com a missão de criar especificações abertas para a entrega de múltiplos serviços sobre redes geograficamente distribuídas (WANs) para redes locais e dispositivos. Ela é composta por mais de 60 membros, incluindo provedores de serviço de Internet, operadores de rede, fabricantes de equipamentos, desenvolvedores de softwares, entre outros. Ela está focada na camada de aplicação e é aberta para qualquer protocolo, camada de transporte ou dispositivo.

O componente principal do esforço OSGi foi concentrado na especificação de *gateways* de serviços *(services gateway*), que são servidores inseridos em uma rede para conectar a rede externa com clientes internos, atuando como uma plataforma para instalação e execução de software que trabalha em cooperação com dispositivos em uma rede doméstica. Esse software permite que provedores de serviços externos interajam com os dispositivos dessa rede, provendo serviços para seus proprietários. A Figura 2.5 mostra a localização do serviço num sistema instalado.

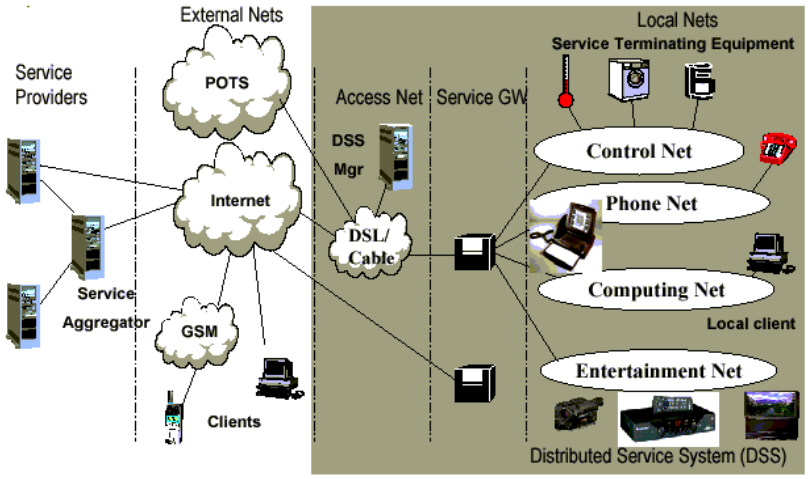


Figura 2.5 : Localização de um *service gateway* (OSGi, 2001).

A idéia principal é interconectar as redes de dispositivos localizados dentro de residências (tais como sensores, interruptores, além de DVDs, aparelhos de som, entre outros) a serviços externos, sendo que fisicamente é proposto um *gateway* entre uma rede externa do tipo conexão de TV a cabo ou DSL e do lado interno redes com os protocolos similares aos analisados no capítulo anterior, permitindo que provedores possam fornecer serviços tais como filmes e músicas sob demanda que possam ser

distribuídos a diversos dispositivos em um prédio, controle e monitoramento do sistema de segurança, gerenciamento dinâmico de energia pela empresa fornecedora, etc.

As especificações OSGi padronizam interfaces de programação que podem ser utilizadas para construir aplicações tipo *gateway*. *Gateways* de serviço abertos devem suportar estas APIs para estarem de acordo com as a especificações. Através do uso dessas APIs é possível endereçar serviços, gerenciar ciclo de vida, controlar a dependência entre os serviços, gerenciar dados, controlar acessos a dispositivos e a recursos e garantir segurança. A especificação divide essas APIs em cinco grupos e define uma estrutura de execução de software projetada para acomodar pacotes de software modulares.

Um *framework* forma o núcleo da especificação da plataforma de serviço OSGI e fornece um ambiente que suporta o desenvolvimento serviços extensíveis que podem ser descarregado via rede que são denominados pacotes (*bundles*). Um pacote é uma entidade colocada à disposição para aplicações baseadas em Java.

Este *framework* fornece um modelo de programação para desenvolvedores de pacotes Java, simplificando o desenvolvimento dos pacotes. Este *framework* é capaz, entre outras tarefas, de gerenciar a dependência entre os pacotes e os serviços.

Qualquer dispositivo padrão OSGi pode buscar e instalar pacotes OSGi e removê-los quando não forem mais necessários. Estes pacotes são construídos a partir de um conjunto de serviços cooperantes disponíveis em um serviço de registro (*service registry*) compartilhado. Um serviço é definido por sua interface (*service interface*) e pelo objeto que a implementa (*service objec*t).

As dependências de uso dos serviços pelos pacotes são gerenciadas pelo *framework*, que mapeia os serviços para seus objetos de serviço subordinados e fornece mecanismos que habilitam um pacote instalado a requisitar os serviços que necessite. Ele fornece também um mecanismo de gerenciamento de eventos para que os pacotes possam receber eventos de objetos de serviços que são registrados, modificados ou desinstalados.

A plataforma oferece ainda um conjunto de serviços para garantia de segurança das aplicações, controle de dispositivo (para localização, instalação e configuração automática de *drivers*), gerenciamento de usuários e servidor HTTP.

2.3.2 Niagara Framework e Baja

O Niagara Framework (TRIDIUM, 2004-a) e (TRIDIUM, 2004-b) é uma infraestrutura de software, desenvolvido pela empresa Tridium, que permite que desenvolvedores construam soluções baseadas em *web* para acesso, automação e controle de dispositivos inteligentes em tempo real, através da Internet. É uma estrutura aberta, baseada em Java, com capacidade de integrar em um ambiente de desenvolvimento diversos sistemas e dispositivos independentes de fabricante, de padrão de comunicação ou software (tais como BACNet, LonWorks, Modbus, DeviceNet, entre outros). Ele é construído a partir de padrões para Internet (Java, TCP/IP, HTTP e XML), sua proposta é tornar possível o controle de dispositivos de qualquer lugar através de um *web browser*.

Sua principal característica é que os dispositivos são convertidos em objetos de software para a construção das aplicações e o *Framework*[1] integra os dispositivos, comunicando-se com eles através da sua rede e de seu protocolo nativos, permitindo a

[1] Framework nesse contexto refere-se ao Niagara Framework™

leitura de dados, o envio de comandos e a utilização de ferramentas de programação para configuração e programação, sem a necessidade de instalar um *gateway* físico entre os diferentes sistemas. Na Figura 2.6 pode-se observar a arquitetura do sistema.

Múltiplas camadas de aplicação, compostas de uma combinação de aplicações gerais, soluções verticais e aplicações específicas podem operar em paralelo. Exemplos de aplicações horizontais são acesso a *web*, segurança, armazenagem de dados históricos e alarmes. Como exemplos de aplicações verticalizadas tem-se automação predial, FMS (*Facility Management Systems*), gerenciamento de manutenção, monitoração remota e diagnósticos, etc.

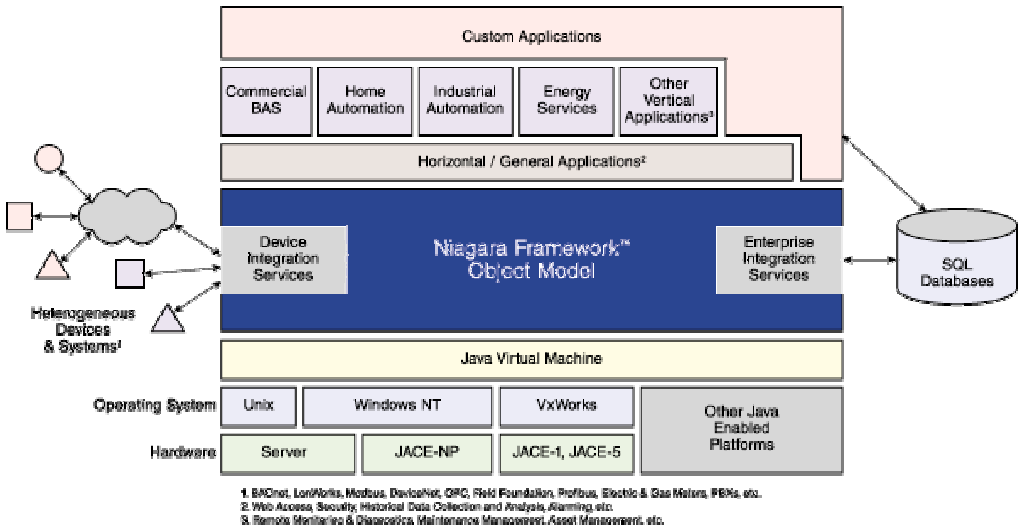


Figura 2.6 : Arquitetura Niagara (TRIDIUM, 2002-a).

O ambiente pode operar com as diversas plataformas de hardware (estações de trabalho Sun, servidores NT, computadores pessoais (PCs) e controladores) e de sistemas operacionais (Wind River's VxWorks RTOS, Microsoft Windows NT ou UNIX) que suportem a máquina virtual Java, escolhidos de acordo com as necessidades específicas da aplicação. Apesar desta portabilidade a empresa comercializa controladores denominados JACE (*Java Application Control Engine*) como dispositivo principal de interconexão entre diferentes sistemas de um prédio.

Uma característica importante do ambiente é o serviço de integração de dispositivos que permite a conexão entre os dispositivos físicos e o modelo de objetos,. Isto possibilita que o ambiente comunique-se com o dispositivo usando o seu próprio protocolo e a sua própria rede.. Os objetos de software usam esses serviços para ler dados de campo dos dispositivos, enviar-lhes comandos e fornecer suporte para reconfiguração e reprogramação dos mesmos. Exemplos de alguns sistemas heterogêneos que podem ser integrados usando esses serviços são: BACNet, LonWorks, Modbus, DeviceNet, OPC, Fieldbus Fundation, Profibus, etc.

Já os serviços de integração de empresa permitem a ligação do ambiente com as aplicações de gerenciamento. Através de tecnologias baseadas em padrões abertos, tais como DCOM, CORBA e XML, essas aplicações podem acessar diretamente os dados coletados dos dispositivos ou calculados pelo ambiente, usando os recursos implementados por esses serviços.

A partir do desenvolvimento do Niagara, a empresa Tridium™ Inc passou a liderar junto com a Sun Microsystems um grupo formado por empresas fornecedoras de sistemas de automação tais como, Siemens, Echelon e Johnson Controls, entre outras,

com a tarefa de criar uma plataforma aberta, padrão Java, para o mercado de automação predial, a qual foi denominada Baja (*Building Automation Java Architecture*), totalmente baseada nas características Niagara.

Baja (TRIDIUM, 2004-c) define um ambiente de aplicação de automação predial com especificações que descrevem um conjunto de APIs Java e esquemas XML para aplicações de sistemas de controle interoperáveis. Ela define uma arquitetura Java padrão para controladores programáveis, uma arquitetura que permite interoperabilidade entre diferentes fornecedores de software e dispositivos heterogêneos, um modelo que permite a usuários construir aplicações de controle e a capacidade de programar uma aplicação quando ela está em execução.

O principal objetivo é que os desenvolvedores venham a ter um *framework* aberto para desenvolver aplicações de software que permita além do funcionamento perfeito de sistemas e dispositivos heterogêneos de diferentes fornecedores, a compatibilidade da aplicação com a Internet.

2.3.3 O Projeto EMBASSI

EMBASSI (EMBASSI, 2004) é um projeto financiado pelo Ministério da Educação e Pesquisa Alemão integrado por universidades e empresas com o objetivo de desenvolver novos paradigmas e arquiteturas para interação intuitiva com dispositivos eletrônicos presentes em ambientes tais como automóveis, residências, terminais públicos, etc. O objetivo principal é pesquisar novos métodos para interação homem-máquina, especialmente no que se refere à sistemas distribuídos.

As duas principais mudanças de paradigmas que estão envolvidas neste projeto são a transição da interação com dispositivo orientada a função para interação com o sistema orientada a objetivo e a transição das estruturas de diálogo unimodal, com vocabulário fornecido pelo sistema, para interação multimodal, com um vocabulário de interação ilimitado fornecido pelo usuário, ou seja, desenvolver técnicas de projeto de sistemas cujo foco esteja no usuário. Assim o projeto busca desenvolver uma arquitetura que forneça assistência inteligente, interação multímodo e interfaces antropomórficas dentro de um *framework* padrão.

Para isto foi especificada uma camada capaz de estender um padrão de hardware existente, desenvolvido dentro do conceito de orientado a função, para o de centrado no usuário, com a interação baseada no objetivo a ser atingido e utilizando técnicas de Inteligência Artificial para aquisição de conhecimento sobre estes sistemas. Esta camada tem por finalidade linearizar a interação com os sistemas físicos, interagindo com o usuário através de comandos naturais ou intuitivos e evitando que ele tenha que aprender em termos de funcionalidade dos dispositivos.

A arquitetura - vide Figura 2.7, pode ser dividida em três grandes níveis:

- interação multimodal: responsável pelo estabelecimento dos objetivos a partir da interação com o usuário;
- assistência: responsável pelo estabelecimento e acompanhamento dos planos traçados com objetivo de atingir a meta estabelecida no nível anterior;
- efeito: responsável por controlar os dispositivos físicos que interagem sobre o ambiente.

Cada nível consiste de um número de processos (componentes) que podem ser adicionados ou removidos dinamicamente e que cooperam coletivamente para implementar a função do nível, o que permite, por exemplo, selecionar componentes para construir sistemas específicos.

O componente de gerenciamento de contexto é responsável pelo gerenciamento da visão do sistema sobre o mundo (informações sobre o usuário, recursos e ambiente) e pelo controle dos componentes do *framework*.

Diversas funções intermediárias foram definidas neste caminho. Primeiro, a interação com dispositivos físicos de entrada (componentes I - *Input*) é transformada em eventos de interação atômicos (*lexical level*), os quais são os menores blocos a partir dos quais as interações compostas podem ser construídas.

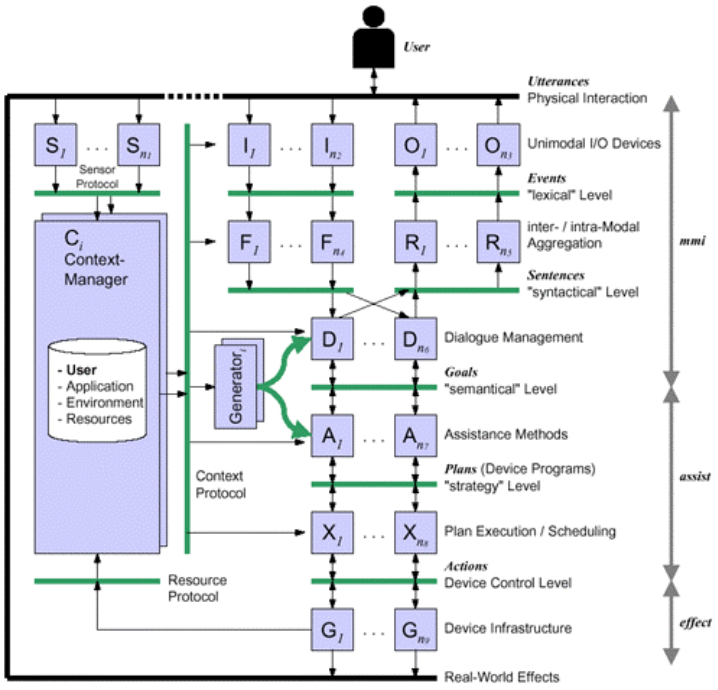


Figura 2.7 : Arquitetura EMBASSI (EMBASSI, 2004).

A seguir eles são enviados a componentes F (*Filters*) responsáveis por agregar estes eventos em sentenças amodais. Estas sentenças são recebidas por componentes D (*Dialogue Manager*) os quais são responsáveis por transformá-las em objetivos e também pelo relacionamento entre as diferentes sentenças recebidas. O processo reverso ocorre quando uma saída para o usuário for produzida.

Na próxima etapa ocorre o mapeamento do objetivo em alteração no ambiente. O primeiro componente desta fase, o componente A (*Assistant*) estabelece estratégias, denominadas planos, que visam alcançar-se o objetivo recebido. Não existe uma maneira pré-definida para estabelecer um plano, cada componente pode usar uma estratégia diferente.

Os planos resultantes são enviados aos componentes X (eXecution Control) que serão responsáveis por garantir a seqüência correta dos passos do plano, pela execução paralela de múltiplos planos e pelo gerenciamento da execução de recurso necessários para o plano.

Finalmente as requisições individuais são enviadas para dispositivos abstratos, os componentes G, que controlam os dispositivos físicos, os quais mudam o estado do ambiente e provocam o efeito requerido pelo usuário.

Considerando a dinamicidade do sistema, novos componentes e novos objetivos podem ser inseridos a qualquer momento e novos comandos devem poder ser inseridos dinamicamente na meta linguagem que representa os objetivos. O componente *Generator* é responsável por interpretar as definições da metalinguagem para

especificação das funções e objetivos e pela atualização de componentes A e D afetados pela linguagem.

Diferentes técnicas, protocolos e linguagens são utilizados na representação e construção dos componentes. Detalhes podem ser encontrados em (EMBASSI, 2004).

2.3.4 Considerações Finais sobre *Frameworks*

A plataforma OSGI é concebida para permitir a conexão de dispositivos de diferentes padrões num *gateway* residencial, com o objetivo de permitir que diferentes provedores forneçam serviços aos usuários que executem nesses dispositivos. O foco do desenvolvimento é para conexão de sistemas externos com sistemas internos, buscando criar uma infraestrutura de suporte a provedores de serviços para aplicações tais como monitoração, áudio e vídeo sob demanda, entre outros. O grupo de empresas associadas, fortemente vinculado à área de Telecomunicações, deixa evidente este objetivo.

A partir do material disponível para estudo, pode-se concluir que o Niagara conceitualmente comporta-se como um *midleware* baseado em Java. Diferentes protocolos são mapeados para uma estrutura comum, sobre a qual podem ser realizados processamentos e os resultados são convertidos novamente na estrutura padrão da tecnologia. Parte desta estrutura deve corresponder ao padrão BAJA, porém durante a escrita deste trabalho ainda não estava disponível a sua especificação.

O projeto EMBASSI por sua vez busca atacar o problema de Interface Homem Máquina, onde máquina num sentido mais amplo representa o sistema como um todo. Neste projeto o sistema não é mais visto como um conjunto de funcionalidades que devem ser manuseadas pelo usuário para atingir o seu objetivo. A proposta é passar para o *framework* a definição das funcionalidades e dispositivos que serão ativados para atingir uma meta estabelecida pelo usuário, poupando-o da tarefa de realizar o controle do sistema. Este *framework* deve incorporar sistemas de IA a fim de superar a limitação de recursos da maioria dos dispositivos e sistemas disponíveis atualmente. Outra questão sob o foco do projeto e o estudo de interfaces naturais de interação com o usuário.

2.4 Metodologias de Projeto Baseada em Visões

A crescente complexidade dos atuais sistemas de automação predial impõe a necessidade de subdividi-los a fim de conseguir-se capturar corretamente os seus requisitos e implementá-los de forma eficiente.

Ao mesmo tempo, a existência de diversos protocolos e de diferentes *frameworks* e *midlewares,* aponta para a necessidade de desenvolvimento de sistemas abertos que permitam a interoperabilidade entre diferentes equipamentos e sistemas, independente da tecnologia de comunicação que eles implementam.

Esta seção tem por objetivo tratar dessas duas questões no que se refere ao desenvolvimento de Sistemas de Automação Predial e analisar alguns trabalhos que se enquadram nesses conceitos.

2.4.1 O Conceito de Visão

Uma abordagem usual no desenvolvimento de software, cujo objetivo é simplificar a análise e o desenvolvimento de sistemas complexos, é dividi-lo em camadas. Partindo-se da especificação de requisitos inicial, cada camada subseqüente representa perspectiva menos abstrata do sistema.

Esta multiplicidade de perspectivas no projeto de software é suportada pelo conceito de ponto de vista ou visão (*viewpoint*) (SOMMERVILLE, 1997). Um ponto de vista é o encapsulamento de informações parciais sobre as requisições de um sistema e a integração de diferentes visões compõe a especificação final do mesmo.

Dentre as vantagens dessa abordagem, destacam-se:

- pontos de vista podem ser utilizados para colecionar e classificar os diferentes tipos de informações necessárias para a especificação, tais como informação sobre o domínio da aplicação, sobre ambiente e sobre desenvolvimento do sistema;
- pontos de vista podem ser utilizados para encapsular modelos diferentes do sistema onde cada um fornece uma parte da especificação.

Este conceito é utilizado, por exemplo, em (DOUGLASS, 2003), na abordagem de padrões de projeto para sistemas de tempo real. Douglass propõem cinco visões para a divisão da arquitetura física de um sistema de tempo real: subsistemas, distribuição, concorrência, tolerância à falhas e mapeamento físico.

Outros exemplos de visões já bastante usuais no desenvolvimento de sistemas de informação são os de segurança, de marketing, de base de dados e de rede, entre tantos possíveis.

2.4.2 O modelo de Referência para Processamento Distribuído Aberto

O RM-ODP (*Reference Model for Open Distributed Processing*) – vide (ISO/IEC, 1995), é o resultado de um trabalho conjunto da ISO/ITU para estabelecer um *framework* de desenvolvimento de sistemas distribuídos heterogêneos de grande escala em ambiente multiplataforma, baseado no conceito de visões. Ele define cinco visões para representação de um sistema distribuído aberto: a visão de empresa, a visão de informação, a visão de computação, a visão de engenharia e a visão de tecnologia.

Visão de empresa:

O objetivo principal desta visão é especificar os objetivos e as restrições do sistema de interesse. Para isto ele é representado por um ou mais objetos dentro de uma comunidade de objetos que representam a empresa e pelas funções (*roles*) nas quais estes objetos estão envolvidos. Estas funções representam, por exemplo, usuários, proprietários, fontes de informação processadas pelo sistema, etc.

Algumas definições importantes desta visão são:

- **Políticas**: conjunto de regras associadas com um determinado propósito. Políticas registram regras na qual ações de um objeto são permitidas ou proibidas e também que ações os objetos são obrigado à executar;
- **Obrigação**: a prescrição de que um determinado comportamento é requerido. Uma obrigação é executada pela ocorrência do comportamento prescrito;
- **Permissão**: a prescrição de que um determinado comportamento é permitido que ocorra;
- **Proibição:** a prescrição de que um determinado comportamento não deve ocorrer.

A idéia principal desta visão é a de um contrato, ligando os objetos realizadores de diferentes funções e expressando suas obrigações mútuas.

Visão de informação:

Esta visão contém os conceitos para permitir a especificação da semântica da informação manipulada e armazenada em um sistema de processamento distribuído

aberto. A especificação da informação consiste de um conjunto de esquemas os quais estão relacionados com os dados que necessitam ser armazenados e processados.

Visão de Computação:

O sistema descrito como um conjunto de objetos que interagem através de suas interfaces. A interação entre os objetos é definida como assíncrona e é classificada em:

- Sinal: interação atômica elementar;
- Operação: similar a procedimentos (*procedures)* que são invocadas através de interfaces. É classificada em interrogação quando há retorno de resposta e anúncio quando não espera resposta;
- Fluxo: seqüência contínua de dados entre interfaces.

Também está definido que a interação entre interfaces somente é possível se for estabelecida uma conexão (*binding*) explícita entre elas (Operações permitem *binding* implícito) e que um objeto pode estar envolvido simultaneamente em diversas conexões.

Visão de Engenharia

Relacionada aos mecanismos de suporte de distribuição do sistema. Define a estrutura computacional que suporta a estrutura definida na visão anterior, promovendo os diversos níveis necessários para suporte à distribuição, ou seja, como a distribuição é alcançada e quais recursos necessários para isto, definindo regras para estruturar canais de comunicação entre os objetos e estruturar os sistemas visando o gerenciamento de recursos.

Cada objeto computacional deve corresponder a pelo menos um objeto de engenharia, podendo corresponder a mais de um em caso de replicação.

Estes objetos podem ser agrupados em clusters, cápsulas e nós. Objetos são agrupados em clusters a fim de reduzir o custo de interação e o custo de manipulação entre eles. As cápsulas geralmente correspondem à menor unidade de Sistema Operacional (ex . todos os objetos da cápsula correspondem a um único processo). Os objetos de engenharia são conectados através de canais.

Visão de Tecnologia

Esta visão está associada aos detalhes dos componentes construtivos do sistema (escolha da tecnologia de hardware e software para composição do sistema).

O modelo também define as seguintes funções:

- Função de gerenciamento do nó: realiza tarefas de gerenciamento de *threads*, escalonamento, etc;
- Função de gerenciamento da cápsula/cluster: responsável pela criação, desativação, reativação e recuperação desses objetos;
- Função de replicação: responsável pela replicação de objetos e pela consistência das réplicas;
- Função de transação: define as ações de interesse, verifica as consistências, estabelece as dependências e verifica a recuperação de falhas;
- Função de segurança: responsável pelos serviços de segurança, controle de acesso, fiscalização aplicados a objetos e interações entre eles.

Finalmente define que num sistema distribuído deve existir transparência quanto a acesso, persistência, localização, relocação, migração e falha.

2.4.3 Alguns Trabalhos que Utilizam o Conceito de Visões e o RM-ODP

Alguns trabalhos que utilizam os conceitos de visões e o RM-ODP em diferentes contextos podem ser encontrados em (GOEDICKE, 1999), (KANDÉ, 1998), (BASTIDAS, 2003) e (BASTIDAS, 2004).

Goedicke, em (GOEDICKE, 1999), propõe uma ferramenta que suporte o desenvolvimento de software orientado a visões com o objetivo de verificar a consistência dentro de cada visão e entre as visões durante o processo de especificação.

Para isto cada visão é dividida em *slots* que representam diferentes perspectivas da mesma. Os *slots* definem os esquemas e a notações para representar a visão, sua área de domínio, a própria especificação da visão de acordo com a notação definida, as ações que ela executa (*workplans*). Dois *slots* são responsáveis por verificar a consistência dentro as diferentes visões e dentro de uma determinada visão.

O foco desse trabalho e fornecer uma metodologia que detecte inconsistências, usando um mecanismo formal de transformações gráficas a partir da manipulação dos esquemas definidos nos *slots*.

Kandé (KANDÉ, 1998) apresenta uma proposta para modelar as visões do RM-ODP usando UML, apontando os diagramas que podem ser usados para modelar cada visão.

A proposta é desenvolvida a partir em um estudo de caso, a implementação de um sistema de telecomunicações e a especificação (visão de empresa) é realizada com o foco numa solução para este sistema. O ponto de partida é a definição dos domínios da aplicação e a modelagem é baseada nesses domínios.

A visão de computação expande os diagramas de classe UML definidos na visão de informação com foco na definição das interfaces entre as classes. Diagramas de seqüência e de interação UML complementam esta visão com o objetivo de representar a interações entre os objetos do sistema.

Na visão de engenharia são usados os diagramas de componentes e os *deployments* e para representar o mapeamento tecnológico ele propõe o uso texto plano.

Batisdas (2003; 2004) aborda a utilização do RM-ODP para a especificação de sistemas abertos, UML para representá-los e redes de Petri para assegurar a coerência formal entre as diferentes fases de especificação, usadas para modelar a dinâmica do sistema.

Embora os trabalhos enfatizem a aplicação de uma metodologia para modelar sistemas de automação predial, os estudos de casos são realizados usando-se apenas um subsistema de controle de incêndios, não fazendo considerações sobre a generalização para outros subsistemas.

Em Bastidas (2004) não existe a preocupação de relacionar os diagramas UML com visões especificas do RM-ODP. A definição e construção dos diagramas de classe é, genericamente, tratada como um atividade relacionada com os pontos de vista de informação e computação, sendo o comportamento dos objetos especificados através de redes de Petri.

O trabalho não aborda a representação das interações entre os objetos e em ambos os trabalhos a análise pára na visão computacional: a representação da implementação ainda não está resolvida.

3 PROPOSTA DE *FRAMEWORK*

Neste capítulo é definida a proposta de *framework* orientado a objetos para o desenvolvimento de projetos de sistemas de automação predial, tema da presente dissertação de mestrado. Para especificação deste *framework* será utilizado a *Unified Modeling Language* (UML) (OMG, 2003). Embora existam outras opções para representação, tais como blocos funcionais (CHRISTENSEN, 2004), escolheu-se a UML por ser esta um padrão já consolidado (durante a escrita deste texto encontra-se na versão 1.5) e largamente utilizado, ter ampla bibliografia disponível e também pela experiência do grupo de pesquisa no qual este trabalho foi desenvolvido.

3.1 Introdução

Segundo (GAMMA, 2000) um *framework* é um conjunto de classes cooperantes que constroem um projeto reutilizável para uma específica classe de software que possui as seguintes características:

- dita a arquitetura da aplicação;
- define a divisão em classes e objetos e
- define as responsabilidades chaves das classes e objetos: como elas colaboram, os fluxos de controle, etc.

Ele reduz as decisões de projeto, pois as aplicações passam a ter estruturas similares, permitindo construir soluções mais rapidamente. O projetista aposta que a arquitetura especificada funcionará para todas as aplicações do domínio, sendo exatamente esta arquitetura a principal contribuição de um *framework*.

Neste trabalho objetiva-se, a partir da descrição funcional de um sistema de automação predial, selecionar dentre os objetos especificados no *framework* proposto, aqueles que melhor representem as funcionalidades requeridas por uma dada aplicação e conectá-los de forma a atender a especificação funcional da mesma.

Esta etapa é denominada projeto lógico e é independente da tecnologia que será utilizada para implementação do sistema, uma vez que o mapeamento tecnológico somente irá ocorrer na etapa final do projeto, após a conclusão do projeto lógico.

Os objetos utilizados no projeto lógico são denominados, no âmbito deste trabalho, dispositivos lógicos. O mapeamento tecnológico corresponde a agrupar os dispositivos lógicos especificados no projeto lógico de maneira conveniente, conforme funcionalidades oferecidas pelos dispositivos físicos disponíveis. A Figura 3.1 representa, de forma simplificada, as etapas principais de projeto propostas.

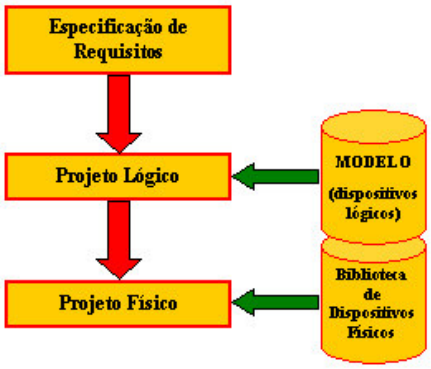


Figura 3.1 : Etapas de projeto

A metodologia de projeto proposta neste trabalho é baseada no conceito de visões, porém usa para definição das visões do RM-ODP, descrito no capítulo anterior, por ser um padrão estabelecido por um órgão de padronização internacional. Ao longo desse capítulo o sistema de automação predial será dividido em camadas de acordo com os conceitos definidos no modelo de referência.

3.2 Construção do *Framework*

O ponto de partida para realização do *framework* proposto neste trabalho foi a análise do modelo de objetos dos diversos padrões abertos disponíveis para implementação de sistemas de automação predial descritos no capítulo anterior. Esta análise buscou identificar conceitos comuns entre diferentes padrões, a fim de identificar um conjunto mínimo de objetos que pudessem ser mapeados para as diferentes tecnologias disponíveis.

Esta análise foi importante na definição da hierarquia de classes do modelo, a qual deverá permitir representar as informações de um sistema de automação predial, etapa correspondente à visão de informação no RM-ODP e que será detalhada posteriormente neste capítulo.

Ao mesmo tempo fez-se também uma análise *bottom-up* dos sistemas de automação predial, buscando identificar requisitos e comportamentos típicos desses sistemas. O restante do capítulo define uma proposta de *framework*, construído a partir dessas duas abordagens, seguindo as definição do RM-ODP.

3.2.1 Definição das Funcionalidades

Tendo como referência o RM-ODP, a primeira visão (camada) do projeto de um sistema tem por finalidade especificar os objetivos e as restrições do sistema de interesse.

Um sistema de automação predial tem por objetivo controlar as diferentes funcionalidades do ambiente sobre controle. Algumas funcionalidades típicas de uma instalação predial são:

- Manter a temperatura, a umidade e qualidade do ar em valores pré-determinados. Estas funcionalidades podem ser refinadas posteriormente de acordo com as condições ambientais em aquecer, esfriar, ventilar, exaurir, desumidificar, umidificar, etc;
- Manter a luminosidade em valores pré-estabelecidos, o que pode resultar em aumentar e diminuir a luminosidade conforme as condições ambientes

- Controlar acesso que pode ser refinado em identificar e autenticar o usuário, autorizar ou negar o acesso, etc;
- Identificar acessos não autorizados que poder ser refinado em detectar e comunicar intrusão;
- Controlar Fogo que pode ser refinado em detectar, comunicar e controlar fogo: controlar portas corta fogo, controlar *sprinkler*, controlar *dumpers*, etc;
- Controlar interação entre ambientes que pode ser refinado em abrir e fechar cortinas, janelas, brises, portas, portões, etc.

As funcionalidades do mesmo tipo são usualmente agrupadas formando subsistemas. Na definição de cada subsistema faz-se necessário identificar quais são as variáveis que o compõem e quais estarão relacionadas com cada funcionalidade do subsistema. A estratégia que será utilizada no controle é importante na definição das variáveis de interesse do controle e também para a definição do tipo de dados que será utilizado para representá-las. A Figura 3.2 ilustra a estruturação funcional de um subsistema Y para controle de um determinado ambiente X.

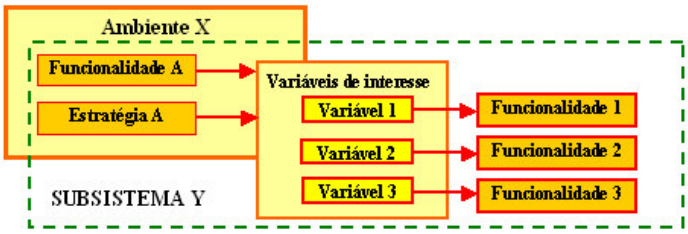


Figura 3.2 : Estruturação funcional de um subsistema.

Suponha-se que a definição dos requisitos especificasse a necessidade de manter a temperatura em um valor predeterminado e que a análise das condições locais identificasse a necessidade de resfriar o ambiente, pois a temperatura externa é sempre superior a temperatura determinada para o mesmo.

E que, adicionalmente, tivesse sido definido como estratégia de controle que se o ambiente estivesse desocupado ou abaixo de uma determinada temperatura o sistema de resfriamento devesse ser desligado.

A partir desses requisitos identifica-se, neste exemplo, as seguintes funcionalidades: resfriar (representada pela funcionalidade A), controlar a temperatura interna (representada pela funcionalidade 1), controlar a temperatura externa (representada pela funcionalidade 2) e controlar a ocupação do ambiente (representada pela funcionalidade 3), estas últimas definidas em função das variáveis de interesse temperatura interna, temperatura externa e ocupação, necessárias para implementar a estratégia de controle definida.

Numa análise inicial identifica-se que as diferentes variáveis envolvidas no processo podem ser classificadas em três grupos distintos:

- **variáveis de processos:** temperatura, umidade, luminosidade, ocupação, ruído, qualidade do ar, consumo de energia (eletricidade, gás, água quente/fria), presença de gases (fumaça, CO_2, etc), indicação de vazamentos (água, gás, etc), identificação de defeitos em equipamentos, faltas (água, gás, eletricidade, etc), entre outras;
- **variáveis temporais:** data, horário, intervalos, temporizações (retardos - *delays*/ temporizações - *timers*), etc;
- **variáveis relativas a interfaces com usuários:** identificadores, mensagens de autenticação, notificações e configurações, comandos, entre outras.

Estas variáveis possuem atributos específicos conforme o seu tipo (resolução, faixa de valores, etc) e diferentes funcionalidade podem estar associadas a uma variável, de acordo com a estratégia de controle a ser implementada para o ambiente.

Assim, por exemplo, a detecção de que um ambiente passou do estado desocupado para ocupado, através da mudança do valor de uma variável que esteja associada à ocupação do ambiente, pode dar início a diferentes ações no que se refere ao controle de luminosidade, temperatura e controle de acesso/intrusão no ambiente.

