UNIVERSIDADE ESTADUAL DE CAMPINAS

FACULDADE DE ENGENHARIA CIVIL, ARQUITETURA E URBANISMO

PROJETO AUXILIADO PELO PARADIGMA DE ORIENTAÇÃO A OBJETOS: UM EXERCÍCIO

Gelly Mendes Rodrigues

CAMPINAS

2007

UNIVERSIDADE ESTADUAL DE CAMPINAS

FACULDADE DE ENGENHARIA CIVIL, ARQUITETURA E URBANISMO

PROJETO AUXILIADO PELO PARADIGMA DE ORIENTAÇÃO A OBJETOS: UM EXERCÍCIO

Gelly Mendes Rodrigues

Orientadora: Maria Gabriela Caffarena Celani

Dissertação de Mestrado apresentada à Comissão de pós-graduação da Faculdade de Engenharia Civil da Universidade Estadual de Campinas, como parte dos requisitos para obtenção do título de Mestre em Engenharia Civil, na área de concentração de Arquitetura e Construção.

CAMPINAS

2007

FICHA CATALOGRÁFICA ELABORADA PELA
BIBLIOTECA DA ÁREA DE ENGENHARIA E ARQUITETURA - BAE - UNICAMP

Rodrigues, Gelly Mendes

R618m Projeto auxiliado pelo paradigma de orientação a objetos: um exercício / Gelly Mendes Rodrigues.-- Campinas, SP: [s.n.], 2007.

Orientador: Maria Gabriela Caffarena Celani

Dissertação (Mestrado) - Universidade Estadual de Campinas, Faculdade de Engenharia Civil, Arquitetura e Urbanismo.

1. Projeto auxiliado por computador. 2. Projeto arquitetônico. 3. Programação orientada a objetos (Computação). I. Celani, Maria Gabriela Caffarena. II. Universidade Estadual de Campinas. Faculdade de Engenharia Civil, Arquitetura e Urbanismo. III. Título.

Titulo em Inglês: Object-oriented-aided design: an exercise
Palavras-chave em Inglês: CAD, Object-oriented programming, Design method, Typology
Área de concentração: Arquitetura e Construção
Titulação: Mestre em Engenharia Civil
Banca examinadora: Anja Prataschke e Regina Coeli Ruschel
Data da defesa: 30/08/2007
Programa de Pós-Graduação: Engenharia Civil

UNIVERSIDADE ESTADUAL DE CAMPINAS

FACULDADE DE ENGENHARIA CIVIL, ARQUITETURA E

URBANISMO

PROJETO AUXILIADO PELO PARADIGMA DE ORIENTAÇÃO A OBJETOS: UM EXERCÍCIO

Gelly Mendes Rodrigues

Dissertação de Mestrado aprovada pela Banca Examinadora, constituída por:

Profa. Dra. Maria Gabriela Caffarena Celani
Presidente e Orientadora/UNICAMP

Profa. Dra. Anja Pratschke
USP

Profa. Dra. Regina Coeli Ruschel
UNICAMP

CAMPINAS

2007

Dedico este trabalho a **Carlos Costa.**

“Para que possamos compreender o processo de abstratificação no homem moderno, devemos examinar primeiro a função ambígua de abstração em geral. É evidente que as abstrações em si não são um fenômeno moderno. Na realidade, a capacidade crescente para formar abstrações é característica do desenvolvimento cultural da espécie humana. Se digo “mesa”, emprego uma abstração: não me refiro a uma determinada mesa, mas ao gênero “mesa”, que compreende todas as mesas concretas possíveis. Se digo “homem”, não falo desta ou daquela pessoa em toda sua constituição concreta e singular, mas do gênero “homem”, que compreende todas as pessoas individuais. Em outras palavras, faço uma abstração. O desenvolvimento do pensamento filosófico e científico se baseia em uma capacidade cada vez maior para a abstratificação, e renunciar a essa capacidade significa retroceder ao tipo de pensamento primitivo”.

Erich Fromm, 1974.

Agradecimentos

É com imenso carinho que expresso minha gratidão à professora Maria Gabriela Caffarena Celani, pela sua dedicação e ensinamentos prestados ao longo do desenvolvimento deste trabalho, bem como pela sua imprescindível orientação, mostrando sempre como realizar adequadamente uma pesquisa científica.

Agradeço à minha família pelo apoio e compreensão nos longos meses que passei distante de todos.

Agradeço às professoras da FEC-UNICAMP, Prof. Dra Regina Ruschel, Prof. Dra Doris Kowaltowski, Prof. Dra Stelamaris Bertoli e Prof. Dra Lucila Labaki, as quais tive a oportunidade de participar dos seus cursos.

Agradeço aos amigos Autimio Guimarães, Érica Pinheiro, Fidel Navarro, Giovana Godoi e Regiane Pupo pelos momentos de alegria que passamos durante esta jornada. E a todos os novos amigos que fiz na FEC deixo aqui minha mensagem de gratidão.

Expresso também meus agradecimentos ao CNPq pelo apoio financeiro prestado durante o desenvolvimento desta pesquisa.

E expresso a minha gratidão a todas as pessoas que foram solidárias comigo, e que de alguma forma puderam contribuir para a realização deste trabalho. Se por algum motivo seus nomes não estão aqui, saibam que na minha memória seus gestos estarão guardados para sempre. Muito obrigada!

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Quadros

Resumo

RODRIGUES, Gelly Mendes. **Projeto auxiliado pelo paradigma da orientação a objetos:** um exercício. 2007. Dissertação (Mestrado em arquitetura e construção) – Faculdade de Engenharia Civil, Universidade Estadual de Campinas, Campinas, 2007.

Diferentes métodos computacionais têm sido discutidos desde o movimento dos métodos da década de 1960, na tentativa de ajudar os arquitetos a melhor estruturar e gerenciar o processo de projeto. Entretanto, atualmente o uso do computador encontra-se cada vez mais voltado para a representação do projeto arquitetônico, enquanto conceitos e métodos computacionais raramente são expressos no processo de projeto. Na tentativa de explorar o uso da computação como uma técnica que efetivamente auxilie o arquiteto no processo de projeto em arquitetura, a presente pesquisa buscou investigar as possíveis relações entre o paradigma da orientação a objetos e o processo de projeto arquitetônico. Para ilustrar o processo de projeto arquitetônico, baseado no método da orientação a objetos, algumas implementações foram desenvolvidas com o uso da linguagem de programação *Visual Basic for Applications* para AutoCAD. Os conceitos de classes e de objetos – bem como suas propriedades e métodos – foram usados para estabelecer uma analogia com a forma como o arquiteto estrutura o problema de projeto. Espera-se que essa pesquisa possa ajudar estudantes de arquitetura a desenvolverem uma nova compreensão do processo de projeto.

Palavras-chave: Metodologia de projeto, automação do processo de projeto, orientação a objetos.

Abstract

RODRIGUES, Gelly Mendes. **Object-oriented-aided design:** an exercise. 2007. Dissertation (Master in Architecture and Building Construction) – School of civil Engineering, State University of Campinas, Campinas, 2007.

Different computational methods have been used since the design methods movement in the 1960's with the aim of helping architects to better structure and manage the design process. However, the use of computers is currently more commonly used for design representation, while computational concepts and methods are rarelly expressed in the design process. Trying to explore the use of computation as a technique that can effectively assist the architect in the design process, the present investigated the relationship between the object-oriented programming paradigm and the architecturual design process. To illustrate the architectural design process based on the object-oriented method, some implementations were developed with the use of *Visual Basic for Applications* in AutoCAD. Concepts such as object and class, along with their properties and methods, were used to establish an analogy with the way the architect structures the design problem. This research is expected to help architecture students develop a new understanding of the design process.

Keywords: methodology of design, design process automation, objects-oriented.

1 INTRODUÇÃO

A produção criativa em arquitetura, como resultado de um processo mental, recebe grande atenção da psicologia cognitiva, que tenta compreender as tomadas de decisões adotadas para a resolução de problemas de projeto. Acredita-se que tarefas projetuais complexas exigem o emprego de determinados métodos heurísticos de criação e, também, a construção de um pensamento estruturado e ordenado para serem resolvidas adequadamente.

Lawson (1997), em *How Designers Think,* aborda a comunicação eletrônica e a informática como uma nova perspectiva para a estruturação desse pensamento, sugerindo analogias entre a resolução de problemas por humanos e por estruturas computacionais. Na verdade essas teorias computacionais, em sua maioria, foram modeladas a partir da compreensão do funcionamento do cérebro. Como efeito, diversas técnicas de inteligência artificial - como as redes neurais - foram desenvolvidas a partir de estudos sobre o funcionamento dos processos cognitivos do cérebro humano.

Fica evidente que o processo de estruturação do problema de projeto, que é um mecanismo consciente de organização de tarefas, necessita de uma metodologia. Contudo, a representação desse processo, de maneira lógica e ordenada, não é tarefa fácil, já tendo sido tema de diversos estudos ao longo dos anos. O mais influente psicólogo que se empenhou em medir o processo da produção criativa é Guilford (1967), um dos pioneiros na área. No livro *The Nature of Human Intelligence* o autor identifica, na atividade intelectual, dois objetivos principais: a solução de problemas e a produção criativa, afirmando que "sempre há algo criativo a respeito da solução do problema, e que a produção criativa é tipicamente realizada com o intuito de resolver algum problema” (GUILFORD, 1967).

Guilford (1967) enumera alguns passos estabelecidos para a solução do problema e a produção criativa segundo a visão de três autores: Dewey (1910), Wallas (1926) e Rossman (1931). O resumo dessas versões, apresentado pelo autor, mostra a diversidade das maneiras de modelar o processo criativo, conforme disposto na Tabela 1.1.

Tabela 1.1. Passos para a solução de problemas e produção criativa

Dewey	Wallas	Rossman
Dificuldade prevista		Necessidades
Dificuldade encontrada e definida		Formulação do Problema
	Preparação (Reunião de informações)	Avaliação da Informação
	Incubação (Trabalho inconsciente)	Formulação da Solução
Possibilidade de solução	Iluminação (Solução emergente)	Exame crítico da solução
Conseqüências consideradas	Verificação (Solução testada)	Formulação de novas idéias
Solução aceitável		Testes e aceitação.

Fonte - Guilford, J. P .The Nature of Human Intelligence (1967)

Embora as propostas citadas por Guilford (1967) não esclareçam com perfeição a estrutura do pensamento criativo, é possível, a partir delas, determinar as fases principais desse processo, que envolve sempre uma etapa de estruturação ou formalização do problema. Através dessa relação dialética existente entre o pensamento humano e a produção criativa, conceitos e métodos computacionais passaram, então, a ser utilizados no processo de projeto como mecanismos para tentar elucidar o pensamento arquitetônico.

Uma dessas abordagens pode ser encontrada no trabalho *The logic of architecture: Design, computation, and cognition*, publicado por Mitchell na década de 1990. Nessa obra o autor destaca as bases teóricas do projeto computacional no intuito de oferecer suporte ao arquiteto no processo de projeto e, conseqüentemente, esclarecer a estruturação do pensamento arquitetônico. A teoria computacional do projeto definida por Mitchell (1990), no capítulo 6 do livro, sugere que os vocabulários arquitetônicos sejam classificados e interpretados como classes de objetos,

de maneira muito semelhante ao que ocorre no paradigma da orientação a objetos. Essa estratégia é a solução sólida e abrangente encontrada pelo autor para fornecer uma base adequada para o desenvolvimento de projetos computacionais, contrapondo-se ao uso mais comum do CAAD (*Computer Aided Architectural Design*) atualmente, em que métodos e conceitos computacionais são raramente explicitados no processo de projeto e, quando acontecem, não apresentam impactos na maneira como os arquitetos pensam sobre o projeto de arquitetura.

Tendo em vista esses antecedentes, a pergunta central que se pretendeu responder nesta pesquisa foi: **como a computação, e mais especificamente o paradigma da orientação a objetos, pode contribuir para o processo de projeto em arquitetura?** Para responder a esta questão buscou-se estabelecer uma analogia entre essas duas áreas de conhecimento, a arquitetura e a computação, na tentativa de descrever o processo de projeto de maneira lógica e ordenada.

Nesse sentido, a presente pesquisa partiu da suposição de que o paradigma da orientação a objetos poderia ajudar os arquitetos na estruturação dos problemas de projeto. Isto facilitaria o desenvolvimento de implementações computacionais efetivamente úteis para os arquitetos, de maneira a auxiliar o processo de projeto em arquitetura.

1.1 OBJETIVOS

A presente pesquisa teve o intuito principal de buscar na ciência da computação métodos que pudessem contribuir para a maneira como pensamos sobre o projeto de arquitetura. Esta pesquisa não pretendeu propor uma solução rígida e definitiva para o pensar em arquitetura, nem o desenvolvimento de software capaz de produzir projetos adequados a qualquer situação. Tratou-se de um trabalho de caráter exploratório, que objetivou apresentar um método alternativo aos métodos tradicionais de pensar e estruturar os problemas de projeto.

2 REVISÃO BIBLIOGRÁFICA

A metodologia e o processo de projeto têm recebido grande atenção nas últimas décadas. Compreender esse processo não é tarefa simples. No entanto, desde o movimento dos métodos da década de 1960, tem havido várias iniciativas de descrevê-lo de maneira lógica e ordenada, sempre objetivando aumentar a qualidade dos projetos de arquitetura, a eficiência do trabalho colaborativo, além de possibilitar a implementação do processo de projeto em computador.

Atualmente são diversas as estruturas computacionais propostas com o objetivo de ajudar os arquitetos a melhor estruturar e gerenciar o processo de projeto. Entretanto, embora a utilização do computador esteja cada vez mais presente nas etapas de esquematizar, modelar e implementar projetos de arquitetura, conceitos e métodos computacionais são raramente explicitados no processo de projeto (CELANI, 2002). Uma possível razão para isso deve-se à visão simplificada que muitos arquitetos possuem em relação ao CAD (*Computer Aided Design*), o que acaba deixando dúvidas quanto à sua efetiva utilização no auxilio do processo de projeto e resultando na preferência dos arquitetos em usar programas prontos e fáceis de usar, mesmo que isto implique em muitas limitações.

Terzidis (2006), em seu livro intitulado *Algorithmic Architecture*, esclarece que se tornou muito comum confundir um procedimento de "computadorização[1]", atualmente desempenhado por intermédio das ferramentas CAD, como um trabalho resultante da computação propriamente dita. Para o autor, a computação propriamente dita diz respeito à utilização de procedimentos lógico-matemáticos na solução de problemas. Trata-se de um procedimento que envolve o raciocínio, lógica e algoritmos, além de estruturas mentais, cognição, simulação e regras baseadas na inteligência. Em contraste, a "computadorização" é definida como o ato de preencher ou

[1] Tradução livre do termo "computerization" proposto por Terzidis (2006).

armazenar informações no computador ou sistemas de computadores, automação e mecanização, que geralmente envolve a digitalização de entidades ou processos que são predeterminados e bem definidos.

Nesse sentido, fica claro que o uso da computação como ferramenta de projeto baseada no computador ainda é bastante limitado em arquitetura. Os processos desenvolvidos na mente do projetista continuam sendo ações de preenchimento, manipulação ou armazenamento nos sistemas de computadores, prevalecendo um modelo de utilização dos computadores baseado na "computadorização". Esse problema deve-se ao fato de os projetistas não aproveitarem o poder computacional dos computadores, acreditando muitas vezes que qualquer modelo criado em computador é um produto de computação (TERZIDIS, 2006).

Uma técnica alternativa para o uso efetivo dessa idéia dialeticamente oposta é o desenvolvimento de algoritmos em arquitetura. Terzidis (2006) afirma que através do uso de *scripting languages* o projetista pode ir além do uso do mouse, transcendendo um conjunto de limitações dos softwares atuais, além de incorporar a complexidade computacional e o uso criativo dos computadores. Conceitos como repetição linear e não-linear, geração de números aleatórios, busca estocástica, fractais, autômatos celulares e hibridização são apresentados por Terzidis (2006) em *Algorithmic Architecture* como pontos de partida para o desenvolvimento de projetos de arquitetura.

A tentativa de se introduzir a programação de computadores no projeto de arquitetura, no entanto, não é uma novidade. Em *The Art of Computer Graphics Programming* os autores Mitchell, Ligget e Kvan (1987) propuseram-se a ensinar a linguagem de programação Pascal aos arquitetos, aproveitando a oportunidade para introduzir conceitos computacionais no projeto, tais como: simetria, recursão e parametrização. Por outro lado, o livro *Microcomputer Aided Design for Architects and Designers,* publicado por Schmitt (1988 *apud* CELANI, 2007), embora não tenha tido a intenção de ensinar a programação de computadores aos arquitetos, abordou diversos exemplos de programas desenvolvidos em AutoLisp. Nesta obra, o autor propôs-se a explorar conceitos como fractais, recursividade, *shape grammar*, projeto baseado em regras e outros.

Na *University of East London* (UEL), por exemplo, Coates e Thum (1995 *apud* CELANI, 2007) implantaram conceitos de projeto evolucionário através de técnicas como autômatos celulares, números aleatórios, *shape grammar* e recursividade, implementadas em diferentes linguagens de programação tais como: AutoLisp, Mini Pascal, GDL (*Geometric Description Language*) e na

linguagem de *scripts* do ArchiCAD. Outro exemplo do uso de algoritmos no projeto de arquitetura é encontrado no trabalho de Duarte (2001) que, a partir da análise das obras de Álvaro Siza no conjunto residencial da Malagueira, em Lisboa, o autor desenvolveu um sistema capaz de gerar automaticamente projetos semelhantes aos do arquiteto, possibilitando recombinar as regras de composição, gerando novas variações perfeitamente plausíveis dentro da linguagem do arquiteto.

Nesse sentido, torna-se necessário destacar que o uso de métodos computacionais no desenvolvimento do projeto não é apenas importante do ponto de vista da eficácia do trabalho. Além de permitir a automação de tarefas, sua maior contribuição para os arquitetos é, na verdade, possibilitar uma compreensão aprofundada deste processo. Acredita-se que o desenvolvimento de implementações no computador possa gerar um impacto positivo na prática do projeto, permitindo estruturar de maneira lógica as etapas do projeto, além de ajudar o projetista a ter uma nova compreensão sobre seu trabalho em arquitetura.

Por razões como essas que a presente pesquisa buscou aplicar uma teoria da ciência da computação (orientação a objetos) à área de metodologia do projeto de arquitetura. Essa estratégia, denominada por Moles (1998) de “método da mistura de duas teorias”, é um dos métodos heurísticos propostos por esse autor para a investigação científica. Segundo ele, a combinação consciente de dois sistemas de teorias individualmente válidos como forma de “explorar um sistema de pensamento, uma doutrina ou conceitos já estabelecidos” levaria ao estabelecimento de novas relações e, conseqüentemente, propiciaria a criação de novas teorias (MOLES, 1998, p.157).

A idéia de transladar um conceito de um plano a outro para obter novos *insights* tem suas origens nas teorias sobre a criatividade. Segundo Koestler (1964 *apud* KNELLER, 1978), por exemplo, toda produção criativa teria origem na bissociação, o resultado da intersecção de planos de raciocínio independentes. Para Koestler (1964 *apud* KNELLER, 1978) o efeito alcançado através da mistura de duas matrizes de raciocínio distintas é, na verdade, a definição de “uma nova síntese intelectual”.

Assim, diante desse contexto, espera-se que a mistura de duas linhas de pensamento aparentemente desprovidas de relação - a metodologia de projeto arquitetônico e o paradigma da orientação a objetos - leve a uma nova compreensão do processo de projeto arquitetônico. Para alcançar esse propósito foi necessário revisar a literatura em duas etapas primordiais.

O primeiro momento da revisão bibliográfica teve como foco o trabalho *The Logic of Architecture: Design, Computation, and Cognition* (1990) de William J. Mitchell, obra em que o autor desenvolve uma discussão sobre a estruturação lógica do pensamento de arquitetura e propõe uma teoria computacional do projeto no intuito de fornecer uma base adequada para o desenvolvimento de sistemas CAD. No capítulo 6 de Mitchell (1990), intitulado "*Types and Vocabulary*", sugere-se que tipos arquitetônicos sejam abordados como classes de objetos em clara alusão ao paradigma da orientação a objetos. O autor compara essa abordagem àquela proposta por Colquhoun (1967) no artigo *Typology and design method*, sugerindo o estabelecimento de uma analogia entre tipologias arquitetônicas e classes de objetos. Essa conexão de idéias firmada por Mitchell (1990), conduziu a presente pesquisa à necessidade de compreender melhor, por um lado, os tipos arquitetônicos - abordando autores como Argan (1962 *in* NESBITT, 2006) e o próprio Colquhoun (1967) – e, por outro, o vocabulário adotado pelo autor para a descrição das qualidades do edifício e os princípios fundamentais da orientação a objetos.

O segundo momento da revisão bibliográfica partiu para a análise de outros estudos que se propuseram a utilizar conceitos computacionais relacionados ao processo de projeto e ao desenvolvimento de implementações CAD. Para isso realizou-se, inicialmente, uma sistematização das idéias do arquiteto e matemático Christopher Alexander, apresentadas em *Notes on the synthesis of form* (1964), referência na área exatamente por apresentar métodos de projeto semelhantes aos métodos utilizados na ciência da computação, e também um delineamento de suas idéias apresentadas em *A Pattern Language: Towns, Buildings, Construction* (1977), obra em que o autor defende a utilização de padrões comprovadamente eficientes no desenvolvimento de projetos arquitetônicos. Uma vez estabelecida essa estruturação de idéias, passou-se ao desenvolvimento de uma ferramenta computacional, como forma de ilustrar o processo de projeto arquitetônico auxiliado pela programação, com uma abordagem verdadeiramente computacional.

A Tabela 2.1 mostra uma visão cronológica das áreas e teóricos abordados durante a evolução da presente pesquisa.

Tabela 2.1. Áreas e teóricos abordados na pesquisa

Década		**1960**			**1970**	**1990**
Autores		Argan	Colquhoun	Alexander	Alexander	Mitchell
Obra		Sobre a tipologia em arquitetura	*Typology as design method*	*Notes on the synthesis of form*	*A Pattern language*	*The logic of architecture*
ÁREAS	Teoria da arquitetura	x	x			x
	Ciência da computação					x
	Metodologia do projeto			x	x	x
	Fundamentos do CAD					x

2.1 CONCEITOS

Para que houvesse uma melhor compreensão da analogia proposta por Mitchell (1990) em *The Logic of Architecture* foi necessário estudar, de maneira aprofundada, os conceitos apresentados pelos autores acima citados, além dos princípios fundamentais das técnicas de programação orientada a objetos, conforme descritos a seguir.

2.1.1 TIPOS ARQUITETÔNICOS

A tipologia arquitetônica é considerada por alguns autores, a exemplo de Rossi e Krier (NESBITT, 2006), como um instrumento analítico preciso para a arquitetura, já que permite, através de um traçado histórico da arquitetura, organizar uma base racional para a concepção de projeto em arquitetura em função de determinadas modificações culturais e projetuais. Autores como Argan (1962 *in* NESBITT, 2006) e Colquhoun (1967) vêem o tipo arquitetônico mais como um princípio passível de variações do que um conjunto de entidades fixas, sendo estas transformações de soluções do passado um meio de reconhecer o papel das soluções precedentes na concepção do projeto de arquitetura (NESBITT, 2006).