Cada subsistema corresponde a um domínio com funções de controle específicas e é desejável que eles interajam entre si a fim de atingir plenamente seus objetivos. A Figura 3.3 apresenta os subsistemas que foram definidos no escopo deste trabalho e especifica as interações que ocorrem entre eles.

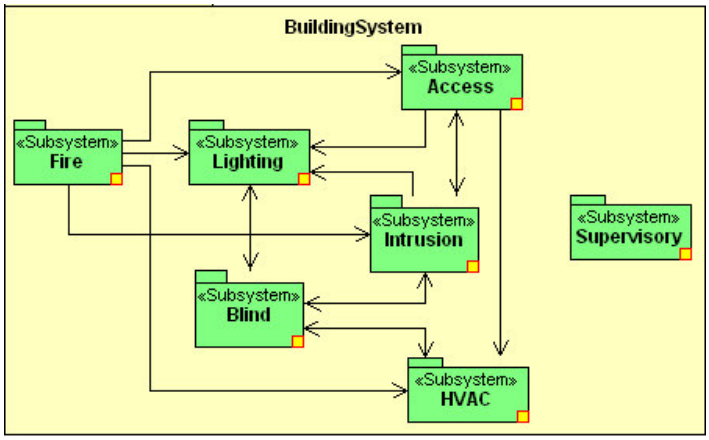


Figura 3.3 : Definição dos Subsistemas

As principais funcionalidades de cada um desses subsistemas são:

- Supervisório (*supervisory*): responsável pelos mecanismos de interação com os usuários (login, autenticação, notificações, etc.) e pela armazenagem e recuperação de dados (configurações, eventos, alarmes);
- Controle de Iluminação (*lighting*): deve permitir o controle de iluminação no ambiente em diferentes condições de uso;
- Controle de HVAC (HVAC): deve permitir o controle da temperatura, da umidade e da ventilação do ambiente, conforme condições pré-estabelecidas;
- Controle de Acessos (*Acess*): responsável pela identificação e autenticação do usuário, permitindo ou não o acesso a áreas restritas;
- Controle de Intrusões (*intrusion*): responsável pela identificação de acesso não autorizado a ambientes e equipamentos;
- Controle de Venezianas (*Blind*): responsável pela da manipulação de cortinas, persianas, brises, portões, etc.
- Controle de Incêndio (*Fire*): responsável por manter o controle a condições ambientais que causem ameaça a segurança de pessoas e instalações, relativamente a controle de gases perigosos e, especialmente, fogo.

Outros domínios típicos que não foram modelados neste trabalho são o de Entretenimento, o Gerenciamento de Energia, o de Controle de Serviços e o de Movimentação. Este procedimento foi adotado tendo em vista que durante a realização dos estudos preliminares para definição do *framework* constatar-se que eles podem ser refinados usando-se os mesmos conceitos adotados para os demais subsistemas.

Para cada subsistema foi determinado um conjunto de funcionalidades típicas. A Figura 3.4 apresenta o diagrama UML de caso de uso que representa as funcionalidades do subsistema HVAC. Para os demais subsistemas os diagramas de caso de uso poderão ser encontrados no Anexo A.

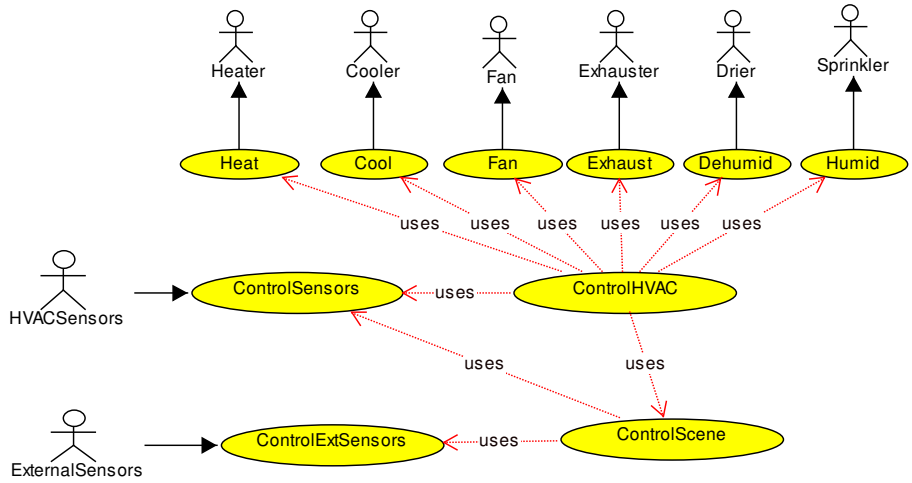


Figura 3.4 : Funcionalidades do subsistema de HVAC

Nesta figura pode-se observar que para este subsistema foram definidas funcionalidades de aquecer (*Heat*), resfriar (*Cool*), ventilar(*Fan/Exhaust*) e aumentar e diminuir a umidade (*Humid/Dehumid*).

Algumas variáveis, tais como temperatura e umidade, estão relacionadas diretamente com a função de controle (*ControlHVAC*) e estão sendo representadas pelo ator *HVACSensors* enquanto outras, tais como as associadas à data, horário ou ocupação do ambiente, definem condições funcionamento (*ControlScene*) para o subsistema e estão sendo representadas pelo ator *ExternalSensors*.

As setas na Figura 3.3 indicam a interação entre os subsistemas. A seguir serão descritas as principais finalidades destas interações:

- Controle de Acessos x Controle de Iluminação: possibilita que o subsistema de controle de iluminação responda a eventos do controle de acessos, possibilitando o nível de iluminação adequado apenas nos ambientes cujo acesso foi autorizado. Permite ainda a personalização dos parâmetros de iluminação, conforme identificação do usuário;
- Controle de Acessos x Controle de HVAC: possibilita que o subsistema de controle de HVAC responda a eventos do controle de acessos, possibilitando o nível de temperatura e umidade adequado apenas aos ambientes cujo acesso foi autorizado. Permite ainda a personalização desses parâmetros, conforme identificação do usuário;
- Controle de Acessos x Controle de Intrusão: possibilita que o subsistema de controle de intrusão seja desativado em áreas cujo acesso tenha sido autorizado pelo controle de acessos;
- Controle de Iluminação x Controle de Venezianas: esta interação possibilita que estes subsistemas cooperem a fim de ajustar o nível de iluminação externa ao nível especificado para o ambiente;
- Controle de HVAC x Controle de Venezianas: esta interação possibilita que estes subsistemas cooperem a fim de ajustar os níveis de temperatura e de umidade levando em conta parâmetros externos ao ambiente sob controle;

- Controle de Intrusão x Controle de Iluminação: permite que o subsistema de controle de iluminação responda a eventos do subsistema de controle de intrusões, garantindo nível adequado de iluminação de acordo com a especificação, em ambientes cuja intrusão tenha sido detectado;
- Controle de Intrusões x Controle de Venezianas: esta interação possibilita que estes subsistemas cooperem a fim de detectar intrusões e realizar isolamentos do ambiente controlado do ambiente externo, conforme a especificação de requisitos do projeto;
- Controle de Intrusões x Controle de Acessos: esta interação possibilita realizar isolamentos do ambiente controlado do ambiente externo, em caso de detecção de intrusão;
- Controle de Incêndios x Controle de Acessos: esta interação possibilita realizar isolamentos do ambiente controlado, bem como liberar saídas de emergência;
- Controle de Incêndios x Controle de Intrusão: esta interação permite liberar acessos das zonas onde tenha sido detectado incêndio, conforme especificação;
- Controle de Incêndios x Controle de HVAC: esta interação possibilita o controle da refrigeração e da ventilação, em caso de detecção de fogo, conforme especificação da instalação;
- Controle de Incêndios x Controle de Iluminação: esta interação possibilita realizar o controle da iluminação de emergência, da iluminação de acessos e das indicações de saídas, entre outros, em caso de detecção de fogo.

A especificação dos requisitos de um sistema do sistema de automação predial, com as definições das condições de funcionamento e a parametrização das suas variáveis, é realizada na fase de especificação da instalação. A partir dessa caracterização pode-se definir quais subsistemas do *framework* serão utilizados, selecionar quais funcionalidades de cada subsistema são necessárias para atender a especificação e definir as parametrizações e as estratégias de controle de cada funcionalidade.

Por exemplo uma funcionalidade especificada como “manter a temperatura em 22^0C”, será refinada de maneiras diferentes, conforme a temperatura ambiente típica do local onde o sistema será implantado. Se a temperatura ambiente for sempre acima da especificada deverá ser modelada como a funcionalidade “resfriar”, caso contrário “aquecer” e ainda se ela variar acima e abaixo desse valor terão que ser modeladas as duas funcionalidades.

A definição de outras funcionalidades que serão necessárias para atender esta funcionalidade irá depender das estratégias de controle ou condições de funcionamento especificadas. Assim, por exemplo, nesse refinamento pode-se ainda acrescentar funcionalidades tais como verificar a ocupação do ambiente, o dia da semana ou o horário para atender às especificações de controle.

3.2.2 Definição do Meta-modelo

Definidos os subsistemas, suas funcionalidades e a interação entre eles, faz-se necessário definir os conceitos e as estruturas que serão usadas para manipular e armazenar as informações do sistema de automação predial. Esta camada do *framework*, focada no significado da informação, corresponde à visão de informação no RM-ODP.

O conceito principal adotado no *framework* para representar as informações é o de dispositivo lógico (*LogicalDevice*). Um dispositivo lógico corresponde a um componente que representa uma funcionalidade lógica, possuindo atributos típicos dessa funcionalidade, independente de como ele será implementado posteriormente.

Todo o sistema pode ser representado como um conjunto de dispositivos lógicos conectados logicamente entre si a fim de permitir a interação entre eles.

A Figura 3.5 ilustra dispositivos lógicos conectados entre si a fim de modelar um controle de temperatura e um controle de luminosidade. As conexões dependem da estratégia de controle que será implementada para o ambiente. Eles estão sendo representados por figuras geométricas diferentes apenas para efeito visual

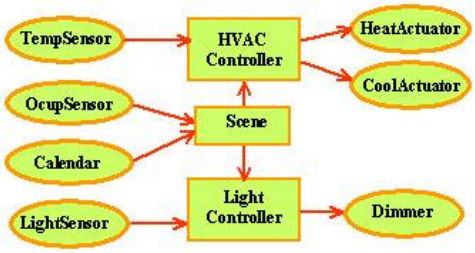


Figura 3.5 : Representação utilizando Dispositivos Lógicos

A raiz do modelo, conforme pode ser observado na Figura 3.6, são as classes *LogicalDevice*, *Subsystem* e *SimpleDevice*. Elas implementam o padrão de projeto *Composite* (Gamma, 2000), o qual permite compor objetos em estruturas de hierárquicas. Dessa forma, um dispositivo lógico (*LogicalDevice*) pode ser especializado como subsistema (*Subsystem*), recursivamente, até que a uma folha da árvore seja especializada como um *SimpleDevice*.

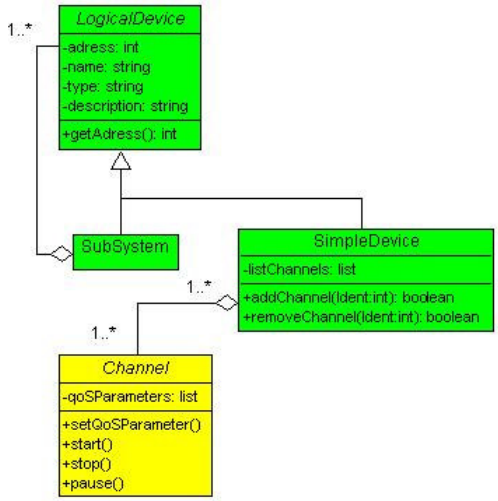


Figura 3.6 : Visão inicial do modelo

A seleção dos atributos e métodos teve como base a definição dos atributos e métodos dos protocolos estudados. Uma descrição detalhada dos métodos e atributos que foram definidos para cada classe encontra-se nas tabelas do Anexo A.

Os tipos básicos (*decimal*, *list*, *string*, *boolean*, etc.) utilizados na especificação dos atributos das classes utiliza a definição de tipos definidos para XML (W3C, 2001) por ser independente de linguagem de programação.

Um *SimpleDevice* pode ser composto por um ou mais canais, os quais permitem representar as conexões lógicas citadas anteriormente. Esta classe é especializada a fim de permitir a representação das diferentes funcionalidades de um sistema de automação predial. A hierarquia de classes foi construída a partir

identificação dos objetos comuns entre os diferentes protocolos estudados e da análise das variáveis citadas anteriormente neste capítulo.

Conforme pode ser observado na Figura 3.7, a classe *SimpleDevice* foi especializada nas classes *ProcessInterface*, *Controller*, *UserInterface*, *ComunicationInterface* e *Persistent*, cada uma com objetivo de permitir a representação de um conjunto de funcionalidades típicas da aplicação.

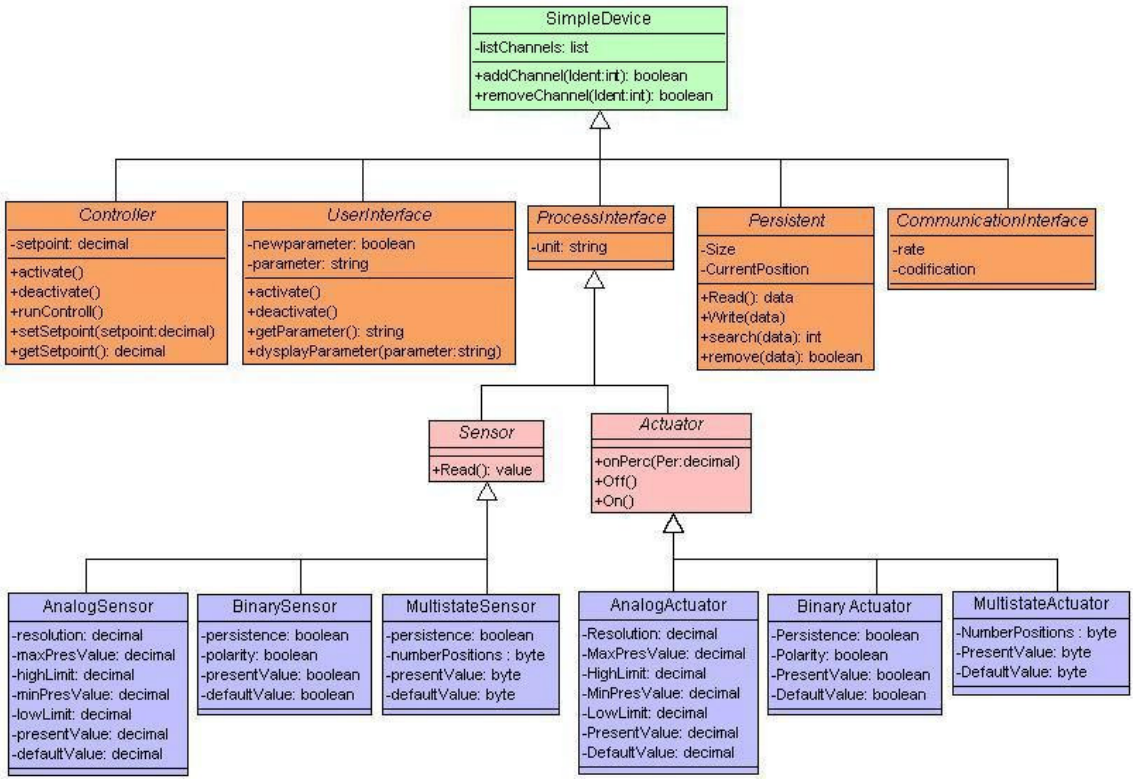


Figura 3.7 : Especialização da classe *SimpleDevice*.

A classe *ProcessInterface* representa as variáveis do processo físico. Em conformidade com a Figura 1.1, ela foi especializada nas classes *Sensor*, responsável pela representação das variáveis controladas do processo e *Actuator*, que permite representar as variáveis de atuação do processo. A classe *Sensor* foi especializada em *BinarySensor*, *AnalogSensor* e *MultistateSensor* e a classe *Actuator* em *BinaryActuator*, *AnalogActuator* e *MultistateActuator*.

A classe *Binary* (*Sensor* ou *Actuator*) tem por finalidade permitir a representação de variáveis que possuam apenas dois estados possíveis, a classe *Analog* (*Sensor* ou *Actuator*) permite a representação de variáveis que possuam um conjunto finito de valores, dados um determinado intervalo e uma resolução e a classe *Multistate* (*Sensor* ou *Actuator*) possibilita representar o estado atual de uma variável de múltiplos estados.

A classe *Controller* destina-se a representar as funções de controle de processos, podendo ser especializada em classes que representam diferentes estratégias de controle.

A classe *Persistent* objetiva modelar dados persistentes no sistema. Estes dados podem ser informações para autenticação de usuários (login e senhas, entre outros), musicas, vídeos, etc, os quais poderiam estar armazenados em um arquivo em mídia, memória ou qualquer meio que garanta a persistência da informação.

A classe *UserInterface* tem por finalidade permitir modelar a interação com os usuários do sistema. Teclados, displays e monitores são especializações típicas dessa classe. A fronteira entre as classes *ProcessInterface* e *UserInterface* não é bem definida, pois se pode considerar, por exemplo, o interruptor de um sistema de iluminação como sendo uma interface com o usuário ou com o processo. No modelo proposto procura-se reservar o conceito de interface com o usuário para representar entradas/saídas de dados de mais alto nível, embora na etapa de mapeamento tecnológico, possa-se optar por implementar esta interface por um dispositivo menos complexo, típico da classe *ProcessInterface*.

Finalmente a última classe, *CommunicationInterface*, tem por objetivo representar componentes do subsistema de comunicação. Esta classe acabou sendo pouco explorada no desenvolvimento do trabalho pois toda a infraestrutura de comunicação é tratada como uma camada, um *middleware*, o que permite considerar toda a conexão de comunicação entre dispositivos lógicos como sendo ponto a ponto. Esta simplificação foi adotada uma vez que a questão do projeto de redes que é um problema bastante complexo e está fora do escopo deste trabalho.

Outra característica que necessita ser representada é o comportamento discreto sistema, ou seja, especificar as ações que devem ser executadas quando da ocorrência de determinados eventos. Para esta representação foi adotado o conceito de cenário, que é definido pela UML (OMG, 2003) como sendo uma seqüência específica de ações que ilustra um comportamento. Este conceito também é utilizado por algumas tecnologias analisadas, principalmente associado à execução de diferentes ações pelos subsistemas de iluminação.

Petriu (PETRIU, 2003) define cenário como uma seqüência de passos que pode incluir caminhos alternativos, caminhos paralelos, laços e refinamentos de um passo por um subcenário. Na sua visão, os cenários podem ser modelados ou por diagramas de interação ou por diagramas de atividade e apresenta as vantagens e as desvantagens de cada diagrama.

Neste trabalho faz-se ampla utilização deste conceito para representar o comportamento discreto em todos os subsistemas: um cenário define os procedimentos que devam ser executados conforme a ocorrência de determinadas pré-condições,

Conforme pode ser observado na Figura 3.8 foram definidas as classes *Scene*, *Precondition* e *Procedure*. A classe *Precondition* permite representar as pré-condições e está associada à classe *LogicalDevice* pois estas podem corresponder ao estado de um subsistema (*Subsystem*) ou de qualquer especialização da classe *SimpleDevice*. Além disso, associadas a ela também estão as classes *Calendar* e *Clock*, as quais permitem representar pré-condições dependentes do tempo, relativas a calendário e a passagem do tempo.

A classe *Procedure* está associada com *SimpleDevice* e indica estado desejado do dispositivo ao qual está associada. Um cenário, representado pela classe *Scene*, associa pré-condições (*Precondition*) a procedimentos (*Procedure*).

Para representar o comportamento expresso pelos cenários foram adotadas regras de produção que consistem basicamente em um conjunto de condições no estilo SE <pré-condição> ENTÃO <ação>, com a possibilidade de inclusão de conectivos lógicos relacionando os atributos.

Podem ser usados operadores lógicos (E, OU e NEGAÇÃO), operadores relacionais (maior, maior igual, menor, menor igual, igual e diferente), operadores

aritméticos (soma , subtração divisão, multiplicação) e funções temporais (`[duração x unidades de tempo]` e `Esperar [x unidades de tempo]`).

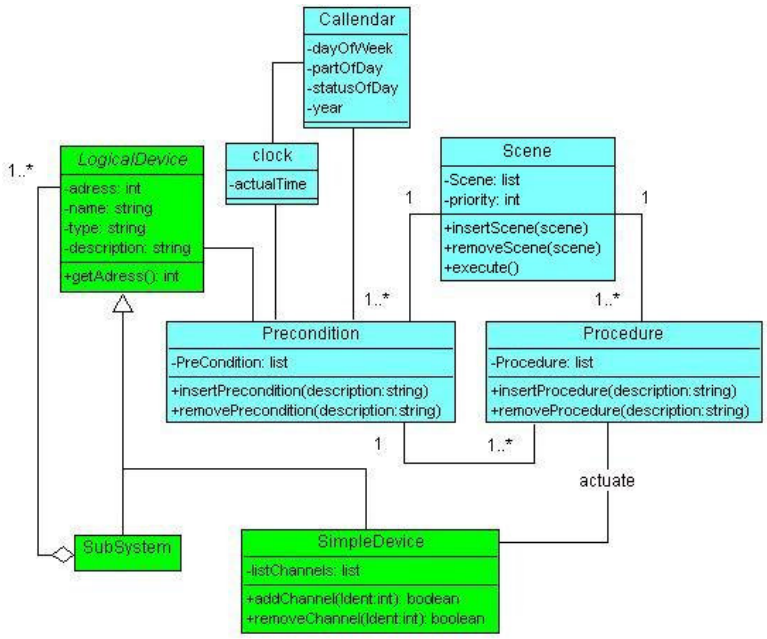


Figura 3.8 : Modelagem dos Cenários

As funções temporais permitem representar temporizações na construção dos procedimentos. A primeira, utilizada após a especificação de uma ação, representa o tempo de duração da mesma e a segunda corresponde a uma ação, representando um tempo de espera antes ou depois de uma outra ação.

Os parênteses devem ser usados para determinar a prioridade da avaliação das sentenças e o ponto e vírgula é utilizado nos procedimentos como separador de uma cadeia de ações. A Figura 3.9 exemplifica a utilização da sintaxe para representação de comportamentos num sistema.

Nome da Pré-condição	**Eventos a serem avaliados**
PreA7	ExtLightSensorSensor < Z & (actualTime > 17h & actualTime < 24h)
Nome do Procedimento	**Dispositivos lógicos a serem acionados**
ProcC1(Ativar. Alarme)	Esperar [15s]; StatusSensor = Desocupado
ProcA2 (L2 manual)	LigthRelay2 = on [duração 15min]
ProcA6 (Ligar L3 100%)	LightDimmer = on
Nome do Cenário	**Descrição do Cenário**
CenA7 (L3: noite antes 24h)	**SE** PreA7 **ENTÃO** ProcA6

Figura 3.9 : Exemplos de utilização da sintaxe

As regras de produção são um mecanismo utilizado para a construção de sistemas especialistas em inteligência artificial e a sintaxe definida mostrou-se suficiente na realização dos estudos de caso realizados ao longo deste trabalho.

Uma questão importante é a resolução de conflitos entre cenários. Um caso típico é o conflito entre comandos automáticos e manuais, quando o sistema define pela execução de um procedimento e o usuário outro, em conflito com o primeiro.

Para o tratamento deste problema foi adotado o conceito de prioridades definido pelo protocolo BACnet para priorização de mensagens (**Erro! A origem da referência não foi encontrada.**, 2003).

Foram definidos 5 níveis de prioridades, conforme a finalidade do cenário, as quais podem ter até 10 subníveis de prioridade (de 0 a 9). A composição do nível mais o subnível dará origem a um número inteiro. Quanto maior o valor do número, maior a prioridade do cenário. Os níveis definidos foram

- nível 1: conforto (Iluminação, HVAC, cortinas, supervisório, etc.);
- nível 2: gerenciamento de Energia;
- nível 3: controle de Acesso;
- nível 4: controle de Intrusões;
- nível 5: controle de Incêndios.

Esta estratégia resolve o problema de resolução de conflitos entre os subsistemas. Dentro de um mesmo subsistema o projetista define, em tempo de projeto, níveis de prioridades diferentes sempre que houver conflito de ações.

A visão conjunta do diagrama de classes pode ser encontrada no Anexo A.

3.2.3 Definição dos Dispositivos Lógicos e de suas Interações

No RM-ODP, após a definição dos objetivos e das restrições (visão de empresa) e da especificação do significado da informação (visão de informação), na próxima visão o sistema deve ser descrito como um conjunto de objetos que interagem através de suas interfaces. Esta camada é denominada visão de computação.

Para atender esta definição, esta etapa de projeto do *framework* especifica os diferentes tipos de objetos que compõem cada um dos subsistemas e como estes objetos interagem. Estes objetos que são denominados como objetos computacionais pelo modelo de referência são denominados, no âmbito deste trabalho dispositivos lógicos, uma vez que tem por objetivo representar logicamente as funcionalidades que deverão ser implementadas pelo sistema

A Figura 3.10 apresenta os dispositivos lógicos que foram definidos para o subsistema HVAC. Estes dispositivos foram definidos em função das funcionalidades desse subsistema que se deseja representar e correspondem a instâncias das classes definidas na etapa anterior. A fim de tornar os diagramas menores e mais fáceis de serem compreendidos foi utilizado o conceito de *stereotypes* da UML para representar a classe de origem.

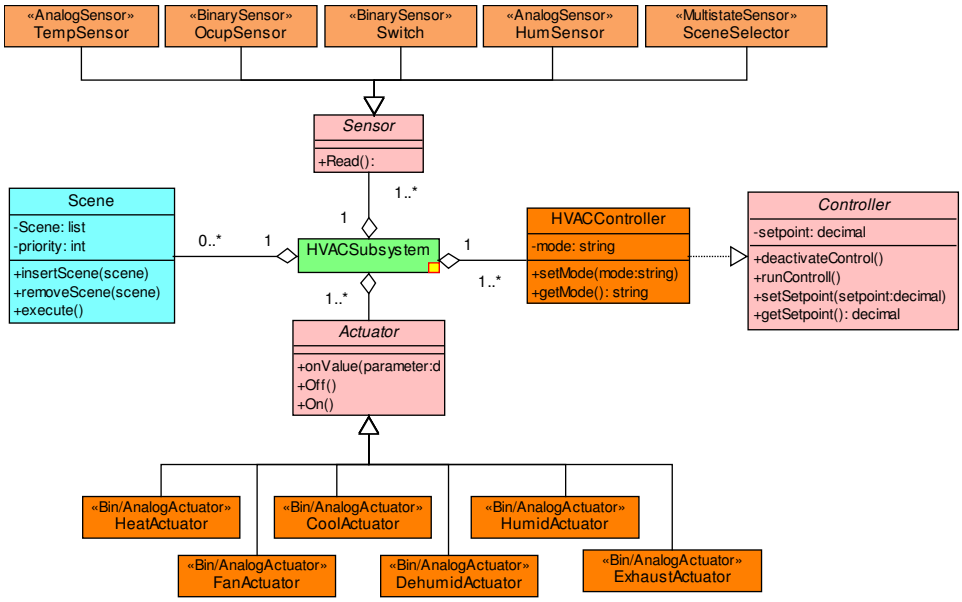


Figura 3.10 : Dispositivos lógicos do subsistema HVAC

Como pode ser observado, o subsistema HVAC é composto por dispositivos lógicos instanciados das classes *Actuator* e *Sensor*, o quais representam variáveis de atuação e de entrada do processo e das classes *Scene* e *Controller*, que permitem representar o comportamento do subsistema, sendo que a primeira permite representar a estratégia para o comportamento em relação aos eventos discretos e a segunda em relação ao controle das variáveis contínuas. As cardinalidade adotadas nas composições permitem a representação de sistemas com as mais diferentes complexidades.

Para representar variáveis de entrada foram instanciados os seguintes dispositivos lógicos:

- *TempSensor*: do tipo *AnalogSensor* com o objetivo de representar a temperatura;
- *OcupSensor*: do tipo *BinSensor* que permite representar se um determinado ambiente está ocupado ou não;
- *Switch*: do tipo *BinarySensor* que permite representar qualquer outra funcionalidade que só tenha dois estados;
- *HumSensor*: do tipo *AnalogSensor* que permite representar a variável de entrada umidade;
- *SceneSelector*: do tipo *MultistateSensor* o qual permite a seleção de um cenário específico de um conjunto de cenários relativos a HVAC.

Para representar variáveis de atuação os seguintes dispositivos lógicos foram instanciados:

- *HeatActuator*: o qual representa uma fonte de calor e pode ser do tipo *BinaryActuator* ou do tipo *AnalogActuator*, conforme a fonte apresente apenas dois estados (ligado/desligado) ou permita seleção de um valor intermediário entre dois limites pré-estabelecidos (representado por 0 e 100%);
- *CoolActuator*: o qual representa uma fonte de frio e pode ser do tipo *BinaryActuator* ou do tipo *AnalogActuator*;
- *HumidActuator*: o qual representa uma fonte de umidade e também pode ser do tipo *BinaryActuator* ou do tipo *AnalogActuator*;
- *DehumidActuator*: o qual representa uma fonte capaz de retirar a umidade do ambiente. Pode ser do tipo *BinaryActuator* ou do tipo *AnalogActuator*;
- *FanActuator*: o qual representa uma fonte de ventilação e pode ser do tipo *BinaryActuator* ou do tipo *AnalogActuator*;
- *ExhaustActuator*: o qual representa uma fonte de ventilação e pode ser do tipo *BinaryActuator* ou do tipo *AnalogActuator*.

O tipo do controlador (*Controller*) não foi especificado no modelo, embora possa se observar a especialização do controlador para atender ao controle das funcionalidades desse subsistema (*HVACController*)

Nesta etapa também devem ser definidos o tipo e o sentido da conexão, relativamente ao sentido do fluxo de informação entre os objetos computacionais. Especializações da classe *Channel*, permite representar os diferentes tipos de conexões previstas na definição do RM-ODP, conforme pode ser observado na Figura 3.11.

Para modelar o fluxo de informação entre os dispositivos especificados para cada subsistema foi adotada uma representação similar aos diagramas de colaboração UML com o foco apenas no sentido da informação.

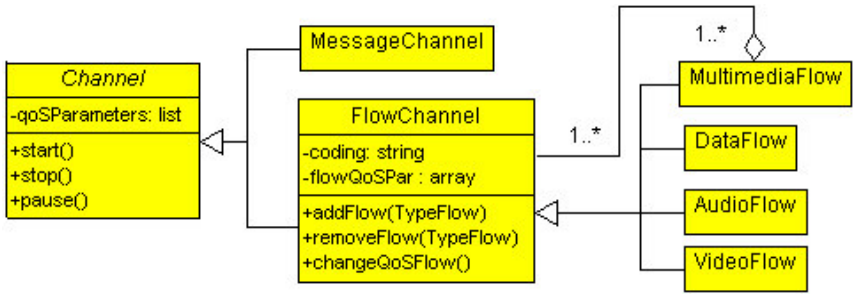


Figura 3.11 : Especialização da classe *Channel*.

Esta simplificação foi necessária porque os diagramas UML que tem por objetivo representar a interação entre objetos (diagramas de colaboração e de seqüência) são focados na solução do problema, não sendo genéricos o suficiente para esta representação. Na Figura 3.12 pode ser observado o diagrama resultante da modelagem do subsistema HVAC.

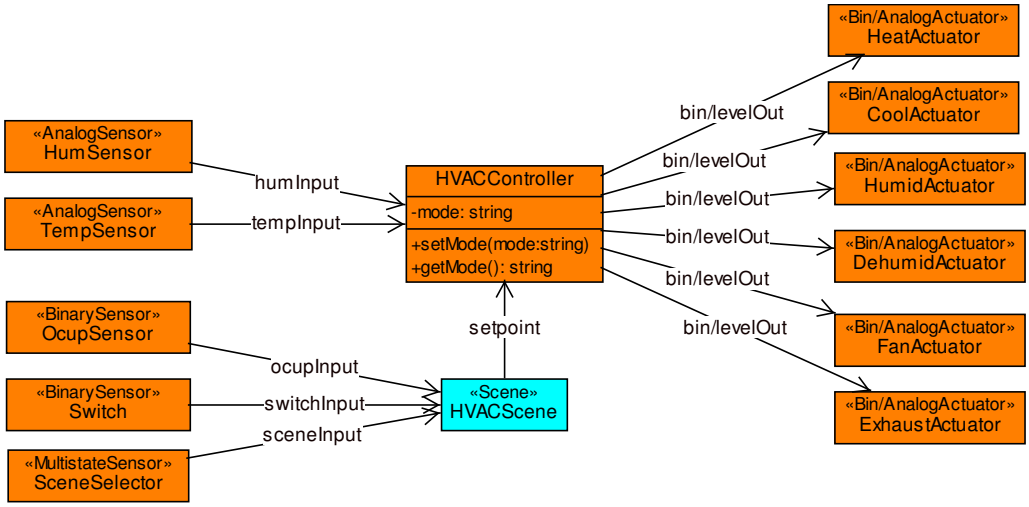


Figura 3.12 : Fluxo de informação no subsistema HVAC.

O tipo da informação está relacionado com o atributo especificado na Classe Pai do dispositivo lógico. Assim, para a classe *Analog*(*Sensor* ou *Actuator*) a variável que representa o valor atual (*PresentValue*) é do tipo decimal (W3C, 2001), para a classe *Binary*(*Sensor* ou *Actuator*) é do tipo boolean, para a classe *Controller* a variável *setpoint* é do tipo decimal e assim sucessivamente para qualquer objeto lógico que tenha sido especificado em qualquer subsistema.

O mesmo critério foi usado para especificação de todos os subsistemas. Os diagramas correspondentes de cada um dos subsistemas especificados no modelo podem ser encontrados no Anexo A.

3.2.4 Mapeamento Tecnológico

Na próxima visão do RM-ODP, os objetos computacionais definidos do nível anterior são mapeados para objetos de engenharia e é incluído todo o suporte a transparência em relação à distribuição física É esta visão, denominada visão de engenharia, que dará suporte à especificação física da próxima visão.

Como as tecnologias para as quais se estará realizando mapeamento tecnológico, na última etapa de modelagem, já implementam os conceitos necessários à distribuição física, conforme já foi abordado no capítulo 2, esta visão torna-se desnecessária no *framework*.

Assim, a quarta e última etapa de projeto do *framework* corresponde à visão de tecnologia no RM-ODP que tem por finalidade a definição da tecnologia de hardware e software para composição do sistema e localização de cada componente de hardware, a nível físico e em nível de rede.

Cada dispositivo lógico definido na etapa anterior será mapeado para dispositivos físicos de uma determinada tecnologia selecionada. Um dispositivo físico pode ser composto por um ou mais dispositivos lógicos, dependendo das funcionalidades suportadas pelo *hardware* escolhido.

Conforme pode ser observado na Figura 3.13 cada dispositivo recebe uma localização física (em determinado ambiente da instalação) e uma localização na rede correspondendo a um ponto (nó) da rede em uma solução distribuída.

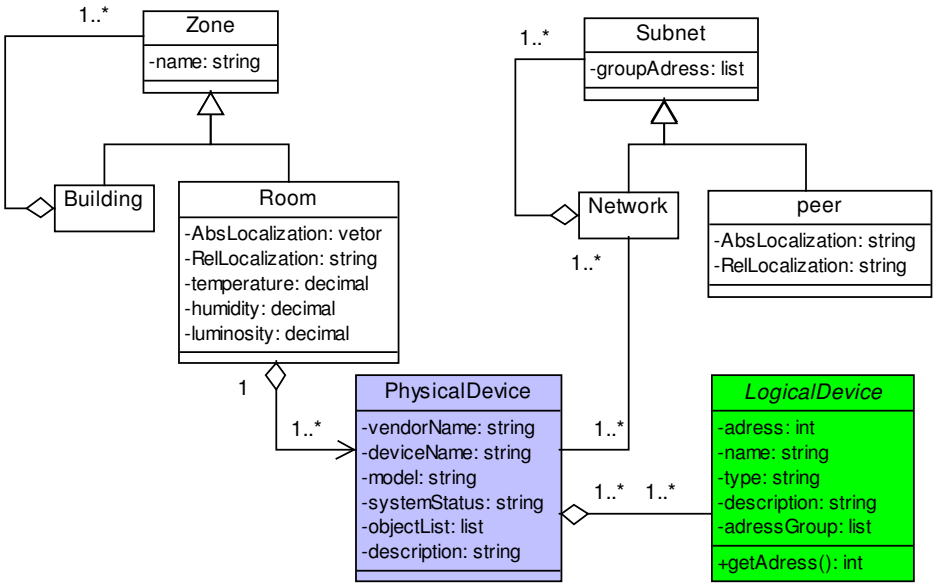


Figura 3.13 : Mapeamento Tecnológico

A fim de possibilitar este mapeamento, dispositivos físicos de diferentes fabricantes devem ter suas funcionalidades mapeadas, conforme proposto no *framework*, compondo uma biblioteca de dispositivos. A partir dessa biblioteca será possível selecionar que componente oferece a funcionalidade que foi modelada.

As conexões entre os dispositivos lógicos corresponderão a uma conexão local, quando os objetos estiverem dentro do mesmo nó, ou a uma conexão externa quando for realizada entre diferentes dispositivos físicos. As conexões locais são providas por mecanismos de comunicação adotados pela tecnologia para a qual se está fazendo o mapeamento e as conexões externas serão dependentes da infraestrutura de rede suportada por ela.

O resultado será o modelo físico de um sistema. É função das ferramentas de configuração importar esta especificação e fazer configurações automáticas de um dado sistema.

Acredita-se que uma boa estratégia seja descrever o sistema no formato XML – *Extensible Markup Language* (W3C, 2000), o qual é um padrão foi proposto pelo W3C para ser uma linguagem para troca de dados entre aplicações e tem rapidamente se tornado um padrão de mercado para representação de dados (AMARAL, 2002).

Uma vantagem da utilização de XML é que é possível validar um arquivo, pela verificação da sua correção gramatical (analisando a estrutura do documento) antes

mesmo da sua utilização por uma ferramenta específica, o que é importante para prevenir erros de formação de um arquivo fonte. Isto é possível porque o consórcio também define uma linguagem para descrever a estrutura do documento, recomendando atualmente o uso de *XML Schema* (W3C, 2001) para esta finalidade

A Figura 3.14 representa uma proposta para o processo de representação (armazenagem) dos dados modelados no *framework* para um determinado sistema. Os dispositivos e as conexões lógicas seriam armazenados em formato XML. Posteriormente, a partir desse(s) arquivo(s) poderia ser realizado o mapeamento automático para diferentes tecnologias disponíveis para implementação.

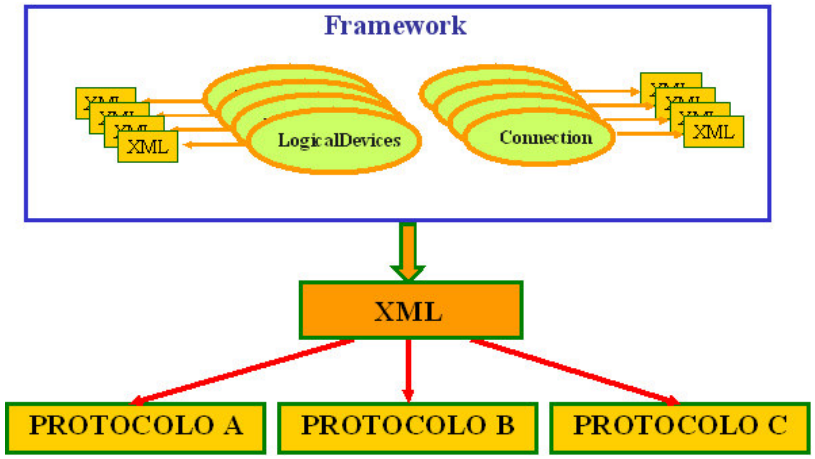


Figura 3.14 : Representação do *framework*

Esta tarefa de mapeamento consiste na definição de filtros para conversão do(s) arquivo(s) gerado(s) pelo framework no formato do(s) arquivo(s) de configuração da tecnologia alvo. O uso de arquivos no formato XML deve facilitar esta tarefa pela existência de padrões (geralmente suportados por diferentes ferramentas) para manuseio de arquivos XML.

Para conversão de formato do(s) arquivo(s) padrão gerado a partir do *framework* para o formato suportado pelo(s) arquivo(s) de uma determinada tecnologia alvo, por exemplo, pode citar-se o padrão **XSL** (***eXtensible Stylesheet Language***) cujo processo está representado na Figura 3.15.

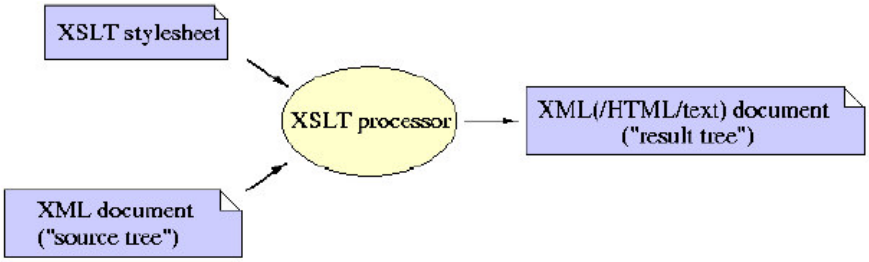


Figura 3.15 : Usando XSL para conversão de formatos.

O anexo C apresenta uma proposta de estrutura de arquivo XML para representação dos dispositivos lógicos e conexões.

3.3 Considerações Finais

A Figura 3.16 apresenta uma visão geral das diversas etapas do projeto de um sistema de automação predial que foram abordadas, situando-as no nível correspondente conforme o RM-ODP.

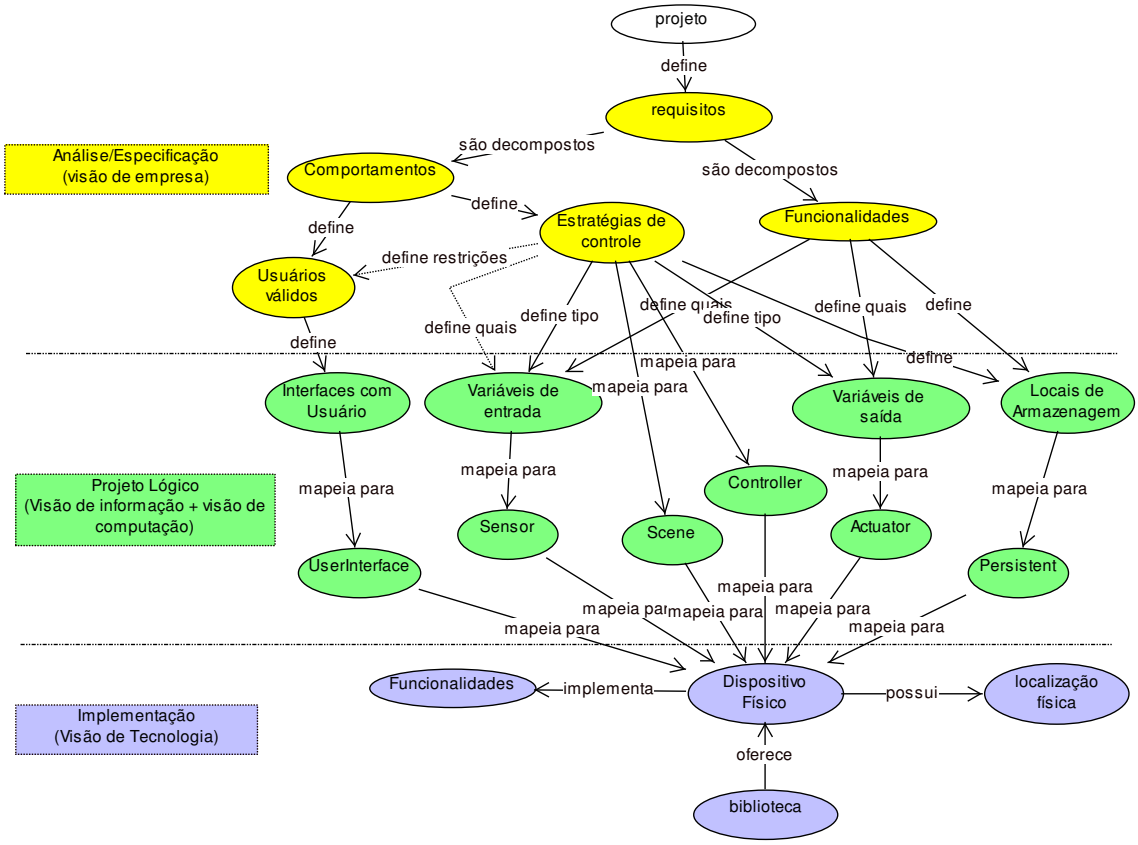


Figura 3.16 : Visão Geral do Projeto

Em Gomaa (GOMAA, 2000) pode-se encontrar uma categorização de classes de aplicação. O autor apresenta as seguintes classes como representativas para as aplicações:

- interfaces: classes que interagem com objetos do ambiente externo tais como dispositivos, usuários e sistemas ou subsistemas externos;
- classe de armazenamento (entity): classes que armazenam informações de objetos persistentes;
- controle: classes que fornecem a coordenação para um conjunto de objetos;
- lógica: classe que contém os detalhes da lógica da aplicação.

Embora o modelo proposto não tenha sido elaborado a partir dessa classificação, como ficou claro no texto apresentado ao longo desse capítulo, pode-se observar que as classes propostas se assemelham muito as da categorização apresentada pelo autor.

Como foi citado um dos pontos de partida para definição das classes do framework foi a análise dos padrões. A Tabela 3.1 apresenta os objetos comuns entre os protocolos EIB, BACnet e HomePnP e os objetos propostos no *framework*.

Tabela 3.1 : Equivalência dos objetos propostos no modelo

EIB-Type_Id (dec) EIB Object types	**BACnet ID(dec) BACnet Object types**	**HomePnP ID(hex) HomePnP Object types**	**Modelo (proposto)**
0 - Device Objec	8 - Device Object	01 - Node Control	**PhysicalObject**
1 - Addresstable Object			
2 - Associationtable Object			
		02 - Context Control	**Scene**
3 – Application program Object	16 - Program Object		
4 - Interface Program			
5 - EIB-Object-Associationtable-Object			
6 - Router Filtertable			
10 - Pollingmaster			
11 - File	10 -File	16 - Data Memory	**Persistent**
100 - Analogue-Input	0 - Analogue-Input	08 - Analog Sensor	**AnalogSensor**
101 - Analogue-Output	1 - Analogue-Output	07 - Analog Control	**AnalogActuator**
102 - Analogue-Value	2 - Analogue-Value		**Scene**
103 - Binary-Input	3 - Binary-Input	06 - Binary Sensor	**BinarySensor**
104 - Binary-Output	4 - Binary-Output	05 - Binary Switch	**BinaryActuator**
105 - Binary-Value	5 - Binary-Value		**Scene**
106 - Counter		1C - Counter/Timer	**Controller**
107 - Loop	12 - Loop	0A - Matrix Switch	**Controller**
108 - Multistate-Input	13 - Multistate-Input	10 - Multi-position Sensor	**MultistateSensor**
109 - Multistate-Output	14 - Multistate-Output	09 - Multi-position Switch	**MultistateActuator**
	6 - Calendar		**Calendar**
	7 - Command		
	9 - Event Enrollment		
	11 - Group		
	15 - Notification Class		
	17 - Schedule		**Scene**
		15 - List Memory	**Persistent**
		03 - Data Ch. RCVR	
		04 - Data Ch. Trans	
		0F - Metter	**Sensor**
		10 - Display	**UserInterface**
		11 - Medium Transport	**Channel**
		13 - Dialer	
		14 - Keypad	**UserInterface**
		17 - Motor	**Actuator**
		19 - Synth/Tuner	
		1A - Tone Generator	
		1D - Clock	**Clock**

4 CICLO DE DESENVOLVIMENTO USANDO O *FRAMEWORK*

Este capítulo tem por objetivo mostrar como o *framework* proposto é utilizado a fim de garantir a consistência durante as diversas fases de projeto de sistema de automação predial

4.1 A Metodologia de Projeto Definida no *Framework*

A metodologia de projeto proposta no framework basicamente divide o projeto de um sistema de automação predial em três etapas.

A primeira corresponde ao nível de análise onde a partir das características desejadas do ambiente são definidos as funcionalidades e os comportamentos desejados do sistema. Esta etapa deve concentrar-se "no quê" o sistema deve realizar. A Figura 4.1 representa esta etapa, onde está sendo utilizado um *use case* UML para representar as funcionalidades desejadas. Estas informações devem ser complementadas através da descrição das políticas (estratégias de controle) que devem ser asseguradas para cada funcionalidade.

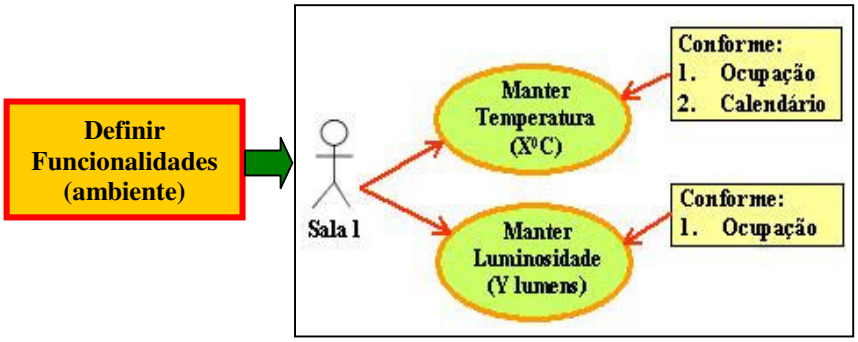


Figura 4.1 : Definição das funcionalidades

A partir desse ponto, o restante do projeto é realizado com o suporte do *framework* proposto. As funcionalidades e o comportamento desejados são mapeados para as funcionalidades dos subsistemas disponíveis no *framework* especificado. Conforme pode ser observado na Figura 4.2, esta etapa corresponde a expandir o *use case* inicial. Entretanto esta expansão já é realizada com foco nas funcionalidades que foram modeladas em cada subsistema.

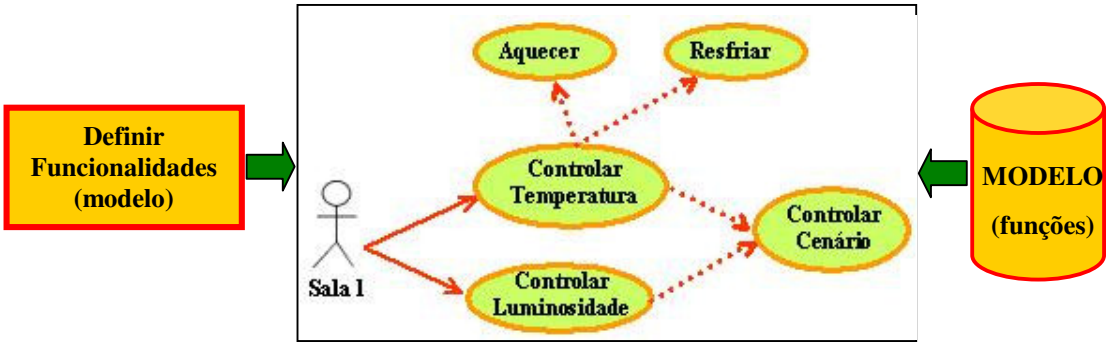


Figura 4.2 : Projeto lógico – seleção das funcionalidades do modelo

Depois de definidas as funcionalidades a próxima etapa corresponde ao projeto lógico: são selecionados os dispositivos lógicos necessários para representar as funcionalidades que foram definidas. É importante ressaltar que nesta etapa são tomadas as decisões de projeto, uma vez que a seleção de um determinado dispositivo lógico que será utilizado para implementação de uma determinada funcionalidade deve levar em conta aspectos relacionados a estratégia de controle que será utilizada no sistema. A Figura 4.3 representa esta camada do projeto.

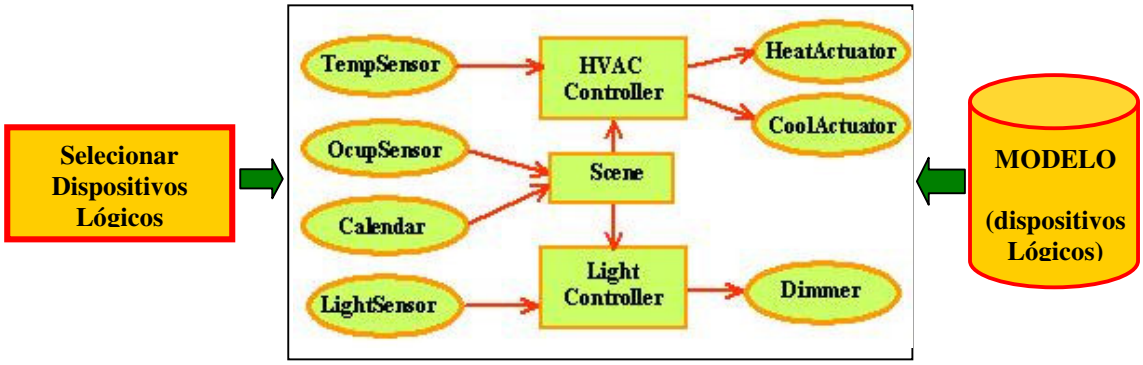


Figura 4.3 : Projeto Lógico – Seleção dos dispositivos lógico

Na terceira etapa os dispositivos lógicos são mapeados (*deployed*) para componentes físicos que suportem as funcionalidades especificadas e as conexões lógicas entre dispositivos serão mapeadas para as conexões físicas disponíveis para uma dada tecnologia.

A fim de permitir este mapeamento, dispositivos de diferentes tecnologias e/ou fornecedores deverão ter suas funcionalidades mapeadas conforme as funcionalidades propostas para os subsistemas do modelo, formando uma biblioteca de dispositivos físicos, a qual relaciona dispositivos com funcionalidades do modelo. Esta camada pode ser visualizada na Figura 4.4, dividida em duas partes (seleção e localização dos dispositivos) para melhor representação dos passos envolvidos.

4.2 Roteiro de Projeto Utilizando o *Framework*

Na realização de estudos de casos para validação do *framework* proposto, observou-se que diversas atividades devem ser executadas em cada uma das etapas citadas no item anterior. Por este motivo optou-se pelo detalhamento das mesmas, resultando no roteiro de projeto que será apresentado a seguir.

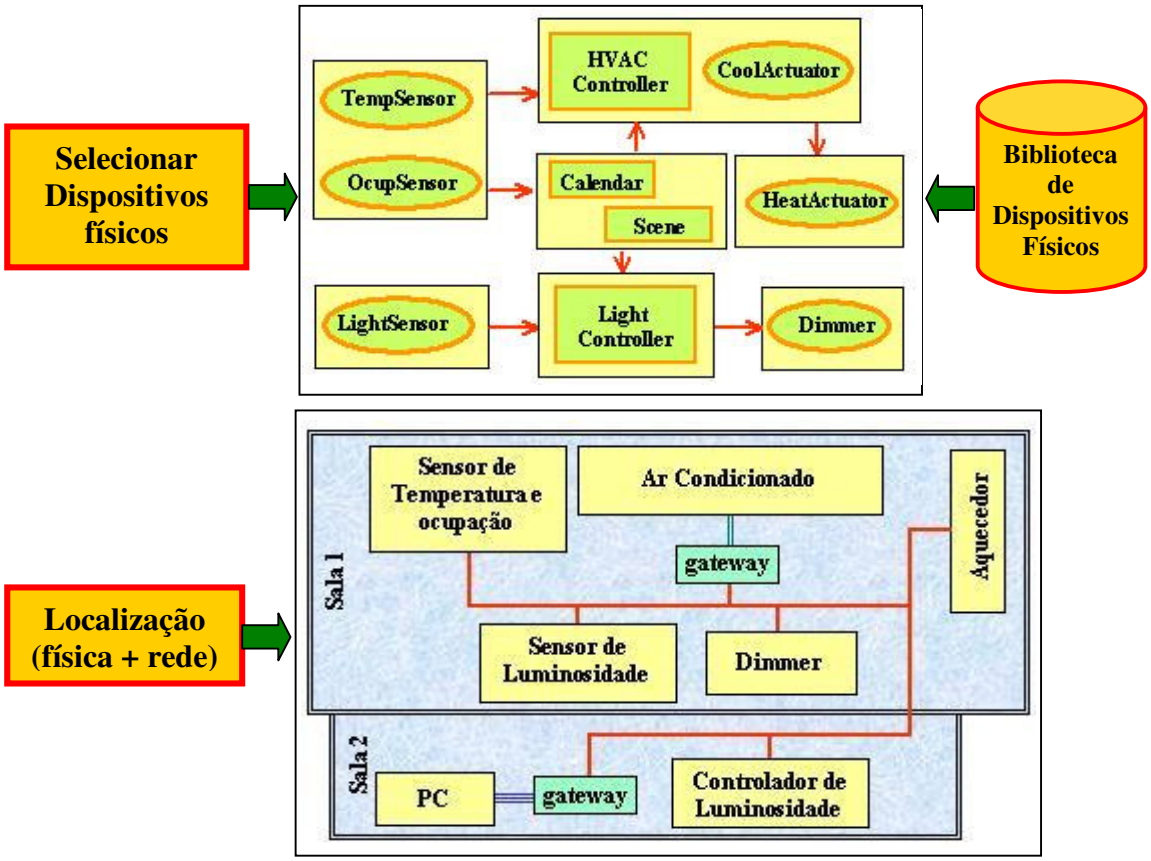


Figura 4.4 : Mapeamento Tecnológico

Este roteiro identifica atividades relevantes que devem ser executadas em cada etapa de projeto e tem como objetivo principal fornecer uma transição suave entre as diferentes camadas que compõem o *framework* proposto. Ele mantém a estrutura de camadas especificada, inserindo atividades dentro de cada camada a fim de facilitar a especificação do sistema.

As etapas desse roteiro foram definidas subjetivamente e resultaram da observação da repetição de tarefas à medida que se executavam estudos de caso para a validação do *framework*, ao longo de suas diversas etapas de especificação.

A seguir serão especificadas as principais atividades que devem ser executadas no projeto de um sistema de automação predial, no âmbito do *framework* especificado no trabalho.

1. Definir as funcionalidades do ambiente que devem ser controladas

A caracterização do ambiente define as funcionalidades que deverão ser atendidas pelo mesmo as quais são selecionadas de acordo com especificação do(s) proprietário(s) e/ou projetista(s) e de acordo com normas e legislação vigentes conforme o tipo de ambiente.

A partir da análise das características do ambiente externo pode-se definir quais as características especificadas para o ambiente definem funcionalidades que devem ser implementadas no sistema de automação predial por não serem atendidas pelo ambiente externo.

Para representação das funcionalidades que devem ser atendidas podem ser construídos diagramas *use case* UML, os quais devem ser complementados por

informações relativas às estratégias (políticas) de controle especificadas para o ambiente.

2. Definir os agentes externos que irão interagir com o sistema e definir as políticas de gerenciamento

Os agentes externos podem ser definidos como usuários do sistema, os quais podem ser classificados em diversos níveis de acesso conforme as políticas de gerenciamento definidas, relativamente à configurações, notificações, permissões de acesso, interfaces de interação com sistema, etc.

3. Dividir o ambiente em sub-regiões ou zonas de acordo com as funcionalidades que devem ser controladas

Um ambiente pode ser subdividido para possibilitar o ajuste da função de controle as características específicas de uma parte do ambiente ou para facilitar a manutenção e a instalação. HomePnp define estas sub-regiões como zonas e as define como uma parte física ou lógica de uma região de controle (CIC, 1998).

Na divisão do ambiente em zonas deve-se levar em conta parâmetros relativos a fatores tais como conforto (controle de luminosidade, de temperatura, de umidade, de qualidade do ar, etc.), segurança (controle de acesso, de intrusão, de incêndio e controle ambiental: vazamentos, inundações, faltas, falha de equipamentos,etc), gerenciamento de energia, etc.

Um subsistema específico pode ter uma ou mais zonas, de acordo com a característica de controle que se deseje implementar e não necessariamente existe relação entre a divisão de zonas entre os diferentes subsistemas.

4. Selecionar funcionalidades e subsistemas que comporão o modelo

A partir da análise das funcionalidades e das políticas definidas no item 1, são definidos os subsistemas necessários para atender a especificação e são refinadas aquelas funcionalidades de acordo com as funcionalidades definidas nos subsistemas especificados no *framework*.

5. Definir as políticas de operação dos subsistemas

A partir das estratégias de controle que foram especificadas no item 1 definir as políticas de operação, criando cenários de operação com enfoque nas ações que devem ocorrer e que não podem ocorrer em cada sub-região (zona) para cada subsistema.

Nesta etapa pode ser interessante definir os estados de ocupação do ambiente ou modos de operação do sistema os quais posteriormente darão origem a cenários específicos nos diferentes subsistemas.

Esta característica é tão relevante na definição dos sistemas de automação predial que alguns protocolos a tratam explicitamente. HomePnP, por exemplo, define um contexto denominado *House mode* (CIC, 1998) com os seguintes estados: Ocupado (normal, baixa atividade, dormindo) – Desocupado (até 1h, até 4h, até 8h, até 24h, até 48h, férias) e Incerto. Por sua vez UPnP define um *template* denominado *house status* (UPnP, 2003) para o qual define as seguintes variáveis : *occupancyState*(Ocupado, Desocupado e Indeterminado), *activityLevel* (regular, alta atividade e dormindo) e *DormancyLevel* (férias, regular e animais na casa).

6. Selecionar os dispositivos lógicos e suas conexões

Nesta etapa serão selecionados os dispositivos lógicos e suas conexões que permitem representar as funcionalidades especificadas no item 4 de forma a atender as políticas de operação definidas no item 5. Definições de projeto são estabelecidas ao definir-se as classes de origem do dispositivo lógico que vai ser utilizado para representar uma determinada funcionalidade e as conexões entre os diferentes dispositivos lógicos que representam cada subsistema.

7. Especificar os cenários

Construir os cenários de operação com enfoque nas funcionalidades dos dispositivos lógicos especificados, definindo as pré-condições e procedimentos que correspondem a cada política de operação definida no item 5.

Como é através dos cenários que é implementada a interação entre diferentes subsistemas esta etapa é fundamental para localizar a necessidade de interação entre os subsistemas.

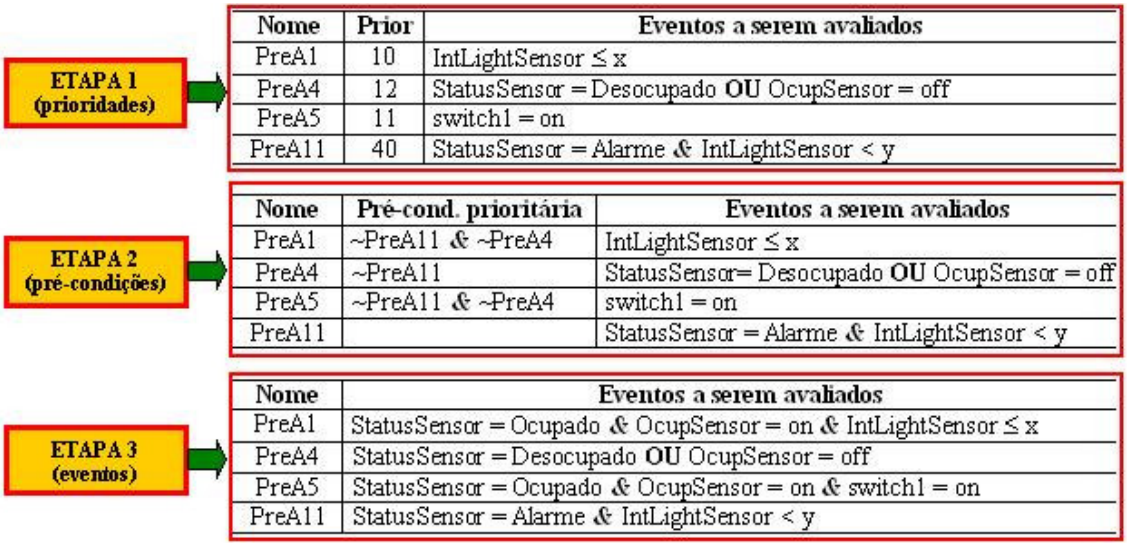
Uma questão importante é a definição e representação das prioridades na solução dos conflitos. No meta-modelo apresentado no capítulo 3 foram definidos níveis de prioridade para cada cenário. Entretanto, a análise de diversas tecnologias mostrou que na maioria dos casos o hardware disponível para execução dos cenários não trabalha orientado a prioridade. Uma alternativa é fazer a construção das pré-condições com a inclusão explícita de prioridades baseada nos eventos que ocorrem nos dispositivos lógicos.

Após avaliar os cenários e verificar as colisões, definindo as prioridades, conforme citado no item 3.2.2, deve-se:

- Estabelecer as prioridades das pré-condições, conforme a prioridade que foi definida para os cenários;
- Estabelecer para cada pré-condição as de maior prioridade que provocam colisão, ou seja, que estão associadas a procedimentos contraditórios e não devem ser válidos no momento da avaliação de uma pré-condição de menor prioridade;
- Avaliar a ocorrência dessas pré-condições antes da de menor prioridade, redefinindo a cadeia de eventos que deve ser verificada para cada pré-condição de menor prioridade, incluindo os eventos das pré-condições de maior prioridade. As avaliações das condições lógicas devem ser realizadas usando-se as leis da lógica booleana.

A Figura 4.5, é um extrato do estudo de caso do capítulo seguinte e ilustra estas etapas. Primeiro, a partir da definição de prioridades dos cenários foi estabelecido a prioridade das pré-condições. A seguir foi avaliado, levando-se em conta os procedimentos que deviam ser executados, que pré-condições de maior prioridade levavam a procedimentos contraditórios. Finalmente foi reconstruída a cadeia de eventos as ser avaliada em cada pré-condição a fim de eliminar o conflito.

Ness exemplo não aparece a pré-condição PreA5 como pré-condição prioritária para a PreA1 porque na definição dos cenários elas não levam a procedimentos contraditórios. Esta solução será explorada posteriormente no estudo de caso.



Nome	Prior	Eventos a serem avaliados
PreA1	10	IntLightSensor ≤ x
PreA4	12	StatusSensor = Desocupado **OU** OcupSensor = off
PreA5	11	switch1 = on
PreA11	40	StatusSensor = Alarme & IntLightSensor < y

Nome	Pré-cond. prioritária	Eventos a serem avaliados
PreA1	~PreA11 & ~PreA4	IntLightSensor ≤ x
PreA4	~PreA11	StatusSensor= Desocupado **OU** OcupSensor = off
PreA5	~PreA11 & ~PreA4	switch1 = on
PreA11		StatusSensor = Alarme & IntLightSensor < y

Nome	Eventos a serem avaliados
PreA1	StatusSensor = Ocupado & OcupSensor = on & IntLightSensor ≤ x
PreA4	StatusSensor = Desocupado **OU** OcupSensor = off
PreA5	StatusSensor = Ocupado & OcupSensor = on & switch1 = on
PreA11	StatusSensor = Alarme & IntLightSensor < y

Figura 4.5 : Construção das prioridades baseadas na ocorrência dos eventos

8. Realizar mapeamento tecnológico

Selecionar dispositivos físicos que implementem as funcionalidades dos dispositivos lógicos modelados no item anterior, definindo sua localização física (*deployment*) e o endereçamento na rede e mapear as conexões lógicas para conexões físicas.

4.3 Exemplos de Modelagem Utilizando o *Framework*

De forma simplificada um padrão de projeto descreve uma solução que foi utilizada para resolver um problema específico a qual se repete em outras aplicações (GAMMA, 2000). De uma maneira geral, em qualquer área de aplicação eles podem ser identificados e documentados, logo também em sistemas de automação predial é possível identificar determinados padrões que se repetem em diferentes contextos.

O objetivo principal desta seção é apresentar algumas soluções que se repetem no domínio sob estudo e mostrar como elas são definidas no âmbito do *framework* proposto, fornecendo alguns exemplos genéricos de modelagem usando a estrutura proposta. Estas soluções foram observadas a partir de análises de soluções em sistemas de automação predial ao longo do trabalho

Serão apresentados três padrões utilizando os dispositivos lógicos especificados, os quais permitem representar sistemas com complexidade crescente e que são utilizados independentes das funcionalidades e dos subsistemas que estão sendo implementados. Para cada um deles será apresentada uma descrição genérica da sua ocorrência, para cuja representação serão utilizadas as classes de origem dos dispositivos lógicos modelados e será apresentado um exemplo.

Um primeiro padrão que se identifica componentes da classe *Sensor* estão conectados diretamente a componentes da classe *Actuator*, o qual recebe o parâmetro relativo a variável de entrada e executa a operação sobre a variável de saída representada, conforme pode ser observado na Figura 4.6.

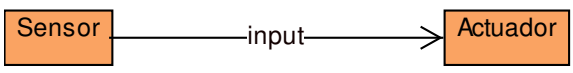


Figura 4.6 : Sensor conectado diretamente ao atuador

Um exemplo de aplicação desse padrão é para representar um interruptor ligado diretamente a uma lâmpada, a qual deve ligar ou desligar conforme o sinal recebido do mesmo. A figura Figura 4.7 ilustra esta aplicação.

Figura 4.7 : Modelagem de um interruptor comandando uma lâmpada.

O segundo padrão tem o objetivo de representar sistemas de controle de variáveis contínuas, no qual se deseja que uma variável de saída mantenha um valor de referência previamente estabelecido.

Um objeto da classe *Controller* executa o algoritmo que objetiva manter a variável de saída, representada por um objeto da classe *Actuator*, no valor previamente estabelecido geralmente pelo usuário, representado por um objeto da classe *UserInterface*. Um objeto da classe *Sensor* destina-se a representar a variável de entrada que está associado ao processo sob controle. A Figura 4.8 ilustra este modelo.

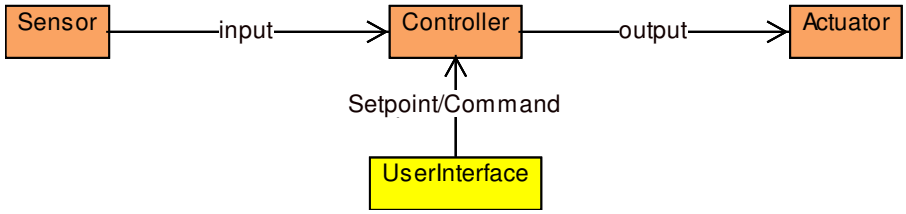


Figura 4.8 : Representação de um processo de controle em malha fechada

Um exemplo de aplicação desse padrão é ilustrado na Figura 4.9 a qual representa um controle de temperatura típico de um Ar Condicionado com as opções quente e frio. As funcionalidades aquecer, resfriar e ventilar estão sendo modeladas por três objetos da classe *Actuator*. O algoritmo de controle está sendo representado por um objeto da classe *Controller* e um objeto da classe *Sensor* está representando a temperatura ambiente. Um objeto da classe *UserInterface* permite representar a interface com o usuário o qual determina o valor de referência (*setpoint*) e selecionar comandos tais como ligar ou desligar, entre outras possibilidades.