No ensaio "Sobre a tipologia em arquitetura" de Argan (1962 *in* NESBITT, 2006) é possível observar algumas indicações do autor sobre o uso da tipologia no processo de concepção do projeto, assim como compreender a construção conceitual do tipo arquitetônico. Para Argan (1962 *in* NESBITT, 2006) esse é um método que se baseia na dedução de uma série de elementos ilustrativos, ou seja, na união e comparação de todos os elementos do edifício em questão. A posição do autor quanto à criação do tipo é de que:

> No processo de comparação e justaposição de formas individuais para determinar o tipo, são eliminadas as características particulares de cada prédio, permanecendo apenas aquelas que são comuns a todas as unidades da série. Portanto, o tipo se constitui pela redução de um complexo de variantes formais à forma básica comum. (...) A forma básica deve ser entendida como uma estrutura interior de uma forma ou como um princípio que contém a possibilidade de infinitas variações formais e modificações estruturais do tipo em si (ARGAN,1962 *in* NESBITT, 2006, p.270).

Dessa maneira, compreende-se que o conceito de tipo representa uma idéia de um elemento que pode servir de regra para um modelo e não a imagem de uma coisa que deve ser perfeitamente imitada e copiada. Quatrèmere de Quincy, em meados do século XIX (18-- *apud* MARTINEZ, 2000, p. 108), já afirmava que "(...) para tudo é necessário um antecedente; nada sai do nada".

O conceito de tipologia é também discutido por Colquhoun (1967) em seu ensaio intitulado *Typology and design method*. Neste artigo o autor critica as metodologias modernistas de projeto e destaca a ineficiência dos métodos intuitivos em lidar com a complexidade dos problemas que precisavam ser resolvidos. Diante de uma análise desse processo de projeto, Colquhoun (1967)

concluiu que na falta de instrumentos mais refinados de análise e classificação, o arquiteto tende a voltar aos antigos exemplos - soluções tipológicas - para resolução de novos problemas.

Dessa maneira, Colquhoun (1967), ao defender o uso da tipologia, afirma que sua intenção não é defender o retorno de uma arquitetura que admite a tradição de modo impensado, isso seria o mesmo que estabelecer uma relação fixa e imutável entre formas e significados. “A mudança é a característica de nosso tempo, e justamente por isso é preciso investigar o papel que as modificações de soluções-tipo desempenham com relação aos problemas e soluções que não têm precedentes em qualquer tradição” (COLQUHOUN, 1967 p.49).

Portanto, diante dos estudos de autores como Argan (1962 *in* NESBITT, 2006) e Colquhoun (1967), fica claro que existe a necessidade de se estabelecer um sistema de análise que considere as formas e soluções do passado no intuito de ganhar controle sobre conceitos que interferirão e auxiliarão no processo de criação em arquitetura.

Por isso acredita-se que a partir da implementação computacional sugerida por Mitchell (1990) em *The Logic of Architecture: Design, Computation, and Cognition* tornar-se-á possível atingir uma compreensão aprofundada do processo de projeto, além de resgatar o uso da tipologia como método projetual.

2.1.2 VOCABULÁRIOS ARQUITETÔNICOS E CLASSES DE OBJETOS

Impulsionado pela necessidade de estabelecer uma linguagem crítica suficientemente expressiva para a formulação completa e precisa das necessidades do problema de projeto, Mitchell (1990), em *The Logic of Architecture: Design, Computation, and Cognition*, propõe a definição de classes e de objetos como maneira de representar as propriedades de um edifício.

Para estabelecer a distinção entre classes (tipos arquitetônicos) e objetos (elementos ou instâncias), Mitchell (1990) baseia-se originalmente na proposta do filósofo americano Peirce (1931 *apud* MITCHELL 1990), para quem definiu um elemento como uma instância de um

determinado tipo. Essa idéia de instanciamento de objetos é ilustrada por Mitchell (1990) através da palavra ‘ARQUITETURA’. Nesse exemplo, “a letra R é instanciada duas vezes, a letra E é instanciada uma vez, e a letra Z não é instanciada nenhuma vez”. O autor expõe em sua obra uma passagem de texto em que Peirce (1931 *apud* MITCHELL 1990) afirma que os “elementos podem ser de um mesmo tipo por possuírem algo em comum. Cada elemento obedece ao tipo a que pertence”.

Nesse raciocínio, Mitchell (1990) estrutura o conceito de tipo através do ato de abstração, identificando “o que há de semelhante entre todos os membros de uma classe de objetos e ignorando suas diferenças” (MITCHELL, 1990). O resultado dessa estruturação consiste na distinção entre duas propriedades: essenciais e acidentais.

Para Mitchell (1990), as propriedades essenciais são as características comuns a todos os objetos pertencentes ao mesmo tipo, já as propriedades acidentais consistem nas características que podem variar entre os exemplares de um mesmo tipo. Diante dessa abordagem já se torna possível afirmar que os objetos que se repetem são as propriedades essenciais para a existência do tipo arquitetônico. Entretanto, para esclarecer com maior precisão essa abordagem o autor descreve a seguinte questão:

> Podemos optar por dizer que a simetria bilateral é uma propriedade essencial dos pórticos clássicos, mas que eles podem possuir diferentes cores. Dessa forma, a simetria bilateral do pórtico do Partenon é uma propriedade essencial, enquanto sua cor branca é uma propriedade acidental. Se o Partenon tivesse cores vivas (como de fato um dia teve), ele ainda seria visto como um pórtico clássico. Contudo, se ele fosse modificado de alguma maneira, perdendo sua simetria lateral (por exemplo, com a remoção de três colunas de um dos lados), então ele deixaria de ser um pórtico clássico. A essência do pórtico clássico não estaria preservado neste caso (MITCHELL, 1990, p.87).

Mitchell (1990), em *The Logic of Architecture: Design, Computation, and Cognition*, propõe essa subdivisão do problema para representar a definição de subtipos arquitetônicos. Embora o diagrama representado pelo autor seja obtido por meio da chamada especificação ou generalização de propriedades essenciais, este conceito está diretamente relacionado a um dos conceitos da orientação a objetos que é a herança. Como exemplo dessa abordagem o autor descreve a divisão dos tipos pórticos clássicos em subtipos dóricos, jônicos e coríntios, e de maneira oposta, a divisão menos abrangente de tipos, removendo algumas de suas propriedades

essenciais, como a generalização de quadrados em retângulos e retângulos em quadriláteros, conforme ilustra a Figura 2.1.

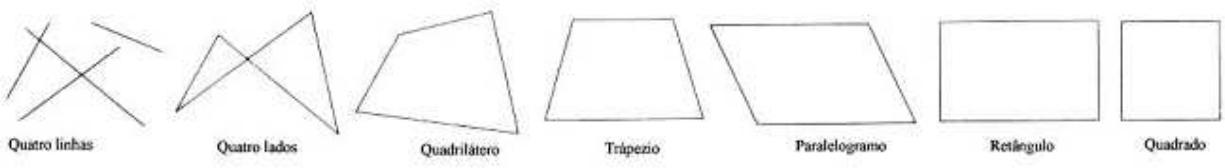


Figura 2.1. Hierarquia de subtipos restritivas.

Fonte - Mitchell, 1990.

As definições técnicas de tipo e propriedades essenciais propostas por Mitchell (1990) são precisas e bem estruturadas, além de coerentes com as definições teórico-filosóficas de Quatemère de Quincy (18-- *apud* MARTINEZ, 2000), Argan (1962 *in* NESBITT, 2006), e Colquhoun (1967). A abordagem feita em *The Logic of Architecture: Design, Computation, and Cognition* representa uma tentativa bem sucedida de Mitchell (1990) em definir tipos arquitetônicos através de conceitos computacionais. Entretanto, a análise feita por Mitchell (1990) em sua obra ainda se prolonga aos conceitos de essências absolutas e essências relativas, como fundamento para as suas idéias sobre propriedades essenciais e acidentais.

De acordo com Mitchell (1990), alguns filósofos afirmam que as propriedades essenciais são dadas e imutáveis. A revelação da essência de um objeto seria obtida por desconsiderar as superficialidades e os erros acidentais, ou seja, através de conceitos universais e idéias gerais. Por outro aspecto, outros filósofos afirmam que as essências são relativas, podendo a distinção entre essencial e acidental ser feita de diferentes maneiras. Nessa abordagem, conclui-se que a interpretação de um objeto pode ser determinada de acordo com diferentes interesses. Não existe uma regra universalmente correta. Rescher (1975 *apud* MITCHELL, 1990) consegue ilustrar essa idéia da seguinte maneira:

> Suponha que um determinado item é introduzido em uma discussão, como por exemplo, a cadeira de madeira da cozinha. Do ponto de vista de um marceneiro, o fato essencial é que se trata de uma peça de mobiliário feita em madeira; do ponto de vista do bombeiro o fato essencial é que se trata de um objeto de madeira (RESCHER, 1975 *apud* MITCHELL, 1990, p.92).

Essa discussão induz à conclusão de que um mesmo objeto pode seguir diversas convenções de abstração. A posição relativista da arquitetura tende a ser utilizada de maneira favorável, mesmo dentro da tradição clássica. Como exemplo, Mitchell (1990) cita em seu texto uma passagem de Summerson (1963 *apud* MITCHELL, 1990), que define um conjunto de tipos arquitetônicos aparentemente bem definidos e imutáveis - as cinco ordens da arquitetura clássica - da seguinte maneira:

> Sejamos claros sobre quão variáveis ou invariáveis são as ordens. Serlio as apresenta com enorme autoridade, dando as dimensões de cada parte como se estivesse estabelecendo suas proporções de uma vez por todas. Contudo, as ordens de Serlio, apesar de parcialmente baseadas em Vitruvius, são também resultado de suas próprias observações dos monumentos antigos, escolhidos por meio de um processo pessoal de seleção, tratando-se, portanto, de certa forma, de sua própria invenção. E não poderia ser de outro modo, já que as descrições de Vitruvius são incompletas e as informações inexistentes só poderiam ser preenchidas a partir da observação dos monumentos sobreviventes. As ordens exemplificadas nesses monumentos, contudo, variam consideravelmente entre si, sendo, portanto, permitido a qualquer um que selecione o que considera as melhores características de cada ordem para estabelecer o que deve ser considerado o estilo ideal: coríntio, jônico ou qualquer que seja. Essa especulação a respeito das ordens continuou através de toda a história da arquitetura clássica, oscilando entre o respeito ao mundo antigo de um lado e a invenção pessoal de outro (SUMMERSON, 1963 *apud* MITCHELL, 1990, p.92).

Certamente Summerson (1963 *apud* MITCHELL, 1990) ilustrou as diferentes características essenciais da ordem dórica, representada por sete autoridades no assunto, como mostra a Figura 2.2.

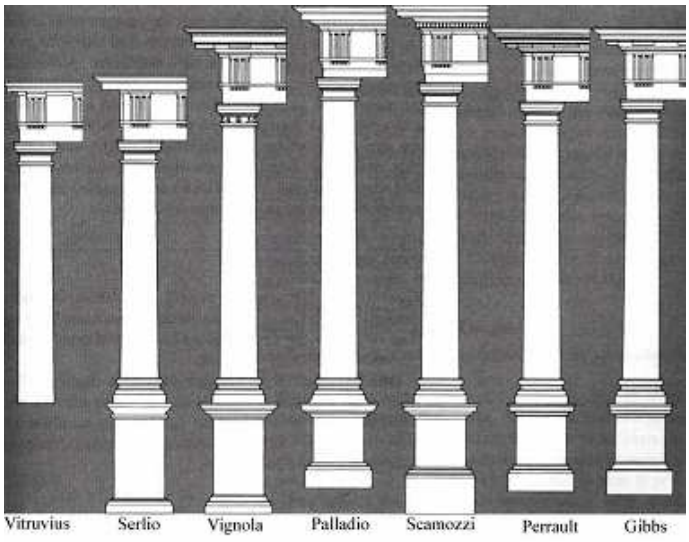


Figura 2.2. Características essenciais da ordem dórica

Fonte - Mitchell, 1990.

Portanto, pode-se concluir que não existe definição única do que sejam propriedades essenciais e acidentais. Tanto na arquitetura, como na computação, a estruturação de uma classe de objeto depende apenas dos interesses particulares de cada indivíduo envolvido no processo de modelagem de dados.

Nesse discurso, Mitchell (1990), ao definir os tipos arquitetônicos como classes de objetos, estrutura primeiramente as condições necessárias e suficientes para que um objeto seja considerado uma instância de um determinado tipo, tais como:

- Necessidade de utilizar diagramas como modelo de representação das propriedades de um objeto;
- Subdividir o diagrama em diagramas parciais que ilustrem diferentes características, possibilitando especificar a essência do tipo por meio do conjunto.
- Estruturar as informações sob forma de hierarquias taxonômicas, em que os subtipos herdam as propriedades dos tipos arquitetônicos.

No caso de instâncias excepcionais, Mitchell (1990) destaca a dificuldade de trabalhar com o esquema proposto. Ele cita o exemplo das colunas clássicas que de maneira geral possuem uma base e são representadas pela notação da lógica de primeira ordem abaixo.

*clássica (**Coluna**) → possui (**Coluna**, base)*

Entretanto, as colunas do Partenon, apesar de serem clássicas, não possuem bases. Essa dificuldade, por sua vez, seria resolvida ao permitir a existência de anormalidades, como a notação abaixo:

*clássica (**Coluna**) e não Partenon (**Coluna**) → possui (**Coluna**, base)*

Essa abordagem estabelecida por Mitchell (1990) demonstra a necessidade de se atrelar, às definições realmente precisas e rigorosas, longa lista de exceções. Assim, o autor sugere outra maneira para trabalhar com o problema das instâncias excepcionais - as famílias de objetos.

Na abordagem por família de objetos, as instâncias são caracterizadas como “típicas”, por possuírem características semelhantes às de outros “membros da família”, ou “atípicas”, por terem apenas algumas dessas características. De acordo com Mitchell (1990) este esquema de tipos e seus limites ainda são definidos de maneira bastante indefinida: “As colunas dóricas do Partenon, por exemplo, são consideradas colunas clássicas levemente atípicas” (MITCHELL, 1990).

Para a resolução de instâncias excepcionais, Mitchell (1990) também apresenta o modelo de representação do conhecimento baseado em *frames,* que foram introduzidos por Minsky (1975 *apud* MITCHELL, 1990) em seu artigo *A framework to represent knowledge*, como é possível observar através dos Quadros 2.1 e 2.2. Essa estrutura permite definir tipos de maneira flexível e genérica. Um *frame* representa elementos com valores específicos, valores-padrão ou ainda um outro *frame*. “Um valor específico é aquele que sabemos ser verdadeiro, enquanto que um valor-padrão é aquele que acreditamos ser verdadeiro até que se prove o contrário. Assim, valores-padrão representam expectativas que podem ser ajustadas caso seja necessário” (MITCHELL, 1990). No exemplo de uma cadeira, a seguinte descrição pode ser feita:

Quadro 2.1. Cadeira

Cadeira
Especialização de: mobília Número de pernas: Default: 4 Se necessário: usar procedimento contar Número de braços: 0, 1 ou 2 Assento: Encosto: Default: o mesmo que o assento Estilo: Apropriado para: sentar

Fonte - Mitchell, 1990.

A descrição de uma instância particular de uma cadeira pode ser, por exemplo:

Quadro 2.2. Cadeira de Sala de Jantar

Cadeira de sala de jantar
Especialização de: cadeira Número de pernas: Número de braços: 2 Assento: trançado Encosto: madeira encurvada Estilo: Thonet Apropriado para: sentar-se à mesa

Fonte - Mitchell, 1990.

Nesse propósito, Mitchell (1990) afirma que, sobre a definição de tipos, “diremos simplesmente, de agora em diante, que é possível utilizar um esquema de classificação associado a um mecanismo de herança de propriedades baseado tanto em necessidades específicas como em valores-padrão” (MITCHELL, 1990).

Dessa forma, fica compreendido que um vocabulário arquitetônico não precisa ser fixo, explícito, ou bem definido, mas, em princípio, deve ser possível enumerar e dar nomes aos tipos nele compreendidos, seja por meio de diagramas de tipos, regras de reconhecimento ou *frames* (MITCHELL, 1990).

2.1.3 ORIENTAÇÃO A OBJETOS

Diante do estudo de tipos e vocabulários arquitetônicos citados por Mitchell (1990), a presente pesquisa direcionou-se à análise do paradigma da orientação a objetos. Definições como classes, instanciação de objetos, atributos e métodos do objeto, encapsulamento, herança e polimorfismo foram indispensáveis para a definição clara e coerente de tipos arquitetônicos como classes de objetos através de sua implementação no computador.

Uma classe de objetos nada mais é que uma definição abrangente de um determinado tipo de objeto. Pode-se definir, por exemplo, uma classe de edifícios residenciais, que possuem a característica de servirem como habitação, devendo possuir obrigatoriamente uma cozinha, uma sala e no mínimo um dormitório e um banheiro. Ao ser instanciada, ou seja, transformada em um edifício residencial específico, essa classe pode resultar em uma casa com 1, 2, 3 ou 4 dormitórios, e com 1, 2, 3 ou 4 banheiros. Objetos criados a partir de uma mesma classe formam uma família de objetos que possuem semelhanças entre si, com propriedades em comum, que definem seu tipo arquitetônico. Desse modo, as casas se parecerão com as outras casas, por exemplo. A Figura 2.3 mostra algumas instâncias de uma mesma classe – as residenciais.

Figura 2.3. Plantas baixas de tipologias residenciais

Fonte – Reis; Lay, 2002.

Os objetos possuem dois tipos de características principais: atributos e métodos. Um atributo é um aspecto do objeto. Por exemplo, uma casa tem paredes, portas e janelas. Essas características são atributos comuns a todas as casas. Os atributos dos objetos armazenam diferentes propriedades que um objeto pode conter. Já um método é uma ação que o objeto pode realizar. Por exemplo, em um sistema CAD é possível orientar uma casa em determinada direção, espelhar ou mesmo rotacionar esse objeto de acordo com a sua operação.

Na ciência da computação, a orientação a objetos permite a construção de sistemas computacionais formados por diversos objetos que se relacionam entre si. Esses objetos representam uma entidade plausível de criação, manipulação e interação, podendo existir no mundo real ou ser desenvolvido a partir do estudo de outros objetos do mundo real. Keogh e Giannini (2005) afirmam que um objeto pode ser uma pessoa, um lugar, um utensílio, um conceito ou, possivelmente, um evento. Entretanto, um objeto pode simplesmente ser definido como um elemento que possui significado para uma determinada aplicação.

Para esclarecer os alicerces da programação orientada a objetos, algumas definições são necessárias. Os principais conceitos da programação orientada a objetos são:

Classe: é uma descrição de um conjunto de objetos que compartilham os mesmos atributos, comportamentos (métodos), relacionamentos e semântica.

Para construir uma casa, por exemplo, é necessário: paredes, portas e janelas como alguns itens importantes para o projeto de uma residência. Cada um desses itens também possui um conjunto de propriedades. As paredes têm comprimento, largura e altura. As portas também têm comprimento, largura e altura, mas possuem o comportamento que permite que sejam abertas e fechadas em uma determinada direção. As janelas são semelhantes às portas por terem aberturas que passam através das paredes. A representação gráfica de classes é feita conforme mostra a Figura 2.4. Booch, Rumbaugh e Jacobson (2005) afirmam que essa notação permite visualizar uma abstração independente de linguagem de programação específica, e de uma maneira que torna possível dar ênfase às partes mais importantes de uma abstração: seu nome, atributos e métodos.

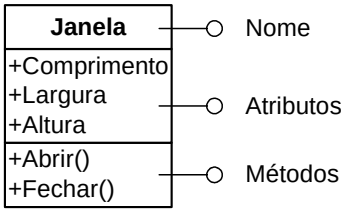


Figura 2.4. Representação de classe

Objeto: é uma instância de uma classe, com os valores específicos atribuídos a cada variável. A instanciação acontece quando a classe produz um objeto como se ela fosse uma espécie de modelo para a criação de objetos. A Figura 2.5 exemplifica o instanciamento de uma classe cadeira.

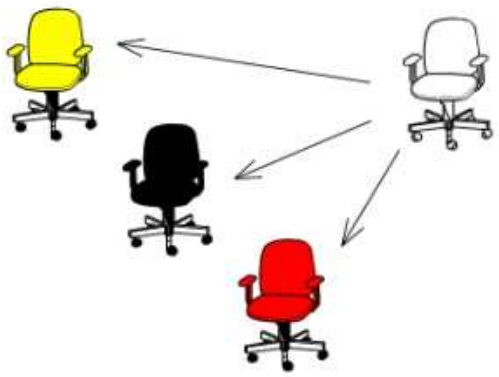


Figura 2.5. Instanciação de Objeto.

Fonte - Carrara; Kalay; Novembri, 1994.

As instâncias são formadas por relações que estabelecem uma hierarquia entre as partes e um objeto como todo. Uma cadeira, por exemplo, pode ser considerada uma montagem de partes como o assento, encosto, pés e rodas, conforme ilustra a Figura 2.6.

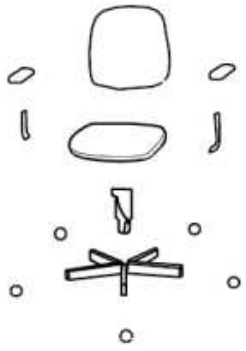


Figura 2.6. Componentes do Objeto

Fonte - Carrara; Kalay; Novembri, 1994.

Encapsulamento: é uma técnica que omite informações necessárias apenas a uma determinada entidade. Consiste num nível de privacidade para controle de dados, não permitindo que o usuário do objeto os acesse diretamente, mas sim através de métodos quando houver necessidade.

Herança: é um mecanismo utilizado para criar subclasses que estendem ou guardam relações com outras classes, unindo partes diferentes de um ou mais objetos, sejam eles simples ou complexos. A herança acontece quando existe uma relação entre pai-filho, ou seja, entre classe e subclasse. Na Figura 2.7, por exemplo, a classe “Edifício” é o pai no relacionamento entre as classes de objetos, sendo suas características herdadas pelas subclasses “residencial”, “educacional” e “industrial” que, por sua vez, têm suas características herdadas pelas demais subclasses. O método da ciência da computação utilizado para a representar, graficamente, a hierarquia de classes, é conhecido como Unified Modeling Language (UML). O uso dessa linguagem-padrão permite estruturar as classes de maneira visual e organizada, facilitando a compreensão do sistema como um todo, corforme ilustra a Figura abaixo.