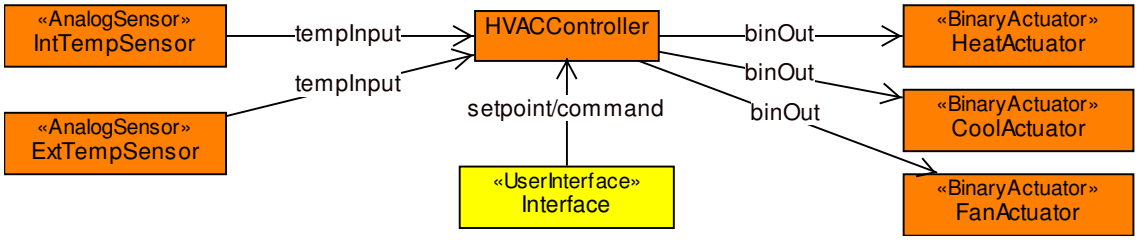


Figura 4.9 : Modelagem de um controle de temperatura

Sistemas mais elaborados devem oferecer uma flexibilidade de operação maior, permitindo a construção de diferentes cenários de operação. Estes sistemas podem ser representados com o terceiro padrão de projeto definido. Aos objetos já citados nos dois modelos anteriores é acrescentado um objeto cenário, da classe *Scene*, o qual permite representar diferentes condições para as variáveis de saída, através da determinação de parâmetros de referência (*setpoints*) ou comandos (*commands*) para os controladores ou pela manipulação direta das variáveis de saída, conforme parâmetros ou eventos que ocorrem no ambiente, representados por objetos da classe *Sensor*, conforme pode ser observado na Figura 4.10.

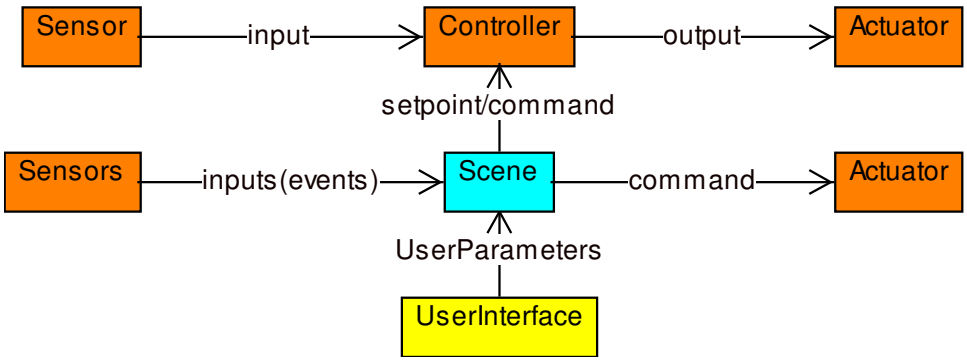


Figura 4.10 : Representação de sistema que permite diferentes cenários.

A Figura 4.11 acrescenta ao controle da temperatura da Figura 4.9 a possibilidade da representação de diferentes cenários, conforme a ocupação do ambiente, representado por um objeto da classe *Sensor* e de acordo com uma agenda previamente determinada, representado por um objeto da classe *Callendar*.

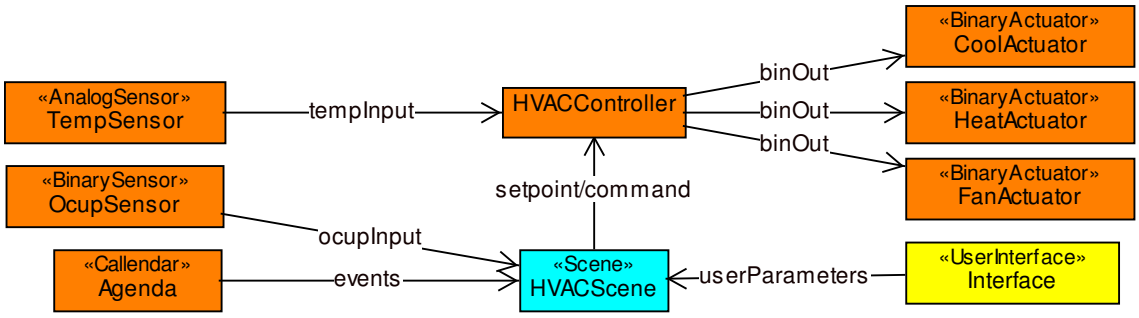


Figura 4.11 : Modelagem de um controle de temperatura com diferentes cenários.

Já a Figura 4.12 representa um sistema de controle de iluminação com dois atuadores das classes *BinaryActuator* e *AnalogActuator* cujos atributos são estabelecidos conforme o cenário, determinado pelos valores dos atributos dos diferentes objetos do tipo *Sensor*, que representam as variáveis de interesse.

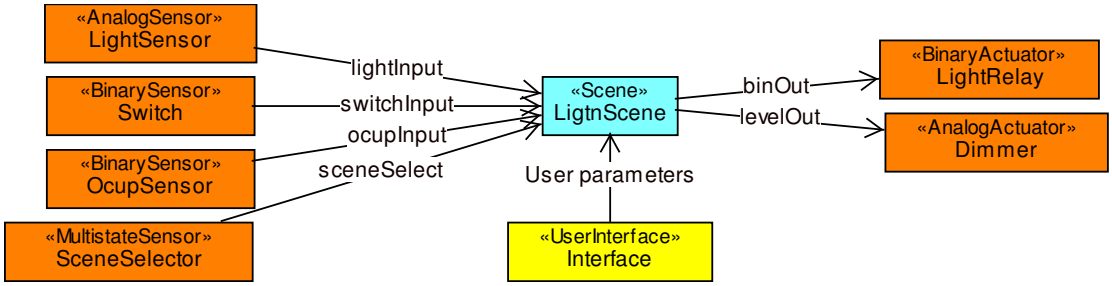


Figura 4.12- Modelagem de um controle de luminosidade com diferentes cenários.

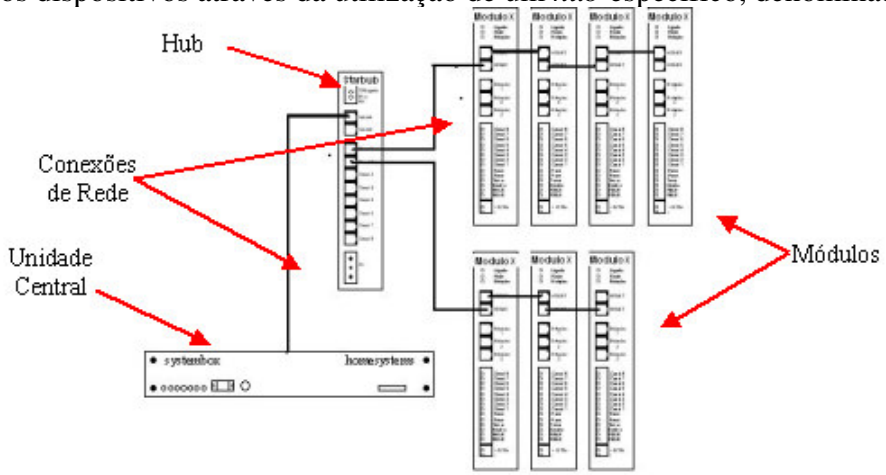
5 VALIDAÇÃO DO MODELO

Ao longo da definição do modelo foram realizados diversos estudos de caso, a fim de validar os conceitos e funcionalidades que estavam sendo definidas. Neste capítulo é apresentado o mapeamento dos objetos especificados para uma arquitetura alvo e também um estudo de caso com o objetivo de validar a metodologia como um todo.

5.1 Sistema de Automação Predial/Residencial da Empresa Homesystems

A arquitetura alvo principal do estudo de caso apresentado é uma plataforma proprietária da empresa Homesystems, cuja matriz localiza-se em Porto Alegre e desenvolve tecnologia em sistemas de automação predial (HOMESYSTEMS, 2005) e com a qual o Grupo de Controle Automação e Robótica (GCAR) da UFRGS possui uma parceria.

A arquitetura se enquadra no terceiro tipo abordado no capítulo 2: diferentes módulos são conectados em rede e possuem um conjunto de funcionalidades que independem de uma unidade central, podendo funcionar em regime *stand-alone*. Entretanto determinadas funcionalidades só são possíveis pelo gerenciamento da unidade de controle central, denominada *systembox*. A plataforma de rede utiliza um protocolo proprietário rodando sobre a camada física 485 e permite diferentes configurações na montagem dos dispositivos através da utilização de um *hub* específico, denominado



Starhub. A Figura 5.1 ilustra uma configuração típica.

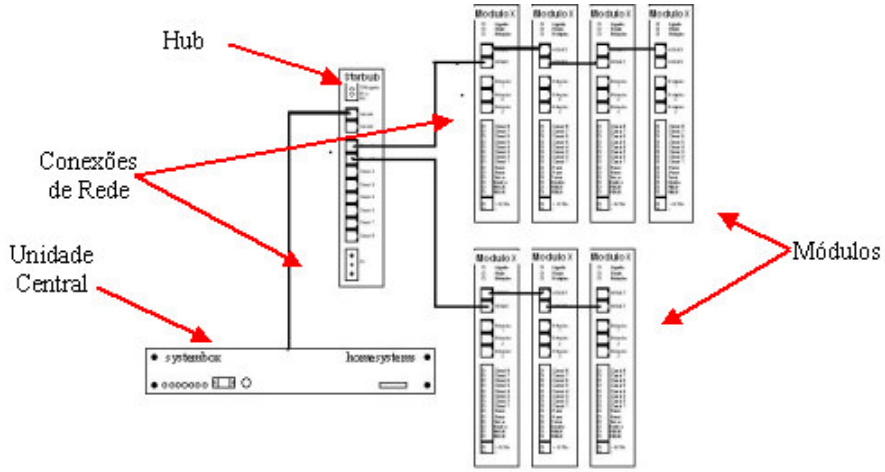


Figura 5.1 : Arquitetura física

Durante a realização deste trabalho, as interfaces de alto nível com usuário eram o aparelho telefônico e navegadores *internet*. Estas funcionalidades são suportadas pela unidade central, a qual faz o gerenciamento seguro das chamadas telefônicas e executa um servidor apache para automatizar a disponibilização de informações do sistema para um usuário através de um *browser* padrão.

Uma ferramenta de configuração e manutenção, denominada *commander*, permite a configuração e a manutenção do sistema. Na Figura 5.2 pode ser visualizada a janela principal da ferramenta. No lado direito da podem ser observados os diversos módulos disponíveis pela arquitetura, no lado esquerdo as funcionalidades oferecidas pela ferramenta e na área central têm-se a área de trabalho, onde é configurada a rede, pela seleção e endereçamento dos dispositivos.

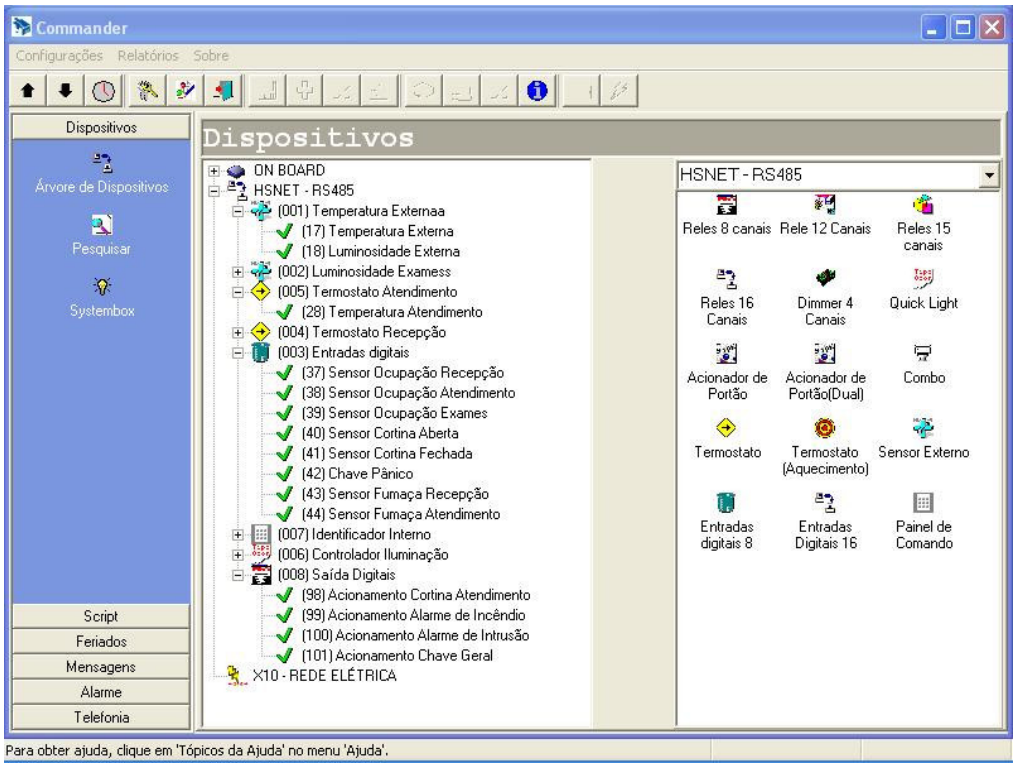


Figura 5.2 : Ferramenta de Configuração

Um conjunto de arquivos de configuração é manuseado pela ferramenta de configuração durante a etapa de configuração do sistema e posteriormente são enviados à unidade central que os utiliza durante a execução do programa de controle.

5.1.1 Dispositivos Disponíveis

A arquitetura Homesystems oferece um conjunto de dispositivos que podem ser utilizados para a automação de sistemas de iluminação, de ar condicionado e aquecimento, controle de portões e cortinas, etc. A seguir são listados alguns módulos disponíveis e suas finalidades:

- **Painel de comandos:** até 9 teclas e 9 *leds* que podem ser configurados para entradas/indicação de comandos e cenários;
- **Sensor Externo:** 2 entradas analógicas, podendo ser usado para indicação de nível de luminosidade, temperatura, etc. A variável pode assumir valores inteiros entre 0 e 255;
- **Relés de 8 e de 15 canais:** 8 ou 15 saídas digitais que podem ser utilizados para acionar cargas do tipo ligado/desligado;
- **Relés de 12 e de 16 canais:** 12 ou 16 saídas digitais, com possibilidade de ligação no próprio módulo de teclados de 9 teclas e definição de até 8 cenários que permitem configurar como ativado ou desativado cada um dos canais de saída para cada um dos cenários;
- **Dimmer 4 canais:** 4 saídas analógicas que podem ser configuradas entre 0 e 100%, com o objetivo de acionar cargas que possam receber tensão variável;
- **QuickLight:** 9 saídas analógicas, com possibilidade de ligação no próprio módulo de teclados de 9 teclas e definição de até 8 cenários que permitem configurar valores entre 0% e 100% cada um dos canais de saída para cada um dos cenários;
- **Entradas digitais de 8 canais:** 8 entradas digitais com possibilidade de configuração NA/NF (Normalmente Aberto/Normalmente Fechado). Podem ser utilizadas para indicar o estado de qualquer grandeza que possa ser representado pelos estados ligado/desligado;
- **Entradas digitais de 16 canais:** 16 entradas digitais com possibilidade de configuração NA/NF (Normalmente Aberto/Normalmente Fechado) e 2 entradas analógicas;
- **Termostato:** utilizado para o controle de atuadores de subsistemas de HVAC e como interface de usuário com este tipo de subsistema. Através dele podem ser definidos *setpoints* de refrigeração e aquecimento e realizado o acionamento dos atuadores do Ar condicionado, através de ligação elétrica entre eles.
- **Acionador de Portão:** este módulo é usado para controle de portões, possuindo entradas para sensores típicos dessa aplicação e saída para ligação dos motores de acionamento;
- **Combo:** módulo leitor de cartões para identificação de usuários.

O arquivo hsconfig.ini possui a configuração dos parâmetros de cada módulo.

5.1.2 Outros Conceitos Usados pela Arquitetura

- **Variáveis:**

O sistema permite a utilização de variáveis. Na manipulação de variáveis estão disponíveis as opções atribuir valor, incrementar valor, decrementar valor, atribuir

E/S e atribuir variável. A configuração de variáveis é armazenada no arquivo variable.ini.

- **Temporizações:**

Diversos módulos permitem definir temporizações para o comando ligar e na execução de qualquer procedimento pode-se implementar um comando específico definido como esperar. Algumas tarefas de controle possuem temporizações específicas e podem ser configuradas diretamente no próprio módulo, como exemplo pode-se citar o acionador de portão que possui alguns tempos definidos diretamente na configuração do módulo.

- **Agrupamentos:**

A arquitetura permite definir grupos. Um grupo comporta-se como uma entrada ou saída digital e sobre ele podem ser executados comandos e verificados estados típicos de uma saída digital.

- **Alarmes e avisos:**

Podem ser definidos diferentes alarmes os quais podem ser enviados a telefones previamente cadastrados. Para cada alarme podem ser especificadas algumas temporizações (atraso na ativação, tempo de ciclo ligado, tempo de ciclo desligado, tempo de acionamento), podem ser definidas saídas que serão acionadas por este alarme e podem ser definidas mensagens para os eventos de invasão, ativação e desativação do alarme.

Uma simplificação do alarme é o aviso. Um aviso pode estar vinculado a qualquer canal de qualquer dispositivo do sistema. As opções de notificação são através de mensagem (vinculada ao canal na configuração) ou através de sinal sonoro (*bip*) e será emitido para uma lista de telefones configurados previamente. Os arquivo alarm.ini possui toda a configuração de alarme e avisos que é programada através da ferramenta de configuração.

Os arquivos alarm.ini e voicemsgs.ini possuem a configuração dos alarmes, de telefonia e de mensagens de voz utilizadas.

- **Calendário e Relógio**

Através de um calendário podem ser definidos os dias úteis, os feriados e os finais de semana e um relógio interno permite configurar eventos temporais. Desta forma é possíveis definir eventos baseados em data, horário (com a possibilidade de definir os dias da semana para os quais está sendo configurado o horário, podendo ser incluídos ou excluídos os feriados) e intervalo de horário o qual permite definir de um horário inicial e um horário final. O arquivo holidays.ini possui a configuração do calendário.

- **Scripts**

A ferramenta de configuração oferece um recurso denominado *scripts*. Através dele é possível definir eventos e associar procedimentos que serão executados na ocorrência da condição verdadeira ou falsa de um determinado evento.

Um evento pode estar associado a uma entrada ou saída (digital ou analógica), a um horário ou intervalo de horário, a uma data ou a uma variável. Diversas condições podem ser previstas para um evento, associadas através dos operadores lógicos “E” e “OU”.

Um procedimento estabelece ações que devem ser executadas no sistema. Uma ação pode corresponder ao acionamento de uma saída, à execução de um outro procedimento, à vinculação de uma saída, à atribuição de uma variável, à determinação

de um tempo de espera, ao toque de uma mensagem ou telefone e à execução de um alarme ou de um aviso.

Os arquivos events.ini e procedure.ini possuem toda a configuração de eventos e procedimentos programados através da ferramenta de configuração.

– **Telefonia**

Como a interface principal com o sistema é através do telefone a ferramenta permite configurar a topologia das conexões telefônicas no *systembox* (ligado diretamente, através de central interna, ramal de atendimento, tipos de toques, temporizações, etc). Também permite definir senhas para acesso externo e interno e níveis de acesso.

Na etapa final de configuração da telefonia podem ser montados menus em nível com números que estarão associados com procedimentos para comando via telefone. O arquivo telephony.ini e phonebook.ini possuem toda a configuração de telefonia programada através da ferramenta de configuração.

5.1.3 Mapeamento do *Framework*

No mapeamento do *framework* deve-se considerar a funcionalidade oferecida por um determinado módulo e a sua configuração. Primeiramente, um dispositivo lógico deverá estar mapeado em um módulo que ofereça a funcionalidade requerida. A seguir está relacionado para que módulos devem ser mapeados as classes do *framework*:

- **AnalogSensor**: sensor de temperatura e luminosidade (2 canais), entradas digitais 16 canais (2 canais analágicos);
- **AnalogActuator**: *dimmer* de 4 canais e *dimmer* de 9 canais;
- **BinarySensor**: entradas digitais *onboard*, entradas digitais de 8 e 16 canais e painel de comando;
- **BinaryActuator**: saídas digitais *onboard*, relés de 8, 12, 15 e 16 canais e *dimmer* de 9 canais (cada canal pode ser configurado opcionalmente para ter um comportamento digital);
- **MultistateSensor / Multistate Actuator**: não implementado diretamente. Implementações dessa funcionalidade usam entradas/saídas analógicas com definição dos valores tais como: 0=ligado, 1=desligado, 2=parado, etc;
- **Controller**: todos os módulos executam função de controle específico, de acordo com a finalidade do módulo;
- **UserInterface**: as interfaces padrão telefone e *internet* são funcionalidades oferecidas pelo *systembox* com configurações específicas. Determinados módulos fornecem interfaces específicas (termostato: display, teclado, *leds* sinalizadores, teclados: *leds* sinalizadores, combo: cartão e acionador de portão: infravermelho, etc.);
- **Persistent**: alguns módulos possuem configuração específica de parâmetros. O módulo Combo armazena número de cartões, o módulo de comando de portões armazena códigos de RF. Os atributos e funcionalidades irão corresponder às configurações e operação desses módulos. As variáveis utilizadas nos scripts correspondem a um meio de armazenamento em RAM, gerenciadas através do sistema operacional do *Systembox*;
- **Scene**: implementado através de eventos (*preconditions*) e procedimentos (*procedures*) no *Systembox*.

Resolvida a questão do mapeamento físico, a segunda questão refere-se à configuração correta dos diferentes arquivos de configuração para que eles reflitam o modelo de um sistema que foi modelado usando o *framework* proposto.

A seguir, no restante desse item procura-se demonstrar como deve ser feito este mapeamento. Uma descrição mais detalhada pode ser encontrada no Anexo B, onde está descrito o mapeamento de cada atributo do modelo para atributos dos arquivos de configuração, bem como uma descrição de como são utilizados os métodos do modelo.

No arquivo *hsconf.ini* devem ser configurados os módulos que farão parte da rede que corresponde ao mapeamento dos dispositivos físicos (da classe *PhysicalDevice*) que foram definidos no *framework*. A Figura 5.3 resume a representação desse mapeamento. No lado esquerdo tem-se a especificação da classe e no lado direito uma parte do arquivo de configuração do dispositivo físico. A parte central mostra a equivalências entre os atributos.

Como pode ser observado, alguns atributos não possuem equivalência nos arquivos de configuração e alguns atributos dos mesmos não possuem equivalência no modelo. No primeiro caso têm-se atributos que não interferem na funcionalidade oferecida pela arquitetura, enquanto no segundo têm-se atributos que são utilizados por aspectos visuais da ferramenta de configuração ou em alguma configuração específica oferecida pela arquitetura que não estão previstas no *framework*.

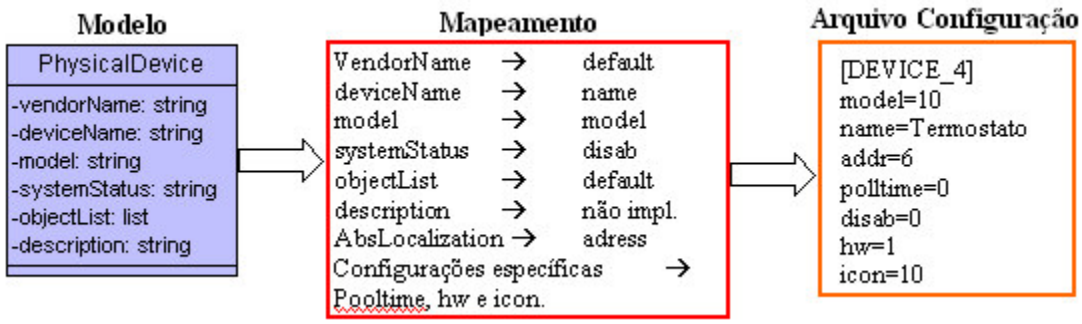


Figura 5.3 : Mapeamento dos Atributos da Classe *PhysicalDevice*

A seguir destacam-se os atributos que pertencem a estes casos que foram observados na figura anterior:

- os atributos `vendorName` e `objectList` são indicados por *default* no mapeamento pois, conforme pode ser observado no item anterior, a definição dos objetos que compõem cada módulo é uma característica fixada na escolha do módulo e o fabricante não indica nenhuma referência nos arquivos de configuração do módulo;
- o atributo `description` não é implementado pela arquitetura Homesystems;
- o atributo `pooltime` é utilizado para indicar o tempo de intervalo entre as leituras sucessivas do módulo (escravo) pelo *systembox* (mestre), característica de configuração do modelo de comunicação mestre-escravo;
- os atributos `hw` e `icon` e são utilizados para implementação de aspectos visuais na ferramenta de configuração.

Após a especificação dos dispositivos físicos deverá ser feito o mapeamento de cada dispositivo lógico. Esta configuração é realizada no mesmo arquivo e o mapeamento dos atributos pode ser observado na

Figura 5.4.

No lado esquerdo tem-se a especificação das classes do modelo que são mapeadas no arquivo de configuração, parcialmente representado no lado direito da figura. A parte central mostra a equivalências entre os atributos.

Aqui também se observa que alguns atributos do *framework* são *default* na arquitetura Homesystems, alguns não são implementados e alguns atributos da arquitetura não são mapeados no *framework*.

O detalhamento dos dois primeiros casos pode ser encontrado no anexo B. Relativamente ao terceiro caso têm-se os atributos:

- [unit_x] utilizado como separador;
- `icon` e `form` que são utilizados para configuração de aspectos visuais no *commander*;
- `disab` que indica que a unidade está desabilitada pois o arquivo de configuração sempre contém todas as unidades de todos os módulos configurados;
- `logged` que indica que o *systembox* deve manter um arquivo com histórico para aquela unidade;

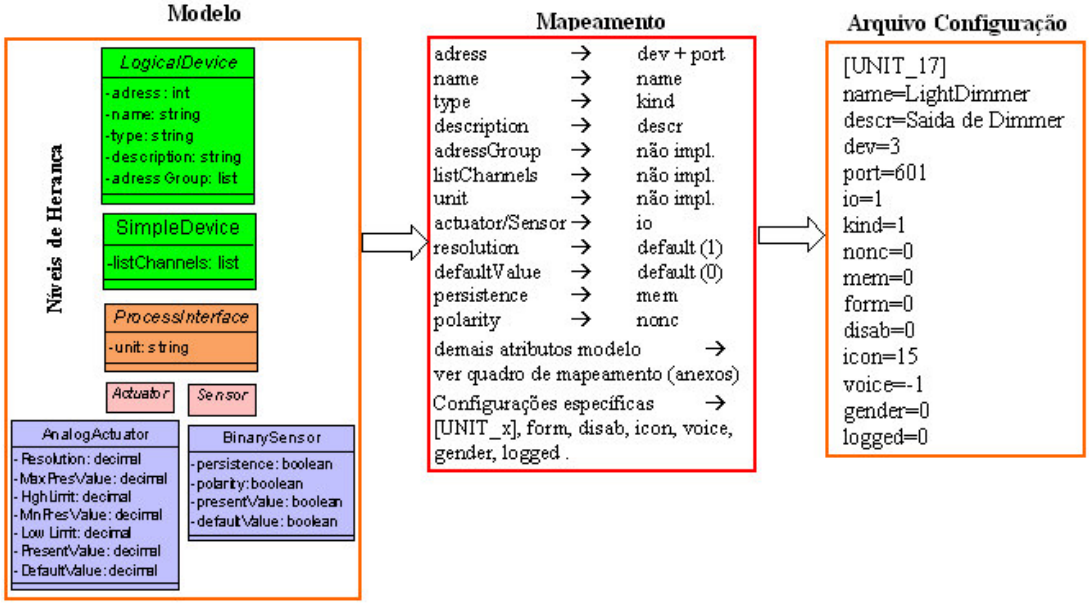


Figura 5.4 : Mapeamento dos Atributos dos Dispositivos Lógicos

- `voice` indica a mensagem de voz que está vinculada à unidade, se houver, e `gender` se esta mensagem será com voz masculina ou feminina.

Por sua vez, o mapeamento dos cenários pode ser observado na Figura 5.5. No lado esquerdo têm-se as classes de origem do modelo e a direita extratos parciais dos arquivos de configuração.

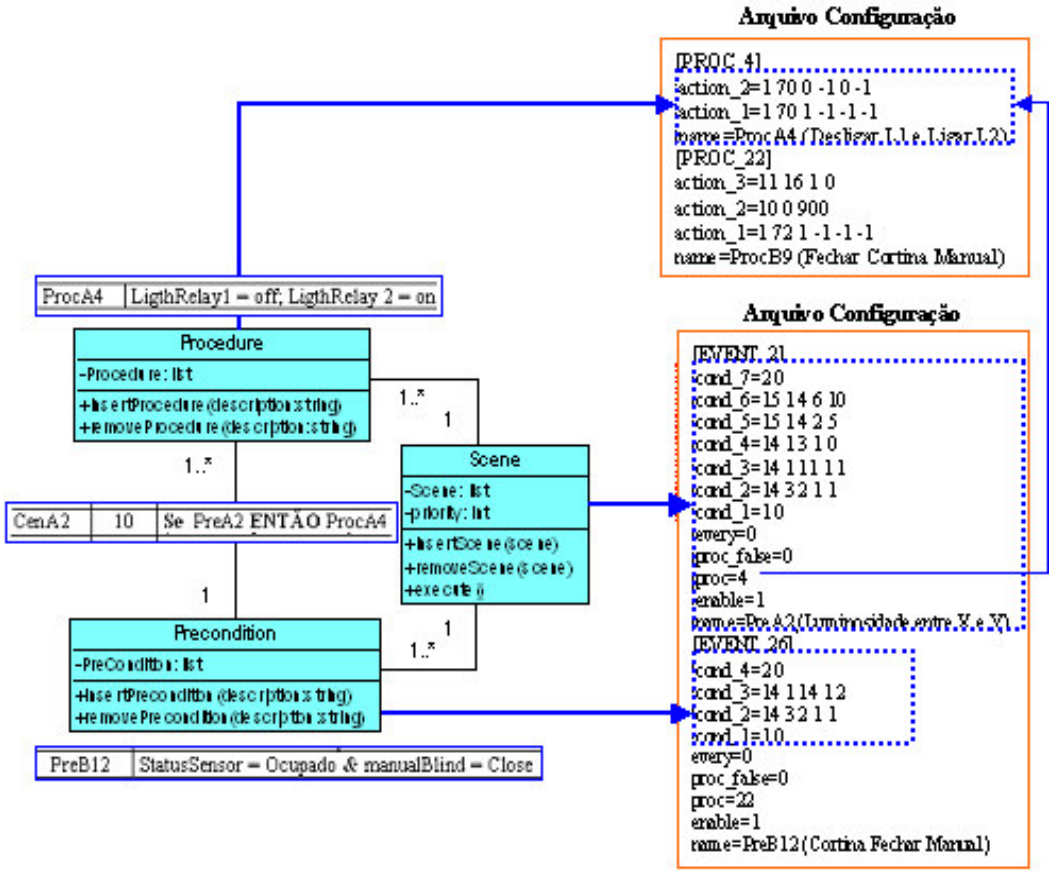


Figura 5.5 : Mapeamento dos Cenários

Para facilitar a identificação de equivalência entre o modelo e os arquivos de configuração junto a estas classes encontra-se a representação dos procedimentos, precondições e cenários de acordo com a sintaxe definida no item 3.2.2 e foram utilizados setas e retângulos delimitadores, indicando onde os conceitos estão sendo implementados nos arquivos de configuração.

Os números após o sinal de igual estão relacionados com a sintaxe adotada pelo fabricante para representar os procedimentos e eventos. Por exemplo na primeira e na segunda linha após o delimitar PROC_4 (`action_2 = 1 70 0 -1 0 -1`) o primeiro campo indica que o comando é o acionamento de uma saída, o segundo indica a `unit` que será acionada e os demais campos estão relacionados com os parâmetros desse comando. A última linha (`name = ProcA4(Desligar L1 e Ligar L2)`) indica o nome do procedimento. Uma sintaxe similar é utilizada na definição dos eventos.

O arquivo *procedures.ini* contém a implementação dos procedimentos (classe *Procedure*) e o arquivo *events.ini* contém a implementação dos cenários (classe *Scene*). Na parte superior de um evento, delimitado pelo marcador `[EVENT_x]` tem-se a descrição das pré-condições (classe *Precondition*) que deve ser avaliadas e na parte inferiro é indicado o procedimento que deve ser executado quando o resultado da avaliação for verdadeiro (`proc`) ou quando for falso (`proc_false`), as quais correspondes às condições SE e SENÃO definidas nos cenários do *framework*.

Dois outros atributos aparecem neste arquivo são `every` indica se o procedimento deve ser executado sempre ou uma única vez e `enable` que indica se o procedimento está habilitado ou não.

Por indisponibilidade da especificação da sintaxe usada nestes arquivos de configuração e limitação de tempo para fazer-se engenharia reversa usando a ferramenta de configuração, não se fez o mapeamento da sintaxe adotada com arquivo de configuração. Durante a realização dos estudos de caso, os procedimentos, as precondições e os cenários foram construídos usando-se os recursos de alto nível oferecido pela ferramenta, muito semelhante à sintaxe especificada no *framework*, conforme pode ser observado posteriormente neste capítulo.

5.2 Exemplos de Mapeamento dos Dispositivos Lógicos

O primeiro padrão estabelecido no capítulo anterior foi de sensor conectado diretamente ao atuador. Na arquitetura em análise isto é possível de ser implementado apenas quando se utiliza um módulo de saída que possui teclado acoplado diretamente (Relés de 12 e 16 canais e o *Dimmer* de 9 canais). Como estes módulos foram projetados para funcionar independentemente do mestre da rede, se o componente da classe *Sensor* for uma entrada do teclado, após configuração local, a função de controle será executada no próprio módulo.

A implementação do exemplo, modelado usando o *framework*, representado na Figura 4.7 pode ser observada na Figura 5.6. Os módulos disponíveis pela arquitetura estão representados por retângulos identificados, a linha cheia que conecta os módulos representa uma conexão de rede e a linha com a seta, entre os objetos do *framework*, indica uma conexão lógica entre estes objetos.

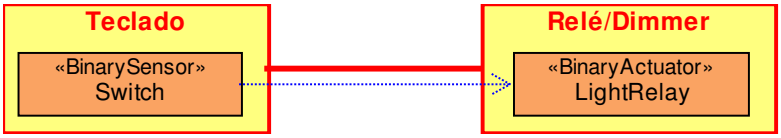


Figura 5.6 : Implementação de um sensor conectado diretamente no atuador

O segundo padrão especificado tem por objetivo representar sistemas de controle de variáveis contínuas, no qual se deseja que uma variável de saída mantenha um valor de referência previamente estabelecido.

Os módulos que se enquadram neste padrão são o termostato para aquecimento, o termostato para refrigeração e o termostato (com ambas funcionalidades), os quais possuem sensor de temperatura interna, uma interface com o usuário local e executam uma função de controle local para manter o *setpoint* estabelecido.

A implementação do exemplo da Figura 4.9 está representado na Figura 5.7. Como o termostato possui ligação elétrica com o aparelho de Ar Condicionado (AC), o qual implementa os atuadores (da classe *Actuator*), no termostato estão modeladas apenas as interfaces (da classe *CommunicationInterface*) com estes dispositivos lógicos.

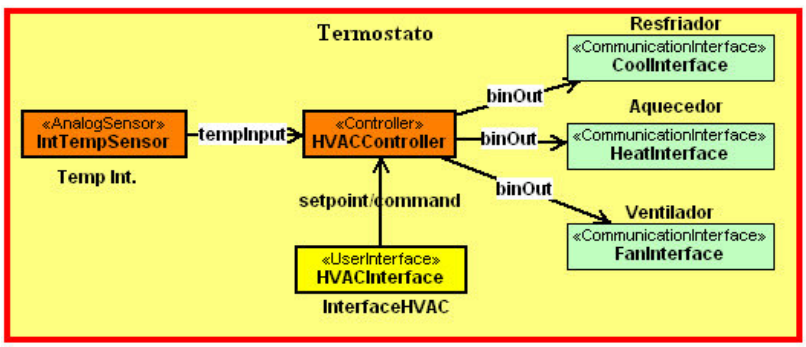


Figura 5.7 : Implementação do controle de variáveis contínuas

O terceiro padrão destina-se a representação de sistemas com diferentes cenários. Duas alternativas de implementação de cenários são possíveis: uma local mais limitada e outra remota, que pode estar associada com qualquer funcionalidade do sistema.