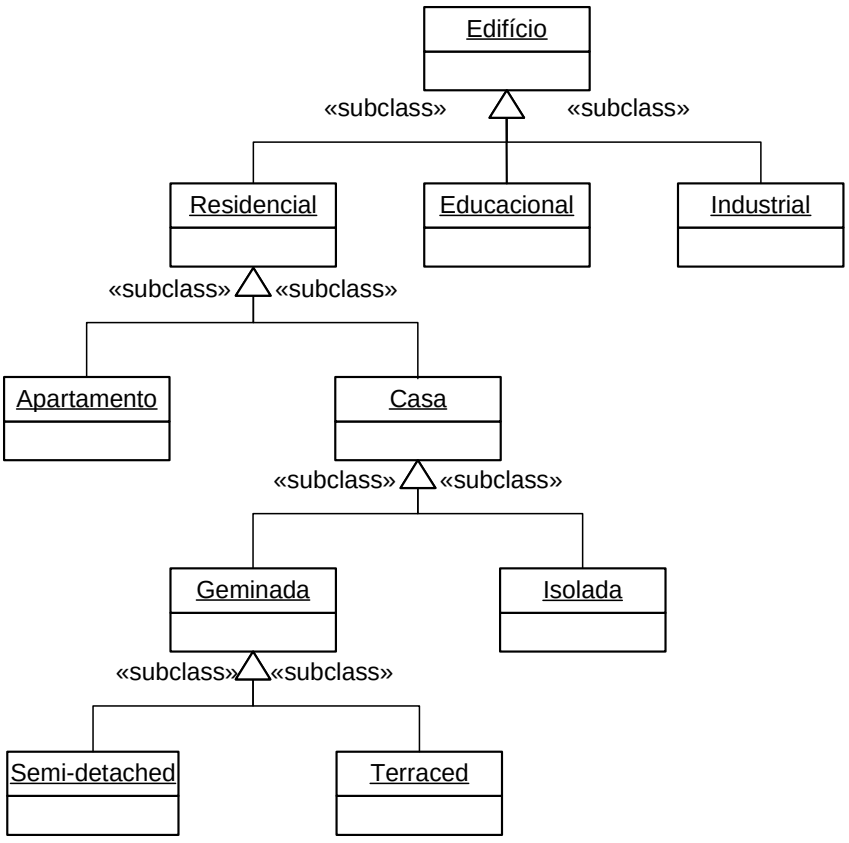


Figura 2.7. Representação de Herança

Da mesma maneira como Mitchell (1990) propõe em *The Logic of Architecture* a divisão do problema de projeto para representar a definição de subtipos arquitetônicos, o paradigma da

orientação a objetos apresenta a especificação ou generalização de classes como forma de descrever o relacionamento existente entre os itens de um problema. O intuito principal por trás dessa conexão entre as classes consiste, na verdade, em compartilhar informações e colaborar umas com as outras para a estruturação clara e coerente do problema em questão. Para isso, é indispensável compreender dois importantes relacionamentos entre classes, isto é, a especificação ou generalização e a agregação, ambas representadas através da UML.

- Especificação ou Generalização: são chamados relacionamentos “é um tipo de”. Isso significa que os objetos das subclasses (filhos) herdarão as propriedades da classe (pai), principalmente seus atributos e métodos. A Figura 2.8 demonstra o relacionamento de especificação para alguns tipos de janelas, o que naturalmente implica em dizer, por exemplo, que a classe “De correr” é um tipo de “Janela”, e que esta, por sua vez, possui dois outros subtipos: “Deslizante” e “Guilhotina”.

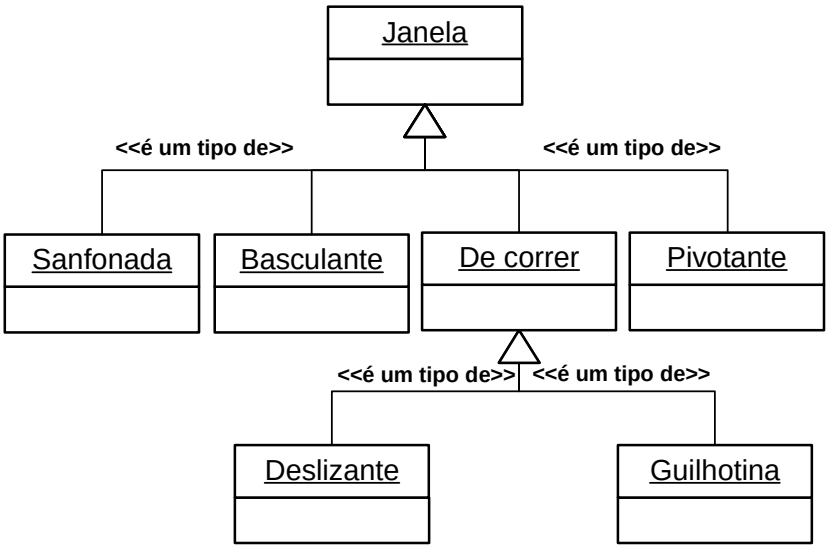


Figura 2.8. Exemplo de especificação de classes

- Agregação: são chamados relacionamentos “faz parte de”. Isso significa que uma determinada classe representará um item maior (“todo”), formado por itens menores (“partes”). A Figura 2.9, por exemplo, ilustra alguns objetos que compõem a classe “janela”. Dessa forma, é possível compreender que os objetos “Moldura”, “Vidro” e “Caixilho” compõem a classe “Janela”.

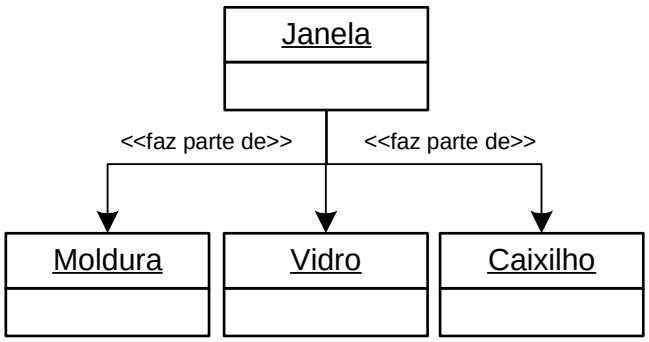


Figura 2.9. Exemplo de agregação

Através da agregação ainda é possível determinar o nível de dependência que uma classe possui com os demais objetos envolvidos no relacionamento. Através da Figura 2.10 é possível perceber que, além da agregação de objetos, existe outra informação representada pelos valores “0..2”, “0..*” e “1..1”, a qual se define como multiplicidade.

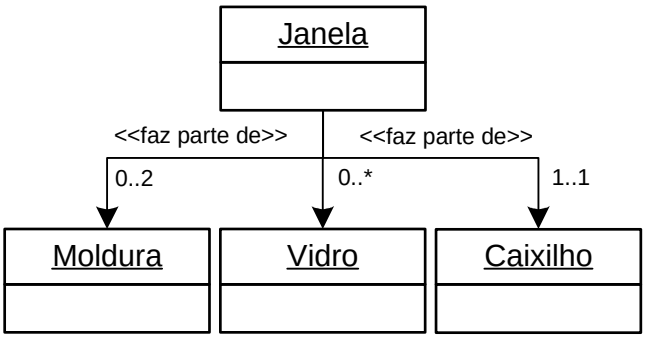


Figura 2.10. Multiplicidade existente entre os objetos.

Dessa forma, com base na Figura 2.10 acima ilustrada, é possível compreender que uma “Janela” pode ser composta por:

- No mínimo zero (nenhum) e no máximo duas Molduras;
- No mínimo zero (nenhum) e no máximo muitos Vidros;
- No mínimo um e no máximo um Caixilho;

A Tabela 2.2 demonstra a notação usada pela UML, para os indicadores de multiplicidade.

Tabela 2.2. Exemplos de Multiplicidade

Multiplicidade	Significado
0..1	No mínimo zero (nenhum) e no máximo um. Indica que os objetos das classes associadas não precisam obrigatoriamente estar relacionados, mas se houver relacionamento indica que apenas uma instância da classe se relaciona com as instâncias de outras classes.
1..1	Um e somente um. Indica que apenas um objeto da classe se relaciona com os outros objetos da outra classe.
0..*	Zero ou mais. Indica que pode ou não haver instâncias da classe participando do relacionamento.
*	Muitos. Indica que muitos objetos da classe estão envolvidos no relacionamento.
1..*	Um ou mais. Indica que há pelo menos um objeto envolvido no relacionamento, podendo haver muitos envolvidos.

Fonte – Guedes, 2004, p. 74.

Polimorfismo: possibilita que um comportamento seja recuperado por diferentes objetos de classe-tipo, resultando em diferentes respostas. Supondo existirem diversas subclasses que herdam um método de uma classe (pai), se uma delas redeclarar este método, ele só se comportará de maneira diferente nos objetos da classe que o modificou, permanecendo igual à forma como foi implementado na classe (pai) para os objetos de todas as outras subclasses (GUEDES, 2004).

No que diz respeito à raiz teórica desta pesquisa é possível afirmar que tanto as definições apresentadas sobre as tipologias arquitetônicas, como as definições sobre o paradigma da orientação a objetos, possuem um mesmo ponto de reflexão, isto é, o conceito de objeto. Esse conceito, que provavelmente teve sua origem na teoria das idéias de Platão, é na verdade o princípio do pensamento orientado a objetos e o responsável por influenciar o vocabulário arquitetônico através de seus métodos de classificação.

Para Platão (2007) cada objeto concreto pertence, juntamente como todos os outros objetos de sua classe, a uma idéia perfeita. Em linhas gerais, esse pensamento sugere que os objetos físicos percebidos pelos sentidos nada mais são do que imitações do mundo perfeito das idéias. Uma determinada caneta, por exemplo, terá determinados atributos (cor, formato, tamanho e outros). Outra caneta terá outros atributos, sendo ela também uma caneta, tanto quanto a outra. Aquilo que faz com que as duas sejam canetas é, para Platão, a idéia de caneta, perfeita, que esgota todas as possibilidades de ser caneta (PLATÃO, 2007).

Na arquitetura, é também possível indicar a teoria das idéias de Platão como a origem dos pensamentos de um artista. Mitchell (1990) afirma em *The Logic of Architecture: Design, Computation, and Cognition* que essa teoria de Platão foi ainda mais desenvolvida por diversos escritores, sugerindo sua aplicação à pintura, escultura e arquitetura. Nesse contexto, o autor cita uma famosa passagem de Summa Theologia, em que São Tomás de Aquino dizia: “a casa pré-existe na mente do arquiteto: e a isto se pode chamar de Idéia de uma casa, pois o artista projeta a casa real de maneira semelhante à que ele imagina em sua mente” (MITCHELL, 1990, p.37).

Assim, é perfeitamente normal que depois de pensar bem sobre determinado objeto se chegue à conclusão de que todas as coisas possuem um denominador comum. “Embora nenhum deles seja absolutamente perfeito, você suspeita que eles devem ter uma origem comum”. A esta realidade Platão chama de “mundo das idéias”. Nele estão as “imagens padrão” e as imagens primordiais que encontramos na natureza. Esta notável concepção é chamada de a teoria das idéias de Platão (GAARDER, 1995, p.100).

2.2 OUTRAS PESQUISAS

No intuito de proporcionar uma maior familiaridade com o tema investigado, a presente pesquisa buscou explorar outros estudos que também se propuseram a utilizar métodos e conceitos computacionais como suporte ao desenvolvimento de projetos arquitetônicos e implementações CAD. Para isso foram analisadas inicialmente as idéias teóricas de Christopher Alexander, apresentadas em *Notes on the synthesis of form* (1964) e *A Pattern Language: Towns, Buildings, Construction* (1977), exatamente por utilizar métodos de projeto similares aos métodos da computação, e para ilustrar uma questão prática do processo de projeto auxiliado pelo computador estudou-se a ferramenta computacional AutoMET, desenvolvida por um grupo de pesquisa de metodologia do projeto e conforto ambiental da FEC-UNICAMP para automatizar processo de projeto de casas populares na cidade de Campinas - São Paulo.

As idéias coletadas a partir desse estudo exploratório são descritas a seguir.

2.2.1 ESTRUTURAÇÃO DO PROBLEMA DE PROJETO

O método de projeto proposto por Alexander (1964) em *Notes on the Synthesis of Form* é fundamentado por um conjunto de teorias da matemática moderna, como a teoria dos grafos, teoria de conjuntos e pelo uso do computador (JUTLA, 1993). Nessa obra, o autor sugere que seja feita uma divisão dos problemas de projeto em partes menores e, conseqüentemente, uma estruturação de prioridades na resolução de problemas como exemplifica a Figura 2.11.

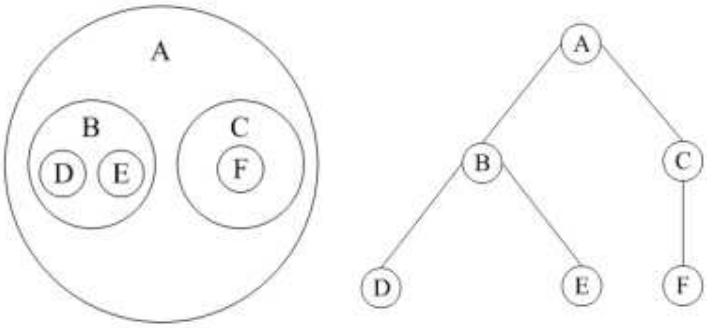


Figura 2.11. Representação de conjuntos

Fonte – Veloso; et al, 1983.

O respectivo trabalho representa uma tentativa do autor de encontrar uma fórmula que garanta um projeto bem sucedido. Apesar de seu objetivo parecer simplista, ao mesmo tempo que pretensioso, Alexander (1964) demonstra em *Notes on the Synthesis of Form* uma preocupação com diversos aspectos importantes para o projeto de arquitetura, tais como: forma-contexto, ajuste-desajuste e consciência-inconsciência.

Nessa obra, Alexander (1964) afirma que todo o problema de projeto se inicia com o esforço para obter um ajuste entre forma e seu contexto. A forma à qual o autor se refere não é a figura do edifício, mas o problema sobre o qual o arquiteto pretende atuar, e o contexto não é o ambiente que receberá o novo objeto, mas os requisitos que o definem. De acordo com Alexander (1964) "a forma é a solução para o problema e o contexto o que define o problema". O que significa que o real objetivo da discussão do projeto em *Notes on the Synthesis of Form* não é centrado unicamente na forma, mas no conjunto que compreende forma e contexto.

Após essa discussão entre forma e contexto, Alexander (1964) ainda aborda questões como ajuste e desajuste no conjunto que compreende o projeto. Mesmo não conseguindo esclarecer um modo prático para a identificação de um bom ajuste, ele sugere que as exigências para um bom ajuste sejam vistas de um ponto de vista negativo, tornando-se assim mais fácil designar os desajustes que a forma pode obter.

Jutla (1993), em seu artigo intitulado *Christopher Alexander's design theory from Notes on the synthesis of form to a Pattern language*, ressalta, contudo, alguns problemas com relação aos conceitos de ajuste e desajuste propostos por Alexander (1964). Para ele:

- Pode ser difícil encontrar todos os desajustes;
- A informação sobre o ajuste pode ser tão importante quanto aquela sobre o desajuste.
- Se não houver informação suficiente sobre os ajustes, algum arquiteto pode ignorar a solução mais apropriada para o projeto e pode terminar criando um outro desajuste;
- O mundo não é dividido em bom/ajuste e ruim/desajuste.

Dessa maneira, fica claro que não é uma regra o fato de que todos os arquietos conseguirão a mesma lista de desajustes, conforme acredita Alexander (1964) em *Notes on the Synthesis of Form*. Alguns fatores que podem ser considerados desajuste de acordo com determinados padrões culturais, por exemplo, podem não ser de acordo com outros. Diferentes arquitetos provavelmente identifiquem desajustes diversos.

Nesse sentido, um dos últimos aspectos a serem abordados por Alexander (1964) diz respeito à visão inconsciente e consciente que uma sociedade possui em relação à arquitetura. Para ele essa distinção entre a cultura inconsciente e a cultura consciente da forma, estabelecida pela ausência ou não de normas específicas para esse processo de concepção, o ajudou a discutir sobre a eficiência cultural de uma sociedade em razão de um bom ajuste e clareza que as formas podem apresentar.

A partir desse momento, Alexander (1964) constatou, em sua pesquisa, a inexistência de um modelo gráfico para a representação do problema de projeto. Para ele seria necessário que o objeto de estudo fosse colocado no mundo real, para então se verificar o bom funcionamento ou não desse projeto. Contudo, essa ausência de uma descrição simbólica impulsionou Alexander (1964) a definir como o problema de projeto deveria ser tratado por meio de uma representação gráfica. Conceitos como "simplicidade, não arbitrariedade e clara organização" foram adotados

como preceitos para esse modelo conceitual de concepção da forma (ALEXANDER, 1964). Diante desse contexto, ele descreveu três possíveis diagramas para o processo de projeto.

O primeiro diagrama representa a chamada cultura inconsciente do processo de projeto, conforme exemplificado pela Figura 2.12. Neste modelo gráfico inicial, “o processo de concepção da forma constitui uma relação bidirecional entre contexto C1 e a forma F1, no mundo real. O ser humano participa apenas como um agente neste processo. Ele reage diante dos desajustes alterando-o” (ALEXANDER, 1964).

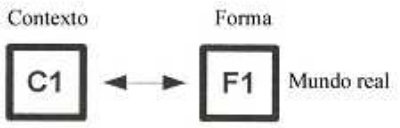


Figura 2.12. Diagrama do projeto na cultura inconsciente

Fonte – Alexander, 1964.

Seguindo essa linha de raciocínio, Alexander (1964) propõe o segundo diagrama do processo de projeto, que representa a cultura consciente do processo de projeto, de acordo com a Figura 2.13. Neste caso, a modelagem da forma é feita mediante uma interação conceitual do contexto C2 que o projetista abstrai e as idéias e desenhos que correspondem à forma F2 a ser projetada. Apesar de não ser um processo exatamente claro, são consideradas informações relevantes para o desenvolvimento da forma, “dependente sempre de uma classe de intuição” (ALEXANDER, 1964).

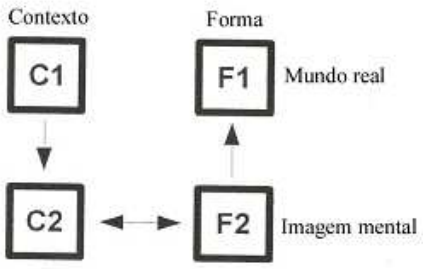


Figura 2.13. Diagrama do projeto na cultura consciente

Fonte – Alexander, 1964.

O terceiro e último esquema de diagramas representa uma nova variação do primeiro mapa mental do problema conforme é observado pela Figura 2.14. O processo consciente ocorre por meio de uma operação precisa, que não está sujeita à parcialidade da linguagem ou da experiência. Assim, “a representação vaga e insatisfatória das exigências do contexto C2, que inicialmente se desenvolve na mente do projetista, é seguida de uma imagem matemática, C3. De maneira análoga, o projeto F2 é precedido por um complexo diagrama F3” (ALEXANDER, 1964). Mesmo sendo intuitiva a derivação de F3 a partir de C3, o mesmo pode ser compreendido claramente.

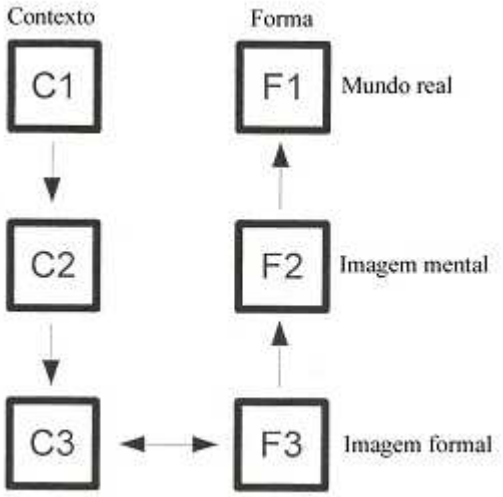


Figura 2.14. Representação formal do contexto (C3) e forma (F3)

Fonte – Alexander, 1964.

O modelo C3 da Figura 2.14 é construído através de entidades matemáticas conhecidas como "conjuntos". Um conjunto consiste em uma coleção de objetos bem definidos, cujas principais propriedades são:

- Se um elemento x pertence ao um conjunto S. Então $x \varepsilon S$;
- Se um conjunto S1 é um subconjunto de outro conjunto S2, cada elemento de S1 pertence a S2;

- A união de dois conjuntos S1 e S2 é o conjunto daqueles elementos que pertencem a S1 ou S2, ou ambos no caso em que S1 e S2 tenham elementos em comum;
- A interseção dos conjuntos S1 e S2 é o conjunto daqueles elementos que pertencem tanto a S1 como a S2. Se S1 e S2 não possuem elementos em comum, a interseção será vazia e os conjuntos serão chamados disjuntos.

O problema de projeto, representado por um conjunto de requisitos, pode ser dividido em outros conjuntos menores (subconjuntos), construindo subproblemas do programa arquitetônico. Dessa maneira, a estrutura de árvore é utilizada para identificar os subconjuntos que possuem relações de hierarquia sob os demais subconjuntos do programa. O problema pode ser organizado sob a forma de árvores, de tal modo que a hierarquia dos subconjuntos fique bem clara e definida, como exemplifica a Figura 2.15.

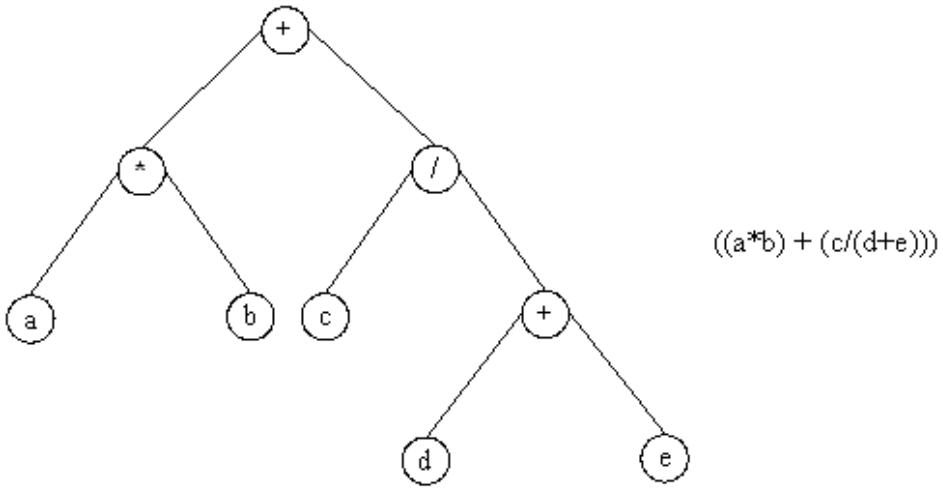


Figura 2.15. Representação de uma expressão aritmética sob forma de árvore

Fonte – Veloso; *et al*, 1983.

Assim, o método descrito por Alexander (1964) em *Notes on the synthesis of form* tem o intuito principal de estruturar os requisitos da forma de maneira simples e ordenada. Devido ao conjunto de informações que o arquiteto deve organizar, Alexander (1964) sugere que o problema de projeto seja resolvido através da divisão desse conjunto. Em *Notes on the synthesis of form*, Alexander (1964) ressalta a importância dos diagramas de modelagem e documentação como

mecanismos de organização das informações projetuais. Devido às informações em projetos de arquitetura tornarem-se cada vez mais complexas e abrangentes, o autor destaca a necessidade de se introduzir alguma técnica de modelagem formal antes de se iniciar o processo de elaboração do projeto. Já na década de 60 ele chamava a atenção para o fato de que "(..) somos autoconscientes, necessitamos de um mapeamento explícito de um problema, de sua estrutura, antes de poder convertê-lo em formas. Portanto, precisamos inventar um modelo conceitual para fazer esses mapeamentos" (ALEXANDER, 1964, p.129).

Conclui-se, portanto, que o programa arquitetônico descrito pelo autor surgiu da necessidade de métodos explícitos para a estruturação do problema de projeto. A proposta de um modelo conceitual, aparentemente óbvio, que pudesse representar as questões envolvidas no projeto foi o grande desafio que Alexander (1964) se propôs a resolver. Dessa forma, *Notes on the Synthesis of Form* representa um momento do trabalho teórico desenvolvido por Alexander (1964) em que se defende uma abordagem racional e a utilização de um método para resolução de problemas do projeto. Essa obra consiste no livro mais convencional e freqüentemente citado do autor, responsável por influenciar diversos arquitetos e também cientistas da computação - como Herbert Simon - desde sua publicação no ano de 1964.

No que se refere à publicação *A Pattern Language: Towns, Buildings, Construction* (1977), a trajetória teórica de Alexander, entretanto, moveu-se em uma nova direção. Jutla (1993) afirma que o autor, insatisfeito com a tradição arquitetônica modernista, percebeu que o método defendido em *Notes of Synthesis of Form* (1964) não dava resposta para a criação de edifícios que tivessem uma "qualidade atemporal". Essa constatação o teria impulsionado a favorecer um trabalho com padrões e enfatizar o uso do modelo exemplar como forma de promover a construção de novas idéias e soluções para o projeto de arquitetura.

Assim, através dessa evolução teórica, presumi-se que as idéias de Alexander sejam sustentadas por métodos de projeto fundamentalmente diferentes. De acordo com Carrara, Kalay e Novembri (1994), existem duas abordagens principais do processo de projeto em arquitetura. De acordo com a primeira abordagem, de influência fortemente positivista, a forma arquitetônica emergiria a partir da correta estruturação dos dados do projeto. E de acordo com uma segunda abordagem,

as soluções de projeto resultariam da adaptação de soluções típicas (antecedentes) a uma situação específica. As obras de Alexander *Notes on the synthesis of form* (1964) e *A pattern language* (1977) ilustram com precisão essas duas maneiras de se pensar sobre o projeto de arquitetura.

O primeiro paradigma apresentado por Carrara, Kalay e Novembri (1994) corresponde ao chamado *Problem-solving*. Um método que tenta, entre outras coisas, fornecer meios de se obter novas soluções do projeto, possibilitando encontrar as respostas mais apropriadas aos objetivos e aos requisitos predefinidos. Esse método de projeto é apoiado por um conjunto de teorias da matemática moderna, como por exemplo, a teoria dos grafos, teoria de conjuntos e os chamados métodos de otimização.

O segundo paradigma, chamado *Puzzle-making*, é baseado na adaptação e implantação de uma solução do projeto previamente desenvolvida. Tais soluções alcançam o estado de "protótipos", na qual soluções similares do projeto podem ser derivadas. De acordo com Carrara, Kalay e Novembri (1994), "na arquitetura, essa abordagem foi claramente manifestada por *A Pattern Language* de Christopher Alexander. A aplicação computacional utilizada para implementar este paradigma é conhecida como *Case-Based Design*" ou chamado *Knowledge-Based Systems* (CARRARA; KALAY; NOVEMBRI, 1994).

Diante do exposto, os autores Carrara, Kalay e Novembri (1994) afirmam que os métodos *Problem-solving* e *Puzzle-making* coexistem na prática e ambos são utilizados alternadamente por arquitetos ao longo do processo de projeto. "As duas abordagens são complementares, praticamente indivisíveis, e freqüentemente permutáveis" (CARRARA; KALAY; NOVEMBRI, 1994).

Dessa maneira, foi possível delinear com clareza a trajetória teórica de Alexander, estabelecida em *Notes on the synthesis of form* e *A pattern language*, e sintetizar o seu processo de projeto conforme ilustra a Tabela 2.3.

Tabela 2.3. Evolução teórica de Alexander

	Obras	Métodos de Projeto	Abordagem Computacional (CAD)
Christopher Alexander	*Notes on the synthesis of form*	*Problem-solving*	Teoria dos conjuntos; Teoria dos grafos; Métodos de otimização;
	A Pattern Language	*Puzzle-making*	*Case-Based Design*; *Knowledge-Based Systems*.

Com base nesse apanhado teórico, tornou-se possível afirmar que, de alguma maneira, os métodos defendidos por Alexander em *Notes on the synthesis of form* (1964) e *A Pattern Language* (1977) influenciaram a analogia que Mitchell (1990) estabeleceu entre tipos arquitetônicos e classes de objetos. Assim, seja pelos seus métodos de resolução de problemas ou pelas adaptações de soluções já existentes, a trajetória teórica de Alexander com toda certeza foi responsável por influenciar diversos arquitetos e também cientistas da computação através de sua grandiosa exploração sobre a metodologia do projeto em arquitetura.

2.2.2 AUTOMET

Na tentativa de ilustrar uma abordagem prática da utilização de métodos computacionais para a elaboração de projetos arquitetônicos, buscou-se analisar um exemplo de ferramenta computacional para o projeto arquitetônico, chamado Automet, desenvolvido por um grupo de pesquisadores do Departamento de Arquitetura e Construção (DAC) da Faculdade de Engenharia Civil (FEC) da UNICAMP. Essa pesquisa que na realidade, teve como ponto de partida um estudo realizado em edifícios residenciais unifamiliares produzidos por sistema de

autoconstrução na periferia de Campinas, levou à compreensão das necessidades e anseios da população. O resultado desse estudo proporcionou a construção de um aplicativo experimental, desenvolvido em AutoLisp para AutoCAD, para a geração de projetos para autoconstrução a partir de informações sobre número de membros da família, dimensões do lote e outros. O programa é capaz de produzir plantas de execução, modelo 3D e plantas de aprovação para prefeitura.

No intuito de levar essa implementação computacional até o seu público alvo, a FEC-UNICAMP, com financiamento da FINEP, ainda desenvolveu um projeto de pesquisa chamado TITAM (Transferência de Inovação Tecnológica na Autoconstrução de Moradias) que resultou na criação de uma “metodologia automatizada de projeto arquitetônico” (KOWALTOWSKI, 2003). Através desse projeto, foi possível analisar os resultados dos projetos em campo e o seu “impacto no fenômeno atual da autoconstrução na região” (KOWALTOWSKI, 2003), incluindo o desenvolvimento de uma "Cartilha do Autoconstrutor" e estudos pós-ocupação. Essa avaliação pós-ocupação incluiu a satisfação das famílias, sua intenção de modificar o projeto, e averiguação do conforto térmico e acústico nas residências construídas por meio de medições. Através do projeto TITAM foi também realizada uma comparação das residências resultantes do Automet com as da COHAB. A Figura 2.16 ilustra algumas tipologias residenciais da base de projetos do aplicativo computacional desenvolvido. O sucesso do projeto deve-se, em grande parte, ao respeito que foi dado às características sócio-culturais do grupo atendido.

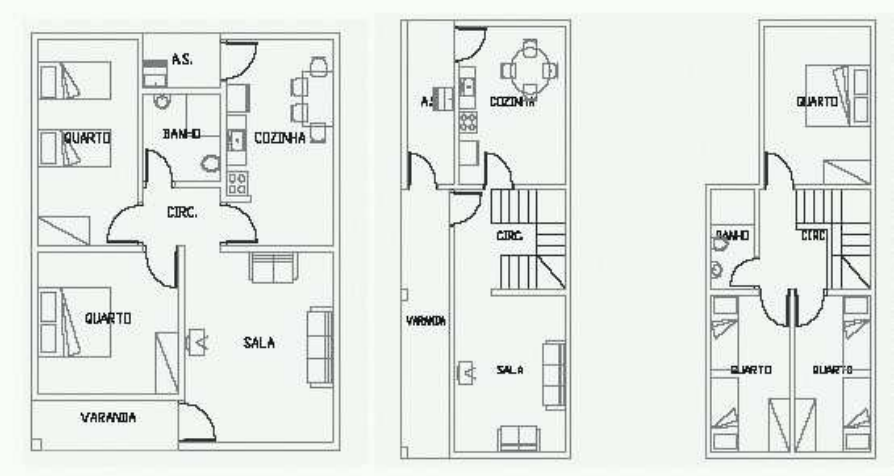


Figura 2.16. Exemplos da base de projetos do Automet

Fonte – Pina; *et al*, 2004.

Vale ressaltar que desde o estudo que originou o Automet, várias pesquisas foram desenvolvidas no intuito de melhorar ainda mais a aceitação dos projetos sugeridos pela ferramenta e sua aplicação em conjuntos habitacionais. Como exemplo é possível citar o caso do trabalho "Adaptação da ferramenta Automet para a simulação de projetos arquitetônicos em conjuntos habitacionais", de Rodrigues e Ruschel (2001), que propôs uma mudança da lógica de programação do Automet, sem que houvesse, no entanto a necessidade de se mudar a interface da ferramenta. O novo programa sugerido pelos autores, baseado em um banco de projetos dinâmico, foi capaz de desenvolver projetos para contextos variados.

O desenvolvimento de uma metodologia automatizada de projeto arquitetônico como o Automet foi, em resumo, uma grande iniciativa na busca por melhorias na qualidade das moradias autoconstruídas. Tratou-se de um exemplo explícito de projeto computacional, capaz de parametrizar algumas características de plantas residenciais, através da utilização eficiente do computador, e não apenas com vistas à representação.

Embora os objetivos da ferramenta Automet sejam diferentes dos propósitos desta pesquisa, o respectivo estudo tornou possível compreender, na prática, como um procedimento resultante da computação pode apoiar de maneira eficiente e robusta a elaboração de projetos arquitetônicos. A semelhança do Automet com a proposta desta pesquisa encontra-se apenas no fato de ambos utilizarem *layouts* pré-estabelecidos, mas difere no sentido em que a implementação de plantas residenciais desta pesquisa se faz como classes de objetos e não baseada em um banco de dados de plantas possíveis, como no caso do Automet.

3 METODOLOGIA

A partir da analogia estabelecida por Mitchell (1990) entre tipos arquitetônicos e classes de objetos, a presente pesquisa buscou estruturar um método de projeto alternativo que pudesse auxiliar o arquiteto no processo de resolução e representação formal do problema de projeto. Diante desse contexto, o processo de projeto sugerido nesta pesquisa tomou como ponto de partida a modelagem formal do problema de projeto proposta por Alexander (1964) em *Notes on the synthesis of form*, e a definição de tipos arquitetônicos estabelecida por Argan (1962 *in* NESBITT, 2006) e Colquhoun (1967), porém seguindo as diretrizes do paradigma da orientação a objetos.

3.1 O MÉTODO DE PROJETO

O método de projeto aqui sugerido se assemelha ao método de projeto defendido por Alexander (1964) em *Notes on the synthesis of form*, principalmente no que diz respeito à divisão do problema de projeto em partes menores de solução e à estruturação hierárquica das informações do problema de projeto em questão. Esse método de projeto é na verdade uma extensão do método de projeto defendido por Alexander (1964) em *Notes on the synthesis of form* (Figura 3.1), porém com um nível a mais de abstração. O quarto nível acrescentado à estrutura do autor tem como objetivo inserir as idéias de Mitchell (1990) no processo de projeto, com a definição das propriedades essenciais e acidentais de um edifício e a introdução do conceito de classes de objetos – neste caso classes de objetos arquitetônicos ou tipos arquitetônicos.

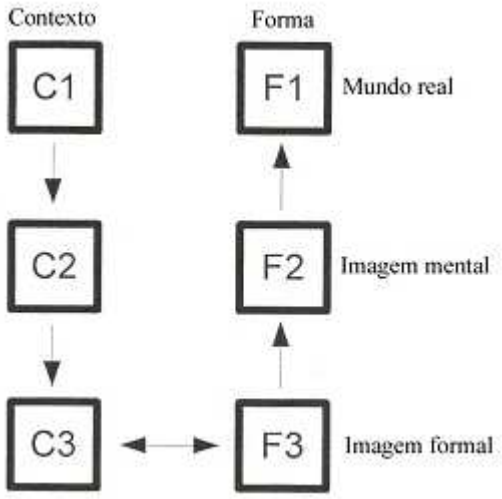


Figura 3.1. O processo de projeto segundo uma cultura consciente

Fonte - Alexander, 1964.

Além de proporcionar um nível maior de abstração do problema de projeto, quando comparado ao método de Alexander (1964), o método aqui sugerido tem o intuito principal de propiciar uma reflexão abrangente sobre classes de soluções preexistentes, apoiando dessa maneira um processo de projeto baseado também em precedentes e não apenas em inferências lógicas.

Essa conexão entre as idéias de Alexander (1964) e os conceitos de classes de objetos abordados por Mitchell (1990), permitiu consolidar a integração entre a metodologia de projeto e o paradigma da orientação a objetos na busca por uma nova compreensão do processo de projeto em arquitetura.

Assim, a analogia estabelecida nesta pesquisa sugere que a estrutura do pensamento do projeto seja dividida nas seguintes etapas: identificação do problema, formação de uma imagem mental, estruturação de uma imagem formal, definição das propriedades essenciais e acidentais do edifício, estudo de classes, implementação computacional da classe, instanciamento do objeto e construção do objeto no mundo real, conforme mostra a Figura 3.2.

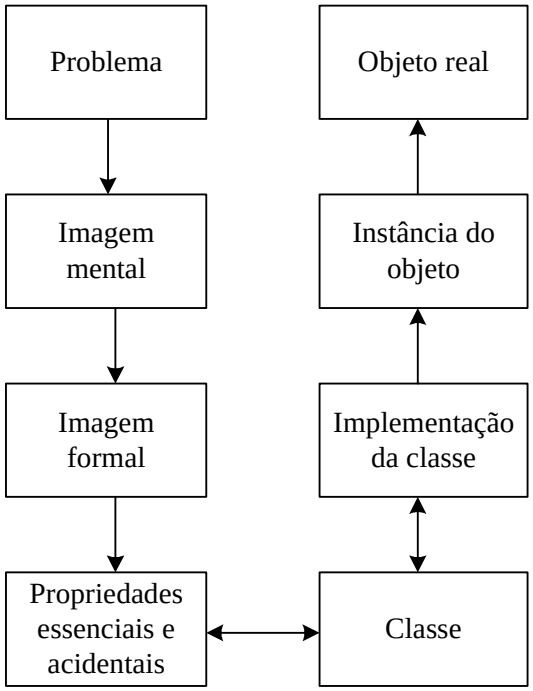


Figura 3.2. Etapas do processo de projeto baseado na idéias de Alexander (1964)

Para esclarecer a metodologia proposta, as etapas do processo de projeto são descritas como:

1. **O PROBLEMA:** corresponde à primeira etapa do processo de projeto, momento em que um determinado problema é abordado em busca de uma possível solução.
2. **IMAGEM MENTAL:** corresponde à segunda etapa do processo de projeto. Pode ser comparado ao estágio de incubação definido por Lawson (1997) em *How Designers Think* que envolve um esforço vago e insatisfatório para a solução do problema.
3. **IMAGEM FORMAL:** corresponde à terceira etapa do processo de projeto. É o momento em que o arquiteto realiza o levantamento de um conjunto de requisitos pertinentes ao problema de projeto e estabelece os vínculos existentes entre as variáveis. Faz parte desse estágio o programa de necessidades do problema em discussão.

4. **PROPRIEDADES ESSENCIAIS E PROPRIEDADES ACIDENTAIS:** corresponde à quarta etapa do processo de projeto. Este estágio tem o intuito principal de explorar os requisitos que são essenciais ou acidentais para a solução do problema. A estrutura em árvore é o mecanismo utilizado para a organização clara e coerente das informações projetuais.
5. **CLASSE:** corresponde à quinta etapa do processo de projeto. É o estágio em que estabelece uma consulta às tipologias (soluções do passado) no intuito de contribuir para o processo de resolução do problema.
6. **IMPLEMENTAÇÃO DA CLASSE:** corresponde à sexta etapa do processo de projeto. É o estágio em que se propõe a implementação computacional da classe, definida com base nas informações modeladas e documentadas em etapas anteriores. Nesse estágio são estruturadas as variáveis do problema de projeto em termos de classes, atributos e métodos.
7. **INSTÂNCIA DO OBJETO:** corresponde à sétima etapa do processo de projeto. É o estágio em que se estabelece a manifestação concreta da classe definida na etapa anterior.
8. **OBJETO REAL:** é a última etapa do processo de projeto. Corresponde à construção concreta do objeto no mundo real.

Dessa maneira, o processo de projeto sugerido nesta pesquisa teve como objetivo propor uma nova maneira de se pensar o processo de projeto, por meio da definição e implementação de uma tipologia arquitetônica como classe de objetos. Embora a utilização desse método de projeto no ensino não tenha feito parte do escopo deste trabalho, esta nova maneira de projetar poderia ser apresentada a alunos de um curso de arquitetura de diferentes maneiras: (1) de modo apenas teórico, com a definição de classes feitas pelos alunos de arquitetura sem, contudo, a necessidade de implementá-las em linguagem orientada a objetos; (2) de modo teórico, com a definição de classes realizadas pelos alunos de arquitetura e a sua implementação estabelecida por especialistas em computação; (3) e de modo teórico e prático, com a definição das classes de

objetos e sua implementação pelos próprios alunos de arquitetura em linguagem orientada a objeto, o que exigiria deles um certo conhecimento de programação de computadores.

Diferentemente de uma abordagem com interesses mais práticos, como é o caso da implementação computacional Automet, citada no capítulo anterior, a presente pesquisa teve o intuito principal de discutir o processo de organização e estruturação do pensamento arquitetônico. Isso significa dizer que a automação das etapas do projeto que foi proposto neste trabalho por meio do paradigma da orientação a objetos tem relevância, sobretudo, no que diz respeito à compreensão aprofundada do processo de projeto. Explorar o trabalho intelectual do arquiteto por meio do desenvolvimento de implementações computacionais pode gerar um impacto positivo na prática do projeto, permitindo estruturar o processo de projeto de maneira lógica e ordenada, além de ajudar o projetista a ter uma nova compreensão sobre seu trabalho em arquitetura.

4 PROCEDIMENTOS METODOLÓGICOS

O delineamento utilizado no desenvolvimento desta pesquisa seguiu alguns critérios importantes no intuito de alcançar os objetivos definidos no estudo da dissertação, conforme descritos a seguir.

4.1 MÉTODO

O ponto de partida, da pesquisa, consistiu na leitura sistematizada do capítulo sexto do livro *The Logic of Architecture: Design, Computation, and Cognition* de Mitchell (1990). Foi nele que o autor estabeleceu uma analogia entre tipos e classes de objetos – o tema central discutido nesta dissertação. Nesse sentido, para que as idéias expostas por Mitchell (1990) pudessem ser compreendidas em profundidade, foram pesquisadas explanações teóricas para todos os conceitos abordados no capítulo do livro. Esse procedimento conduziu a pesquisa à necessidade de abranger os campos da arquitetura, computação e filosofia no intuito de proporcionar uma compreensão minuciosa da analogia proposta. A tipologia arquitetônica, o paradigma da orientação a objetos e a teoria das idéias de Platão foram às proposições essenciais para o aprimoramento de idéias neste estágio da pesquisa.

Com o objetivo de tomar maior familiaridade como o problema estudado e com vista a torná-lo mais explícito, foi desenvolvido, na segunda etapa desta pesquisa, uma investigação criteriosa de outros trabalhos que, assim como o de Mitchell (1990), se propuseram a utilizar conceitos e métodos computacionais no processo de projeto em arquitetura. Nessa etapa da revisão bibliográfica, buscou-se investigar um trabalho de cunho teórico e outro com finalidades práticas,

desenvolvidos respectivamente por Alexander (1964) em *Notes on the Synthesis of Form* e a proposta da ferramenta computacional Automet.

A partir de um extensivo estudo exploratório e de toda a abordagem feita por Mitchell (1990) no capítulo sexto do livro *The Logic of Architecture: Design, Computation, and Cognition*, foi possível, então, realizar a terceira etapa da pesquisa. Esse estágio caracterizou-se pela inserção das idéias propostas Mitchell (1990), em *The Logic of Architecture*, ao modelo do processo de projeto proposto por Alexander (1964) em *Notes on the Synthesis of Form.* Assim, através da conexão firmada entre o paradigma da orientação a objetos e a metodologia de projeto tornou-se possível definir um método de projeto, alternativo aos métodos tradicionais de arquitetura, e refletir o processo de projeto por meio da definição e implementação de tipologias como classes de objetos, conforme exposto no capítulo 3 desta dissertação.

Para ilustrar o processo de projeto por meio da definição e implementação de um tipo como classe de objetos, foram desenvolvidos na quarta etapa desta pesquisa dois protótipos iniciais, ambos com natureza experimental e de acordo com os exemplos propostos por Mitchell (1990) em *The Logic of Architecture: Design, Computation, and Cognition*. Neste estágio da pesquisa, buscou-se utilizar um método bastante utilizado na área da ciência da computação, conhecido como desenvolvimento incremental, que permitiu implementar inicialmente uma classe bastante simplificada até chegar a uma classe mais complexa. Outro método da ciência da computação utilizado foi a *Unified Modeling Language* (UML), uma linguagem de modelagem gráfica que possibilitou representar as classes de objetos de maneira visual e organizada, auxiliando o trabalho de implementação computacional.

Dessa forma, uma vez modeladas todas as informações relacionadas aos protótipos iniciais, os dados - dos tipos “cadeira” e “ordem dórica” - foram transcritos para uma linguagem de programação orientada a objetos tornando-se explícito o estudo de tipos como classe de objetos. O intuito principal deste estudo preliminar foi, na verdade, possibilitar que se atingisse um amadurecimento conceitual sobre as técnicas de modelagem de dados. No entanto, através dessa abordagem inicial, foi possível pré-testar as técnicas para a criação de classe de objetos na linguagem de programação *Visual Basic for Application* (VBA) do AutoCAD, antes da elaboração do protótipo final proposto no escopo deste trabalho.

Assim, como última etapa da pesquisa, buscou-se desenvolver um exercício de projeto no intuito de ilustar a utilização de classes de objetos no desenvolvimento de projetos arquitetônicos. Uma tipologia residencial foi modelada e implementada computacionalmente como classe de objetos, porém seguindo as diretrizes do método de projeto sugerido no capítulo 3 desta dissertação.

Um procedimento de testes no programa implementado ainda foi desenvolvido. Porém, esse estágio não teve a intenção de ajustar aspectos lógicos interno à linguagem de programação, e sim focar nos efeitos produzidos pelo programa. Nesse sentido, o estágio de testes do programa final consistiu nos seguintes passos: geração de plantas, análise de resultados e realização de ajustes na classe implementada.

É importante destacar que a análise dos resultados obtidos por meio do programa final foi realizada à luz das categorias de "níveis de pretensão" no uso do CAD no desenvolvimento de projetos arquitetônicos propostas por Mitchell (1975) em *The theoretical foundation of computer-aided architectural design.* Em vista de se tratar de uma pesquisa de cunho mais teórico que prático, não se buscou medir a eficiência da implementação por meio de testes com usuários. É importante ressaltar que a implementação desenvolvida nesta pesquisa não teve como objetivo automatizar o processo de projeto visando a rapidez e eficiência, mas objetivando uma reflexão sobre o processo de projeto. Dessa forma, fez parte desta pesquisa apenas o contato direto e interativo do pesquisador com o objeto de estudo a fim de situar algumas interpretações sobre os fenômenos estudados. Enfim, o que se pretendeu alcançar através deste estudo foi apenas uma visão mais abrangente e explícita sobre tipos arquitetônicos como classes de objetos, e para isso a implementação prática teve um papel fundamental.

As etapas desenvolvidas neste estudo foram sintetizadas na Tabela 4.1

Tabela 4.1. Procedimentos metodológicos.