Na primeira, quando forem utilizados relé de 12 ou 16 canais e *dimmer* de 9 canais é possível compor até 8 cenários com combinação de valores para as saídas (0 e 100% para relés e qualquer valor intermediário para *dimmer*). Estes cenários ficam armazenados no próprio módulo e podem ser executados a partir do teclado local.

Na segunda opção, o *systembox* será responsável pela execução do cenário estabelecido através dos scripts. Qualquer funcionalidade do sistema (entrada, saída, relógio, calendário, etc.) pode gerar eventos e qualquer combinação de procedimentos pode estar associada a um evento. As interfaces disponíveis (telefonia e internet) também podem ser usadas para emitir comandos. O exemplo ilustrado na Figura 4.11 pose ser implementado conforme representado na Figura 5.8.

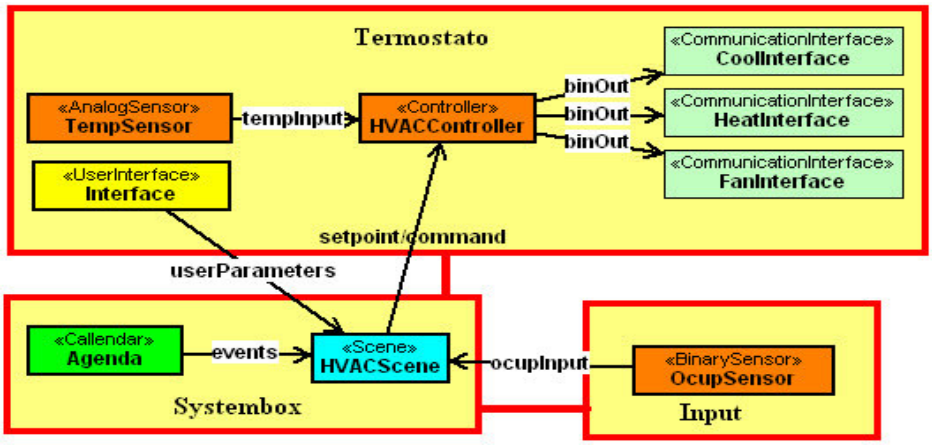


Figura 5.8 : Implementação de diferentes cenários

5.3 Estudo de Caso: Automação de Escritório

O estudo de caso proposto, embora de dimensões reduzidas se comparado com um projeto de automação real, permite utilizar amplamente os conceitos discutidos nas seções anteriores, uma vez que para sua implementação deverão ser utilizados diversos subsistemas especificados no modelo.

A análise de aplicações reais mostrou que um projeto de grande porte difere, de uma maneira geral, apenas pelo aumento dos componentes que integram cada subsistema, o que muitas vezes provoca a divisão em subsistemas menores para facilitar o gerenciamento.

A Figura 5.9 representa a planta baixa de um escritório, onde L1 e L2 são luminárias fluorescentes com acionamento independente, L3 representa luminárias incandescente externas (com possibilidade de dimerização), AC1 é um ar condicionado de janela e cort1 representa uma cortina motorizada.

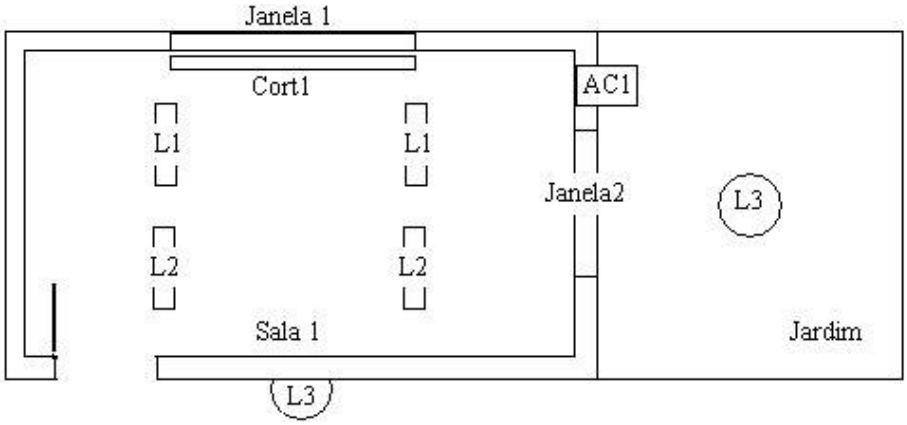


Figura 5.9 : Planta baixa de escritório

A partir dessa descrição geral do ambiente foram definidos diversos requisitos para o prédio com objetivo de permitir a modelagem de diversos subsistemas propostos. Foram especificados os seguintes requisitos:

- As lâmpadas deverão permanecer ligadas apenas se a sala estiver ocupada de acordo com luminosidade do ambiente. Um sistema de acionamento manual deve permitir o acionamento da iluminação durante um determinado intervalo independente do sistema automático de controle. **Objetivo:** controle de iluminação por ocupação e por luminosidade, controle manual/automático com prioridades diferentes.
- A iluminação externa deverá ser acionada apenas à noite e a partir da meia noite deverá ser reduzida para 50%. **Objetivo:** controle de iluminação por luminosidade e por horário.
- Temperatura ambiente controlada em 22^0C. A partir das 18h o sistema de controle de temperatura deverá estar acionado apenas se a sala estiver ocupada. O sistema poderá ligar (conforme temperatura ambiente) nos dias úteis a partir das 7h mesmo que o ambiente esteja desocupado. **Objetivos:** controle de temperatura por horário, calendário e ocupação.
- A cortina da janela 1 deverá estar aberta durante o dia, sempre que o controle de intrusão esteja desativado. Quando a temperatura externa for maior do que X^0C ela deverá ser fechada e a iluminação interna ajustada. Um sistema de acionamento manual deve permitir o acionamento da cortina durante um determinado intervalo independente do sistema automático de controle. **Objetivos:** controle da cortina por luminosidade, temperatura e ocupação do ambiente, controle manual/automático com prioridades diferentes. Interação entre os dois subsistemas
- Diariamente às 18h o sistema de irrigação deverá ser acionado por um tempo X. **Objetivo:** utilização de um subsistema com controle de acionamento por horário e tempo de duração.
- Quando o sistema de controle de intrusão for ativado, a iluminação interna deverá ser desacionada, o sistema de controle de temperatura desativado, exceto dias úteis no intervalo de almoço (12 às 14h) e a cortina deverá ser fechada. **Objetivo:** estabelecer cenários diferentes para controle de intrusão.
- Em caso de detecção de intrusão o alarme deverá ser acionado, a cortina deverá abrir e, se for à noite, a iluminação deverá ser acionada. **Objetivo:** explorar a interação entre diferentes subsistemas
- Sistema de detecção de fumaça deverá acionar alarme de incêndio em caso de detecção de fumaça (dois níveis: maior quando sala ocupada e menor quando

sala desocupada). **Objetivo:** explorar cenários diferentes de ativação para o sistema de controle de incêndio.

A seguir serão apresentadas as etapas de projeto, conforme definidas no capítulo anterior.

1. Definição das funcionalidades e comportamentos

A Figura 5.10 apresenta um diagrama *use case* UML com as funcionalidades que deverão ser implementadas pelo sistema, o qual foi construído a partir da descrição da especificação do ambiente.

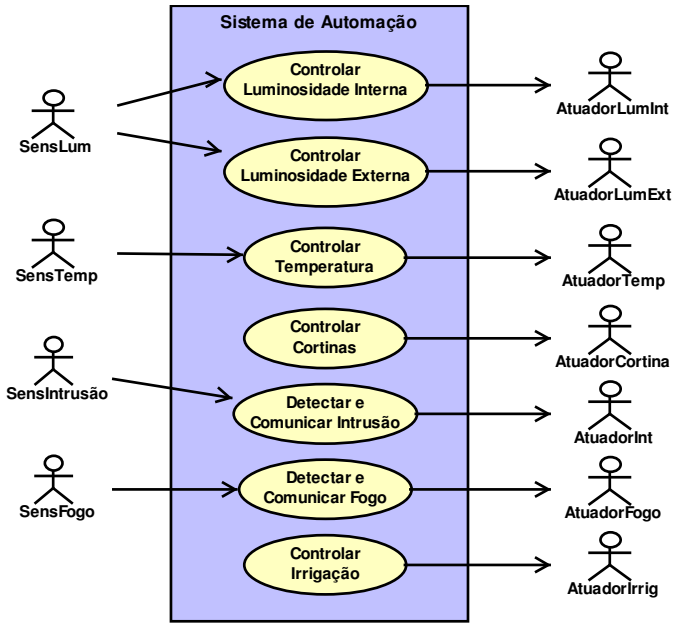


Figura 5.10 : Funcionalidades especificadas

Esse diagrama é complementado com informações de comportamento que estão descritos na Tabela 5.1, as quais posteriormente permitirão o detalhamento de cada subsistema que será implementado.

Tabela 5.1 : Detalhamento dos comportamentos

Funcionalidade	Comportamento
Controlar temperatura (aquecer ou resfriar)	Manter em 22^0C; Após 18h acionar apenas se houver ocupação; Habilitar dias úteis a partir das 7h Desligar ao habilitar controle de Intrusão, exceção dias úteis no horário de almoço (12h às 14h);
Controlar luminosidade interna	Regular conforme luminosidade externa; Desligar se ambiente estiver desocupado; Ligar em caso de intrusão, se noite; Permitir comando manual.
Controlar luminosidade externa	Acionar (100%) quando noite; Às 24h reduzir para 50%; Acionar (100%) em caso de intrusão, se noite; Permitir comando manual com prioridade sobre o automático por um tempo pré-determinado.

Funcionalidade	Comportamento
Controlar Cortina (abrir e fechar)	Aberta durante o dia quando controle de intrusão estiver desabilitado; Fechar quando temperatura externa for maior do que 26^0C; Fechar ao habilitar controle de Intrusão; Abrir quando intrusão detectada; Permitir comando manual com prioridade sobre o automático por um tempo pré-determinado.
Detectar e comunicar Intrusão	Emitir alarme local em caso de intrusão; Abrir cortina e quando noite acionar iluminação.
Detectar e comunicar Fogo	Emitir alarme local em caso de detecção de fogo; Ajustar o nível de detecção a ocupação do ambiente.
Controlar Irrigação	Acionar diariamente as 18h; Manter acionado por 5mim.

2. Definir os agentes externos que irão interagir com o sistema e definir as políticas de gerenciamento

Foram definidos os seguintes usuários e para eles foram definidas as seguintes políticas:

- Administrador: responsável pela habilitação de usuários, pela configuração de parâmetros dos os subsistemas e pela criação/alteração de cenários;
- Usuário: usuário habilitado a desabilitar o sistema de intrusão, controlar a temperatura da sala de atendimento e selecionar cenários previamente estabelecidos.

3. Dividir o ambiente em sub-regiões ou zonas de acordo com as funcionalidades que devem ser controladas

Considerando que existe um ambiente único não se fez necessário dividir o ambiente em sub-regiões de controle, necessidade típica de sistemas maiores com controle mais complexo.

4. Selecionar funcionalidades e subsistemas que comporão o modelo

Os seguintes sistemas serão utilizados para modelar esta aplicação:

- Controle de Iluminação (*lighting*) para modelar as funcionalidades controlar luminosidade interna e controlar luminosidade externa;
- Controle de HVAC (HVAC) para modelar a funcionalidade controlar temperatura
- Controle de Intrusões (*intrusion*) para modelar a funcionalidade detectar e comunicar intrusão;
- Controle de Acessos (*Acess*) utilizada para permitir o habilitação/desabilitação do sistema de intrusões apenas à usuários habilitados;
- Controle de Venezianas (*Blind*) para modelar a funcionalidade controlar cortina;
- Controle de Incêndio (*Fire*) para permitir a modelagem da funcionalidade detectar e comunicar fogo;

A funcionalidade controlar irrigação foi modelada de maneira independente de qualquer dos subsistemas especificados no *framework* com o objetivo de demonstrar possibilidade de expandir os subsistemas definidos, usando-se os mesmos conceitos que suportam os subsistemas definidos na proposta.

Na Figura 5.11 pode ser observado o detalhamento da funcionalidade controlar temperatura, usando as funcionalidades que foram modeladas para o subsistema de controle de HVAC no framework, conforme Figura 3.4

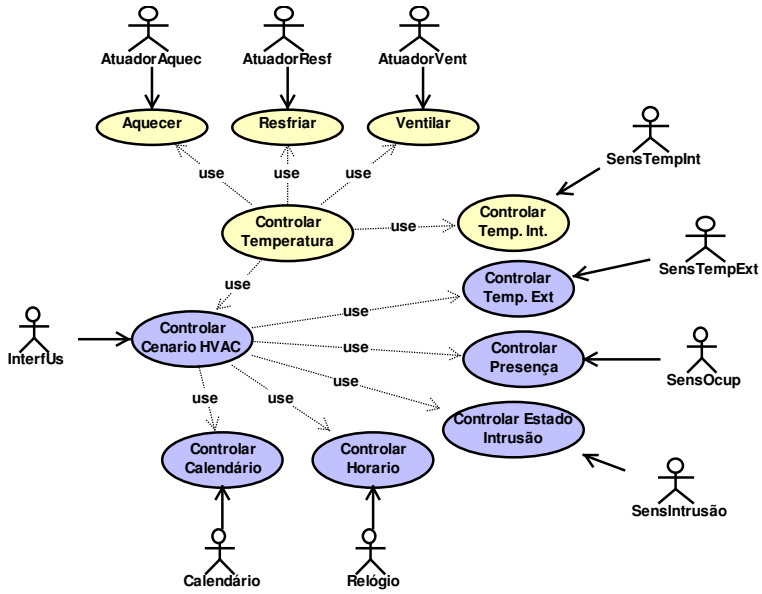


Figura 5.11 : Detalhamento das funcionalidades

Pode-se observar nesta figura que as funcionalidades *ControlExtSensors* e *ControlSensors* definidas na Figura 3.4 foram expandidas a fim de permitir a modelagem da funcionalidades que garantirão o comportamento desejado desse subsistema.

Outra característica que pode ser observada é que a modelagem da funcionalidade *ControlHVAC*, representada nesse diagrama pela funcionalidade Controlar Temperatura, utiliza apenas as funcionalidades que permitem implementar o controle de temperatura que foi especificado para o ambiente (aquecer, resfriar e ventilar).

No anexo C podem ser encontrados os diagramas que representam as funcionalidades a serem implementadas pelos demais subsistemas.

5. Definir as políticas de operação de cada subsistema

A partir da análise do comportamento desejado, foram modelados diferentes três estados de ocupação para o ambiente:

- **ocupado** para quando o sistema de intrusão estiver desativado;
- **desocupado** para quando ele estiver ativado;
- **alarme** para quando o sistema de intrusão for acionado.

A seguir será detalhado o comportamento desejado, com foco para cada subsistema:

- **Subsistema de controle de iluminação:**
  - Se luminosidade interna maior do que Y lumens: L1 e L2 desligadas;
  - Se luminosidade interna entre X e Y lumens: L1 desligada e L2 Ligada;
  - Se luminosidade interna menor do que X lumens: L1 e L2 ligadas;
  - Se noite: ligar L3 com 100%;
  - Se noite, após as 24h: ligar L3 com 50%;
  - Quando estado de ocupação = desocupado ou quando e o controle presença indicar que o ambiente está desocupado: L1 e L2 desligados;
  - Quando estado de ocupação = alarme, se noite: L1, L2 e L3 ligados;
  - L1, L2 e L3 podem ser acionadas manualmente. O comando manual tem prioridade durante um tempo de 15min sobre os demais comandos, exceto se controle de intrusão for ativado.

- **Subsistema de controle de temperatura:**
  - Manter a temperatura em 22^0C (parâmetro alterável pelo usuário);
  - A partir das 18h quando a sala estiver desocupada o subsistema deverá apenas ventilar;
  - Quando controle de intrusão estiver ativado deverá ser desligado;
  - Nos dias úteis, mesmo com o controle de intrusão ativado não deverá desligar no horário de almoço (intervalo das 12h às 14h);
  - Nos dias úteis a partir das 7h o controle de temperatura deverá estar ligado.Se até às 9h não houver ocupação no ambiente, ele deve ser desligado.
- **Subsistema de controle de cortinas:**
  - A cortina deverá estar aberta durante o dia quando a sala quando o sistema de intrusão estiver desativado;
  - Se a temperatura externa for maior do que 26^0C a cortina deverá ser fechada;
  - Quando o ambiente estiver desocupado ela deverá ser fechada;
  - A cortina deverá abrir quando a sala estiver em alarme;
  - Permitir comando manual. Os comandos manuais têm prioridade durante um tempo de 15min sobre os demais comandos, exceto se controle de intrusão for ativado;
- **Subsistema de controle de acesso:**
  - O usuário deverá ser identificado após abrir a porta para desacionar o sistema de controle de intrusão, sempre que ele for o primeiro a entrar na sala. O procedimento é semelhante é realizado na saída da sala, quando ele for o último a deixar o ambiente.
- **Subsistema de controle de intrusões:**
  - Se o sistema de controle de acesso autenticar o usuário, o sistema de controle de intrusão muda para estado habilitado após um tempo X se ele estiver no estado ocupado ou para o estado desabilitado imediatamente se ele estiver no estado desocupado ou no estado de alarme.
  - Após entrar na sala, se o sistema de intrusão estiver habilitado, o usuário possui um tempo Y para desabilitar o sistema, caso contrário o sistema de intrusão passa para o estado de alarme;
  - Se o sistema de intrusão entrar no estado de alarme deverá emitir um sinal sonoro;
- **Subsistema de controle de Incêndio:**
  - Ajustar o nível de detecção conforme a ocupação do ambiente (considerando se o sistema de controle de intrusão está habilitado ou desabilitado);
  - Emitir um sinal sonoro em caso de detecção de fogo.
- **Subsistema de controle de irrigação:**
  - Ligar diariamente às 18h e manter ligado por 15min.

6. Selecionar os dispositivos lógicos e suas conexões

A partir das funcionalidades especificadas na no item 4 foram selecionados os dispositivos lógicos necessários para representar cada funcionalidade. A tabela Tabela 5.2 mostra o equivalências entre as funcionalidades especificadas e o dispositivo lógico usado para modelagem.

Ao definir-se o dispositivo lógico que será usado para representar cada funcionalidade deve-se analisar o comportamento desejado para o subsistema e

determinar a classe de origem do mesmo. Por exemplo, deve-se determinar se um dispositivo lógico necessário para modelar uma variável de controlada, derivado da classe Sensor, será um *AnalogSensor*, um *BinarySensor* ou um *MultistateSensor*..

Tabela 5.2 : Definição dos Dispositivos Lógicos

Funções associadas a variáveis de entrada	Controlar Luminosidade Externa	ExtLightSensor (analogSensor)	**Dispositivos Lógicos**
	Controlar Luminosidade Interna	IntLightSensor (analogSensor)	
	Controlar Presença	OcupSensor (binarySensor)	
	Indicar Status do Ambiente (desocupado/ocupado/alarme)	StatusSensor, StatusActuator (multistateSensor/Actuator)	
	Operação Manual L1	ManualL1 (binarySensor)	
	Operação Manual L2	ManualL2 (binarySensor)	
	Operação Manual L3	ManualL3 (binarySensor)	
	Controlar Horário	Clock	
	Controlar Calendário (dias úteis)	Callendar	
	Controlar Temperatura Interna	IntTempSensor (analogSensor)	
	Controlar Temperatura Externa	ExtTempSensor (analogSensor)	
	Indicar cortina aberta	BupSensor (binarySensor)	
	Indicar cortina Fechada	BdownSensor (binarySensor)	
	Operação Manual Cortina (open/ close)	ManualL3 (multistateSensor)	
	Detectar Intrusão	OcupSensor (binarySensor)	
	Detectar Fogo	SmokeSensor (analogSensor)	
Interfaces	Identificação do Usuário (no acesso)	IntInterface (UserInterface)	
	Setpoint do HVAC	HVACInterface (UserInterface)	
Funções associadas	Controlar Temperatura	HeatActuator, CoolActuator, FanActuator (binaryActuator)	
	Controlar L1, L2 e L3	LightRelay1 e LightRelay2 (binaryActuator) LightDimmer (analogActuator)	
	Controlar Cortina atendimento	BlindRelay (binaryActuator)	
	Indicar alarme de incêndio	FireAlarmRelay (binaryActuator)	
	Indicar alarme de intrusão	IntAlarmRelay (binaryActuator)	
Controladores	Controlar Temperatura	HVACController (HVACController)	
	Controlar Cortina	BlindController (BlindController)	
	Temporizadores	Implementados nos cenários	
Cenários	Iluminação	LigthScene	
	HVAC + Cortina	HVACScene	
	Incêndio	FireScene	
	Intrusão	IntrusionScene	
	Irrigação	IrrigationScene	
Base de Dados	Cadastro de Usuários (autenticar Usuário)	UserInfo	

Na Figura 5.12 pode ser observado o modelo de uma parte do sistema correspondente aos subsistemas de HVAC e de cortina, onde além dos dispositivos lógicos que modelam as funcionalidades especificadas pode ser observado também as conexões necessárias entre os dispositivos, as quais representam a interface necessária entre os dispositivos lógicos.

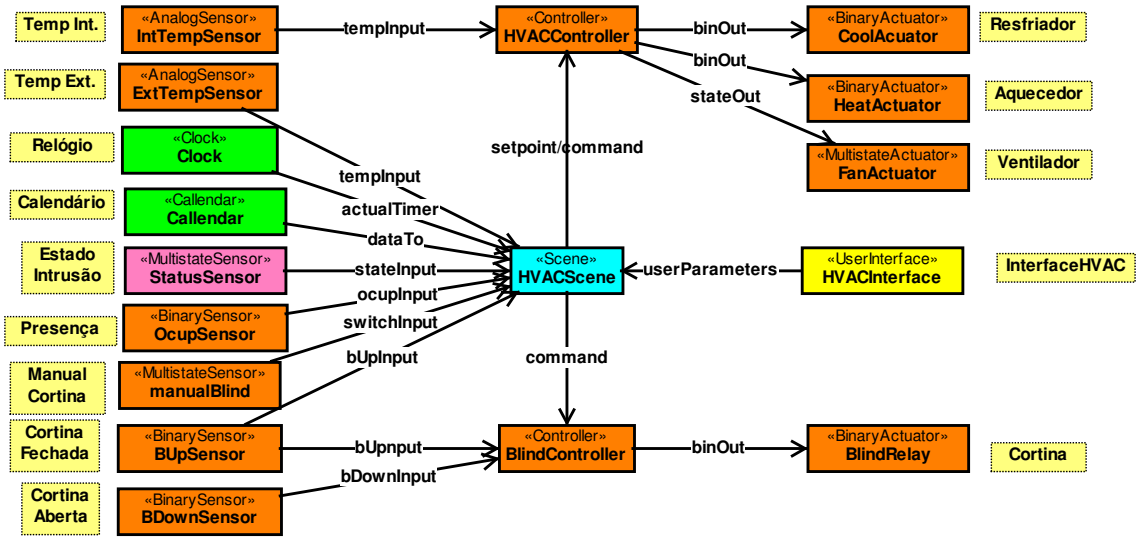


Figura 5.12 : Modelagem lógica

Para facilitar a compreensão do diagrama, ao lado de cada dispositivo lógico foram adicionados comentários relativos à funcionalidade que está sendo modelada. O anexo C contém o modelo completo do estudo de caso.

7. Especificar os cenários (com foco nos dispositivos selecionados)

A partir das políticas, definidas no item 5 desta subseção, serão especificados os cenários para os diferentes subsistemas com foco nas funcionalidades oferecidas pelos dispositivos selecionados no item anterior.

No capítulo anterior foi discutido o problema da implementação de prioridades em gerenciadores de cenários não orientados a prioridade. As pré-condições apresentadas a seguir já implementam a solução discutida. No Anexo C pode ser observado o detalhamento das etapas anteriores que originaram esta solução.

A seguir podem ser observadas as pré-condições, os procedimentos e os cenários para os cenários de iluminação e HVAC, modelados pelos dispositivos lógicos *LightScene* e de *HVACScene*.

Todas as políticas especificadas no item 5 devem ser atendidas, no que se refere às pré-condições e procedimentos.

- **LightScene**

Pré-condições

Nome	Eventos a serem avaliados
PreA1	StatusSensor = Ocupado & OcupSensor = on & IntLightSensor ≤ x
PreA2	StatusSensor = Ocupado **&** OcupSensor = on **&** manualL1 = off **&** IntLightSensor > x **&** IntLightSensor ≤ y
PreA3	StatusSensor = Ocupado **&** OcupSensor = on **&** manualL1 = off **&** manualL2 = off **&** IntLightSensor > y
PreA4	StatusSensor = Desocupado **OU** OcupSensor = off
PreA5	StatusSensor = Ocupado **&** OcupSensor = on **&** manualL1 = on
PreA6	StatusSensor = Ocupado **&** OcupSensor = on **&** manualL2 = on
PreA7	ExtLightSensor < Z **&** (actualTime > 8h **&** actualTime =< 24h)
PreA8	Status <> Alarme & manualL3 = off & ExtLightSensor < Z **&** (actualTime > 0h **&** actualTime =< 8h)

PreA9	StatusSensor <> Alarme **&** manualL3 = off **&** ExtLightSensor > Z
PreA10	StatusSensor <> Alarme **&** manualL3 = on
PreA11	StatusSensor = Alarme **&** IntLightSensor < y

Procedimentos

Nome	Dispositivos lógicos a serem acionados
ProcA1 (Ligar L1 manual)	LigthRelay1 = on [duração 15min]
ProcA2 (Ligar L2 manual)	LigthRelay2 = on [duração 15min]
ProcA3 (Ligar L1 e L2)	LigthRelay1 = on; LigthRelay 2 = on
ProcA4 (Desligar L1 e Ligar L2)	LigthRelay1 = off; LigthRelay 2 = on
ProcA5 (Desligar L1 e L2)	LigthRelay1 = off; LigthRelay 2 = off
ProcA6 (Ligar L3 100%)	LightDimmer = on
ProcA7 (Ligar L3 Manual)	LightDimmer = on [duração 15min]
ProcA8 (Ligar L3 com 50%)	LightDimmer = onValue (50)
ProcA9 (Desligar L3)	LightDimmer = off
ProcA10 (L1, L2 e L3 ligados)	LigthRelay1 = on; LigthRelay 2 = on; LightDimmer = on

Cenários

Nome	Prior	Descrição do Cenário
CenA1 (Luminosidade menor do que X)	10	**SE** PreA1 **ENTÃO** ProcA3
CenA2 (Luminosidade entre X e Y)	10	**SE** PreA2 **ENTÃO** ProcA4
CenA3 (Luminosidade maior do que Y)	10	**SE** PreA3 **ENTÃO** ProcA5
CenA4 (Iluminação: sala desocupada)	12	**SE** PreA4 **ENTÃO** ProcA5
CenA5 (Acionamento manual de L1)	11	**SE** PreA5 **ENTÃO** ProcA1
CenA6 (Acionamento manual de L2)	11	**SE** PreA6 **ENTÃO** ProcA2
CenA7 (L3: noite antes 24h)	10	**SE** PreA7 **ENTÃO** ProcA6
CenA8 (L3: noite após 24h)	10	**SE** PreA8 **ENTÃO** ProcA8
CenA9 (L3: durante o dia)	10	**SE** PreA9 **ENTÃO** ProcA9
CenA10 (Acionamento manual de L3)	11	**SE** PreA10 **ENTÃO** ProcA7
CenA11 (Alarme acionado durante a noite)	40	**SE** PreA11 **ENTÃO** ProcA10

- **HVACScene**

Pré-condições

Nome	Eventos a serem avaliados
PreB1	((actualTime < 18h **&** actualTime > 07h) **OU** OcupSensor = on) & ExtTempSensor > HVACController.setpoint
PreB2	actualTime > 18h **&** actualTime < 07h **&** OcupSensor = off
PreB3	actualTime > 07h **&** atualTime < 09h **&** StatusOfDay = dia útil
PreB4	actualTime > 12h **&** atualTime < 14h **&** StatusOfDay = dia útil
PreB5	(((actualTime < 07h **&** atualTime > 09h) **OU** (actualTime < 12h **&** atualTime > 14h) **OU** StatusOfDay <> dia útil) **&** StatusSensor = Desocupado
PreB6	StatusSensor = Ocupado
PreB7	manualBlind = off **&** ExtTempSensor < 26^0C **&** StatusSensor = Ocupado **&** ExtLgihtSensor > Z
PreB8	manualBlind = off **&** ExtTempSensor >= 26^0C
PreB9	StatusSensor = Desocupado
PreB10	StatusSensor = Alarme
PreB11	StatusSensor = Ocupado **&** manualBlind = Open
PreB12	StatusSensor = Ocupado **&** manualBlind = Close

Procedimentos

Nome	Dispositivos lógicos a serem acionados

ProcB1 (Selecionar modo=refrigeração)	HVACController.setmode = cool
ProcB2 (Selecionar modo=aquecimento)	HVACController.setmode = hot
ProcB3 (Selecionar modo=ventilação)	HVACController.setmode = fan
ProcB4 (Executar controle)	HVACController.runControl
ProcB5 (Desativar controle)	HVACController.deactivateControll
ProcB6 (Abrir cortina)	BlindController.up
ProcB7 (Fechar cortina)	BlindController.down
ProcB8 (Abrir cortina manual)	BlindController.up; manualBlind = open[duração 15min]
ProcB9 (Fechar cortina manual)	BlindController.down; manualBlind = close[duração 15min]

Cenários

Nome	Prior	Descrição do Cenário
CenB1 (HVAC: setar modo = Refrigeração ou Aquecimento)	10	**SE** PreB1 **ENTÃO** ProcB1 **SENAO** ProcB2
CenB2 (HVAC: noite, sala desocupada)	11	**SE** PreB2 **ENTÃO** ProcB3
CenB3 (HVAC: dia útil entre 7 e 9h)	11	**SE** PreB3 **ENTÃO** ProcB4
CenB4 (HVAC: intrusão ativado entre 12 e 14h)	11	**SE** PreB4 **ENTÃO** ProcB4
Nome	**Prior**	**Descrição do Cenário**
CenB5 (HVAC: intrusão = ativado)	10	**SE** PreB5 **ENTÃO** ProcB5
CenB6 (HVAC: intrusão = desativado)	10	**SE** PreB6 **ENTÃO** ProcB4
CenB7 (Cortina durante o dia)	10	**SE** PreB7 **ENTÃO** ProcB6
CenB8 (Cortina temperatura externa maior 26 graus)	11	**SE** PreB8 **ENTÃO** ProcB7
CenB9 (Cortina: Intrusão = ativado)	13	**SE** PreB9 **ENTÃO** ProcB7
CenB10 (Cortina: Intrusão = alarme)	40	**SE** PreB10 **ENTÃO** ProcB6
CenB11 (Cortina: abrir manual)	12	**SE** PreB11 **ENTÃO** ProcB8
CenB12 (Cortina: fechar manual)	12	**SE** PreB12 **ENTÃO** ProcB9

A mesma metodologia é utilizada para a construção dos demais cenários. O conjunto completo dos cenários pode ser encontrado no Anexo C.

8. Realizar mapeamento tecnológico

Nesta etapa os dispositivos lógicos selecionados serão mapeados para dispositivos físicos/unidades oferecidas pela arquitetura. A Tabela 5.3 mostra onde estão mapeados os dispositivos lógicos na arquitetura alvo.

Tabela 5.3 : Implementação dos dispositivos lógicos

Dispositivo lógico	HomeSystems
IntLightSensor	(var) IntLightSensor Systembox – scripts
ExtLightSensor	(61) ExtLightSensor
OcupSensor	(62) MovSensor (var) OcupSensor
StatusSensor, StatusActuator	(var)StatusSensor Systembox – scripts
ManualL1	(108) Switch1 (var) ManualL1
ManualL2	(109)Switch2 (var) ManualL2
ManualL3	(110)Switch3 (var) ManualL3
Clock	Relógio Interno (Systembox – scripts)
Callendar	Calendário Interno (Systembox – scripts)
IntTempSensor	(06) termostato
ExtTempSensor	(60) ExtTempSensor
BUpSensor	(63) BUpSensor
BDownSensor	(64) BDownSensor
ManualBlind	(111) Switch4 (var) ManualBlind
SmokeSensor	(134)SmokeSensor

Dispositivo lógico	HomeSystems
IntInterface	Switch5
HVACInterface	(54) CoolSp (55) HeatSp
HeatActuator CoolActuator FanActuator	(53) HeatActuator (52) CoolActuator (56) FanActuator
LightRelay1 LightRelay2 LightDimmer	(70) LightRelay1 (71) LightRelay2 (17) LightDimmer
BlindRelay	(72) BlindRelay
FireAlarmRelay	(75) FireAlarmRelay
IntAlarmRelay	(74) IntAlarmRelay
IntAlarmRelay	(76) IrrigRelay
HVACController	(006) Termostato + Systembox - scripts
BlindController	Systembox - scripts
Temporizadores	Systembox - scripts

LigthScene	Systembox - scripts
HVACScene	Systembox - scripts
FireScene	Systembox - scripts
IntrusionScene	Systembox - scripts

WaterScene	Systembox - scripts
UserInfo	Systembox – scripts (var)userSearch

Nesta tabela pode-se observar que os dispositivos lógicos são mapeados de três formas distintas:

- através de uma unidade de um dispositivo físico (quando aparece um número entre parentes que representa o endereço do canal onde está implementado o dispositivo lógico);
- através uma variável interna (quando aparece a expressão "var" entre parênteses);
- através da programação de *scripts* na unidade central (*systembox*).

A Figura 5.13 mostra a tela onde podem ser observadas as variáveis que estão sendo utilizadas.

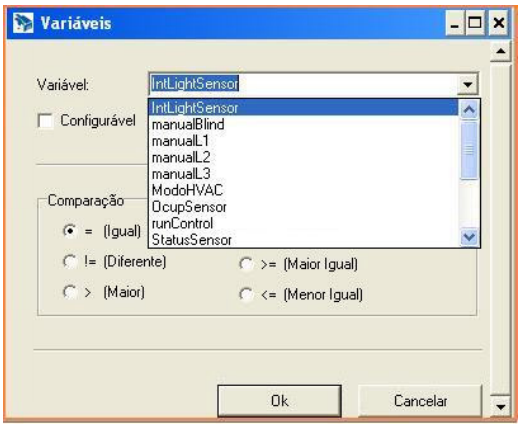


Figura 5.13 : Variáveis

As variáveis permitem representar funcionalidades que podem ser manuseadas de forma independente dos dispositivos físicos. Elas permitem representar o estado de um subsistema tais como *StatusSensor*, *userSearch*, *manualL1*, *manualL2*, *manualL3*, *manualBlind*, *ocupSensor* e também funcionalidades que podem ser obtidas indiretamente, sem a necessidade de um dispositivo físico. Para ilustrar uma aplicação deste segundo caso, o dispositivo lógico ExtTempSensor foi modelado através da variável *IntLightSensor* que é obtida por operações matemáticas a partir do valor de *ExtLightSensor*, *BUpSensor* e *BDownSensor*.

A Figura 5.14 mostra a árvore de dispositivos da configuração que implementa o estudo de caso. Nela podem ser encontrados os dispositivos físicos onde são implementados os dispositivos lógicos citados anteriormente.

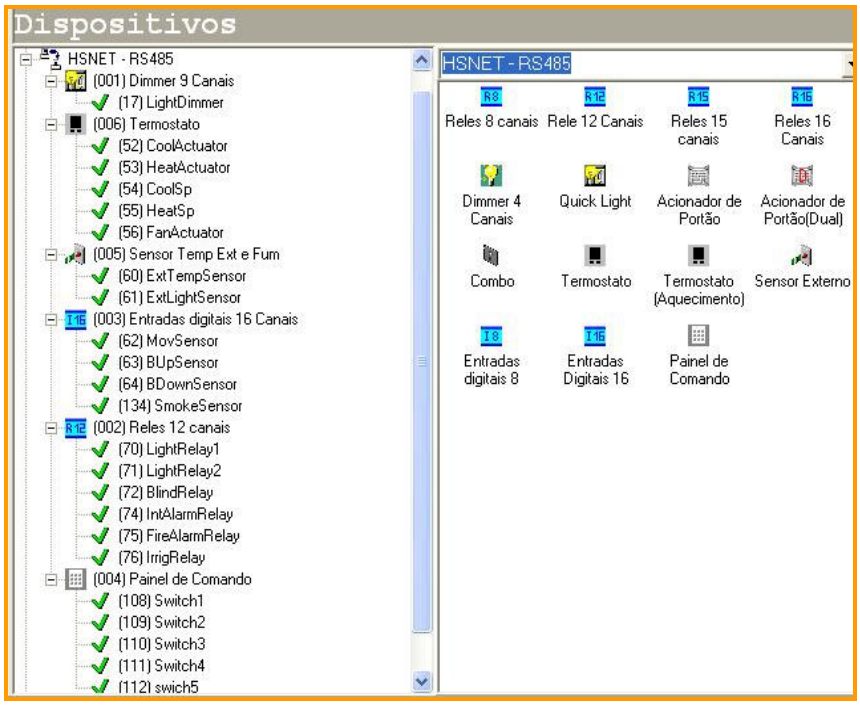


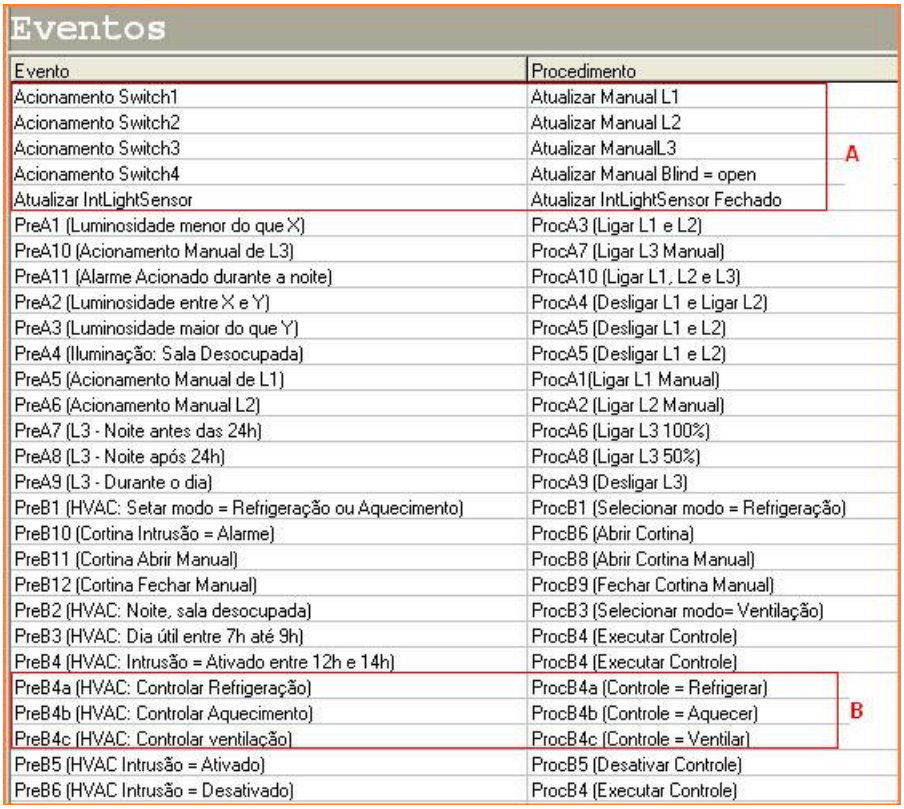
Figura 5.14 : Dispositivos mapeados

Embora o estado de um subsistema possa estar associado ao estado de um determinado dispositivo, é interessante implementá-lo usando variáveis a fim de permitir que o ele possa ser alterado também por outras condições além do dispositivo com a qual está associada.

Um exemplo típico em que isto deve ser levado em conta é a detecção de acionamentos manuais, uma vez que em muitos casos deve-se resolver problemas de conflito e não apenas o estado do dispositivo físico é suficiente para determinar o estado do subsistema em questão, o estado atual do e o estado de outros dispositivos podem ser necessários para definir o estado futuro do sistema.

Além das variáveis definidas na tabela de mapeamento, as variáveis *modoHVAC* e *runControl* são utilizadas para permitir a implementação de parte do controle do Ar Condicionado que não é implementado pelo controlador do termostato e sim por scripts do *systembox*.

A Figura 5.15 mostra uma vista da tela de scripts do *commander*: na coluna da esquerda estão as precondições e na coluna da direita os procedimentos que estão associados a cada precondição. Esta associação corresponde à implementação de um cenário do modelo.



Eventos

Evento	Procedimento	
Acionamento Switch1	Atualizar Manual L1	
Acionamento Switch2	Atualizar Manual L2	
Acionamento Switch3	Atualizar ManualL3	A
Acionamento Switch4	Atualizar Manual Blind = open	
Atualizar IntLightSensor	Atualizar IntLightSensor Fechado	
PreA1 (Luminosidade menor do que X)	ProcA3 (Ligar L1 e L2)	
PreA10 (Acionamento Manual de L3)	ProcA7 (Ligar L3 Manual)	
PreA11 (Alarme Acionado durante a noite)	ProcA10 (Ligar L1, L2 e L3)	
PreA2 (Luminosidade entre X e Y)	ProcA4 (Desligar L1 e Ligar L2)	
PreA3 (Luminosidade maior do que Y)	ProcA5 (Desligar L1 e L2)	
PreA4 (Iluminação: Sala Desocupada)	ProcA5 (Desligar L1 e L2)	
PreA5 (Acionamento Manual de L1)	ProcA1(Ligar L1 Manual)	
PreA6 (Acionamento Manual L2)	ProcA2 (Ligar L2 Manual)	
PreA7 (L3 - Noite antes das 24h)	ProcA6 (Ligar L3 100%)	
PreA8 (L3 - Noite após 24h)	ProcA8 (Ligar L3 50%)	
PreA9 (L3 - Durante o dia)	ProcA9 (Desligar L3)	
PreB1 (HVAC: Setar modo = Refrigeração ou Aquecimento)	ProcB1 (Selecionar modo = Refrigeração)	
PreB10 (Cortina Intrusão = Alarme)	ProcB6 (Abrir Cortina)	
PreB11 (Cortina Abrir Manual)	ProcB8 (Abrir Cortina Manual)	
PreB12 (Cortina Fechar Manual)	ProcB9 (Fechar Cortina Manual)	
PreB2 (HVAC: Noite, sala desocupada)	ProcB3 (Selecionar modo= Ventilação)	
PreB3 (HVAC: Dia útil entre 7h até 9h)	ProcB4 (Executar Controle)	
PreB4 (HVAC: Intrusão = Ativado entre 12h e 14h)	ProcB4 (Executar Controle)	
PreB4a (HVAC: Controlar Refrigeração)	ProcB4a (Controle = Refrigerar)	
PreB4b (HVAC: Controlar Aquecimento)	ProcB4b (Controle = Aquecer)	B
PreB4c (HVAC: Controlar ventilação)	ProcB4c (Controle = Ventilar)	
PreB5 (HVAC Intrusão = Ativado)	ProcB5 (Desativar Controle)	
PreB6 (HVAC Intrusão = Desativado)	ProcB4 (Executar Controle)	

Figura 5.15 : Vista parcial dos eventos programados

Como a ferramenta não permite especificar nomes para os cenários, a fim de permitir a observação da equivalência entre o modelo e a implementação, os nomes das precondições da implementação correspondem a uma combinação do nome adotado para a precondição com o nome adotado para o cenário no modelo. O nome adotado para o procedimento corresponde exatamente ao nome adotado para o procedimento no modelo.

Nesta figura podem ser observados dois destaques. Os cenários destacados em A, são necessários para atualizar o valor das variáveis que representam estados (*manualL1*, *manualL2*, *manualL3*, *manualBlind*) de acordo com eventos gerados por um dispositivo físico disponível em hardware (por exemplo uma tecla de um painel de comando) e para permitir a atualização da variável *IntLightSensor* que é obtida indiretamente pela operação sobre outras funcionalidades disponíveis.

Estes cenários não faziam parte do modelo e foram necessários tendo em vista que a funcionalidade foi mapeada em variáveis, as quais devem ser atualizadas conforme as condições do ambiente.

O segundo destaque, os cenários destacados em B, mostra a implementação manual (através dos *scripts*) de uma parte do controle que não é executado pelo hardware de controle do HVAC, o termostato. Isto é necessário porque a implementação do termostato só permite a fazer o laço de controle local, lendo a temperatura interna e atuando sobre o atuador que esteja ativado no momento (refrigeração, aquecimento ou ventilação) a fim de manter o *setpoint* especificado. Assim como no modelo, a atividade de controle previa a leitura da temperatura externa (*ExtTempSensor*) e a seleção automática do modo de funcionamento, esta etapa do controle teve que ser especificada através de cenários no *systembox*.

A Figura 5.16 visa ilustrar todo o processo de configuração de um cenário do modelo através da programação de um evento na ferramenta. A montagem é

composta de três partes: sobreposta à janela de configuração de eventos (janela 1) está a de configuração de procedimentos (janela 2), ambas disponíveis pela ferramenta de configuração e ao centro tem-se a especificação de um cenário com as pré-condições e os procedimentos especificados no âmbito do *framework* e extraídos do estudo de caso (janela 3).

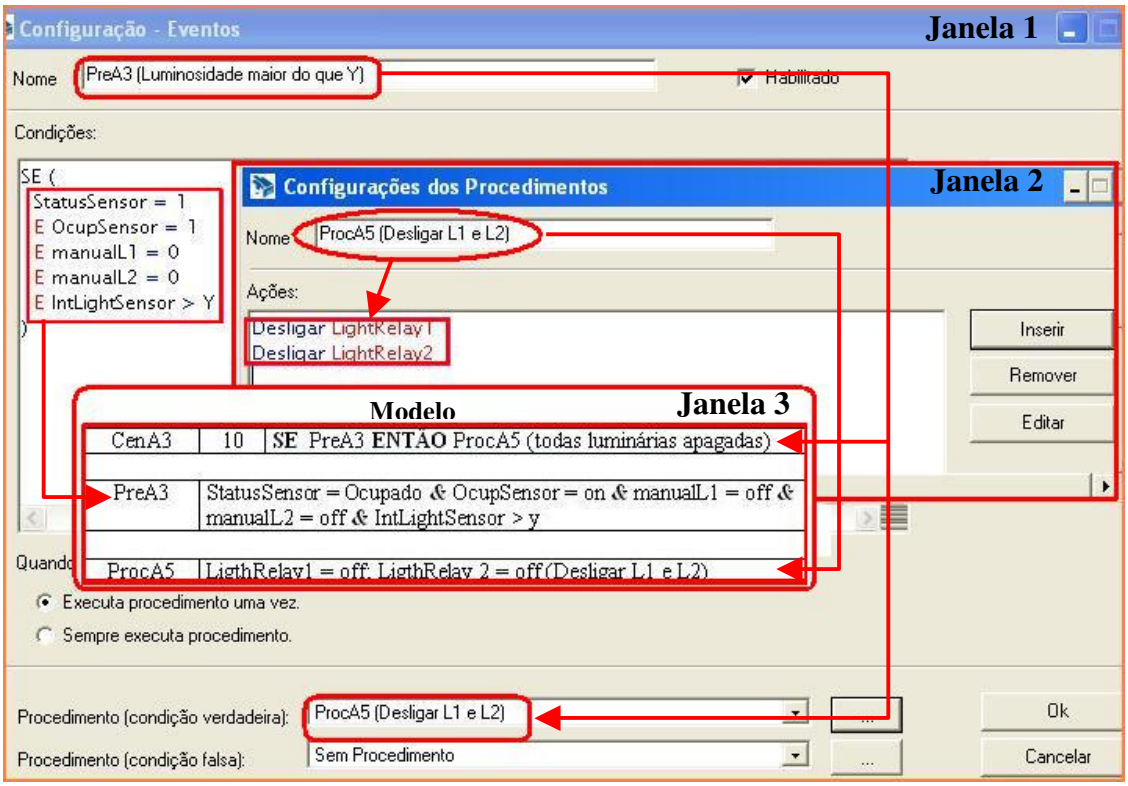


Figura 5.16 : Mapeamento dos Cenários

Os delimitares e as setas indicam as correspondências entre as diversas partes representadas no modelo e onde elas estão sendo implementadas pela ferramenta de configuração. Pode-se observar a estreita relação entre o modelo e a implementação

5.4 Mapeamentos Diferentes para o Subsistema de HVAC

Esta seção tem por objetivo demonstrar diferentes modelagens possíveis a partir da descrição lógica de um subsistema. Os mapeamentos descritos são possíveis implementações para o subsistema de HVAC modelado na Figura 5.12.

Na Figura 5.17 pode-se observar o mapeamento usando-se os dispositivos disponíveis pela arquitetura Homesystems. O termostato implementa o sensor de temperatura interna, o controlador e a interface com o usuário, conforme descrito no item 5.2. Ele possui ligação elétrica com o aparelho de Ar Condicionado, o qual implementa os dispositivos lógicos que representam os atuadores.

O *systembox* implementa o controlador de cenários, o relógio e o calendário. O módulo de entradas implementa os sensores de temperatura externa e de presença. Por simplificação, neste último caso está se omitindo o hardware especifico desses sensores, conectados nas portas do módulo de entrada.

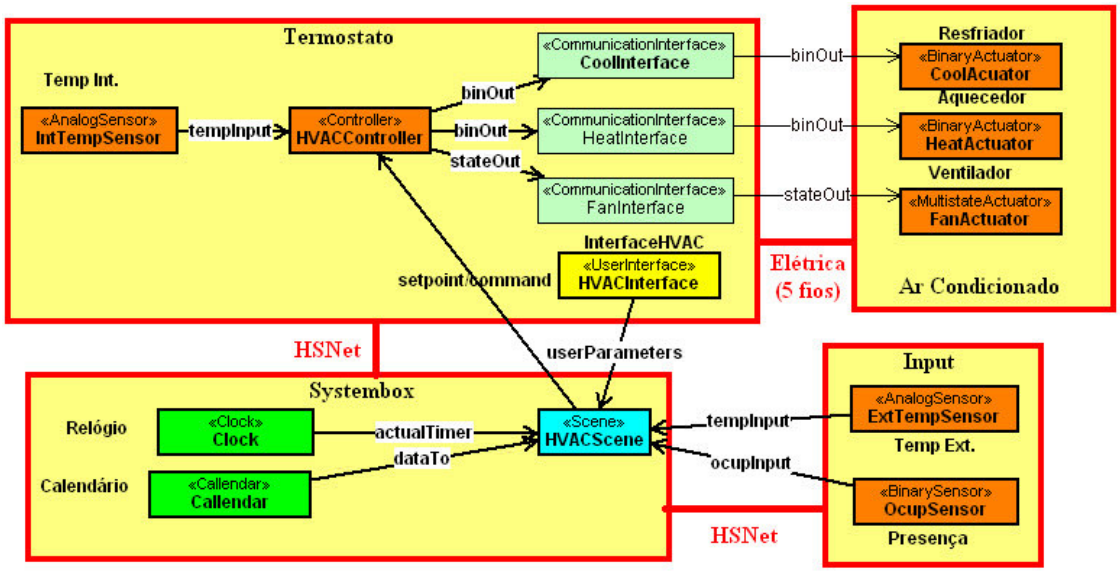


Figura 5.17 : Implementação do subsistema HVAC usando apenas dispositivos disponíveis pela arquitetura Homesystems.

A Figura 5.18 apresenta um mapeamento diferente para o mesmo projeto lógico citado anteriormente. Esta implementação utiliza o Ar Condicionado de janela com controle eletrônico da empresa Springer-Carrier (SPRINGER, 2005), o qual é alvo de projeto de pesquisa no DELET/UFRGS com objetivo de possibilitar a operação em rede destes aparelhos.

Nessa solução o aparelho de ar condicionado implementa além dos atuadores, o controlador e os sensores de temperatura interna e externa. A interface com o usuário é implementada no controle remoto do aparelho.

O controlador de cenários, o relógio e o calendário continuam sendo implementados no *systembox* e sensor de presença também continua sendo implementado pelo módulo de entradas. Um *gateway* implementa as interfaces entre os dois protocolos.

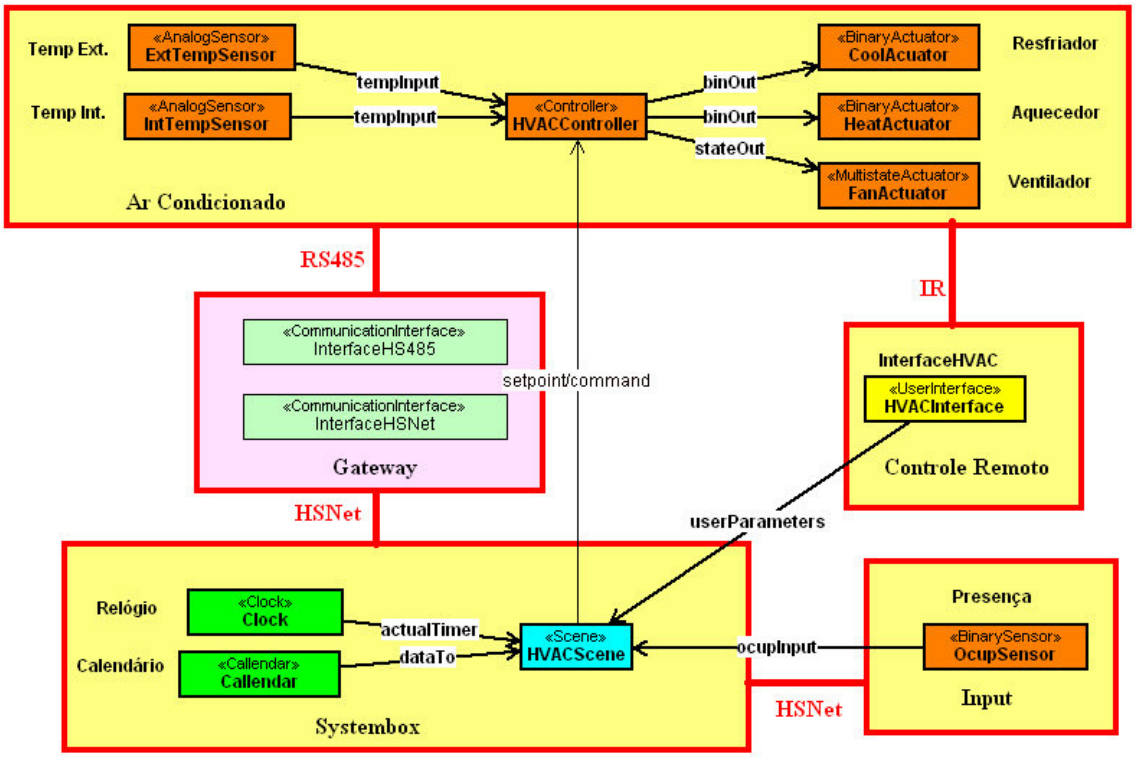


Figura 5.18 : Implementação do subsistema HVAC usando AC Carrier.

5.5 Considerações Finais

O mapeamento apresentado discute diferentes possibilidades de mapeamento do mesmo projeto lógico para diferentes arquiteturas alvo. Alguns recursos disponíveis não foram explorados tais como as configurações específicas de alarme e de telefonia, os quais implicariam um outro mapeamento.

Embora o mapeamento tenha sido realizado manualmente com a utilização da ferramenta de configuração, isto é, não foram editados diretamente os arquivos de configuração, acredita-se que o mapeamento automático seja uma tarefa factível, principalmente se forem utilizados os recursos disponíveis pela tecnologia XML, conforme discutido no capítulo anterior e considerando-se, obviamente, que se conheça a estrutura de armazenamento disponível pela tecnologia alvo.

6 CONCLUSÕES E TRABALHOS FUTUROS

Este trabalho teve por objetivo principal propor um *framework* orientado a objetos para projeto de Sistemas de Automação Predial independente da tecnologia adotada durante a sua implementação.

A fim de que os objetos especificados pelo *framework* sejam representativos desse domínio de aplicação e possam representar satisfatoriamente a informação dos sistemas de automação predial, foram analisados vários padrões abertos e selecionados objetos comuns entre os mesmos. A metodologia de projeto definida adota a divisão do projeto em camadas, seguindo-se as visões propostas pelo RM-ODP (Modelo de Referência para Processamento Distribuído Aberto) da ISO.

A última parte do trabalho teve por objetivo a validação do trabalho. Para isto uma tecnologia de mercado é analisada, os conceitos propostos são mapeados para dispositivos disponíveis nesta arquitetura e um estudo de caso teve por objetivo validar o processo de análise e projeto de um sistema de automação predial suportado pelo *framework.*

Em relação ao *framework* proposto, o trabalho apresenta as seguintes contribuições:

- a identificação de um conjunto de funcionalidades típicas dos sistemas de automação predial, as quais foram agrupadas formando subsistemas;
- a definição de um conjunto de dispositivos lógicos que permitem representar estas funcionalidades independentes do dispositivo físico que será usado para implementação do sistema e a modelagem das relações entre os dispositivos lógicos as quais permitem verificar o tipo de informação que deve ser trocada entre os diferentes dispositivos lógicos que compõem cada um dos subsistemas modelados;
- a utilização do conceito de cenários como elemento fundamental para representar o comportamento dos subsistemas e a especificação dos cenários de funcionamento baseada nas políticas de operação de cada subsistema.

Como o comportamento do sistema está concentrado nos cenários é possível acrescentar e remover comportamentos sem alterar a representação do sistema como um todo. Também deve ser possível alterar a forma de representação desse comportamento sem alterar a representação e a relação entre os dispositivos lógicos modelados, ou seja, alterações no comportamento alterarão apenas esta classe sem alterar a representação do sistema.

A utilização das camadas conceituais (visões) do RM-ODP mostrou-se uma alternativa interessante, por se tratar de um padrão criado pela ISO e fornecer todo o suporte conceitual para fazer-se a transição desde as camadas de alto nível, quando se tem apenas a especificação relativa ao domínio do problema, até a implementação física (de um sistema de automação predial)

Como é um modelo bastante genérico procurou-se identificar os conceitos relevantes para o domínio dos sistemas de automação predial, ao especificar cada camada do *framework*. Neste sentido, o roteiro de projeto apresentado neste trabalho, tem por objetivo oferecer uma transição suave entre as diferentes camadas, com foco nos conceitos relevantes desta área de aplicação. O roteiro busca facilitar o uso do *framework* proposto, uma necessidade que surgiu durante a realização de estudos de caso ao longo do trabalho. Ele mantém as estrutura de camadas especificadas, inserindo atividades dentro de cada camada a fim de facilitar a especificação do sistema.

Relativamente à etapa de validação do modelo, uma questão bastante significativa é que o tipo de tecnologia empregada na arquitetura alvo escolhida não havia sido avaliada na definição dos objetos componentes do modelo, uma vez que foram analisados apenas modelos de objetos de padrões abertos. Mesmo assim todas as funcionalidades propostas no modelo puderam ser mapeadas para dispositivos disponíveis pela arquitetura, o que aponta para a generalidade do modelo proposto.

Durante a elaboração do estudo de caso a maior dificuldade encontrada foi a de configurar o sistema exatamente como havia sido modelado, o que mais uma vez demonstra o quanto é crítica a passagem do projeto para a implementação, principalmente quando não se utilizam ambientes integrados.

Outra questão marcante ao longo da realização do estudo de caso foi a necessidade de reiniciar cada nova implementação praticamente do início, sem a possibilidade de reaproveitamento de estruturas e fazer-se novamente todo o processo de configurações de endereços para cada módulo e para cada unidade, de criação de cada procedimento e de cada script, mesmo em se tratando de estruturas de alto nível bastante similares a um outro caso já implementado.

O mapeamento automático, com a seleção de dispositivos físicos na última etapa do projeto, reduziria os dois problemas citados nos parágrafos anteriores, permitindo que o sistema implementado seja fiel ao que foi modelado na etapa de projeto e reduzindo o retrabalho, pelo aproveitamento de padrões de alto nível.

Como trabalho futuro, destaca-se a implementação de uma ferramenta computacional para suporte ao *framework* proposto. Numa primeira fase, a ferramenta deverá permitir a representação do projeto lógico em arquivo formato XML. Na etapa seguinte deverão ser implementados filtros que permitam converter as definições do modelo lógico em um formato que possa ser importado por cada ferramenta de configuração específica de uma determinada tecnologia, a qual foi definida para implementação de determinado(s) dispositivo(s) lógico(s).

Outra questão interessante é modelar e experimentar outras formas de modelar o comportamento inteligente no sistema seja pela alteração da sintaxe ou pela utilização de outras técnicas de representação de comportamento, o que permitiria explorar o recurso oferecido no *framework* de definir o comportamento de forma independente da representação dos dispositivos que compõem o sistema.

REFERÊNCIAS

AMARAL, J.; PIETROBOM, C. A. M. Using XML to Improve Frameworks Reuse. In: SIMPÓSIO BRASILEIRO DE ENGENHARIA DE SOFTWARE, SBES, 16., 2002, Gramado. **Anais …**Porto Alegre: Instituto de Informática, 2002. p. 254-267.

ARAUJO, J. J. **Análise do Estado da Arte dos Modelos de Objetos utilizados em Sistemas de Automação Predial**. 2003 Trabalho Individual (Mestrado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

ARAUJO, J. J. **Protocolos de Comunicação para Sistemas de Automação Predial**. 2002. 55f. Trabalho Individual (Mestrado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre.

ARAUJO, J. J.; PEREIRA, C. E. Análise de protocolos de Automação Predial/Residencial. In: CONGRESSO BRASILEIRO DE AUTOMÁTICA, CBA, 15., 2004, Gramado. **Anais ...** [Rio de Janeiro]: SBA, 2004. 1 CD-ROM.

ARAUJO, J. J.; PEREIRA, C. E. Object-Oriented Framework for the Design Of Home/Building Automation Systems. In: IFAC WORLD CONGRESS, 16., 2005, Prague, Czech Republic**. Proceedings …**[S.l.:s.n.], 2005.

BASTIDAS, G.; VILLANI, E.; JUNQUEIRA, F. et al. Open Distributed Supervisory System Design using Petri nets. In: IEEE INTERNATIONAL SYMPOSIUM ON INDUSTRIAL ELECTRONICS, 2003, Rio de Janeiro**. Proceedings...** [S.l.:s.n.], 2003.

BASTIDAS, G.; MYAIGI, P. Metologia para Modelagem de Sistemas Abertos e Distrubuídos para Automação Predial. In: CONGRESSO BRASILEIRO DE AUTOMÁTICA, CBA, 15., 2004, Gramado. **Anais ...** [Rio de Janeiro]: SBA, 2004. 1 CD-ROM.

BATIBUS. **Batibus in the first Line**. Disponível em:<http://www.batibus.com/anglais/pre/index.htm>. Acesso em: 12 set. 2004.

BECKER, R. What is an Intelligent Building? In: INTELLIGENT BUILDINGS CONGRESS, 1995, Tel-Aviv. **Proceedings...** [S.l.:s.n.], 1995. p.229-240.

BUSHBY, S. T.; NEWMAN, H. M.; APPLEBAUM, M. A. **GSA Guide to Specifying Interoperable Building Automation and Control Systems Using ANSI/ASHRAE Standard 135-1995, BACnet.** [S.l.]: National Institute of Standards and Technology, 1999. 80p. Disponível em: <fire.nist.gov/bfrlpubs/build99/ PDF/b99051.pdf>. Acesso em: 05 mar. 2003

CHO, S. Y. Framework for the Composition and Interoperation of the Home Appliances Based on Heterogeneous Middleware in Residential Networks. **IEEE Transactions on Consumer Electronics**, [S.l.], v.48, n.3, p. 484 – 489, Aug. 2002.

CHRISTENSEN, J. H. **Basic Concepts of IEC 61499**. Disponível em: <http://www.holobloc.com/papers/1499_conc.zip > Acesso em: 14 jan. 2004.

CIC. **Common Application Language (CAL) Specification**. [S.l.], 1996. 80p. Disponível em: <http://www.cebus.org/Files/eia600.zip>. Acesso em: Acesso em: 14 jan. 2003.

CIC. **HomePnP™ Specification**: version 1.0. [S.l.], 369p. Disponível em: <http://www.cebus.org/Files/hpnp10.zip>. Acesso em: 14 jan. 2003.

DOUGLASS, B. P. **Real-Time Design Patterns**: Robust Scalable Architeture for Real-Time Systems. Boston: Addison-Wesley, 2003. 484p.

ECHELON CORPORATION. **Introduction to the LONWORKS System**: version 1.0. [S.l.], 1999. Disponível em: <http://www.echelon.com/Support/ documentation/Manuals/IntroLonWorks.pdf>. Acesso em: 11 abr. 2002.

ECHELON CORPORATION. **LonMark Application-Layer Interoperability Guidelines**: Version 3.3. [S.l.], 2002. 135p. Disponível em: <http://www.lonmark.org/press/ download/LYR732.pdf>. Acesso em: 05 nov. 2002.

ECHELON CORPORATION. **LONMARK SNVT Master List**: versão 11. [S.l.], 2002. 143p. Disponível em: <http://www.echelon.com/support/documentation/ Manuals/SNVT.pdf>. Acesso em: 17 mar. 2003.

EHS. **European Home Systems Specification**. Disponível em: <http://www.ehsa.com>. Acesso em: 12 set. 2004.

EIBA. **Interworking**. [S.l.], 1999. 42p. Disponível em: <http://www.eiba.ru/dnld.htm>. Acesso em: 03 nov. 2002.

EIBA. **Introduction to the System**. [S.l.], 1999. 36p. Disponível em: <http://www.georg-luber = on linehome.de/data/introduc.pdf>. Acesso em: 03 nov. 2002.

EIBA. **User Layer**. [S.l.], 1999. 20p. Disponível em: <http://www.eiba.ru/dnld.htm>. Acesso em: 03 nov. 2002.

EMBASSI. **The EMBASSI-project**. Disponível em: <http://www.embassi.de/ewas/ project_frame.html >. Acesso em: 15 mar. 2004.

ENDERS, B.; HEVERHAGEN, T. Consistency during the Design of Function Block Adapters using Integration by the Viewpoint Framework.. In: WORLD CONFERENCE ON INTEGRATED DESIGN & PROCESS TECHNOLOGY, 6., 2002. **Proceedings...** [S.l.:s.n.], 2002.

GAMMA, E.; HELM, R.; JOHNSON, R. et al. **Padrões de projeto**: soluções reutilizáveis de software orientado a objetos. Porto Alegre: Bookman, 2000. 364 p.

GOEDICKE, M.; MEYER, T.; TAENTZER, G. ViewPoint-Oriented Software Development by Distributed Graph Transformation: Towards a Basis for Living with Inconsistencies. In: IEEE INTERNATIONAL SYMPOSIUM ON REQUIREMENTS ENGINEERING, 4., 1999. **Proceedings...** [S.l.:s.n.], 2002. p.92–99.

GOMAA, H. **Designing Concurrent, Distributed, and Real-Time Aplication with UML**. Boston: Addison-Wesley, 2000. 785p.

HOMESYSTEMS. **Homepage da Empresa**. Disponível em: <http://www.homesystems.com.br> Acesso em: 15 dez. 2004.

SNOONIAN, D. Smart Buildings. **IEEE Spectrum**, [S.l.], v.4, n.8, p.18-23, Aug. 2003.

ISO/IEC. **ISO/IEC 10746**: Open Distributed Processing - Reference Model. [S.l.], 1995.

KANDÉ, M.; MAZAHER, S.; PRNJAT, O. et al. Applying UML to Design an Inter-domain Service Management Application. In: WORKSHOP ON THE UNIFIED MODELING LANGUAGE, 1., 1998. [S.l.]: **Proceedings...** [S.l.:S.n.], 1998. p. 201-214.

KNX. **KNX System-Architecture**. [S.l.]: Konnex Association. 2003. 32p Disponível em: <www.konnex.org>. Acesso em: 20 jul. 2004.

KU, T.-Y.; PARK, D.-H.; MOON, K.-D. A Java-Based Home Network Middleware Architecture Supporting IEEE 1394 and TCI/IP. **IEEE Transactions on Consumer Electronics**, [S.l.], v.48, n.3, p. 496-504, Aug. 2002.

MICROSOFT CORPORATION. **Understanding Universal Plug and Play**: White Paper. Redmond, 2000. 45p. Disponível em: <http://www.upnp.org/download/ UPNP_UnderstandingUPNP.doc>. Acesso em: 12 fev. 2003.

MICROSOFT CORPORATION. **Universal Plug and Play Device Architecture**: versão 1.0. Redmond, 2000. Disponível em: <http://www.upnp.org/download/ UPnPDA10_20000613.htm>. Acesso em: 12 fev. 2003.

MIYAGI, P. E. **Controle Programável**. Rio de Janeiro: Ed. E. Blücher, 1996. 194p.

MOON, K.-D.; LEE, Y.-H.; SON, Y.-S. et al. Universal Home Network Middleware Guaranteeing Seamless Interoperability among the Heterogeneous Home Network Middleware. **IEEE Transactions on Consumer Electronics**, [S.l.], v.49, n.3, p. 546-553, Aug. 2003.

OMG. **OMG Unified Modeling Language Specification**: versão 1.5. [S.l.], 2003. 736p. Disponível em: <http://www.omg.org/uml> Acesso em: 14 jan. 2004.

OSGi. **OSGi Service Platform.** Release 2. [S.l.], 2001. 288p. Disponível em: <http://www.osgi.org/resources/docs/spr2book.pdf>. Acesso em: 03 abr.2003.

PETRIU, D.C.; WOODSIDE, C.M. Performance Analyses with UML. In: **UML for Real**: Design of Embedded Real-Time Systems. Boston: Kluwer Academic Publishers, 2003. p.220-239

REIS, L. A.; REIS, L. A. Integração de Sistemas – Uma Tendência Irreversível nos Edifícios Inteligentes. In: CONGRESSO BRASILEIRO DE AUTOMAÇÃO E PRÉDIOS INTELIGENTES, 1., 2002, Curitiba. **Anais ...** [S.l.], 2002. p127-135.

SOMMERVILLE, I.; SAWYER, P. **Viewpoints:** Principles, Problems and a Practical Approach to Requirements Engineering. [S.l.]: Computing Department, Lancaster University, 1997. 34p. Disponível em: <http://www.comp.lancs.ac.uk/computing/ research/cseg /97_rep.html > Acesso em: 03 jun. 2003.

SPRINGER. **Homepage da Empresa**. Disponível em: <http://www.springer.com.br> Acesso em: 10 fev. 2005.

SWAN, B. **The Language of BACnet®-Objects, Properties and Services.** [S.l.]: Alerton Technologies. 13p. Disponível em: <http://www.gopolar.com/BACnet/ learning/langbac.html>. Acesso em: 05 mar. 2003.

TANENBAUM, A. S. **Redes de Computadores.** 4.ed. Rio de Janeiro: Campus, 1997. 923p.

TANENBAUM, A. S. **Sistemas Operacionais Modernos.** 2.ed. São Paulo: Pearson Education do Brasil, 2003. 635p.

TRIDIUM. **Baja:** A Java ™ - based Architecture Standard for the Building Automation Industry. Richmond, VA, 2000. 10p. Disponível em: <http://www.tridium.com/products/Baja_White_Paper.pdf >. Acesso em: 08 jun. 2002.