<table>
<tr><th>1ª etapa da pesquisa</th><th>2ª etapa da pesquisa</th><th>3ª etapa da pesquisa</th><th colspan="2">4ª etapa da pesquisa</th><th colspan="8">5ª etapa da pesquisa</th></tr>
<tr><td rowspan="2">Estudo aprofundado do capítulo sexto do livro The Logic of Architecture de Mitchell (1990).</td><td rowspan="2">Estudo exploratório de outros trabalhos que também usaram conceitos e métodos computacionais na arquitetura.</td><td rowspan="2">Proposta de um método alternativo aos métodos tradicionais de projeto.</td><td colspan="2">Definição e implementação computacional de dois protótipos preliminares, um tipo cadeira e um tipo ordem dórica.</td><td colspan="8">Definição e implementação computacional do protótipo final, uma tipologia residencial.</td></tr>
<tr><td>Método de desenvolvimento incremental</td><td>UML</td><td>Problema</td><td>Imagem mental</td><td>Imagem formal</td><td>Prop. essenciais e acidentais</td><td>Classe</td><td>Implemen-tação da classe</td><td>Instan-ciamento do objeto</td><td>Objeto real</td></tr>
</table>

4.2 MATERIAIS

O material utilizado para a implementação de tipos como classe de objetos foi o ambiente de programação *Visual Basic for Applications Interactive Development Environment* (VBAIDE) do AutoCAD. A escolha dessa linguagem de programação foi determinada exatamente pelo fato da sua execução ser interna ao sistema do AutoCAD, abordagem do material que tornou possível trabalhar diretamente com o *Object Model* do AutoCAD e conseqüentemente utilizar o ambiente gráfico padrão da ferramenta, sem a necessidade de implementar interfaces gráficas.

Os objetos do AutoCAD foram na realidade os componentes primordiais para a construção dos protótipos abordados nesta dissertação, pois foi através deles que tornou-se possível:

- Utilizar objetos gráficos como linhas, polilinhas, sólidos, hachuras dentre outros.
- Utilizar estilos de ajustes como tipos e dimensões de linhas.
- Utilizar a estrutura organizacional como layers, grupos e os blocos.
- Visualizar o protótipo gerado e a própria aplicação construída no VBA.

Apesar da linguagem de programação *Visual Basic for Application* (VBA) ser classificada por especialistas da área de computação como uma linguagem orientada a objetos, ficou claro ao longo do desenvolvimento dos protótipos desta pesquisa que o ambiente de programação utilizado é na verdade uma linguagem baseada em objetos e não rigorosamente orientada a objetos. Essa verificação trouxe consigo algumas das limitações da linguagem, principalmente no que diz respeito à implementação estrita de conceitos como herança, encapsulamento e polimorfismo. Contudo, mesmo com as restrições impostas pela linguagem escolhida, para os propósitos centrais desta pesquisa, foi possível implementar com êxito tipos como classes de objetos.

De maneira alguma se pretendeu nesta pesquisa entrar no mérito sobre as linguagens ou *scripts* de programação orientada a objetos. O objetivo principal das implementações computacionais desenvolvidas ao longo do trabalho foi de exemplificar o processo do pensamento arquitetônico

de acordo com as principais características encontradas na linguagem utilizada. Por essa razão, o desenvolvimento dos protótipos implementados no *Visual Basic for Application* (VBA) se limitou aos conceitos de classes, objetos, atributos e métodos. Os demais conceitos da orientação a objetos, como herança, encapsulamento e polimorfismo, ficaram a cargo apenas de uma abordagem teórica sobre o processo de projeto arquitetônico.

5 ESTUDOS PRELIMINARES

Para ilustrar o método de projeto sugerido na pesquisa (p. 39), foram desenvolvidos inicialmente dois protótipos preliminares, de acordo com os exemplos propostos por Mitchell (1990) em *The Logic of Architecture.* Os tipos "cadeira" e "ordem dórica" foram definidos pelo autor sob a forma de regras de conhecimento. Para expressar o pensamento sobre os objetos analisados, o autor utilizou *frames*[2] e a lógica de primeira ordem.

Em sua obra *The Logic of Architecture*, Mitchell (1990) utiliza métodos de áreas da ciência da computação, como por exemplo, a inteligência artificial. Esses métodos constituem mecanismos para a representação do conhecimento a respeito de um domínio de maneira clara e eficiente. Apesar da lógica ser - em tese - uma representação suficientemente expressiva para qualquer tipo de conhecimento, autores como Bittencourt (2001) destacam que os problemas de eficiência e a necessidade de expressar um conhecimento incerto e incompleto na linguagem acabam contribuindo para o desenvolvimento de outros tipos de formalismo de representação de conhecimento.

De acordo com Mitchell (1990), os esquemas de classificação do vocabulário arquitetônico não precisam ser rigorosamente fixos, mas devem ser capazes de relatar e de dar nomes aos tipos neles compreendidos, seja por meio de regras de conhecimento, *frames* ou classes de objetos. Dessa forma, com base na analogia teórica estabelecida por Mitchell (1990) entre tipos arquitetônicos e classes de objetos, buscou-se através deste estudo de campo, formalizar e implementar computacionalmente o conhecimento sobre os tipos e tipologias arquitetônicas. Para

---

[2] Consiste em um conjunto de atributos que, através de seus valores, descrevem as características do objeto representado pelo *Frame*. Os valores atribuídos a esses atributos podem ser outros *Frames*, criando uma rede de dependências entre eles. São também organizados em uma hierarquia de especialização, criando uma outra dimensão de dependência entre eles (BITTENCOURT, 2001, p. 264)

isso os exemplos “cadeira” e “templo dórico” foram modelados como classes de objetos e automatizados por meio da linguagem de programação *Visual Basic for Application* (VBA), conforme descrito em detalhes na seqüência.

5.1 O PROTÓTIPO CADEIRA

A escolha do primeiro protótipo para elucidar o método de projeto proposto foi influenciada, em parte, pela simplicidade das variáveis envolvidas na resolução do problema em questão, já que o intuito principal desse estudo inicial consistiu em partir de uma classe bastante simplificada, como é o caso de um tipo cadeira, até se chegar a uma classe mais complexa de objetos. Essa estratégia denominada como método de desenvolvimento incremental, teve em vista a expressão do conhecimento em vários níveis de abstração.

Dessa forma, para definir claramente o processo utilizado na definição e implementação computacional do protótipo cadeira, as etapas do projeto foram estruturadas da seguinte maneira:

Em primeiro lugar foi analisada a descrição de uma classe de objetos do tipo cadeira proposta por Mitchell (1990) em *The Logic of Architecture*.

Em seguida, buscou-se refletir sobre o problema cadeira de uma maneira geral. Assim, algumas classes distintas de objetos puderam ser consideradas, tais como: a cadeira da sala de jantar, a cadeira escolar, a cadeira do escritório, a cadeira de balanço, a cadeira giratória, a cadeira fixa, dentre outros possíveis resultados que atendem de maneira suficientemente expressiva a questão do problema cadeira, conforme pode se visualizado na Figura 5.1.

Figura 5.1. Algumas representações de cadeiras.

Através dessa abstração inicial do problema, tornou-se explícito o fato do tipo cadeira ser na verdade um item geral que se refere a muitos objetos, cada qual com suas características e necessidades particulares. Nesse sentido, para diferenciar os exemplos de objetos pertencentes a uma classe genérica como o item “cadeira”, passou-se a analisar e refletir sobre o tipo em termos de itens gerais e tipos mais específicos desse item, o que em outras palavras, permitiu com que a classe cadeira fosse analisada e refletida sob a especificação/ generalização de tipos de objetos. Dessa forma, tornou-se possível definir que o tipo cadeira pode perfeitamente se desdobrar em tipos específicos de cadeira, tais como: cadeira fixa, cadeira de balanço, cadeira giratória e outras possibilidades de classificações.

Para auxiliar a formalização do pensamento envolvido no estudo da tipologia cadeira, foi utilizado a sintaxe “é um tipo de”, muito comum ao relacionamento de especificação/ generalização de classes de objetos. Assim, a estruturação hierárquica das classes envolvidas no estudo foi organizada de acordo com o seguinte preceito:

- A cadeira fixa “é um tipo de” cadeira.
    - A cadeira fixa de metal “é um tipo de” cadeira.
    - A cadeira fixa de metal com estrutura redonda “é um tipo de” cadeira.

A partir do momento em que o conceito sobre “cadeira” foi classificado como a denominação de um grupo, isto é, uma classe: a classe cadeira; tornou-se possível definir e estruturar com precisão alguns dos subtipos da classe cadeira, conforme ilustra a Figura 5.2.

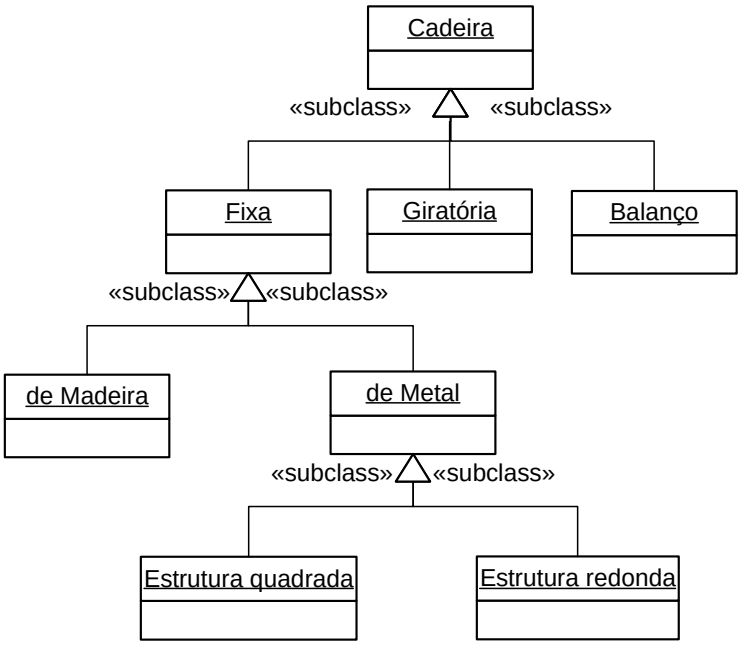


Figura 5.2. Representação hierárquica de alguns tipos associados à classe cadeira.

É importante ter em mente que a proposta aqui apresentada não se trata de uma solução rígida ou definitiva para o problema em discussão. Na verdade, todas e quaisquer etapas do processo de projeto desta pesquisa estão diretamente condicionadas aos critérios e anseios dos envolvidos na resolução do problema em questão. O método de projeto discutido na pesquisa, que é de caráter inteiramente subjetivo, pode tratar uma mesma realidade sob ponto de vistas completamente diferentes.

Assim, são diversas as maneiras de se organizar e classificar o pensamento em termos de classes de objetos. No caso do problema “cadeira”, por exemplo, a classe foi classificada levando em consideração requisitos como modelos (fixa, giratória e balanço) e materiais (de madeira e de metal) como ilustra a Figura 5.2. Contudo o mesmo problema “cadeira” poderia ser abordado com um nível maior de abstração, isto é, com a classificação do item geral “cadeira” em termos de funcionalidade (de sala de jantar, de escritório, de cozinha), modelo (fixa) e material (de madeira, de metal) como representa a estrutura hierárquica definida na Figura 5.3. Esses diferentes enfoques são, na verdade, partes da intenção projetual do arquiteto quando pensa o projeto. Portanto, fica claro que não existe uma regra universalmente correta no que diz respeito

às possibilidades de classificação e raciocínio envolvido no problema. A interpretação de um dado problema é determinada de acordo com diferentes interesses.

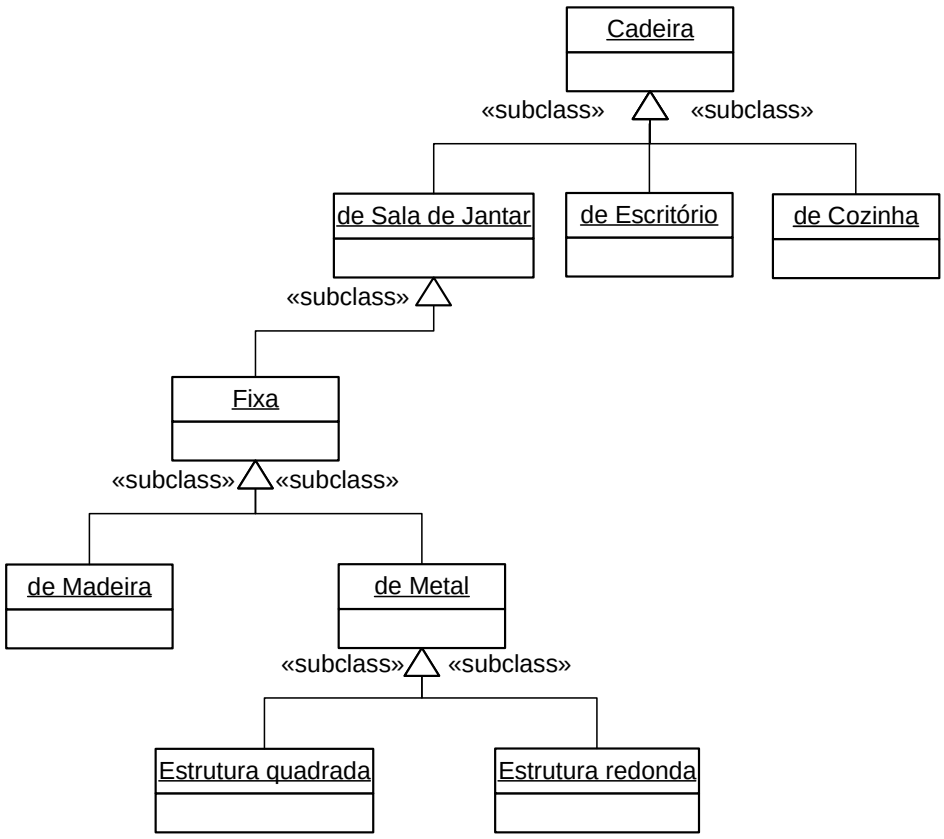


Figura 5.3. Outra representação hierárquica de tipos associados à classe cadeira.

Talvez ainda possa parecer necessária, a definição de uma classe que fosse realmente geral, isto é, uma classe abstrata que a partir desta fosse possível derivar a classe “cadeira”, conforme ilustra a Figura 5.4. Isto não estaria errado, no entanto, para os propósitos do trabalho, é através da classe cadeira que todas as características do tipo serão atribuídas as outras classes, assim além de definir as características de uma classe cadeira, a classe servirá como classe geral a partir da qual serão derivadas classes especificas de objetos como uma cadeira fixa, giratória e balanço, por exemplo.

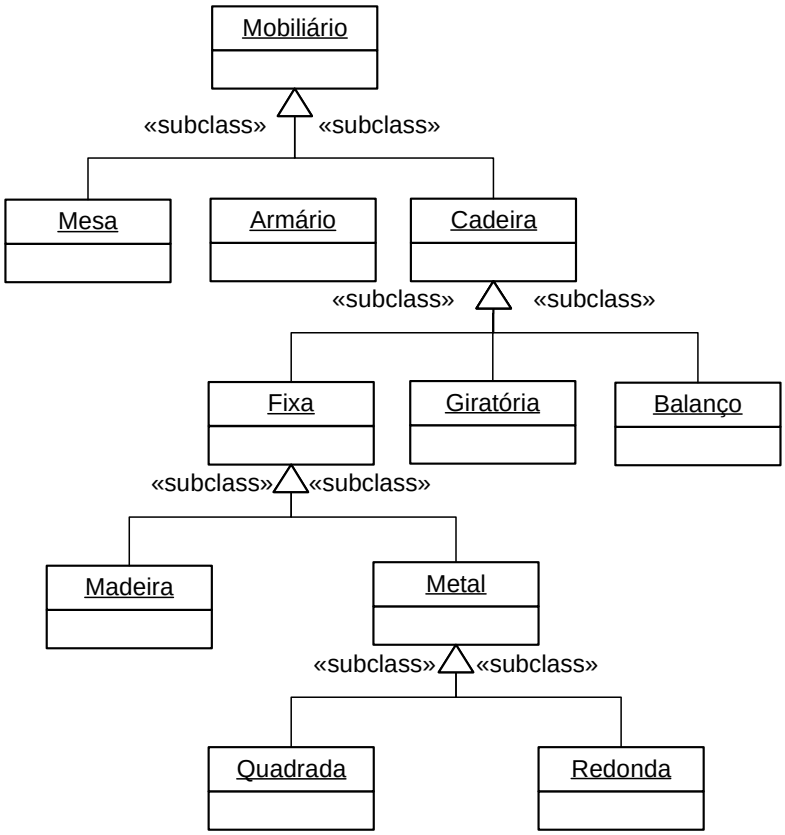


Figura 5.4. Representação hierárquica de alguns tipos associados à classe mobiliário.

Assim, uma vez que as informações relacionadas ao tipo “cadeira” foram organizadas e estruturadas na modelagem[3] formal do problema, tornou-se possível ter uma visão global acerca do problema estudado e dessa maneira, determinar uma solução adequada para a implementação do projeto computacional proposto no escopo da pesquisa. A Figura 5.5 ilustra a seleção de um tipo específico de cadeira após o estudo do problema em partes menores de soluções.

[3] Atividade de construir modelos que expliquem características de um sistema

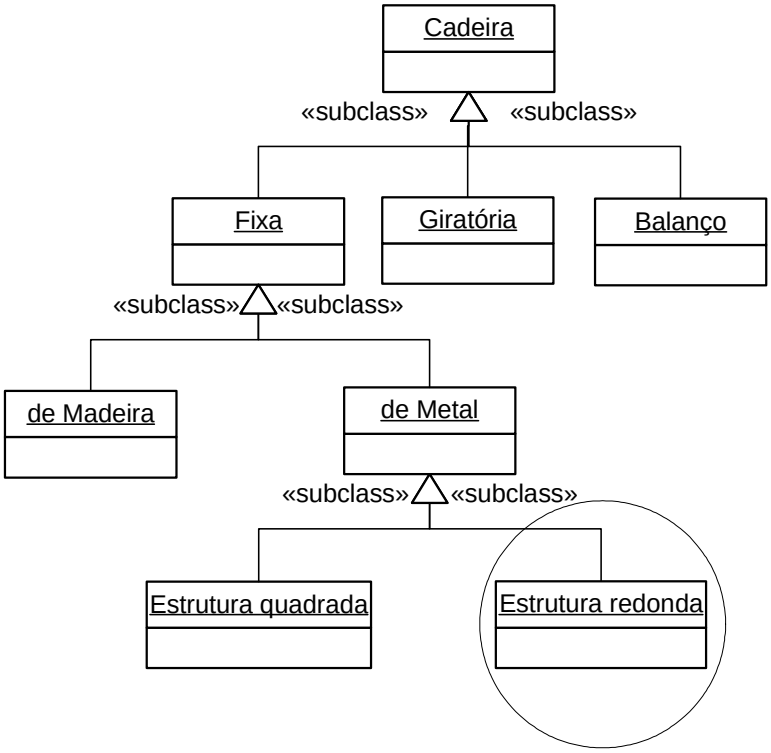


Figura 5.5. Definição de um tipo específico de cadeira a ser implementado no computador como classe de objetos.

Em seguida, buscou-se definir e estruturar as propriedades essenciais para a construção do tipo cadeira. Para esse propósito foram realizados dois passos importantes. A primeira ação consistiu na definição dos objetos comuns a todos os tipos cadeira - isto é assento, encosto e pernas - seguindo as definições teórico-filosóficas propostas por Mitchell (1990) em *The Logic of Architecture*, e a segunda ação consistiu na utilização de um relacionamento da orientação a objetos conhecido como agregação. Através da utilização dessa estratégia tornou-se possível estruturar os objetos essenciais à composição do tipo cadeira e determinar o nível de dependência que a tipologia analisada possui com os demais objetos envolvidos no relacionamento, conforme ilustra a Figura 5.6.

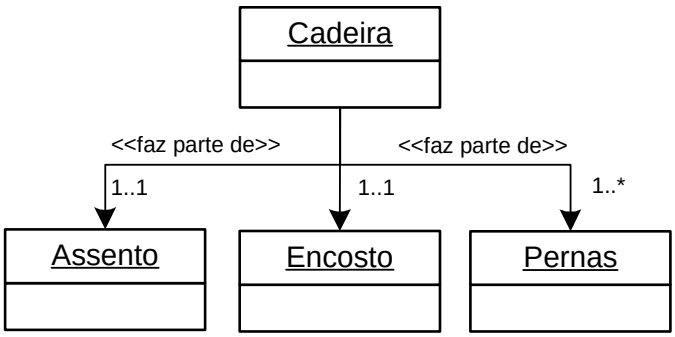


Figura 5.6. Propriedades essenciais à composição do tipo cadeira.

Essa abstração do problema em nível de objetos possibilitou que o relacionamento “faz parte de” fosse utilizado para a estruturação clara e precisa dos objetos de composição do tipo cadeira, pemitindo conseqüentemente, reconhecer a tipologia estudada como um item maior – o “todo” – que é formado por itens menores de objetos – as “partes”.

A dependência existente entre a tipologia estudada e os seus objetos essenciais de composição se configuraram da seguinte maneira:

- O tipo cadeira possui um e somente um assento;
- O tipo cadeira possui um e somente um encosto;
- O tipo cadeira possui no mínimo uma e no máximo muitas pernas.

Em resumo, através do estudo das propriedades essenciais do tipo, tornou-se possível compreender que uma cadeira nas suas mais diferenciadas instâncias de objetos possuirá no mínimo um assento, um encosto e uma perna. Esses objetos correspondem à essência da tipologia, portanto são comuns a todas as instâncias do objeto.

A etapa seguinte consistiu na definição das propriedades acidentais de um tipo cadeira. Esse estágio, não se propôs a definir a essência de uma cadeira, portanto tornou-se perfeitamente possível introduzir nesta etapa do processo, características variáveis entre os exemplares da mesma tipologia.

Ao realizar um estudo exploratório acerca do tipo “cadeira”, foi possível encontrar na literatura a seguinte definição:

> Cadeira é assento provido de um encosto, destinado a um descanso, mas não a um repouso. Assim, a cadeira nada mais é que um banco, um moncho ou tamborete com um espaldar onde se pode repousar as costas. Nada impede, no entanto, que esse tamborete provido de encosto às vezes também possua braços; mas esses braços não significam nenhum acréscimo ao fim a que a cadeira se destina, que é o descanso. Ela cumpre sua função, com ou sem braços. O braço talvez seja apenas um acessório decorativo, ou mesmo, um elemento que dá status à cadeira. (GALLI, 1988).

Através dessa abordagem geral do problema, foi possível confirmar as definições técnicas de propriedades acidentais pospostas por Mitchell (1990) em *The Logic of Architecture* e dessa maneira, afirmar que o tipo cadeira pode perfeitamente possuir características como braços, além de outras como estofamento, pintura e rodinhas - por exemplo - que não alterarão em nada a essência da tipologia. Se por alguma razão o tipo cadeira precisasse ser modificado a ponto de perder a sua sustentação como, por exemplo, com a remoção de suas pernas, o tipo então deixaria de ser uma cadeira e passaria a ser um balanço. A essência do tipo cadeira não estaria preservada neste caso.

Para ilustrar algumas das propriedades acidentais pertencentes a um tipo cadeira utilizou-se, igualmente à etapa anterior, o relacionamento da orientação a objetos conhecido como agregação. Esse mecanismo permitiu estruturar os objetos acidentais que compõem o tipo cadeira e determinar o nível de dependência que a tipologia analisada possui com os demais objetos envolvidos no relacionamento, conforme ilustra a Figura 5.7.