TRIDIUM. **Niagara Framework™.** Richmond, VA: Tridium Inc. Disponível em: <http://www.tridium.com/products/niagara.asp>. Acesso em: 08 jun. 2004.

TRIDIUM. **Press Kit.** Richmond, VA: Tridium Inc. Disponível em: <http://www.tridium.com/company/press_kit.pdf>. Acesso em: 08 jun. 2002.

UPnP. **HouseStatus:**1 Service Template. [S.l.], 2003. Disponível em: < http://www.upnp.org/download/> Acesso em: 21 jun. 2004.

W3C. **Extensible Markup Language (XML) 1.0 (Second Edition).** [S.l.], 2000. 59p. Disponível em:<http://www.w3.org/TR/2000/REC-xml-20001006.pdf> Acesso em: 9 jun. 2003.

W3C. **XML Schema Part 0:** Primer. [S.l.], 2001. Disponível em: <http://www.w3.org/TR/xmlschema-0/> Acesso em: 14 jan. 2003.

W3C. **XML Schema Part 2:** Datatypes. [S.l.], 2001. 137p. Disponível em: <http://www.w3.org/TR/xmlschema-2/> Acesso em: 14 jan. 2003.

WILS, A.; MATTTHIJS, F.; HOLVOET, T. et al. Device Discovery via Residential Gateways. **IEEE Transactions on Consumer Electronics**, [S.l.], v.48, n.3, p. 478-483, Aug. 2002.

X10. **X10 Powerline Carrier (PLC) Technology.** Disponível em: <http://www.x10.com/ support/technology1.htm>. Acesso em: 04 abr. 2002.

ANEXO A DETALHAMENTO DO MODELO

A.1 - Diagramas UML

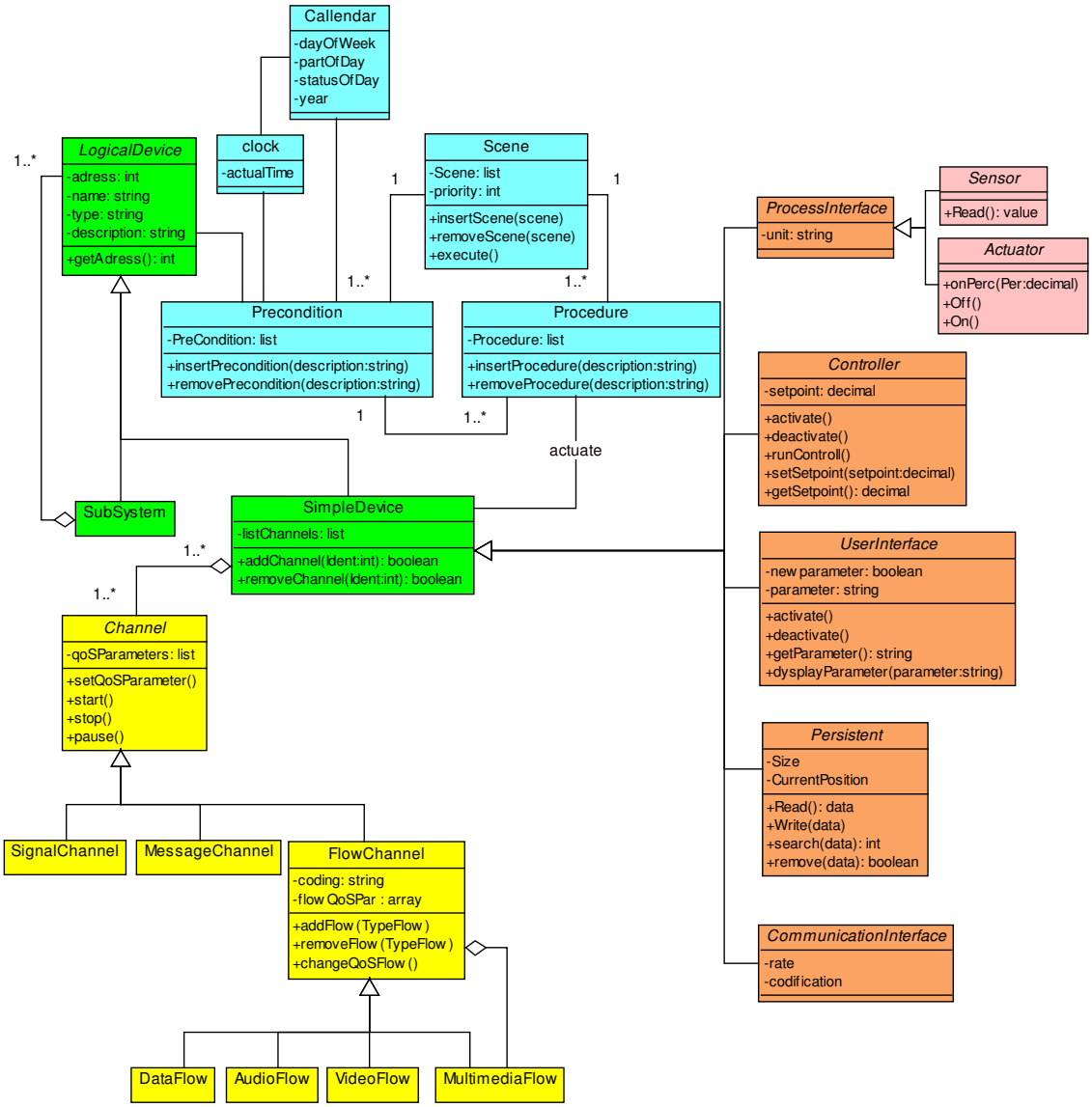


Figura A.1: Visão geral do modelo (visão de informação)

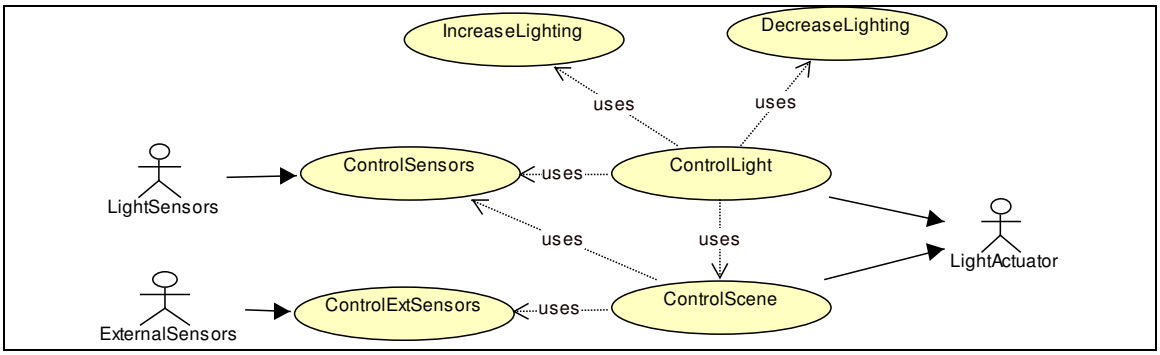


Figura A.4: Funcionalidades do subsistema de controle iluminação

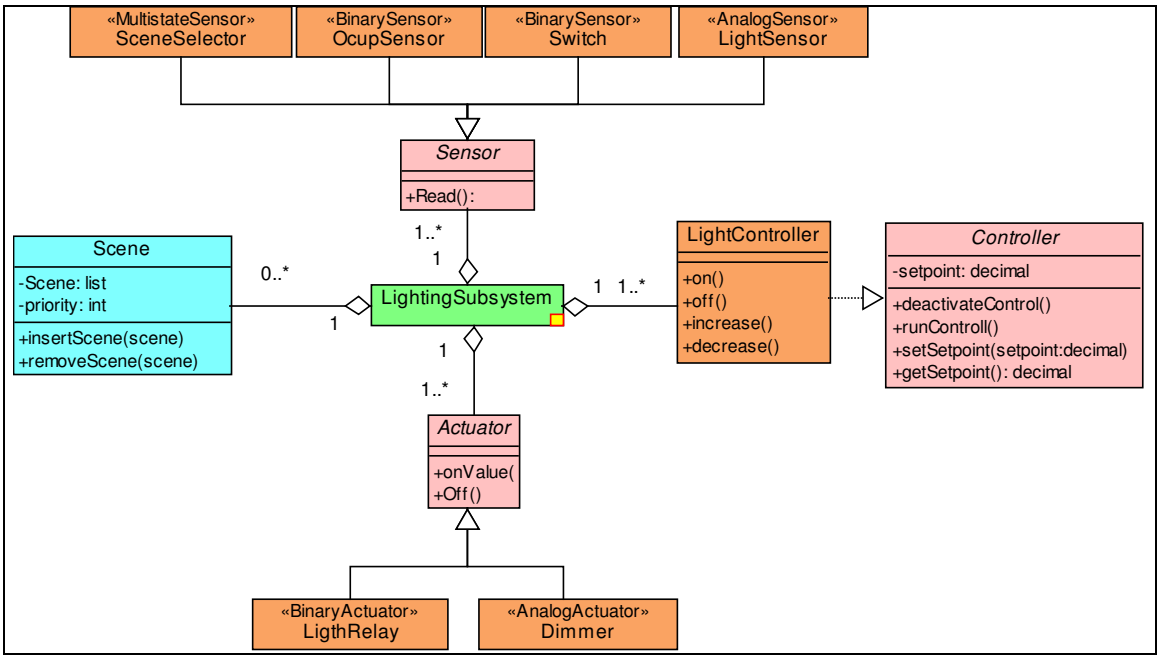


Figura A.3: Diagrama de Classe do subsistema de controle de Iluminação.

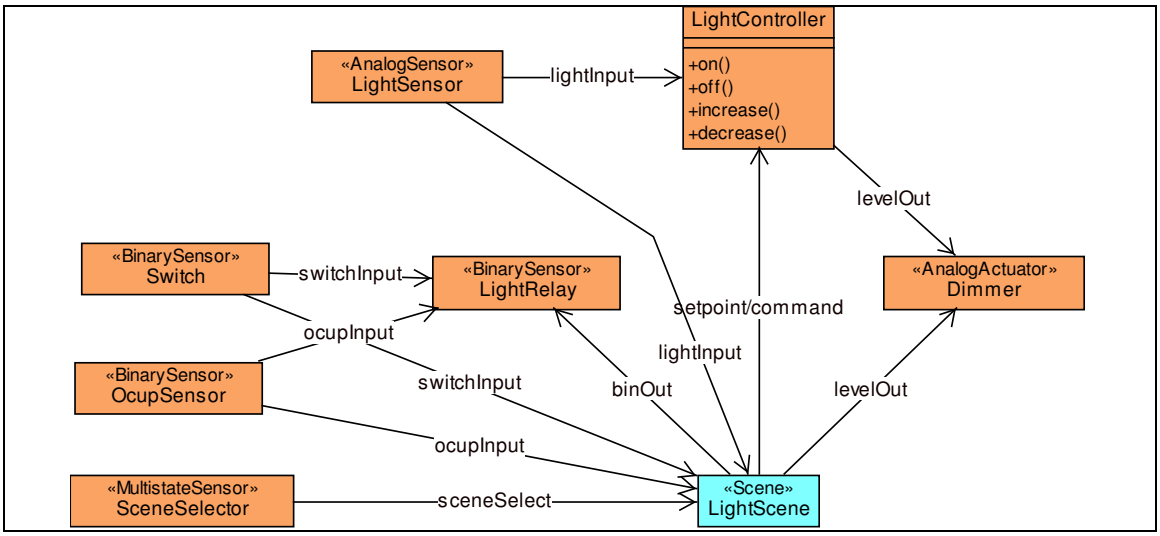


Figura A.4: Diagrama de contexto do subsistema de controle de iluminação

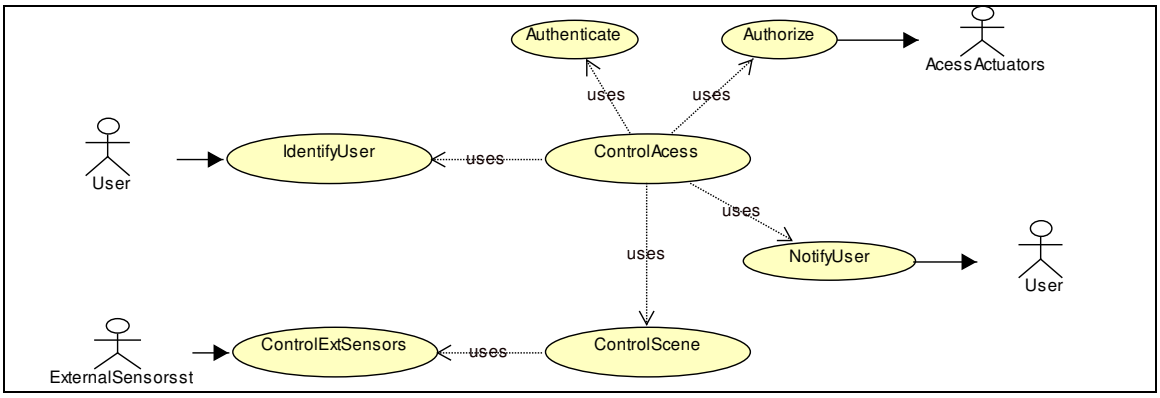


Figura A.5: Funcionalidades do subsistema de controle de acessos

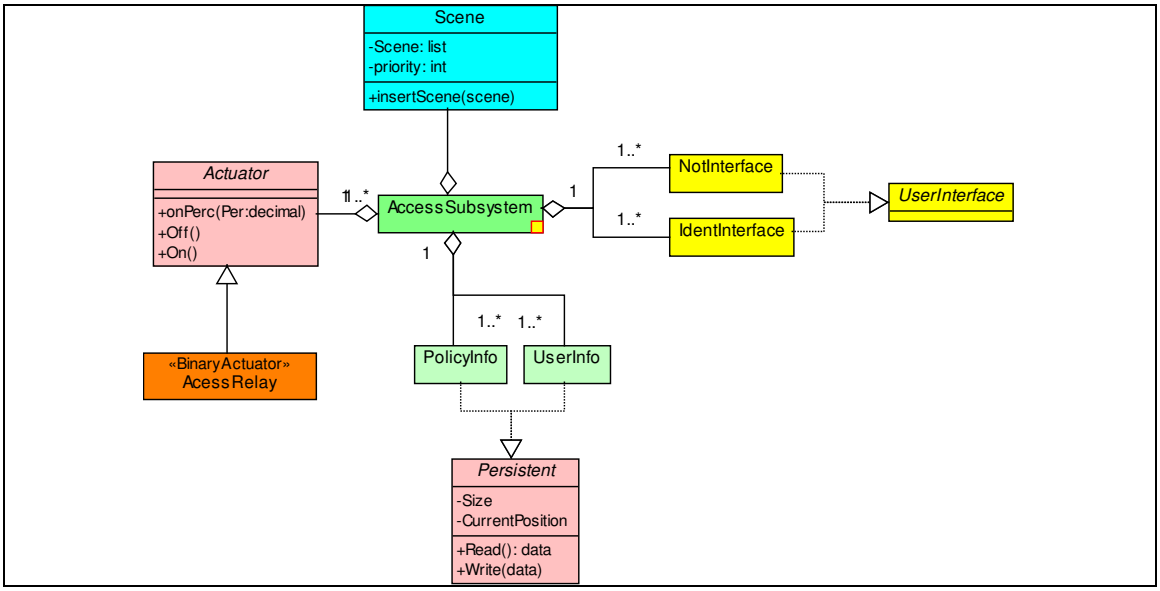


Figura A.6: Diagrama de Classe do subsistema de controle de acessos.

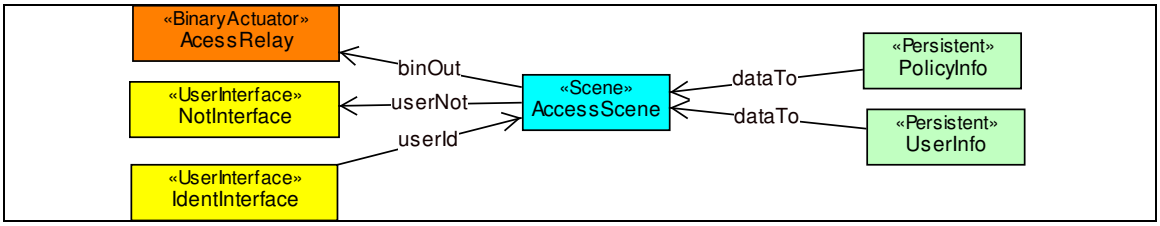


Figura A.7: Diagrama de contexto do subsistema controle de acessos

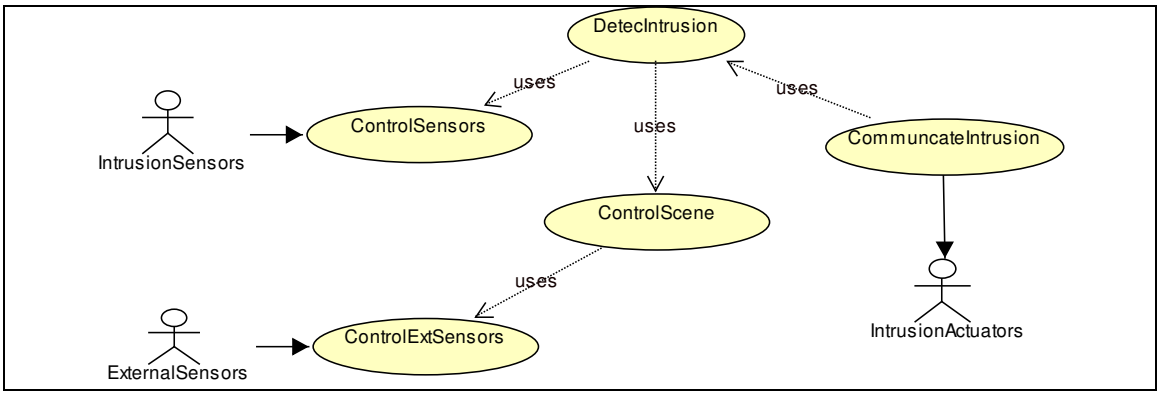


Figura A.8: Funcionalidades do subsistema de intrusões

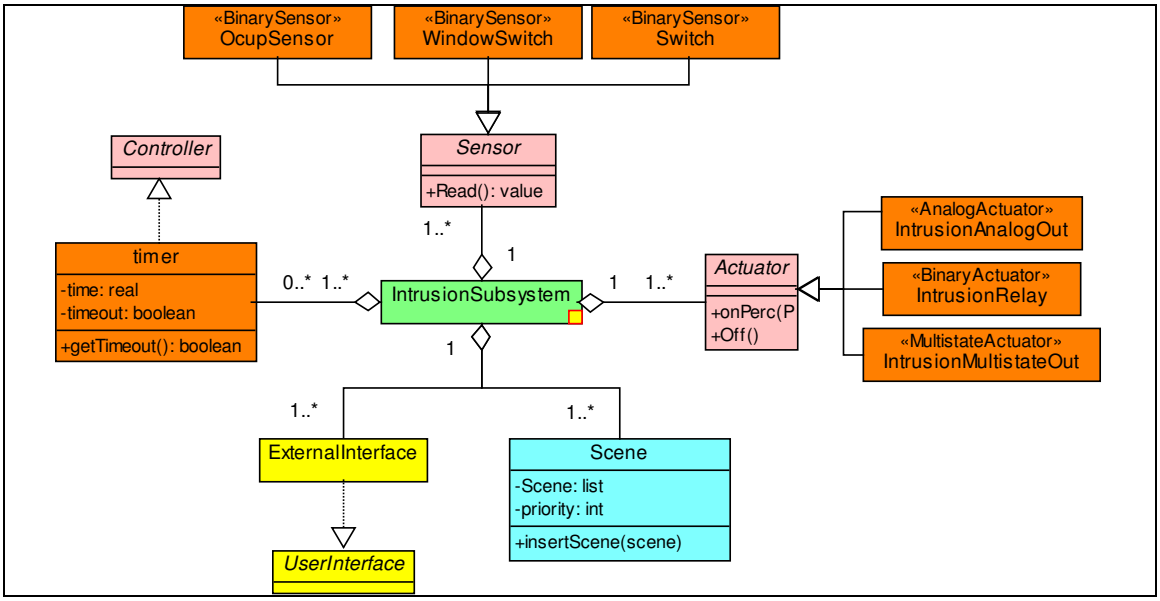


Figura A.9: Diagrama de Classe do subsistema de controle de intrusões.

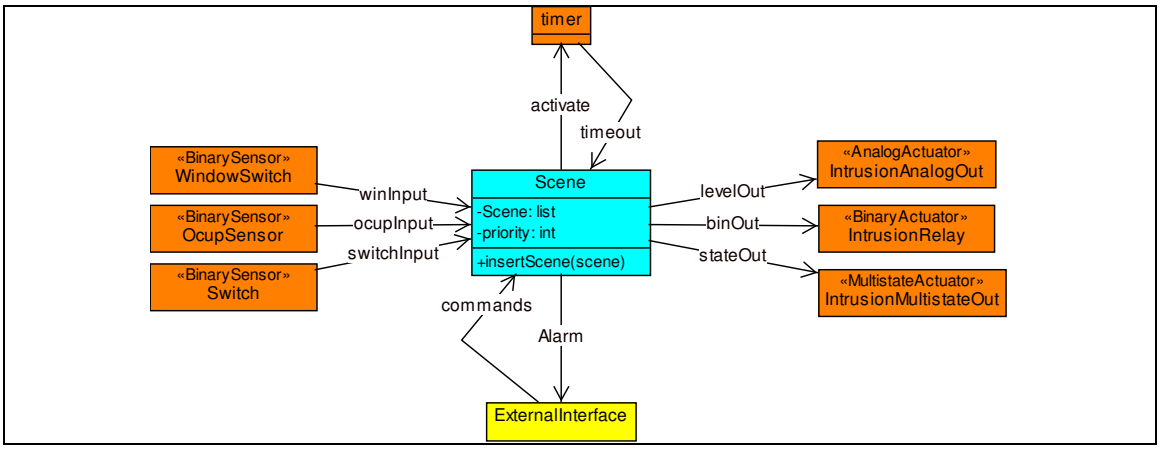


Figura A.10: Diagrama de contexto do subsistema de controle de intrusões

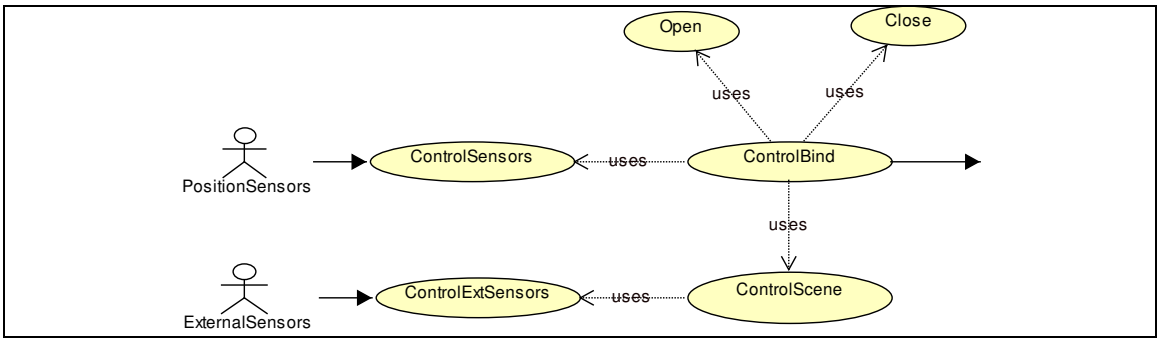


Figura A.11: Funcionalidades do subsistema de controle persianas.

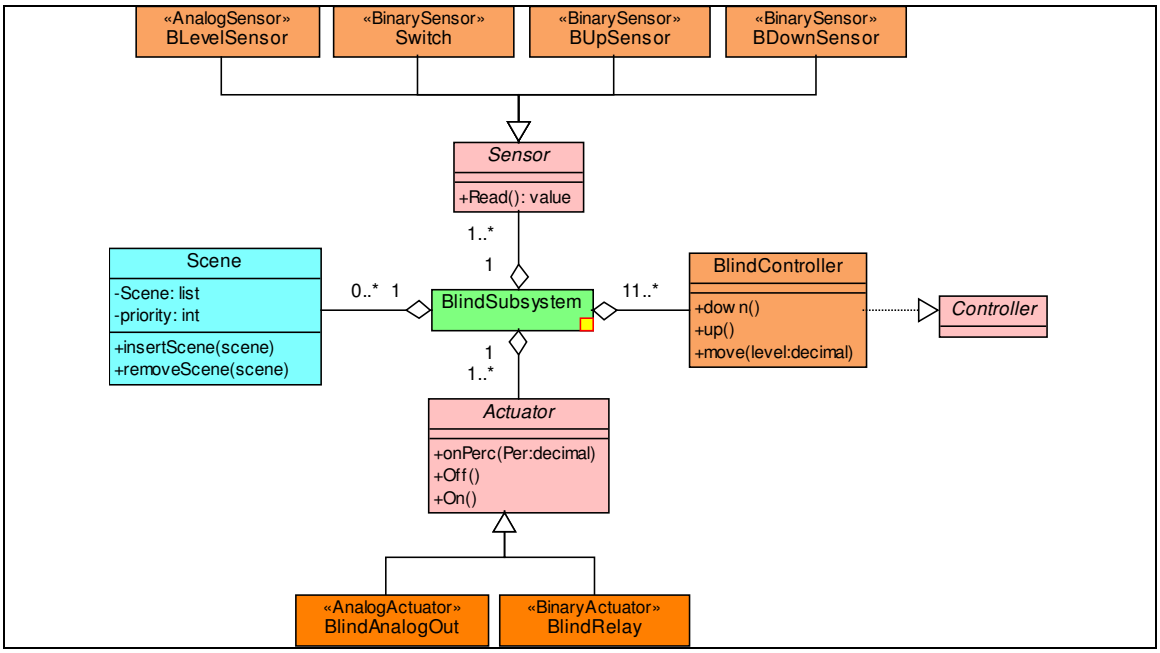


Figura A.12: Diagrama de Classe do subsistema de controle de persianas.

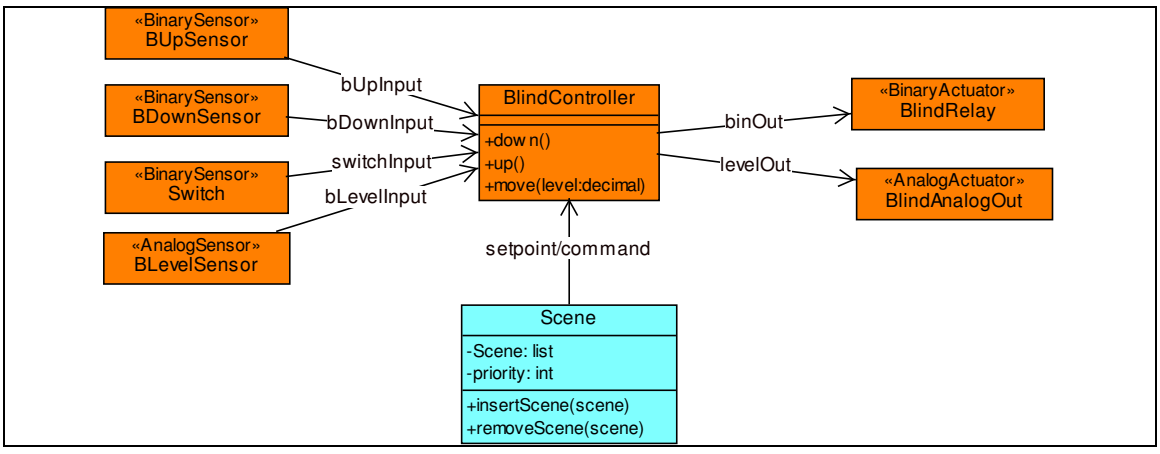


Figura A.13: Diagrama de contexto do subsistema de controle de persianas.

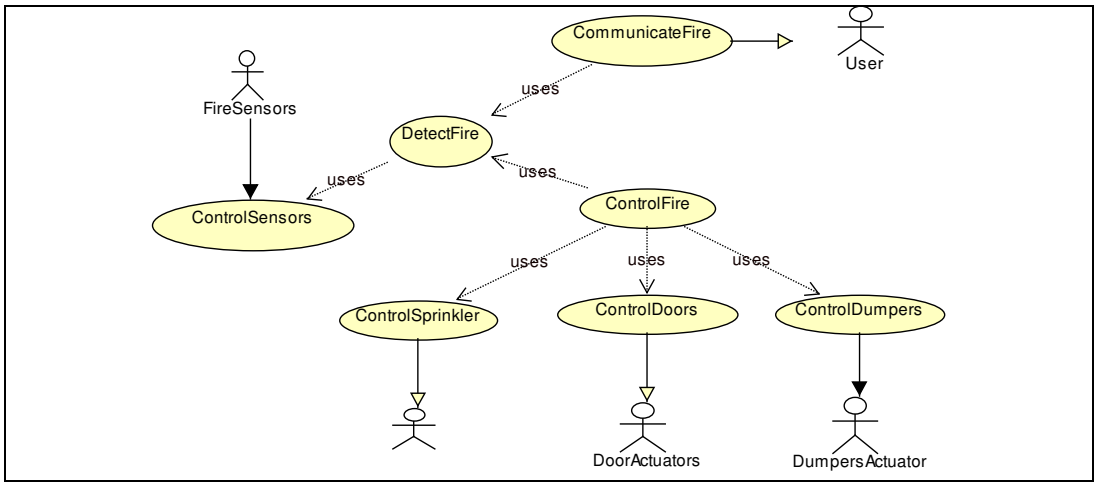


Figura A.14: Funcionalidades do subsistema de controle de incêndios

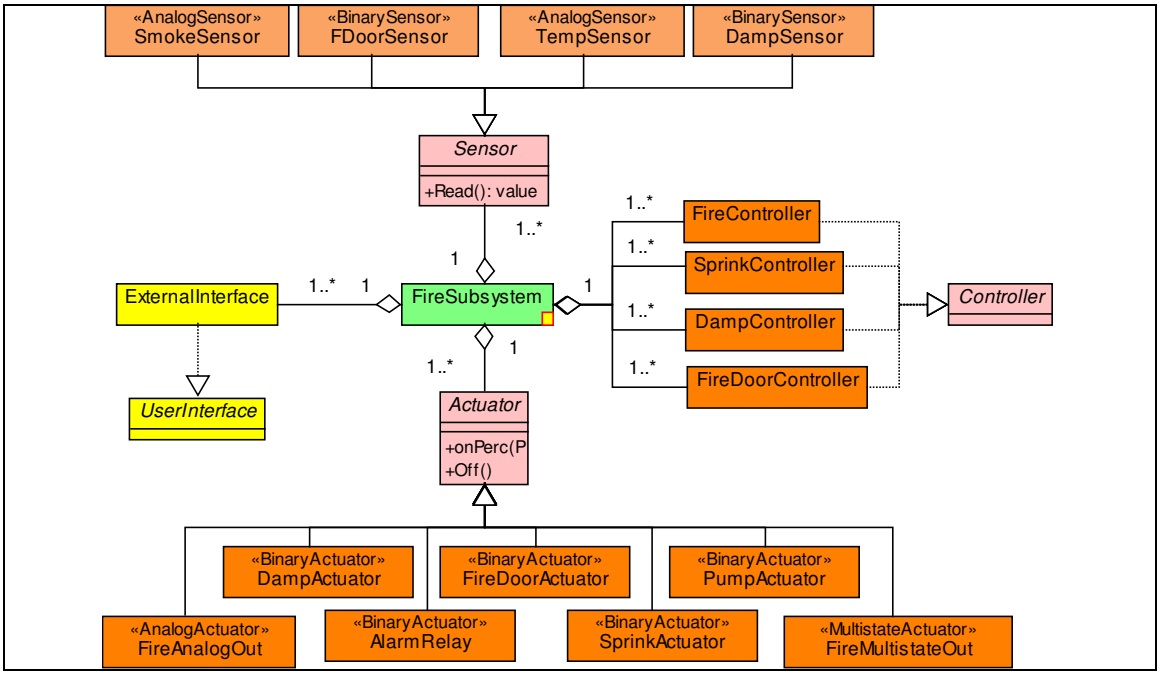


Figura A.15: Diagrama de Classe do subsistema de controle de incêndios.