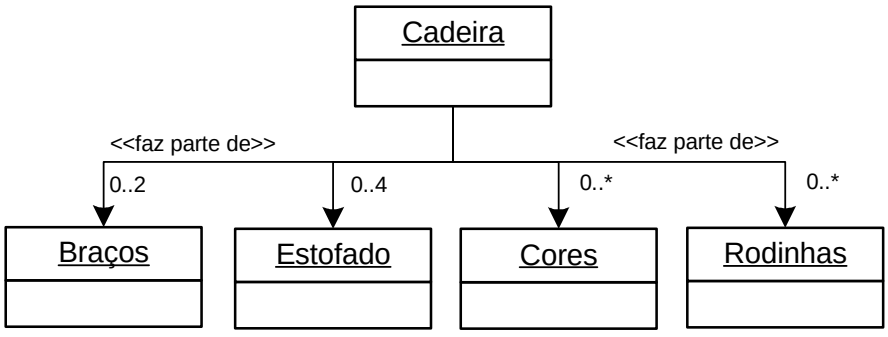


Figura 5.7. Propriedades acidentais à composição do tipo cadeira.

Assim, nos que diz respeito às propriedades acidentais de um tipo cadeira em geral, a dependência entre os objetos configurou-se da a seguinte:

- O tipo cadeira possui no mínimo zero (nenhum) e no máximo dois braços;
- O tipo cadeira possui no mínimo zero (nenhum) e no máximo quatro estofados, considerando estofamento para cada assento, encosto e braços;
- O tipo cadeira possui no mínimo zero (nenhum) e no máximo muitas cores.
- O tipo cadeira possui no mínimo zero (nenhum) e no máximo muitas rodinhas.

O estudo das propriedades acidentais da tipologia, assim com das suas propriedades essenciais, além de possibilitar atingir um amadurecimento teórico no que diz respeito ao conceito de objetos, veio proporcionar uma visão ampla e abrangente da tipologia de partes menores de soluções. Através do relacionamento de agregação que se utilizada da sintaxe “faz parte de”, tornou-se possível estruturar de maneira clara e coerente todos os objetos de composição da tipologia em questão, sejam estes propriedades acidentais ou propriedades essenciais.

Para realizar a construção do tipo cadeira como uma classe de objetos, buscou-se através da representação do conhecimento desenvolvido nos estágios anteriores, o subsídio necessário para implementar computacionalmente a tipologia estudada.

Por motivos de simplificação, foi decidido limitar as alternativas referentes ao material e à rigidez da estrutura, de modo a facilitar a implementação, que é capaz de gerar apenas cadeiras do tipo fixa com estrutura de metal redonda. Alguns objetos de composição da tipologia foram definidos de acordo com as propriedades essenciais e acidentais discutidas na quinta e sexta etapas do processo de projeto. A Figura 5.8 apresenta uma modelagem formal dos elementos abordados para a implementação de uma cadeira do tipo fixa com estrutura de metal redonda como se ela fosse classe de objetos.

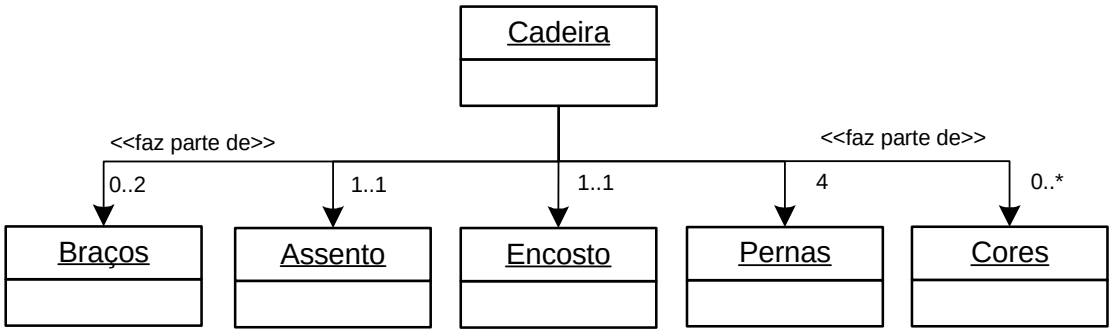


Figura 5.8. Modelagem formal do tipo cadeira implementado como classe de objetos.

Dessa maneira, a classe cadeira implementada no computador definiu-se de acordo com as seguintes características:

- O tipo cadeira possui no mínimo zero (nenhum) e no máximo dois braços;
- O tipo cadeira possui um e somente um assento;
- O tipo cadeira possui um e somente um encosto;
- O tipo cadeira possui quatro pernas;
- O tipo cadeira possui no mínimo zero (nenhum) e no máximo muitas cores.

Todo esse processo desenvolvido no estudo do protótipo inicial cadeira, além de ter possibilitado atingir uma compreensão aprofundada do problema em questão, permitiu formalizar e documentar todo o pensamento desenvolvido para a solução da tipologia. Através do processo de projeto desempenhado tornou-se possível transladar com coerência as informações do tipo cadeira abordado para a linguagem de programação *Visual Basic for Application* (VBA), e dessa maneira estruturar com precisão as variáveis do problema em termos de classe de objetos.

A Figura 5.9 ilustra um esquema geral da classe cadeira desenvolvida no *Visual Basic for Application* (VBA), representado no sistema conhecido como UML[4]. Todas as informações

[4] A UML (*Unified Modeling Language*) é uma linguagem para especificação, documentação, visualização e desenvolvimento de sistemas orientados a objetos. Sintetiza os principais métodos existentes, sendo considerada uma

relacionadas ao processo de projeto da classe cadeira estão documentadas e podem ser visualizadas no Apêndice A desta dissertação.

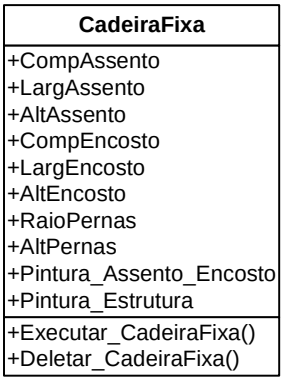


Figura 5.9. Atributos e métodos da classe cadeira implementada no computador.

Uma classe cadeira, representada pela linguagem UML, é simbolizada como um retângulo com até três divisões, descritas a seguir:

- A primeira divisão contém o nome da classe, que neste exemplo é CadeiraFixa;
- A segunda divisão armazena os atributos da classe, neste caso a classe CadeiraFixa contém atributos como: comprimento, largura e altura do assento, comprimento, largura e altura do encosto, raio e altura das pernas da cadeira, cor do assento, encosto e estrutura da cadeira;
- Finalmente, a terceira divisão lista os métodos da classe, neste exemplo a classe CadeiaFixa contém os métodos executar e limpar os objeto CadeiraFixa.

Com a definição e implementação dos atributos e métodos do protótipo computacional, tornou-se possível produzir diferentes instâncias da classe cadeira. Essas instâncias da tipologia permitem variações em algumas características de seus objetos como, por exemplo, as dimensões de

---

das linguagens mais expressivas para modelagem de sistemas orientados a objetos. Por meio de seus diagramas é possível representar sistemas de softwares sob diversas perspectivas de visualização. Facilita a comunicação de todas as pessoas envolvidas no processo de desenvolvimento de um sistema por apresentar um vocabulário de fácil entendimento.

assento e encosto, além de permitir a atribuição de propriedades acidentais ao objeto cadeira como, por exemplo, a definição de cores e braços. A Figura 5.10 mostra algumas instâncias do tipo cadeira obtidas através do protótipo computacional desenvolvido.

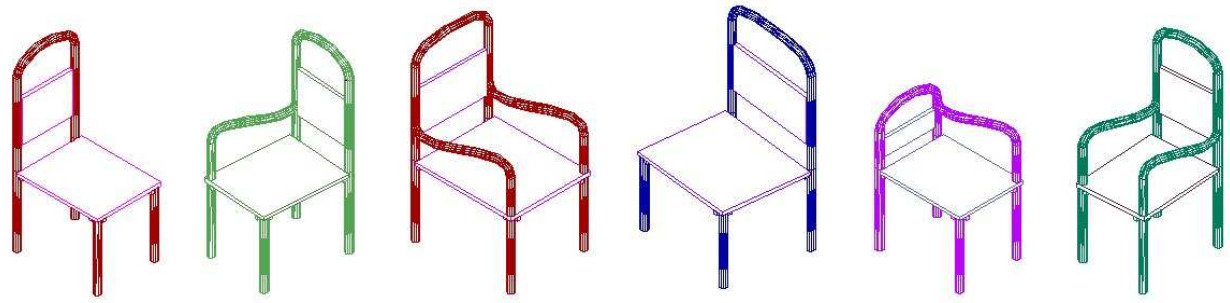


Figura 5.10. Instâncias do tipo cadeira fixa.

A Figura 5.11 apresenta a interface da classe cadeira desenvolvida como estudo preliminar.

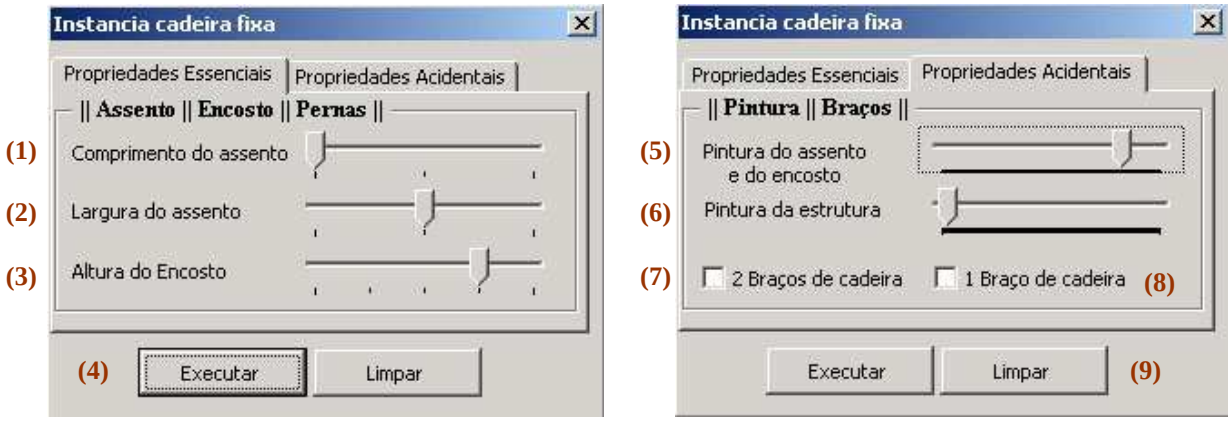


Figura 5.11. Interfaces da classe cadeira fixa implementada.

Através dos botões e menus acima o usuário do protótipo pode alterar alguns atributos que correspondem às propriedades essenciais e acidentais da classe cadeira fixa, como ilustra a descrição do protótipo abaixo:

(1) Define o comprimento do assento da cadeira fixa que é igual ao valor do encosto.

(2) Define a largura do assento da cadeira fixa.

(3) Define a altura do encosto da cadeira fixa.

(4) Instancia a classe cadeira fixa.

(5) Define a pintura do assento e do encosto da cadeira fixa.

(6) Define a pintura da estrutura da cadeira fixa.

(7) Pode instanciar a classe cadeira fixa com dois braços.

(8) Pode instanciar a classe cadeira fixa apenas com um braço.

(9) Apaga todos os objetos presentes no *Model Space* do AutoCAD.

Finalmente, a última etapa do processo de projeto consistiria na sua construção no mundo real, a partir de sua representação projetual. No caso do protótipo de cadeira, cuja representação consiste em um modelo tridimensional virtual, essa construção poderia se dar por meios tradicionais ou por meios automatizados, com o uso de máquinas de controle numérico, em um processo conhecido como “fabricação rápida”.

5.1.1 RESULTADOS PARCIAIS: ANÁLISE DO PROCESSO DE DESENVOLVIMENTO DO PRIMEIRO PROTÓTIPO

De maneira geral, pretendeu-se aqui neste estágio da pesquisa atingir um grau de maturidade inicial, porém bastante consistente, em relação ao uso de técnicas de modelagem orientada a objeto e suas aplicações diretas no auxílio ao processo do projeto arquitetônico.

Em nenhum momento o processo de projeto desenvolvido ao longo desta pesquisa visou discutir sobre questões como originalidade projetiva; o intuito principal foi de explorar o pensamento sobre um projeto em termos de tipos e tipologias. Por isso, por mais simples que possa parecer a tipologia inicial abordada, todo e qualquer tipo deve ser modelado antes de se iniciar a sua construção, entre outras coisas, por que as informações do projeto de arquitetura freqüentemente costumam aumentar de tamanho, complexidade e abrangência. Esta forma de abordagem do processo de projeto pode ser particularmente interessante quando se trabalha com variações sobre um mesmo tema, ou seja, tipos de edifícios que apresentam diversas características comuns, mas ao mesmo tempo precisam se adequar a diferentes terrenos, clientes, orçamentos e outras condições.

5.2 O PROTÓTIPO ORDEM DÓRICA

Tendo ainda em vista o método de desenvolvimento incremental, buscou-se através do segundo protótipo, abordar um número maior de variáveis no processo de resolução do problema de projeto. Essa estratégia, além de ter proporcionado fortalecer o estudo do método de projeto sugerido na pesquisa, permitiu explorar de maneira favorável um problema em termos de um posicionamento relativista, isto é, o estudo de uma tipologia - aparentemente imutável - sob diferentes convenções de abstração. O desenvolvimento do segundo protótipo se deu nas seguintes etapas:

Inicialmente, estudou-se a proposta de definição de um tipo “ordem dórica” proposta por Mitchell (1990) em *The Logic of Architecture.*.

Em seguida, buscou-se investigar e refletir sobre o problema da “ordem dórica” de uma maneira geral. Dessa forma, através do estudo inicial que foi desenvolvido a respeito da tipologia em questão, tornou-se possível compreender que mesmo um tipo aparentemente bem definido como o templo dórico possui diferentes versões para a sua ordem arquitetônica. Esses diversos tipos de “ordem dórica” consideram em geral questões como: o número de colunas na fachada,

distribuição das colunas, espaçamento de colunas/ intercolúnio e disposição de tríglifos e métopas, conforme exemplifica a Figura 5.12.

Figura 5.12. Algumas representações de ordens dóricas.

Fonte – Mitchell, 1990.

Nesse sentido, a partir dessa abstração inicial do problema, ficou explícita a possibilidade da "ordem dórica" variar consideravelmente entre os exemplos da mesma tipologia. Esse posicionamento relativista encontrado no item "ordem dórica" é capaz de conduzir o arquiteto a diversas linhas de raciocínio, em que características distintas da tipologia são consideradas, mas nenhuma delas única e universalmente corretas.

Portanto, para diferenciar alguns dos tipos de objetos pertencentes a uma classe relativista como o item "ordem dórica", optou-se por analisar e refletir sobre o tipo apenas em termos de espaçamentos de colunas/ intercolúnio. Assim, tornou-se possível desdobrar tipo "ordem dórica" em subtipos como "diastilo" e "sistilo". Essa abstração do problema foi influenciada pelas idéias de essências relativas apresentadas por Mitchell (1990) em *The Logic of Architecture* e enriquecida com informações provenientes da obra *Ten Books on Architecture* (1999) de Vitruvius.

Para realizar a formalização do pensamento envolvido no estudo da tipologia "ordem dórica" foi utilizado, de maneira análoga ao primeiro protótipo desenvolvido nesta pesquisa, a sintaxe "é um tipo de" que é muito comum ao relacionamento de especificação/ generalização de classes de objetos. Assim, a estruturação hierárquica das classes envolvidas no estudo foi organizada de acordo com o seguinte preceito:

- O pórtico diastilo "é um tipo de" ordem dórica.
- O pórtico sistilo "é um tipo de" ordem dórica.

A partir do momento em que o item “ordem dórica” foi classificado como a uma classe: a classe “ordem dórica”, tornou-se possível definir e estruturar alguns dos subtipos da classe em termos de espaçamento entre as colunas, conforme ilustra a Figura 5.13.

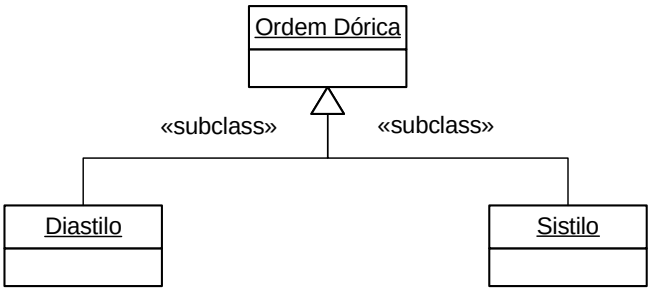


Figura 5.13. Representação hierárquica de alguns tipos de pórticos associados à classe ordem dórica.

É claro que o problema “ordem dórica” poderia se desdobrar inicialmente em números de colunas e depois em espaçamento de colunas, conforme exemplifica a Figura 5.14. Isto não estaria errado, no entanto, como os números das colunas dos pórticos diastilo e sistilo foram pré-estabelecidos por Vitruvius em *Ten Books on Architecture* (1999), optou-se por trabalhar com os objetos colunas em termos de propriedades essenciais e não como subtipos da classe “ordem dórica”, conforme discutido mais à frente na quinta etapa deste processo de projeto.

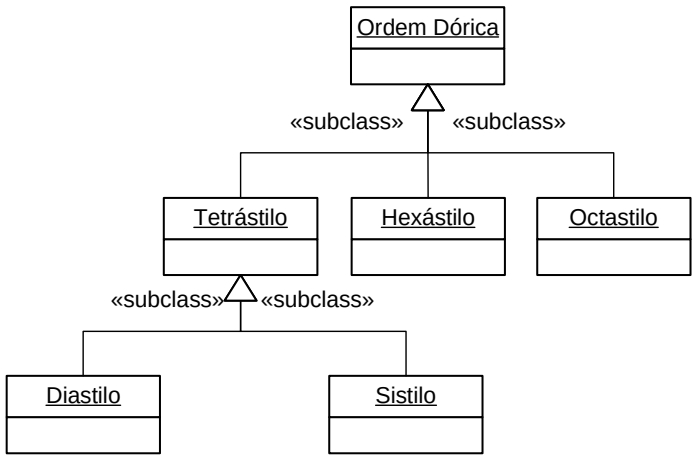


Figura 5.14. Outra representação hierárquica de alguns tipos de pórticos associados à classe ordem dórica.

Uma vez que as informações relacionadas ao tipo “ordem dórica” foram organizadas e estruturadas na modelagem formal do problema, tornou-se possível ter uma visão global acerca do problema em questão e, dessa maneira, determinar uma solução adequada para a implementação do projeto computacional proposto no escopo da pesquisa. A Figura 5.15 ilustra os dois tipos específicos de “ordem dórica” após o estudo do problema em partes menores de soluções: o subtipo diastilo, cuja característica é possuir um intercolúnio com medida de três diâmetros da coluna da “ordem dórica”, e o subtipo sistilo que possui intercolúnio com medida de dois diâmetros de coluna da “ordem dórica”.

Como a “ordem dórica” em geral possui algumas variações entre os exemplos de uma mesma tipologia, optou-se nesta etapa do processo de projeto por selecionar ambos os subtipos da classe “ordem dórica” para serem implementados computacionalmente como classes de objetos, conforme mostra a Figura 5.15. Essa decisão além de ter possibilitado explorar uma variação significativa de subtipos da “ordem dórica”, veio ressaltar sobre quão variáveis ou invariáveis são as ordens arquitetônicas do tipo dórica.

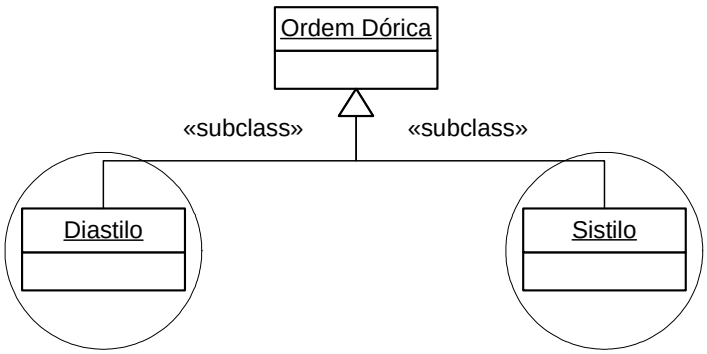


Figura 5.15. Definição dos tipos específicos de ordem dórica a serem implementados no computador como classe de objetos.

No que diz respeito aos elementos de composição da tipologia estudada, buscou-se através desta etapa do processo de projeto definir e estruturar as propriedades essenciais para a construção de um tipo “ordem dórica”. Como o estilo arquitetônico analisado costuma variar significativamente entre os exemplos da mesma tipologia, não seria possível enumerar neste estudo todas as

propriedades comuns a uma tipologia “ordem dórica” em geral. Nesse sentido, por motivos de simplificação, optou-se nesta etapa do processo de projeto por analisar apenas as principais propriedades essenciais de dois subtipos da “ordem dórica”, já que eles possuem objetos comuns na sua composição, ou seja, foram abordadas as características dos subtipos “diastilo” e “sistilo” respectivamente.

Diante desse contexto, para conseguir definir e organizar as propriedades das tipologias selecionadas, o processo de projeto se desmembrou em dois momentos importantes. O primeiro momento consistiu na definição dos objetos comuns aos subtipos da “ordem dórica” - “diastilo” e “sistilo” - que foi estabelecida de acordo com os relatos de Vitruvius apresentados em *Ten Books on Architecture* (1999). Uma vez que todos os objetos essenciais à formação das tipologias foram analisados, o segundo momento do processo de projeto consistiu na organização formal desses objetos através de um relacionamento da orientação a objetos que é conhecido como agregação. Essa estratégia além de ter possibilitado formalizar o raciocínio envolvido na abstração dos subtipos da “ordem dórica” em termos de objeto, permitiu determinar o nível de dependência que os subtipos analisados possuem com os demais objetos envolvidos no relacionamento. As Figuras 5.16 e 5.17 ilustram a estrutura hierárquica dos objetos pertencentes aos subtipos da “ordem dórica” diastilo e sistilo respectivamente.

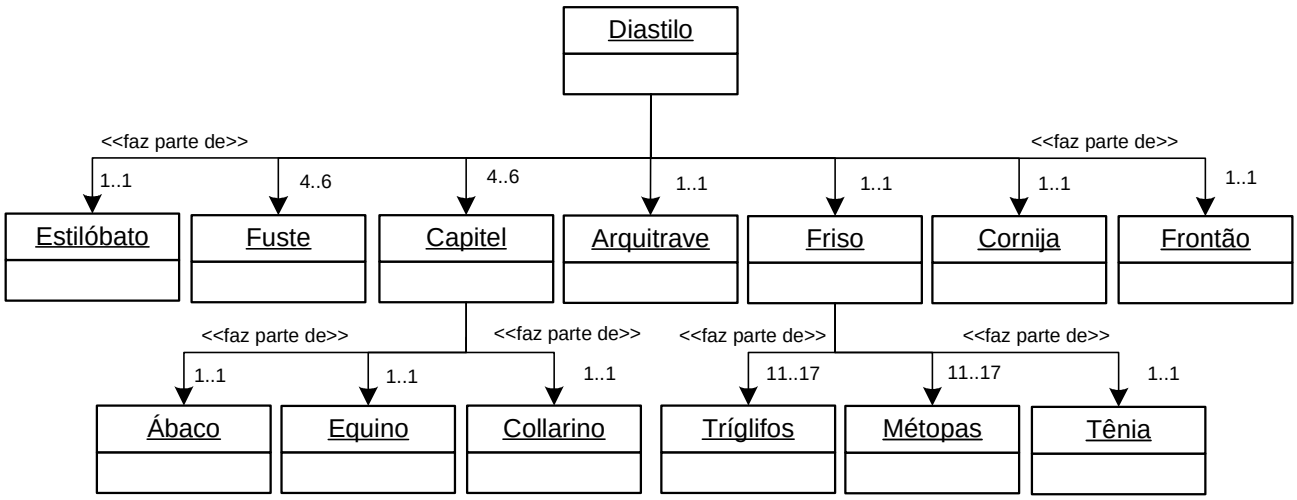


Figura 5.16. Propriedades essenciais à composição do subtipo da “ordem dórica” diastilo.

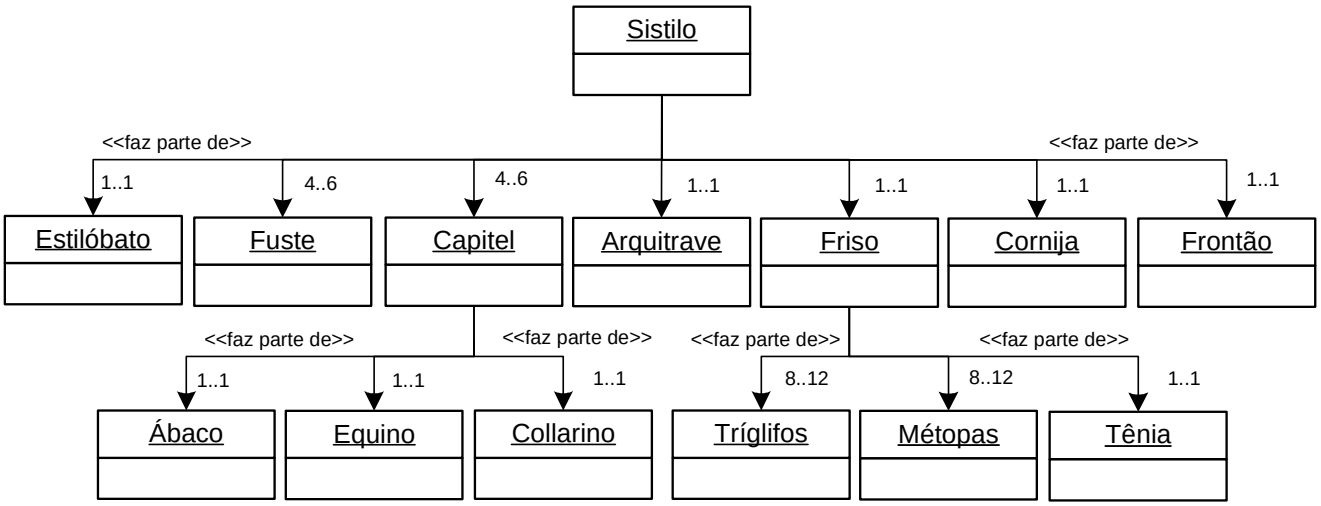


Figura 5.17. Propriedades essenciais à composição do subtipo da “ordem dórica” sistilo.

Dessa forma, ao definir a dependência das tipologias analisadas e os seus objetos essenciais de composição, tornou-se possível encontrar a seguinte configuração:

- A “ordem dórica” diastilo e sistilo possuem no mínimo um e no máximo um Estilóbato;
- A “ordem dórica” diastilo e sistilo possuem no mínimo quatro e no máximo seis Fuste;
- A “ordem dórica” diastilo e sistilo possuem no mínimo quatro e no máximo seis Capitel;
- A “ordem dórica” diastilo e sistilo possuem no mínimo uma e no máximo uma Arquitrave;
- A “ordem dórica” diastilo e sistilo possuem no mínimo um e no máximo um Friso;
- A “ordem dórica” diastilo e sistilo possuem no mínimo uma e no máximo uma cornija;
- A “ordem dórica” diastilo e sistilo possuem no mínimo um e no máximo um Frontão;

A principal diferença entre os pórticos diastilo e sistilo encontrou-se no número de triglífos e métopas dispostos na tipologia “ordem dórica”. Como pode ser observado através das Figuras 5.16 e 5.17, cada objeto da classe friso relacionou-se a no mínimo 11 e no máximo 17 tríglifos e métopas, no caso da “ordem dórica” do tipo diastilo, e no mínimo a 8 e no máximo 12 tríglifos e métopas, no caso de “ordem dórica” do tipo sistilo. Através dessa abstração formal das propriedades pertencentes às tipologias, tornou-se possível documentar e validar informações, impedindo dessa forma que as “ordens dóricas” do tipo diastilo e sistilo fossem construídas sem ter o número exato de tríglifos e métopas para a sua composição.

Em resumo, através do estudo das propriedades essenciais das tipologias, tornou-se possível enxergar a “ordem dórica” - em geral - como uma classe não universal. A interpretação da tipologia está diretamente relacionada aos interesses particulares dos envolvidos na resolução do problema em questão, portanto não existem propriedades inteiramente corretas ou únicas, na verdade, são diferentes as maneiras e alternativas para a solução do mesmo problema em discussão.

Em relação às propriedades acidentais dos subtipos da “ordem dórica”, concluiu-se neste estágio que não seria coerente atribuir novos elementos à composição das tipologias, haja vista que o problema em questão possui um caráter estritamente relativista e foi procurado na etapa anterior do processo de projeto simplificar ao máximo os objetos de composição das tipologias em discussão. Os elementos definidos como propriedades essenciais para as tipologias são na verdade de grande importância para a simetria bilateral da “ordem dórica” em geral, qualquer modificação ou remoção desses objetos poderia alterar o tipo a tal ponto de perder a essência do objeto. Portanto, por motivos de relevância, optou-se apenas por atribuir cores às tipologias estudadas. Sendo esta a única propriedade acidental definida no problema em discussão.

A Figura 5.18 ilustra a propriedade acidental que compõe os subtipos da “ordem dórica” diastilo e sistilo respectivamente, e o nível de dependência que as tipologias analisadas possuem com o objeto envolvido no relacionamento.

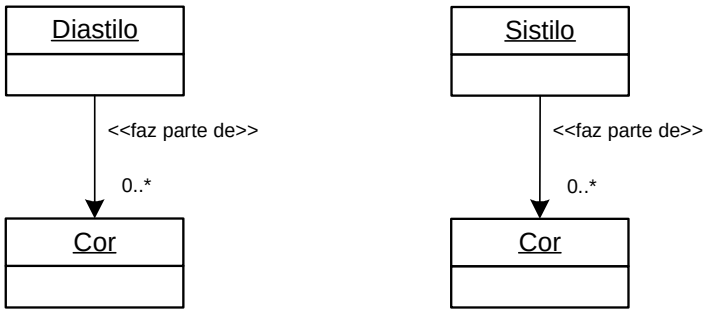


Figura 5.18. Propriedades acidentais à composição os subtipos da "ordem dórica" diastilo e sistilo respectivamente.

Dessa maneira, no que diz respeito às propriedades acidentais das tipologias, a dependência entre os objetos configurou-se da a seguinte:

- A "ordem dórica" diastilo e sistilo possuem no mínimo zero (nenhum) e no máximo muitas cores.

Para realizar a construção de um tipo "ordem dórica" como classe de objeto, buscou-se através da representação do conhecimento desenvolvido nos estágios anteriores, o auxílio necessário para implementar computacionalmente a tipologia estudada. Diante desse contexto, por motivos de simplificação, foi decidido limitar os subtipos de composição da "ordem dórica", de modo a facilitar a implementação, que é capaz de gerar apenas pórticos do tipo diastilo e sistilo. Esses subtipos da "ordem dórica" foram transcritos para a linguagem de programação *Visual Basic for Application* (VBA) de acordo com as propriedades essenciais e acidentais discutidas na quinta e sexta etapas deste processo de projeto, e estruturados com precisão em termos de classes de objetos.

A Figura 5.19 ilustra um esquema geral das classes diastilo e sistilo desenvolvidas no *Visual Basic for Application* (VBA), representadas no sistema conhecido como UML. Todas as informações relacionadas ao processo de projeto das classes desenvolvidas estão documentadas e podem ser visualizadas no Apêndice A desta dissertação.

Figura 5.19. Atributos e métodos da ordem dórica diastilo e sistilo implementada no computador.

A implementação dos subtipos da “ordem dórica” como classes de objetos foi inteiramente influenciada pelos relatos feitos por Vitruvius em *Ten Books on Architecture* (1999). Nessa obra o autor descreve com grande precisão alguns procedimentos algorítmicos para a construção da “ordens dórica” e detalha minuciosamente os elementos de composição das tipologias diastilo e sistilo investigadas neste estudo de campo.

Dessa maneira, com base nas idéias apresentadas por Vitruvius em *Ten Books on Architecture* (1999), tornou-se possível utilizar na implementação computacional das classes, o método de proporção desenvolvido pelo autor e, conseqüentemente, estruturar de maneira inteiramente congruente a distribuição de todos elementos arquitetônicos da “ordem dórica” diastilo e sistilo. Assim, a grande maioria dos atributos definidos nas classes diastilo e sistilo, conforme é possível observar através da Figura 5.19, refere-se diretamente a características como: altura e largura dos elementos arquitetônicos, módulo de proporção dos elementos arquitetônicos, espaçamento entre as colunas, número de colunas e outros atributos importantes para a resolução do problema da “ordem dórica”.

O cálculo de todos os atributos estipulados nas classes diastilo e sistilo foram implementados de acordo com malha de proporção desenvolvida por Vitruvius, no intuito de atingir a simetria bilateral de todos os objetos de composição da “ordem dórica” diastilo e sistilo. A Figura 5.20 e a Figura 5.21ilustram a malha de divisão das tipologias analisadas em pequenas partes solução que é nomeada pelo autor como conhecida como módulos.

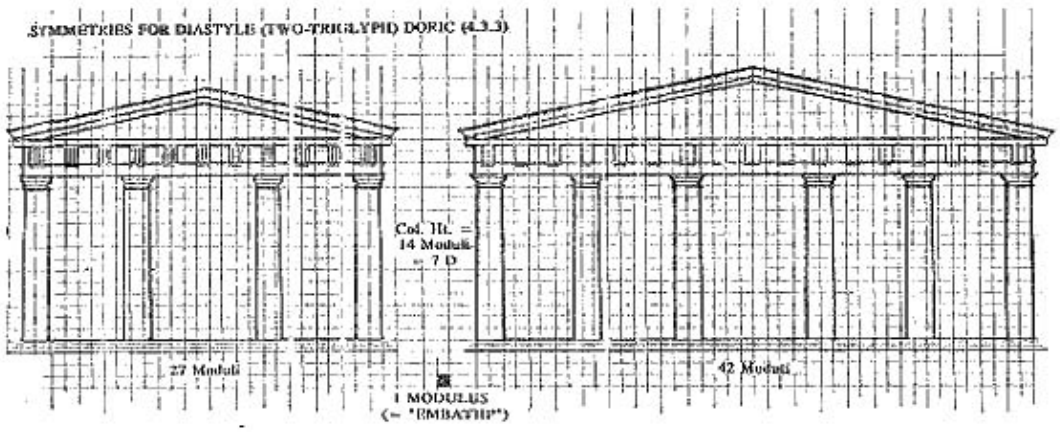


Figura 5.20. Módulos de proporção da “ordem dórica” diastilo com 4 e 6 colunas respectivamente.

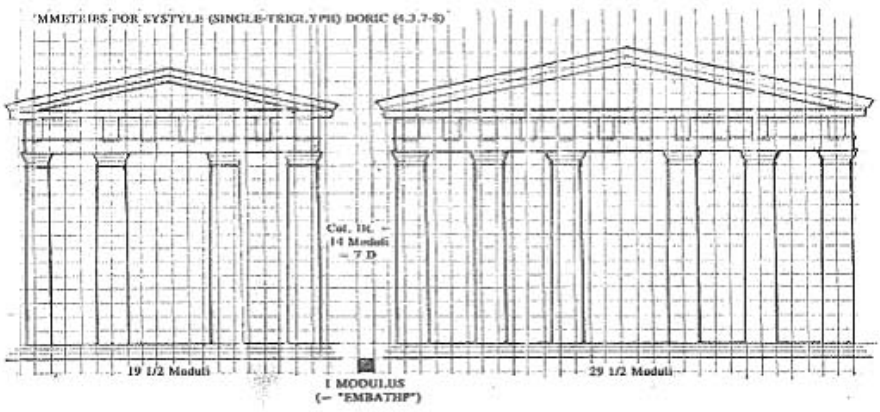


Figura 5.21. Módulos de proporção da “ordem dórica” sistilo com 4 e 6 colunas respectivamente.

Uma vez que o módulo de proporção de todos os elementos arquitetônico da “ordem dórica” pôde ser enumerado, todo o cálculo de simetria das tipologias foi realizado, permitindo dessa maneira, estabelecer uma perfeita harmonia na distribuição dos tríglifos e métopas que é uma característica marcante das ordens dóricas de Vitruvius.

Assim, com a definição e implementação de todas as informações necessárias para a construção do protótipo computacional, tornou-se possível alcançar os exemplos de “ordem dórica” diastilo e sistilo defendidos por Vitruvius em *Ten Books on Architecture* (1999). A Figura 5.22 e a Figura 5.23 ilustram os resultados de instanciamento de classes de objetos como o pórtico diastilo e sistilo respectivamente.

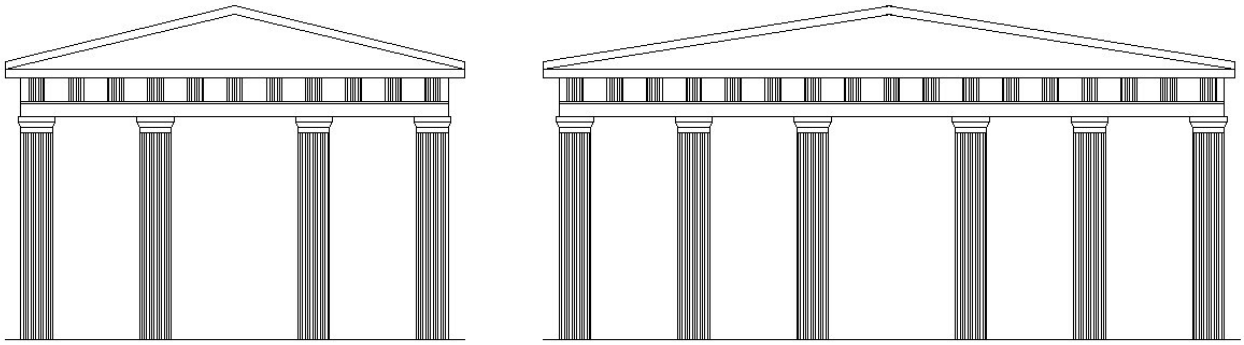


Figura 5.22. Instanciamento da “ordem dórica” diastilo com 4 e 6 colunas respectivamente.

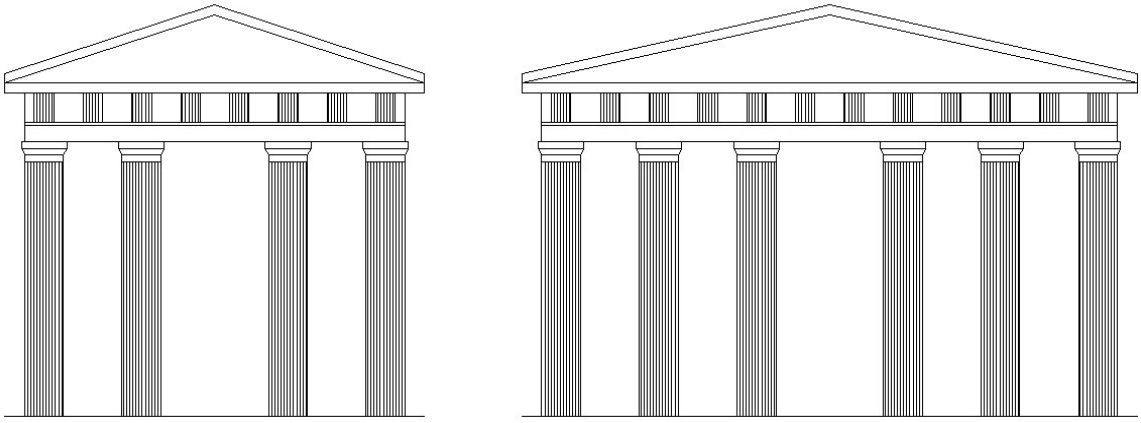


Figura 5.23. Instanciamento da “ordem dórica” sistilo com 4 e 6 colunas respectivamente.

A Figura 5.24 mostra interface do protótipo “ordem dórica” desenvolvida no *Visual Basic for Applications* (VBA), com os subtipos diastilo e sistilo respectivamente.

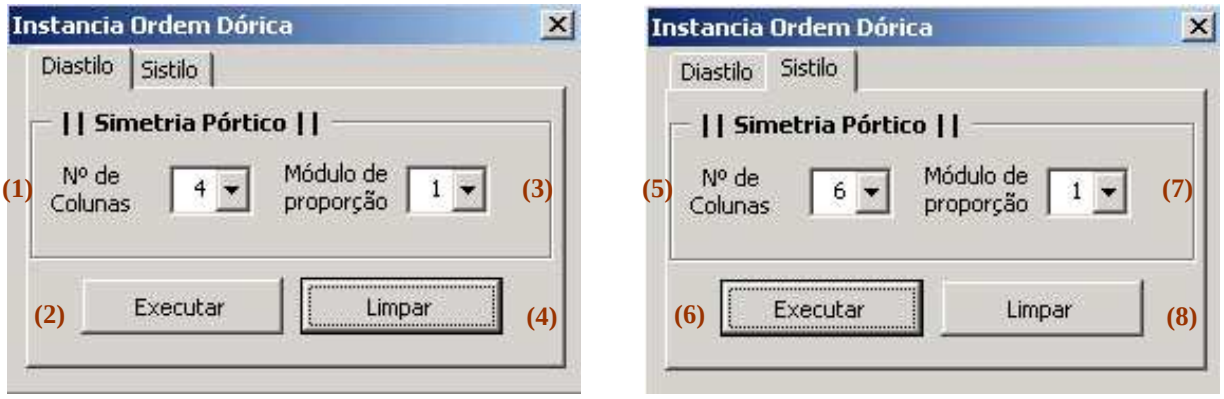


Figura 5.24. Interface da classe "ordem dórica" diastilo e sistilo respectivamente.

Através dos botões e menus acima o usuário do protótipo pode alterar alguns atributos que correspondem às propriedades essenciais da classe diastilo e sistilo, como ilustra a descrição do protótipo abaixo:

(1) Define o número de colunas da "ordem dórica" diastilo.

(2) Instancia a classe "ordem dórica" do tipo diastilo.

(3) Define o número de colunas da "ordem dórica" diastilo.

(4) Apaga todos os objetos presentes no Model Space do AutoCAD.

(5) Define o número de colunas da "ordem dórica" sistilo.

(6) Instancia a classe "ordem dórica" do tipo sistilo.

(7) Define o número de colunas da "ordem dórica" sistilo.

(8) Apaga todos os objetos presentes no Model Space do AutoCAD.

Finalmente, a última etapa do processo de projeto consistiria na sua construção no mundo real, a partir de sua representação projetual. No caso do protótipo "ordem dórica" diastilo e sistilo, cuja representação consiste em um modelo virtual bidimensional, essa construção poderia se dar por meios automatizados, com o uso de máquinas como cortadora a laser para se confeccionar modelos em escala reduzida em materiais como madeira ou acrílico, por exemplo.

5.2.1 RESULTADOS PARCIAIS: ANÁLISE DO PROCESSO DE DESENVOLVIMENTO DO SEGUNDO PROTÓTIPO

Por motivos de limitação da linguagem de programação utilizada, a implementação computacional dos subtipos abordados neste segundo protótipo precisou limitar-se à utilização dos conceitos primordiais da orientação a objetos, tais como classes, objetos, atributos e métodos. Esse estudo dos subtipos da “ordem dórica” ficou restrito ao desenvolvimento de classes isoladas de objetos por não ter sido possível, implementar computacionalmente conceitos como herança ou polimorfismo entre as classes analisadas. Contudo, mesmo com essa restrição existente no *Visual Basic for Applications* (VBA), tornou-se possível através da abordagem teórica desenvolvida ao longo do processo de projeto proposto, consolidar o mecanismo de estruturação do conhecimento arquitetônico em termos as classes e os objetos de um dado problema em questão. Portanto, a complexidade envolvida na conceituação de tipos e objetos arquitetônicos foi perfeitamente subtraída ao se analisar o problema de projeto sob à luz do paradigma da orientação a objetos. Através da estruturação de tipos como classes de objetos tornou-se possível visualizá-los como grupos de objetos que possuem as mesmas características de qualquer outro objeto dos grupos em questão.

6 O EXERCÍCIO DE PROJETO

O estudo dos protótipos iniciais que se desenvolveu no capítulo anterior desta dissertação, de acordo com os exemplos sugeridos por Mitchell (1990) em *The Logic of Architecture*, foi uma experiência favorável para se ampliar as capacidades técnicas de desenvolvimento de implementações computacionais, principalmente no que diz respeito às técnicas da orientação a objetos; tendo sido também uma grande oportunidade para se atingir um amadurecimento teórico no que se refere aos conceitos de tipos e vocabulários arquitetônicos abordados ao longo da pesquisa.

Dessa maneira, a partir da exploração preliminar de tipos como classes de objetos, foi possível estabelecer, através deste protótipo final, a aplicação dos conceitos da orientação a objetos a um processo de projeto em arquitetura. Para este estudo em específico, optou-se por desenvolver um projeto computacional de uma tipologia residencial. As diretrizes do processo de projeto sugerido na pesquisa foram seguidas e uma descrição da aplicação computacional desenvolvida.

6.1 O PROCESSO DE PROJETO DE UMA RESIDÊNCIA

A descrição deste processo de projeto foi estabelecida de acordo com a sequência de passos do método de projeto proposta no capítulo 3 desta dissertação. Baseada originalmente nas idéias de Alexander (1964) apresentadas em *Notes on the synthesis of form*, isso implica em dizer que as três primeiras etapas do processo de projeto descritas a seguir correspondem exatamente às três primeiras etapas que Alexander (1964) definiu em sua obra. A partir da quarta etapa do processo de projeto, contudo, são introduzidas novas ações no sentido de alcançar uma generalização do problema de projeto, por meio do pensamento sobre tipos e objetos arquitetônicos e da definição

de uma classe de edifícios residenciais, tal como sugerido por Mitchell (1990) em *The Logic of Architecture: Design, Computation, and Cognition.*

Diante desse contexto, as etapas desenvolvidas para a definição de uma tipologia residencial se estruturaram da seguinte maneira:

DEFINIÇÃO DO PROBLEMA

Correspondeu ao primeiro estágio do processo de projeto. Essa etapa equivale à primeira etapa do processo de projeto citado por Alexander (1994) em *Notes on the synthesis of form.* Para ilustrar um processo de projeto arquitetônico por meio do paradigma da orientação a objetos, elaborou-se neste estágio do processo de projeto, um desafio que consistiu na organização de um projeto residencial que atendesse as necessidades básicas de moradia para um jovem solteiro, sem filhos e independente. Através da definição deste problema buscou-se alcançar uma solução viável para o projeto de arquitetura.