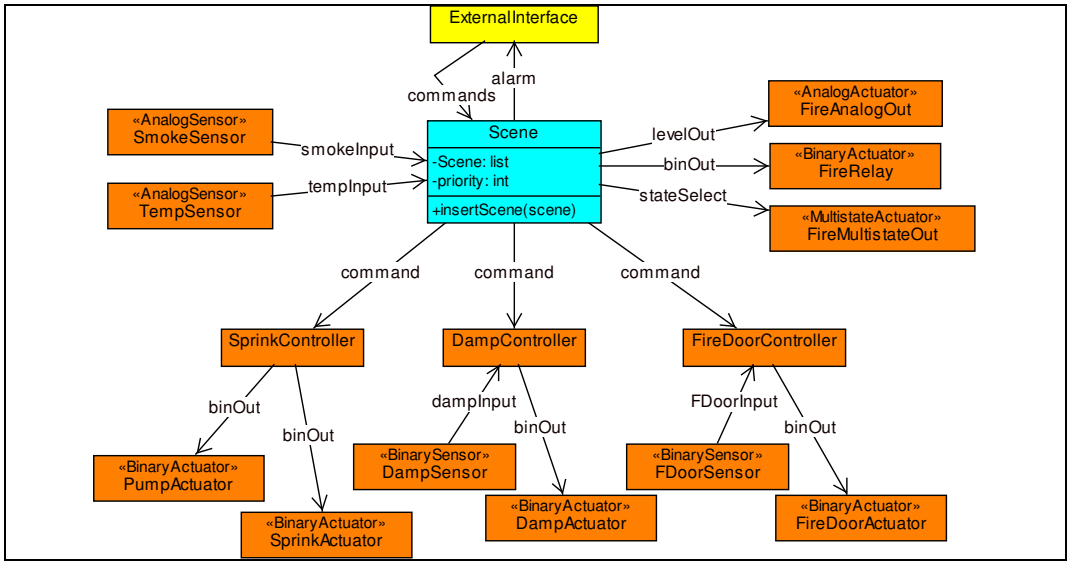


Figura A.16: Diagrama de contexto do subsistema de controle de incêndios

A.2 - Descrição de Atributos e Métodos

Tabela A.1: Descrição do Dispositivo Físico

PhysicalDevice	
Modela o dispositivo físico	
Atributos	**Finalidade / Tipo**
absLocalization	Endereço absoluto do dispositivo na rede. Tipo: int.
relLocalization	Endereço relativo do dispositivo na rede. Tipo: string
groupAdress	Lista de grupos ao qual o dispositivo pertence. Tipo: list.
vendorName	Identificação do fornecedor (nome, referência, etc.). Tipo: string
deviceName	Identificação do nome do dispositivo. Tipo: string.
model	Identificação da referência do dispositivo. Tipo: string.
description	Descrição do dispositivo. Tipo: string.
systemStatus	Representar o estado do dispositivo. Estados possíveis: ON/OFF – online/offline. Tipo: string.
objectList	Lista dos dispositivos lógicos. Tipo: list.
Métodos	**Descrição**

Tabela A.2: Descrição dos Dispositivos Lógicos

Métodos e atributos herdados	
Métodos e atributos herdados das classes anteriores	
Atributos	**Finalidade / Tipo**
adress	Representa o endereço lógico. Tipo: int
adressGroup	Representa o grupo lógico do dispositivo Tipo: list.
name	Identificação do nome lógico do dispositivo. Tipo: string.
description	Descrição do dispositivo. Tipo: string.
type	Tipo: string.
listChannels	Representa a lista de conexões lógicas.Tipo: list
Métodos	**Descrição**
addChannel (Ident:int): boolean	Método para gerenciamento de conexões lógicas
removeChannel (Ident:int): boolean	Método para gerenciamento de conexões lógicas
AnalogSensor	
Objeto cujas propriedades representam as características visíveis de um dispositivo físico entrada analógico	
Atributos	**Finalidade / Tipo**
unit	Unidade física que está sendo representada. Tipo: string
presentValue	Representa o valor atual. Tipo: decimal
defaulValue	Representa o valor default. Tipo: decimal
resolution	Representa o incremento de variação do valor atual. Tipo: decimal
maxPresValue	Representa o valor máximo permitido para o valor atual. Tipo: decimal
highLimit	Representa um valor próximo de valor máximo.Tipo: decimal
minPresValue	Representa o valor mínimo permitido para o valor atual Tipo: decimal
lowLimit	Representa um valor próximo de valor mínimo.Tipo: decimal
Métodos	**Descrição**
Read(): presentValue	Devolve o valor do atributo presentValue
AnalogActuator	
Objeto cujas propriedades representam as características visíveis de um dispositivo de saída analógico	
atributos	**Finalidade / Tipo**
unit	Unidade física que está sendo representada. Tipo: string
presentValue	Representa o valor atual. Tipo: decimal
defaulValue	Representa o valor default. Tipo: decimal
resolution	Representa o incremento de variação do valor atual. Tipo: decimal
maxPresValue	Representa o valor máximo permitido para o valor atual. Tipo: decimal

highLimit	Representa um valor próximo de valor máximo.Tipo: decimal
minPresValue	Representa o valor mínimo permitido para o valor atual Tipo: decimal
lowLimit	Representa um valor próximo de valor mínimo.Tipo: decimal
Métodos	**Descrição**
onValue(parameter:data)	Altera o valor do atributo *presentValue* para o valor estabelecido entre 0 e 100%
on()	Altera o valor do atributo *presentValue* para 100%
off()	Altera o valor do atributo *presentValue* para 0

BinarySensor

Objeto cujas propriedades representam as características visíveis de um dispositivo de entrada que pode ter apenas dois estados

Atributos	**Finalidade / Tipo**
unit	Unidade física que está sendo representada. Tipo: string
presentValue	Representa o valor atual. Tipo: boolean
defaulValue	Representa o valor default. Tipo: boolean
persistence	Representa o comportamento quando na alteração do valor atual. Estados possíveis: *momentary*/*persistent*. Tipo: boolean
polarity	Indica a polaridade na posição de inicial. Estados possíveis: NA / NF .Tipo: boolean
Métodos	**Descrição**
Read(): presentValue	Devolve o valor do atributo presentValue

BinaryActuator

Objeto cujas propriedades representam as características visíveis de um dispositivo de saída que pode ter apenas dois distintos estados

Atributos	**Finalidade / Tipo**
unit	Unidade física que está sendo representada. Tipo: string
presentValue	Representa o valor atual. Tipo: boolean
defaulValue	Representa o valor default. Tipo: boolean
persistence	Representa o comportamento quando na alteração do valor atual. Estados possíveis: *momentary*/*persistent*. Tipo: boolean
polarity	Indica a polaridade na posição de inicial. Estados possíveis: NA / NF .Tipo: boolean
Métodos	**Descrição**
onValue(parameter:data)	Atribui 0 ou 1 para o atributo *presentValue*
on()	Altera o valor do atributo *presentValue* para 1
off()	Altera o valor do atributo *presentValue* para 0

MultistateSensor

Objeto cujas propriedades representam as características visíveis de um dispositivo de entrada que pode ter múltiplos estados

Atributos	**Finalidade / Tipo**
unit	Unidade física que está sendo representada. Tipo: string
presentValue	Representa o valor atual. Tipo: byte
defaulValue	Representa o valor *default*. Tipo: byte
numberPositions	Número de estados possíveis.Tipo: byte
persistence	Representa o comportamento quando na alteração do valor atual. Estados possíveis: *momentary*/*persistent*. Tipo: boolean
Métodos	**Descrição**
Read(): presentValue	Devolve o valor do atributo presentValue

MultistateActuator

Objeto cujas propriedades representam as características visíveis de um dispositivo de saída que pode ter múltiplos estados

Atributos	**Finalidade / Tipo**
unit	Unidade física que está sendo representada. Tipo: string
presentValue	Representa o valor atual. Tipo: byte

defaulValue	Representa o valor *default*. Tipo: byte
numberPositions	Número de estados possíveis.Tipo: byte
persistence	Representa o comportamento quando na alteração do valor atual. Estados possíveis: *momentary*/*persistent*. Tipo: boolean
Métodos	**Descrição**
onValue(parameter:data)	Altera o valor do atributo *presentValue* para o valor estabelecido entre 1 e *numberPositions*
on()	Incrementa 1 posição em *numberPositions*
off()	Altera o valor do atributo *presentValue* para 0

Controller	
Permite representar dispositivos de controle	
atributos	**Finalidade / Tipo**
setpoint	Representa o valor de referência. Tipo: decimal
Métodos	**Descrição**
activate()	Ativa o controlador
deactivate()	Desativa o controlador
runControll()	Executa a função de controle

UserInterface	
Permite representar interfaces com o usuário	
Atributos	**Finalidade / Tipo**
parameter	Representa o dado que está sendo manipulado pela interface. Tipo: data
newParameter: boolean	Indica a disponibilidade de um novo parâmetro. Tipo: boolean
Métodos	**Descrição**
activate()	Ativa a interface
deactivate()	Desativa a Interface

Persistent	
Permite representar dados persistentes armazenados em algum tipo de mídia	
atributos	**Finalidade / Tipo**
size	Representa o tamanho da informação. Tipo: int
currentPosition	A posição atual no local de representação da informação. Tipo: int
Métodos	**Descrição**
read(): data	Permite a leitura de um conteúdo com um determinado formato
write(data)	Permite a escrita de um conteúdo com um determinado formato
search(data): int	Permite a localização de um conteúdo com um determinado formato
remove(data): boolean	Permite a remoção de um conteúdo com um determinado formato

CommunicationInterface	
Modelando-se a rede como um subsistema, permite representar os dispositivos de rede	
atributos	**Finalidade / Tipo**
rate	Representa o taxa de transmissão do dispositivo. Tipo: string
codification	Representa o padrão de codificação da informação Tipo: string

ANEXO B ARQUITETURA HOMESYSTEMS

B.1 - Descrição dos Módulos

Configuração Geral do dispositivo

Configuração obrigatória em todos os módulos.

- Nome do dispositivo (cadeia de caracteres);
- Endereço (inteiro entre 0 e 128); – *default*: 100;
- Pool Time (0, 10s, 15s, 12s, 30s, 1min, 2min, 3min, 4min, 5min) – *default*: 0;
- Desabilitado (verdadeiro/falso) – *default*: verdadeiro;

Parâmetros para canais de saída

Configurações obrigatórias em todos os canais de saída

- Nome (cadeia de caracteres);
- Memorizável (verdadeiro/falso) – *default*: falso;
- Desabilitado (verdadeiro/falso) – *default*: verdadeiro;
- Log (verdadeiro/falso) – *default*: falso;
- Vincular mensagem de voz (verdadeiro/falso) – *default*: falso;
- Gênero (masculino/feminino) – *default*: masculino;
- Identificador da mensagem (cadeia de caracteres).

Parâmetros para canais de entrada

Configurações obrigatórias em todos os canais de entrada

- Nome (cadeia de caracteres);
- Desabilitado (verdadeiro/falso) – *default*: verdadeiro;
- Log (verdadeiro/falso) – *default*: falso;
- Vincular mensagem de voz (verdadeiro/falso) – *default*: falso;
- Gênero (masculino/feminino) – *default*: masculino;
- Identificador da mensagem (cadeia de caracteres).

Painel de comandos

- 09 teclas e 09 *leds* com os parâmetros especificados em item c;
- Na verificação do estado de uma tecla são possíveis os seguintes valores: ligou, desligou, ligado, desligado e mudou de estado;
- Cada *led* pode receber um dos seguintes comandos: ligar, desligar, piscar lento e piscar rápido;
- Na verificação do estado de um *led* são possíveis os seguintes valores: ligado, desligado, piscando lento e piscando rápido;

Sensor Externo

- 2 entradas (analógicas) com os parâmetros especificados em item c;

- Uma entrada analógica pode assumir um valor entre 0 e 256 (0-100%). Comparações podem ser feitas usando o operador igual, diferente, maior, menor, maior ou igual e menor ou igual.

Relés de 8 e de 15 canais (para cada canal):

- 8 e 15 saídas (digitais) com os parâmetros especificados em item b;
- Cada saída pode receber um dos seguintes comandos: ligar, desligar e toggle.
- Na verificação do estado de uma saída são possíveis os seguintes valores: ligou, desligou, ligado, desligado e mudou de estado;

Relés de 12 e de 16 canais (para cada canal):

- 12 ou 16 saídas (digitais) com os parâmetros especificados em item b;
- 09 teclas e 09 *leds* configurados conforme especificado para o painel de comandos;
- Cada saída pode ser configurada conforme especificado para o relé de 8 canais
- 08 cenários permitem configurar como ativado (0%) e desativado (100%) cada um dos canais para cada um dos cenários.

Dimmer 4 canais

- 4 saídas (analógicas) com os parâmetros especificados em item b;
- Cada saída pode receber ser configuração entre 0 e 100%.

QuickLight

- 09 saídas (analógicas) com os parâmetros especificados em item b;
- 09 teclas e 09 *leds* configurados conforme especificado para o painel de comandos;
- Cada saída pode assumir um valor entre 0 e 100%.
- Cada saída pode ser habilitada para funcionar como digital. Parâmetro: Habilitar relé para canais de 1 até 9 (verdadeiro/falso)
- Fade (s) (inteiro entre 0 e 255). Observação: este parâmetro corresponde ao tempo de rampa;
- Cena inicial: nome da cena;
- 08 cenários permitem configurar valores entre 0% e 100% cada um dos canais cada um dos cenários.

Entradas digitais de 8 canais

- 8 entradas (digitais) com os parâmetros especificados em item c;
- Parâmetro: Habilitar NF (verdadeiro/falso) – *default*: falso;
- Na verificação do estado de uma entrada são possíveis os seguintes valores: ligou, desligou, ligado, desligado e mudou de estado;

Entradas digitais de 16 canais

- Parâmetro: Habilitar NF (verdadeiro/falso) – *default*: falso;
- 16 entradas digitais com os parâmetros especificados em item c;
- Os estados das entradas são configurados conforme especificado para entradas digitais de 8 canais.
- 2 entradas (analógicas) configuradas conforme especificado para o sensor externo;
- 16 teclas e 16 *led* configurados conforme especificado para o painel de comandos.

Termostato

Configuração geral do dispositivo:

- Temperatura de aquecimento: temperatura inferior e temperatura inferior (tipo inteiro)
- Temperatura de refrigeração: temperatura inferior e temperatura inferior (tipo inteiro)

- KeyLock (verdadeiro/falso) – default: falso. Permite o bloqueio do teclado, não permitindo alteração manual no mesmo.
- 8 canais de saída configurados conforme item b e com a seguinte especificação:
  - COOL (saída digital, NA): permite habilitar comando remoto da refrigeração;
  - HEAT (saída digital, NA) permite habilitar comando remoto do aquecimento;
  - SP COOL (saída analógica): determina qual a temperatura que deve ser atingida quando se ligar COOL.
  - SP HEAT (saída analógica): determina qual a temperatura que deve ser atingida quando se ligar COOL.
  - FAN SPEED (saída analógica): Permite habilitar comando remoto da ventilação. Opções: 0 (automático), 1 (velocidade 1) e 2 (velocidade 2).
  - Modo Display (saída analógica): habilita a exibição da temperatura externa
  - Temperatura Externa (saída analógica): ???
  - Temperatura Interna (saída analógica): ???

Um dispositivo denominado Termostato para aquecimento apenas as saídas HEAT e SP HEAT.

Acionador de Portão (Dual)

Configuração geral do dispositivo:
- tempo para fechar G1 – *default*: 0;
- tempo após IVA SENSE – *default*: 0;
- tempo para fechar G2 – *default*: 0;
- modo IVA (verdadeiro/falso) – *default*: falso;
- modo noturno (motor, luz, luz+system, system e desligado) – *default*: motor;

Parâmetros do G1 (idem para G2):
- tempo do portão aberto – *default*: 0;
- tentativas para fechar – *default*: 0;
- tempo de luz ligada – *default*: 0;
- modo IVA (verdadeiro/falso) – *default*: falso;
- modo relé (motor, luz, luz+system, system e desligado) – *default*: motor;
- IVA Sensor (NA, NF, desligado) – *default*: desligado;
- tempo após IVA SENSE – *default*: 0;
- Chave/Sensor (chave/sensor) – *default*: sensor;

16 canais configurados conforme no item 2 ou 3 e com a seguinte especificação:
- Relé Abre/Fecha/Para G1: saída analógica. Opções: abrir, fechar, parar, toggle
- Saída Relé Aux G1: saída digital NA
- Relé Abre/Fecha/Para G2: saída analógica. Opções: abrir, fechar, parar, toggle
- Saída Relé Aux G1: saída digital NA
- Noite: saída digital NA
- Sensor (superior) G1: saída digital NA
- Sensor (inferior) G1: saída digital NA
- Sensor IVA G1: saída digital NA
- Sensor auxiliar G1: saída digital NA
- Sensor (superior) G2: saída digital NA
- Sensor (inferior) G2: saída digital NA
- Sensor IVA G2: saída digital NA
- Sensor auxiliar G2: saída digital NA
- 3 saídas digitais Analógicas, código RF

Combo:

- 4 entradas digitais, configuradas conforme entradas digitais de 8 canais
- 5 saídas digitais, configuradas conforme relé de 8 canais
- 2 Leds
- 1 Leitor de cartão

B.2 - Mapeamento do Modelo para a Arquitetura Homesystems

Tabela A.3: Mapeamento para o Dispositivo Físico

PhysicalDevice	
ATRIBUTOS MODELO	**ATRIBUTO HOMESYSTEMS (arquivo hsconfig.ini)**
absLocalization	Endereço (addr)
relLocalization	
groupAdress	Não implementado no nível físico (tratado pelo systembox)
vendorName	Não implementado – *default* HomeSystems
deviceName	Nome do Dispositivo (name)
model	Conforme o dispositivo (model)
description	Não implementado
systemStatus	Desabilitado/Habilitado (disab)
objectList	Os dispositivos oferecem uma lista fixa de funcionalidades, denominado porta na arquitetura. Posteriormente cada unidade de programação (que corresponde a um dispositivo lógico no modelo) irá referenciar o dispositivo físico onde ela está localizada.
	Tempo de pooling (Pooltime)
Padrão para o parâmetro model	
Entradas digitais onboard = 1	Saídas digitais onboard = 2
Relé 8 canais = 3	Relé 12 canais = 4
Relé 15 canais = 5	Relé 16 canais = 19
Dimmer 4 canais = 6	Dimmer 9 canais = 7
Acionador de Portão dual = 20	Termostato = 10
Termostato aquecimento = 11	Sensor Temperatura e luminosidade = 12
Entradas digitais 8 canais = 13	Entradas digitais 16 canais = 21
Painel de comando = 14	grupo = 22

Tabela A.4: Mapeamento dos Dispositivos Lógicos

Atributos e métodos herdados	
atributos	**Atributo HomeSystems (arquivo hsconfig.ini)**
adress	Endereçamento é automático ao inserir o módulo (dev + port)
adressGroup	O objeto grupo é que possui as referências para o endereço do dispositivo
name	Nome (name)
description	Inserido automaticamente conforme módulo (descr)
type	Permite identificar apenas se é analógico ou digital (kind – 0: analógico e 1: digital)
listChannels	Não implementado
AnalogSensor	
Implementado nos seguinte módulos: sensor de temperatura e luminosidade, Entradas digitais 16 canais (2 canais)	
atributos	**Atributo HomeSystems (arquivo hsconfig.ini)**
unit	Não implementado
presentValue	Valor entre 0 e 255 que pode ser monitorado através de operadores relacionais nos *scripts*, gerando eventos.

defaulValue	Não implementado
resolution	*Default* – inteiro = 1
maxPresValue	Dependente de programação: pode ser monitorado através dos *scripts* comparando-se com variáveis.
highLimit	Dependente de programação: pode ser monitorado através dos *scripts* comparando-se com variáveis.
minPresValue	Dependente de programação: pode ser monitorado através dos *scripts* comparando-se com variáveis.
lowLimit	Dependente de programação: pode ser monitorado através dos *scripts* comparando-se com variáveis.
Métodos	**Implementação HomeSystems**
Read(): presentValue	O valor atual do sensor pode ser monitorado em eventos através de operadores relacionais
AnalogActuator	
Implementado nos seguinte módulos: *dimmer* de 4 canais e *dimmer* de 9 canais	
atributos	**Atributo HomeSystems (arquivo hsconfig.ini)**
unit	Não implementado
presentValue	Valor entre 0 e 255. pode ser monitorado através de operadores relacionais em eventos nos *scripts*.
defaulValue	O valor *default* é zero
resolution	*Default* – inteiro = 1
maxPresValue	Dependente de programação: pode ser monitorado através dos *scripts* comparando-se com variáveis.
highLimit	Dependente de programação: pode ser monitorado através dos *scripts* comparando-se com variáveis.
minPresValue	Dependente de programação: pode ser monitorado através dos *scripts* comparando-se com variáveis.
lowLimit	Dependente de programação: pode ser monitorado através dos *scripts* comparando-se com variáveis.
Métodos	**Implementação HomeSystems**
onValue(parameter:data)	Na execução de um procedimento corresponde a opção ligar com percentual regulável
on()	Na execução de um procedimento corresponde a opção ligar
off()	Na execução de um procedimento corresponde a opção desligar
BinarySensor	
Implementado nos seguinte módulos: entradas digitais onboard, entradas digitais de 8 e 16 canais e painel de comando	
Atributos	**Atributo HomeSystems (arquivo hsconfig.ini)**
unit	Não implementado
presentValue	Os estados ligou, ligado, desligou, desligado e mudou de estado que podem ser monitorado através em eventos nos *scripts*.
defaulValue	O valor *default* é desligado
persistence	O estado da entrada é persistente. Um evento mudou de estado pode ser monitorado em eventos. Para o caso de falta de energia o estado pode ser memorizado (mem)
polarity	Nonc (0: Normalmente Aberto – 1: Normalmente Fechado)
Métodos	**Implementação HomeSystems**
Read(): presentValue	O valor atual do sensor pode ser monitorado em eventos através dos testes ligou, ligado, desligou, desligado e mudou de estado
BinaryActuator	
Implementado nos seguinte módulos: saídas digitais onboard, relés de 8, 12, 15 e 16 canais e *dimmer* de 9 canais (opcionalmente)	
Atributos	**Atributo HomeSystems (arquivo hsconfig.ini)**
unit	Não implementado

presentValue	Os estados ligou, ligado, desligou, desligado e mudou de estado que podem ser monitorado através em eventos nos *scripts*.
defaulValue	O valor *default* é desligado
persistence	Conforme descrito para BinarySensor. A opção memorizável (atributo mem) é utilizada para restabelecer o estado anterior a falta de energia.
polarity	Para os módulos de entrada digital a configuração NA/NF é realizada diretamente no dispositivo.
Métodos	**Implementação HomeSystems**
onValue(parameter:data)	não implementado
on()	Na execução de um procedimento corresponde a opção ligar
off()	Na execução de um procedimento corresponde a opção desligar

MultistateSensor / Multistate Actuator

Não implementado diretamente: implementações dessa funcionalidade usam entradas / saídas analógicas com definição dos valores tais como: 0=ligado, 1=desligado, 2=parado, etc.

Controller

Todos os módulos executam função de controle específico, de acordo com a finalidade do módulo.

Descrição Geral da Função de Controle

Todas as unidades ficam realizando *pooling* nas entradas/saídas e atualizando o mapa de memória com o estado atual de cada posição. Atualizações locais no módulo são executadas conforme programação resultante do projeto do módulo. A cada *pooling* da unidade central o estado do módulo é informado ao *systembox*, o qual pode enviar comandos específicos para determinados módulos. Como se pode observar o controle é feito em duas camadas: uma etapa no próprio módulo quando se tratar de atualizações locais, conforme programação residente no módulo e que independe de comunicação, e outra etapa, gerenciada pelo mestre rede (dependente de comunicação), resultante da execução do programa construído a partir dos scripts. Parâmetros locais são definidos a partir da configuração do módulo e os parâmetros remotos são gerenciados pelo *systembox* e programados a partir dos scripts.
Dois módulos de controle com funções específicas são o Termostato e o Acionador de portão

Descrição geral de métodos para os módulo de controle

Métodos	**Implementação HomeSystems**
activate()	Configurar o módulo como Habilitado (configurar o parâmetro disab = 0)
deactivate()	Configurar o módulo como Desabilitado (configurar o parâmetro disab = 1)
runControll()	Módulo habilitado automaticamente executa seu procedimento de controle e responde a requisições do mestre (*systembox*)
getMode	scritps
setMode	scrips

Termostato

atributos	**Atributo HomeSystems (arquivo hsconfig.ini)**
mode	Não implementado
unit	Default: ^{0}C. O módulo possui configuração direta que impede a determinação de setpoints fora da faixa especificada (corresponderia aos atributos do modelo maxPresValue, highLimit, minPresValue, lowLimit)
setpoint	Ajustado diretamente pelo usuário, via teclado, ou através de scripts, usando as saídas analógicas SP COOL, SP HEAT e Fan Speed.

UserInterface

As interfaces padrão telefone e internet são funcionalidades oferecidas pelo *systembox* com configurações específicas. Se considerar o painel de comando como interface sua configuração já foi abordada em *BinarySensor*. Se considerar o Termostato como interface, sua configuração já foi abordada em *Controller*

Persistent

Alguns módulos possuem configuração específica de parâmetros. O módulo Combo armazena número de cartões, o módulo de comando de portões armazena códigos de RF. Os atributos e funcionalidades irão corresponder as configurações e operação desses módulos. As variáveis utilizadas nos scripts correspondem a um meio de armazenamento em RAM, gerenciadas através do sistema operacional do *Systembox*.

Operações sobre variáveis	
Métodos	**Implementação HomeSystems**
read(): data	O valor atual de uma variável pode ser monitorado em eventos através de operadores relacionais em eventos nos *scripts*.
write(data)	Para uma variável pode-se atribuír um valor, atribuir uma variável, atribuir uma E/S, incrementar e decrementar um valor em procedimentos nos *scripts*.
search(data): int	Não implementado
remove(data): boolean	Uma variável pode ser excluída através da ferramenta de configuração. Qualquer dado de uma variável pode ser reescrito.
CommunicationInterface	
Não implementado	

ANEXO C ESTUDO DE CASO

C.1 - Definição das Funcionalidades Conforme Subsistema

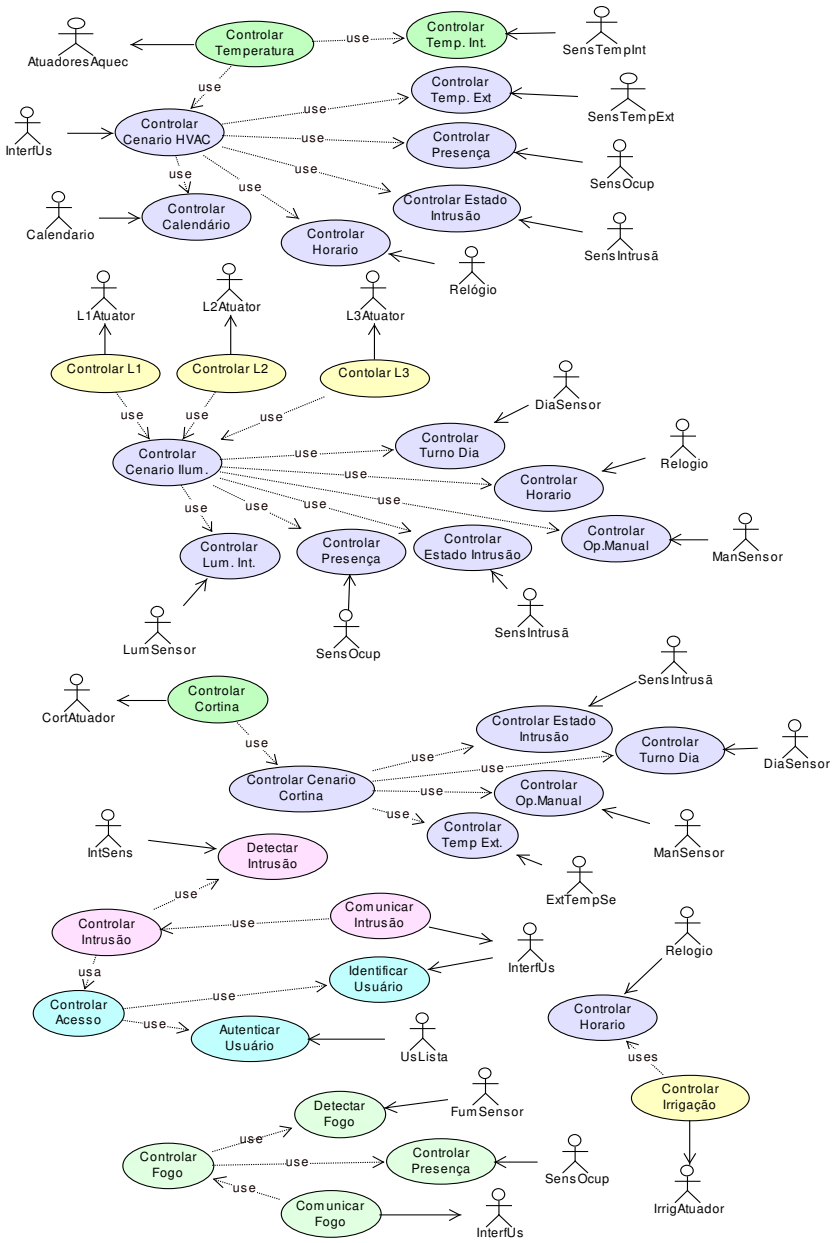


Figura A.17: Estudo de caso: especificação das funcionalidades

C.2 - Definição dos Dispositivos Lógicos

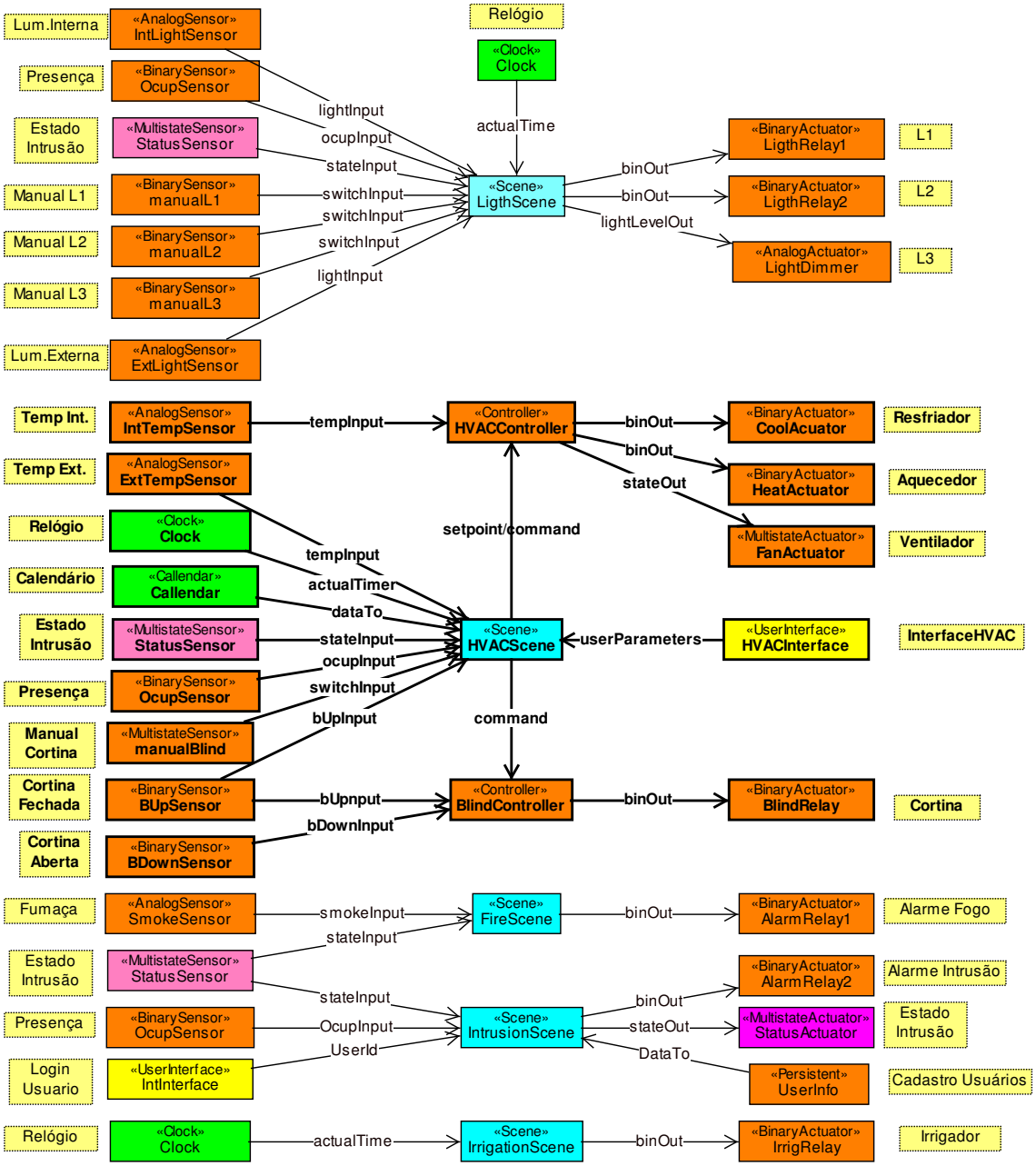


Figura A.18: Estudo de caso: projeto lógico

C.3 - Definição dos Cenários

LightScene

Pré Condições: com definição de prioridades conforme cenários

Nome	Prior	Eventos a serem avaliados
PreA1	10	IntLightSensor ≤ x
PreA2	10	IntLightSensor > x **&** IntLightSensor ≤ y
PreA3	10	IntLightSensor > y
PreA4	12	StatusSensor = Desocupado **OU** OcupSensor = off
PreA5	11	ManualL1 = on
PreA6	11	ManualL2 = on
PreA7	10	ExtLightSensorSensor < Z **&** (actualTime > 17h **&** actualTime < 24h)
PreA8	10	ExtLightSensor < Z **&** (actualTime > 0h **&** actualTime < 8h)
PreA9	10	ExtLightSensor > Z
PreA10	11	ManualL3 = on
PreA11	40	StatusSensor = Alarme **&** IntLightSensor < y

Pré Condições: com definição das pré-condições prioritárias

Nome	Pré-cond. prioritaria	Eventos a serem avaliados
PreA1	~PreA11 **&** ~PreA4	IntLightSensor ≤ x
PreA2	~PreA11 **&** ~PreA4 **&** ~PreA5	IntLightSensor > x **&** IntLightSensor ≤ y
PreA3	~PreA11 **&** ~PreA4 **&** ~PreA5 **&** ~PreA6	IntLightSensor > y
PreA4	~PreA11	StatusSensor = Desocupado **OU** OcupSensor = off
PreA5	~PreA11 **&** ~PreA4	switch1 = on
PreA6	~PreA11 **&** ~PreA4	switch2 = on
PreA7	*	ExtLightSensorSensor < Z **&** (actualTime > 17h **&** actualTime < 24h)
PreA8	~PreA11 **&** ~PreA10	ExtLightSensor < Z **&** (actualTime > 0h **&** actualTime < 8h)
PreA9	~PreA11 **&** ~PreA10	ExtLightSensor > Z
PreA10	~PreA11	Switch3 = on
PreA11	*	StatusSensor = Alarme **&** IntLightSensor < y

* Não ocorre colisão

HVACScene

Pré Condições: com definição de prioridades conforme cenários

Nome	Prior	Eventos a serem avaliados
PreB1	10	ExtTempSensor > HVACController.setpoint
PreB2	11	actualTime > 18h **&** actualTime < 07h **&** OcupSensor = off **&** StatusSensor = Ocupado
PreB3	11	actualTime > 07h **&** atualTime < 09h **&** StatusOfDay = dia útil
PreB4	11	actualTime > 12h **&** atualTime < 14h **&** StatusOfDay = dia útil
PreB5	10	StatusSensor = Desocupado
PreB6	10	StatusSensor = Ocupado
PreB7	10	ExtLgihtSensor > Z
PreB8	11	ExtTempSensor > 26^0C
PreB9	13	StatusSensor = Desocupado
PreB10	40	StatusSensor = Alarme
PreB11	12	manualBlind = open
PreB12	12	manualBlind = close

Pré Condições: com definição das pré-condições prioritárias

Nome	Prior	Eventos a serem avaliados
PreB1	~PreB2	ExtTempSensor > HVACController.setpoint
PreB2	*	actualTime > 18h **&** actualTime < 07h **&** OcupSensor = off **&** StatusSensor = Ocupado
PreB3	*	actualTime > 07h **&** atualTime < 09h **&** StatusOfDay = dia útil
PreB4	*	actualTime > 12h **&** atualTime < 14h **&** StatusOfDay = dia útil
PreB5	~PreB3 & ~PreB4	StatusSensor = Desocupado
PreB6	*	StatusSensor = Ocupado
PreB7	~PreB8& ~PreB9 & ~Pre B10 & ~Pre B12	ExtLgihtSensor > Z
PreB8	~PreB9 & ~Pre B10 ~PreB11	ExtTempSensor > 26^0C
PreB9	*	StatusSensor = Desocupado
PreB10	*	StatusSensor = Alarme
PreB11	~PreB9	manualBlind = open
PreB12	~PreB10	manualBlind = close

* Não ocorre colisão

IntrusionScene

Pré-condições

Nome	Eventos a serem avaliados
PreC1	UserInfo.Search (userId)> 0 & StatusSensor = Ocupado
PreC2	UserInfo.Search > 0 & StatusSensor = Desocupado
PreC3	StatusSensor = Desocupado & OcupSensor = on
PreC4	StatusSensor = Alarme

Procedimentos

Nome	Dispositivos lógicos a serem acionados
ProcC1	Esperar [X]; StatusSensor = Desocupado
ProcC2	StatusSensor = Ocupado
ProcC3	Esperar [Y]; StatusSensor = Alarme; IntrAlarmeRelay = on
ProcC4	IntrAlarmeRelay = on

Cenários

Nome	Prior	Descrição do Cenário
CenC1	40	**Se** PreC1 **ENTÃO** ProcC1 (Desativar: usuário reconhecido)
CenC2	40	**Se** PreC2 **ENTÃO** ProcC2 (Ativar: usuário reconhecido)
CenC3	40	**Se** PreC3 **ENTÃO** ProcC3 (Intrusão detectada)
CenC4	40	**Se** PreC4 **ENTÃO** ProcC4 (Alarme Acionado)

FireScene

Pré-condições

Nome	Eventos a serem avaliados
PreD1	(StatusSensor = Ocupado & SmokeSensor > Y) OU (StatusSensor = Desocupado & SmokeSensor > X)

Procedimentos

Nome	Dispositivos lógicos a serem acionados

ProcD1	FireAlarmeRelay = on

Cenários

Nome	Prior	Descrição do Cenário
CenD1	50	**Se** PreD1 **ENTÃO** ProcD1 (Aciona Alarme de Incêndio)

IrrigationScene

Pré-condições

Nome	Eventos a serem avaliados
PreE1	actualTime = 18h

Procedimentos

Nome	Dispositivos lógicos a serem acionados
ProcE1	IrrigRelay = on [durante 5min]

Cenários

Nome	Prior	Descrição do Cenário
CenE1	10	**Se** PreE1 **ENTÃO** ProcE1 [durante 5mim] (Acionar Irrigação)

ANEXO D