O cliente

IMAGEM MENTAL

Como resultado de um processo mental, objetivou-se na segunda etapa do processo de projeto, refletir sobre algumas questões pertinentes ao problema em discussão. Embora Alexander (1994) em *Notes on the synthesis of form* não tenha explicitado com precisão essa etapa do processo de projeto, foi possível comparar este momento ao estágio de “incubação” proposto por Lawson (1997) em *How designers think*, ou seja, foi o momento em que se começou a produzir as primeiras idéias sobre o problema de projeto.

IMAGEM FORMAL

Uma vez atingida a terceira etapa do processo de projeto, buscou-se estabelecer neste momento uma formalização do pensamento desenvolvido em torno do problema de projeto. Um levantamento dos requisitos necessários para solucionar o problema em discussão foi desenvolvido e a definição de vínculos entre as variáveis do projeto estabelecido exatamente como Alexander (1964) sugeriu em *Notes on the synthesis of form.*

Por motivos de simplificação, optou-se neste estágio do processo de projeto por estabelecer relações apenas entre duas variáveis do projeto residencial - área total e custo - conforme indica a Figura 6.1. Dessa forma, a área total da residência que é uma das variáveis do projeto estará vinculada ao custo total da construção de maneira diretamente proporcional. Logo, quanto maior for a área total residencial construída maior será o gasto para o cliente em questão.

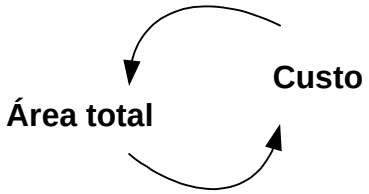


Figura 6.1. Relação entre duas variáveis do projeto.

Após realizar uma reflexão e também uma investigação sobre algumas indicações do projeto residencial, tornou-se possível chegar a um programa arquitetônico suficientemente expressivo para as necessidades do cliente e também estabelecer um dimensionamento inicial de áreas para o projeto. Fez parte desta etapa do processo de projeto, a análise das obras: Módulo de vivenda y grupo residencial: tipologias e La vivenda mínima, que foram essenciais por orientar a imagem formal do problema em termos de propriedades essenciais e acidentais. A

Tabela **6.1** lista as condições projetuais elaboradas para atender as necessidades do cliente.

Tabela 6.1. Programa de necessidades e pré-dimensionamento dos ambientes.

PROGRAMA ARQUITETÔNICO				
	Ambientes	**Função**	**Equipamentos**	**Área Mínima (m²)**
PROPRIEDADES ESSENCIAIS	Sala	Será o espaço mais universal da casa. Neste ambiente será possível estabelecer as funções de: receber visitas, comer, estudar, trabalhar, descansar, lazer como assistir tv e ouvir música, além de permitir guardar acessórios pessoais do usuário.	Sofá cama, Tv/ som, mesa e cadeira para computador, armário e outros.	Mínimo de 20m²
	Cozinha	Será o compartimento da casa onde serão preparados e armazenados os alimentos, além da limpeza e armazenamento dos igredientes e utensílios da cozinha. Receberá as funções de: comer, cozinhar e limpar alimentos. Poderá ter, ou não, uma ligação direta com a sala.	Pia, armários altos e baixos, bancada, geladeira, forno/fogão, micro-ondas, mesa e cadeiras.	Mínimo de 9m²
	Banheiro	Será o espaço adequado para limpeza e higiene do cliente. Neste ambiente serão realizados as atividades de: Lavar-se, Banhar-se, Escovar dentes, Pentear cabelos e outros.	Banheiro completo com: um lavatório, um sanitário e uma ducha.	Mínimo de 2,4m²

PROPRIEDADES ACIDENTAIS	Dormitórios	Será o espaço para repouso do cliente. Poderá servir também como espaço de trabalho e lazer.	Cama, armário, tv/som.	Mínimo de 9m²
	Varanda	Será um ambiente que além de possibilitar uma aproximação entre o cliente e área externa, terá uma importante função climática na habitação. Através deste espaço pretende-se criar uma zona de sombreamento que impedirá o contato direto com a fachada da edificação. Poderá receber funções como estar e refeições rápidas ao ar livre.	Poltronas, almofadas, mesa e cadeiras.	Mínimo de 10m²
	Área de Serviço	Áreas de serviço, na verdade, terá a função de lavanderia.	Tanque, armários, lava-roupa e secadora.	Mínimo de 6m²
	Hall	Terá a função de recepção, atendimento de quem chega, convite à entrada.	--	Mínimo de 2,25m²

PROPRIEDADES ESSENCIAIS E ACIDENTAIS

Assim, na tentativa de explorar os elementos arquitetônicos formadores de um tipo residencial, buscou-se através da quarta etapa do processo de projeto estruturar – hierarquicamente – as propriedades comuns e variáveis entre os exemplares da tipologia em termos de classe e objetos, exatamente como ilustram a Figura 6.2 e a Figura 6.3.

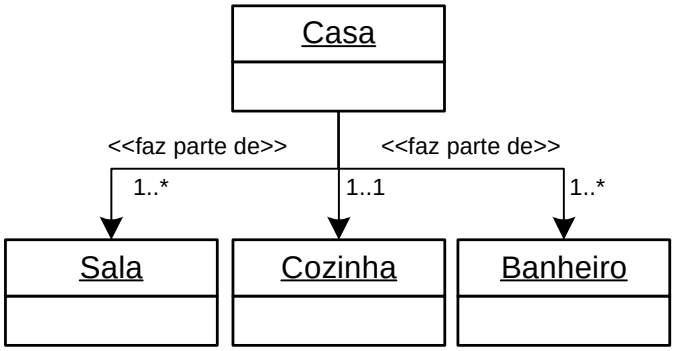


Figura 6.2. Propriedades essenciais à composição da tipologia residencial.

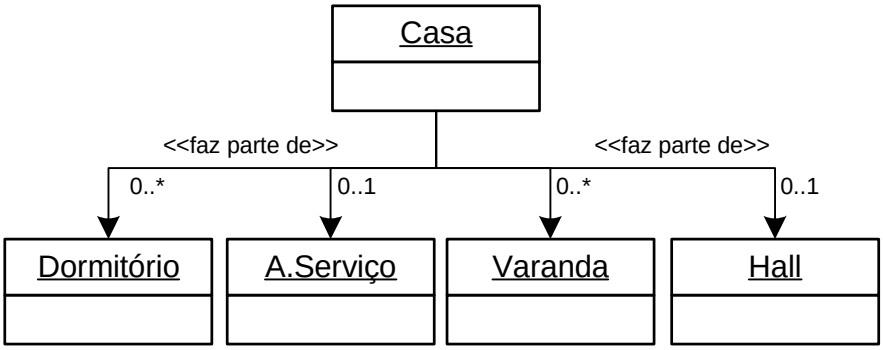


Figura 6.3. Algumas das propriedades acidentais à composição de uma tipologia residencial.

Essa estruturação hierárquica das propriedades do projeto permitiu visualizar a dependência existente entre a tipologia estudada e os elementos considerados essenciais e acidentais a sua composição arquitetônica, possibilitando desta maneira chegar a conclusão de que a casa do

jovem cliente, solteiro, sem filhos e independente considerado neste estudo deveria essencialmente possuir:

- No mínimo uma e no máximo muitas salas;
- No mínimo uma e no máximo uma cozinha;
- No mínimo um e no máximo muitos banheiros;

Os demais elementos arquitetônicos, classificados como propriedades acidentais, suprem as necessidades do cliente, mas não são consideradas indispensáveis para a resolução do problema em questão. Assim, a dependência existente entre a tipologia residencial e os seus objetos acidentais de composição podem ser configurados da seguinte maneira:

- A casa pode possuir nenhum ou muitos dormitórios;
- A casa pode possuir nenhuma ou uma área de serviço;
- A casa pode possuir nenhuma ou uma varanda;
- A casa pode possuir nenhum ou um hall;

O intuito principal desta abordagem do problema foi de auxiliar o arquiteto e o seu processo de projeto. Através do estudo das propriedades essenciais e acidentais, tornou-se possível encontrar uma solução viável para o problema do projeto de arquitetura em discussão, além de alcançar novas variações para o projeto residencial. A tipologia de uma casa refere-se a muitos objetos, portanto é certo encontar exemplares diversos de seu tipo, cada qual com suas características particulares.

CLASSE

Uma vez estruturada as propriedades essenciais e acidentais da tipologia estudada, tornou-se possível através desta etapa do processo de projeto, realizar uma abstração ainda maior do problema em discussão. Através da definição de classes de objetos, buscou-se examinar algumas soluções do passado no intuito encontrar respostas para a concepção do projeto de arquitetura. Questões como a disposição da tipologia residencial e a topologia de uma casa foram analisadas e estruturadas hierarquicamente como é possível observar através da Figura 6.4 e Figura 6.5 respectivamente.

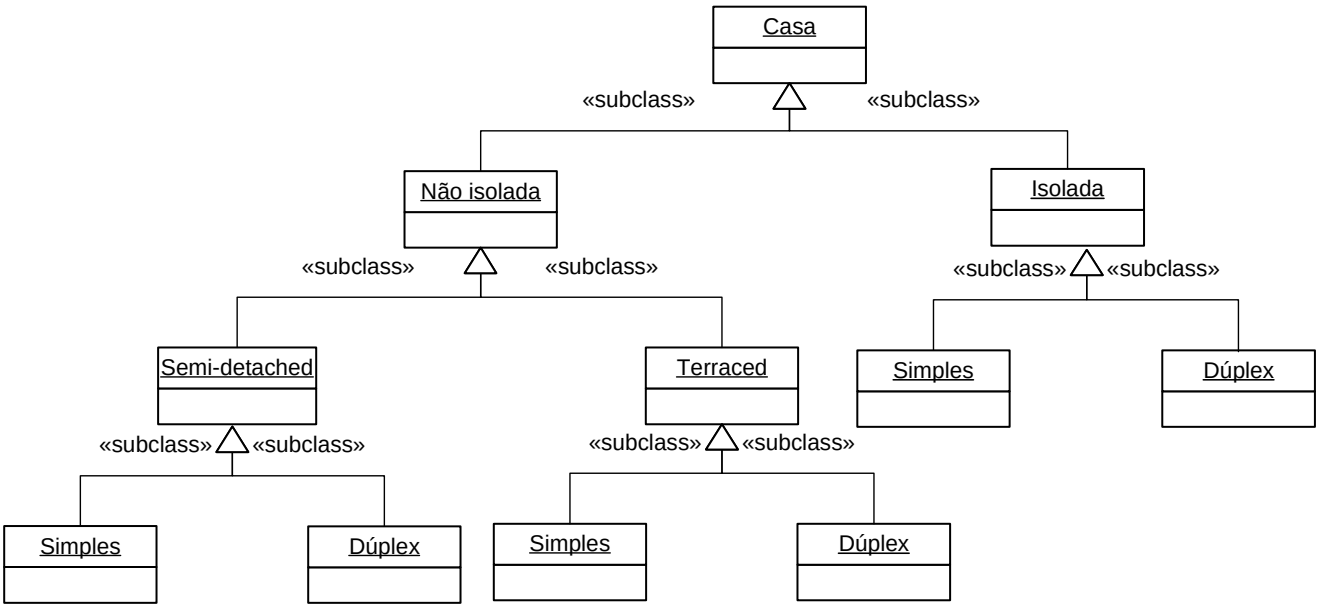


Figura 6.4. Estudo de tipologias residenciais

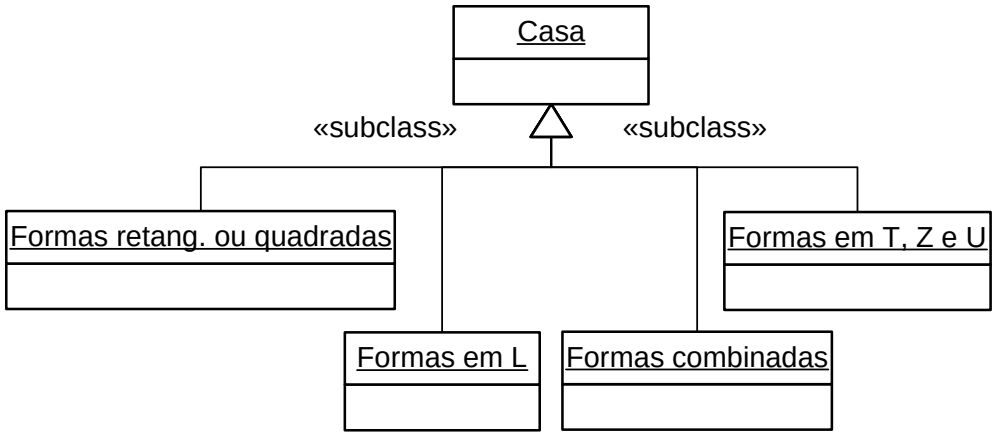


Figura 6.5. Estudo das topologias de uma casa.

IMPLEMENTAÇÃO DA CLASSE

Dessa maneira, diante do estudo realizado nas etapas anteriores foi possível neste estágio do processo de projeto implementar computacionalmente uma tipologia residencial como se fosse uma classe de objetos. Para atender às necessidades do problema em discussão, optou-se por desenvolver um projeto residencial correspondente a uma casa do tipo isolada, de um pavimento (simples) e com o modelo de planta retangular, ou seja, os cômodos da casa do cliente serão dispostos de maneira seqüencial. Nesse estágio, o problema de projeto foi estruturado em termos de: objetos, classes, atributos e métodos desenvolvidos por intermédio da linguagem de programação *visual basic for applications* (VBA) do AutoCAD. As propriedades essenciais e acidentais da tipologia residencial - anteriormente apresentadas - foram consideradas como parâmetros para a resolução do projeto computacional.

INSTÂNCIA DO OBJETO

Como resultado do processo de projeto que se desenvolveu nas etapas anteriormente descritas, foi possível alcançar uma solução de projeto que se adequasse às necessidades essenciais do cliente jovem, solteiro, sem filhos e independente, conforme ilustra a Figura 6.6. Duas tipologias básicas foram consideradas para este estudo: (1) projeto residencial com sala e cozinha separada e (2) projeto residencial com cozinha integrada a sala.

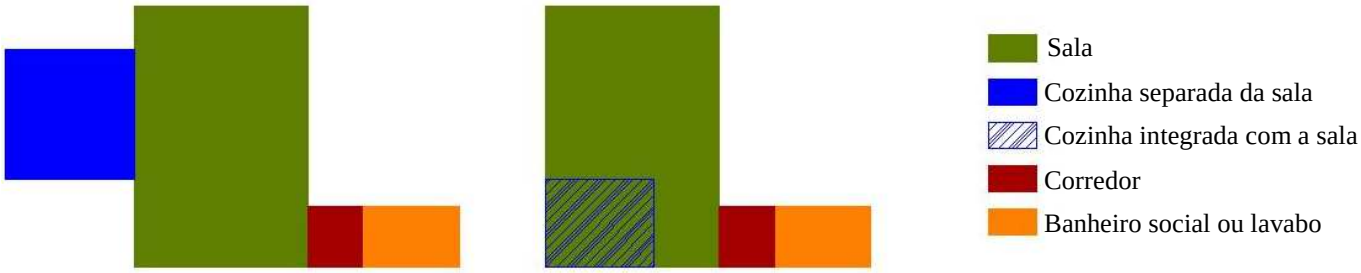
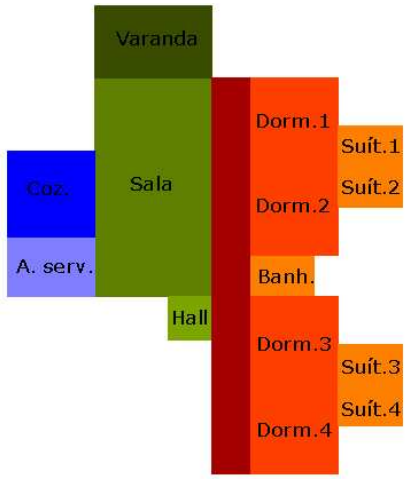
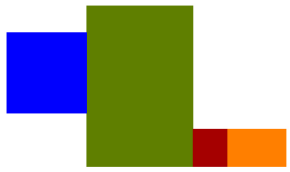
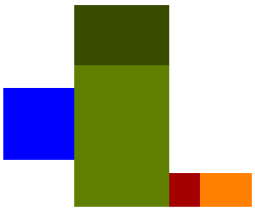
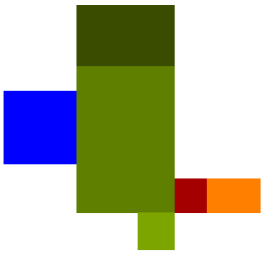
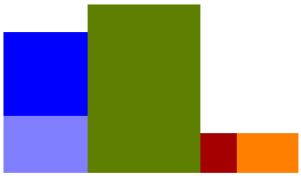
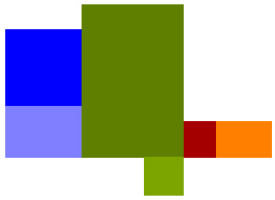
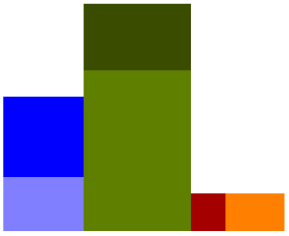
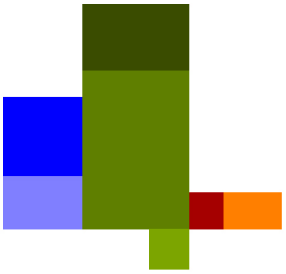


Figura 6.6. Instâncias da classe de objetos casa.

A partir desta síntese do projeto arquitetônico, foi possível alcançar novas derivações do problema em discussão - pertencente sempre a classe reconhecível do edifício em questão. Estes exemplares variávies da tipologia residencial puderam ser implementados a partir de algumas propriedades acidentais do objeto, tais como: dormitórios, área de serviço, varanda e hall. Essa combinação diversificada entre os elementos arquitetônicos do projeto possibilitou criar novos objetos a partir de uma mesma classe e formar dessa maneira, uma família de objetos com semelhanças entre si, com propriedades em comum, mas com suas próprias características arquitetônicas.

Neste estudo também foram consideradas as tipologias com: (1) sala e cozinha separada e (2) cozinha integrada a sala conforme ilustram a Tabela 6.2 e a Tabela 6.3.

Tabela 6.2. Tipologias residências com sala e cozinha separada

TIPOLOGIAS RESIDENCIAS (A)			
			
			
			

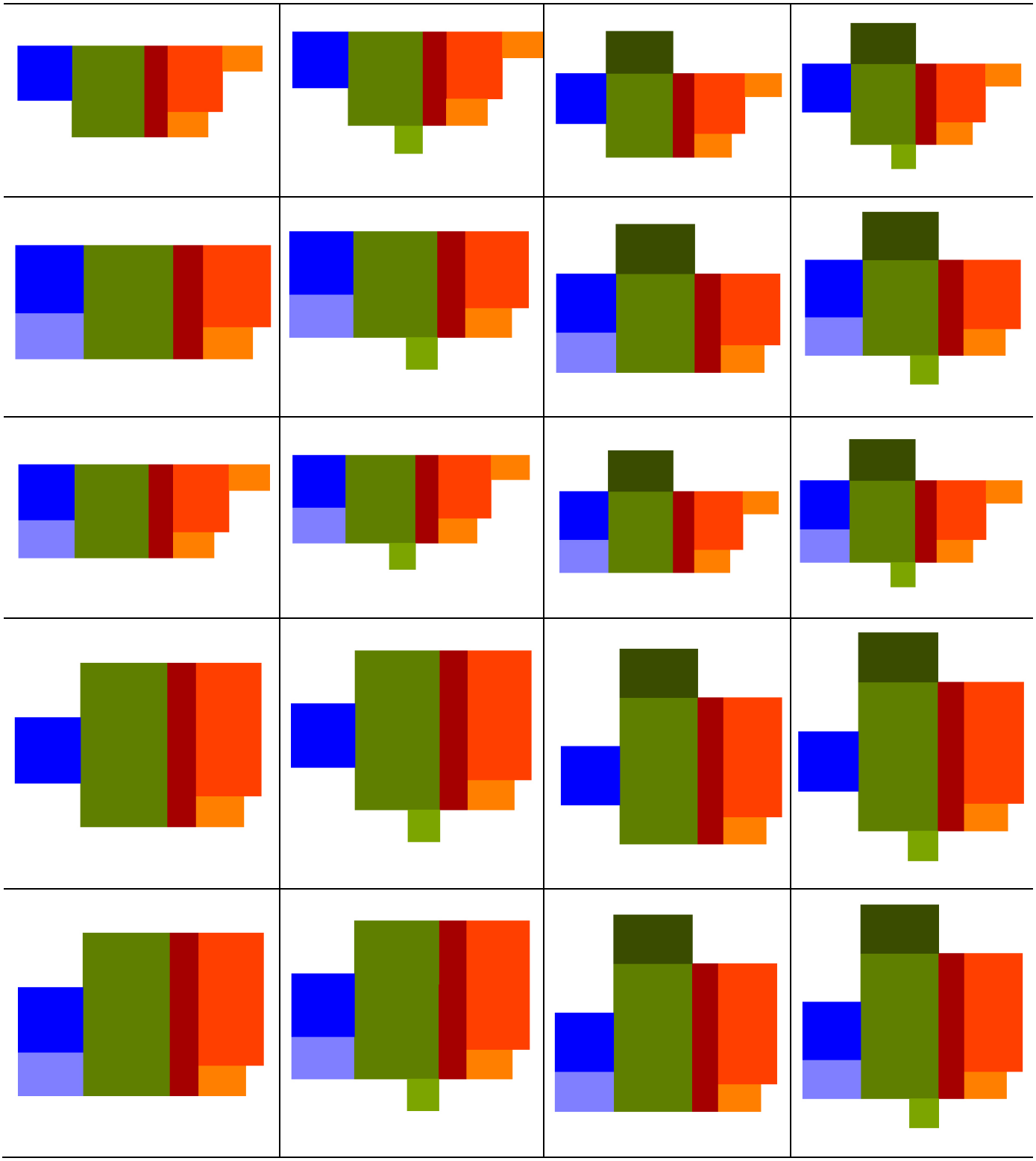
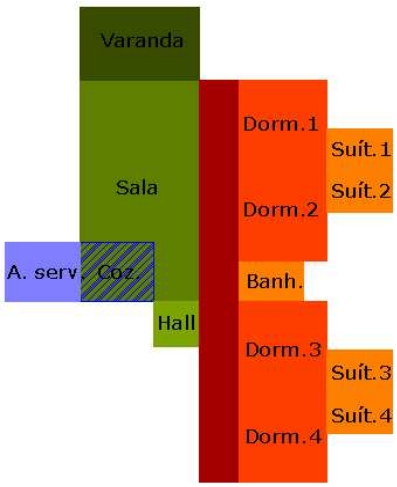
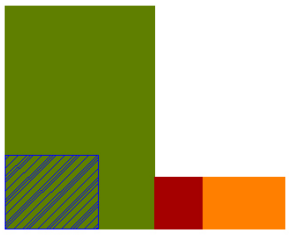
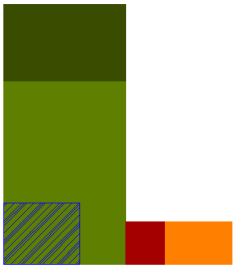
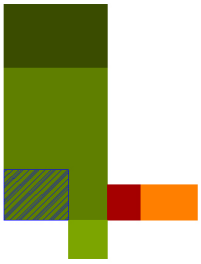
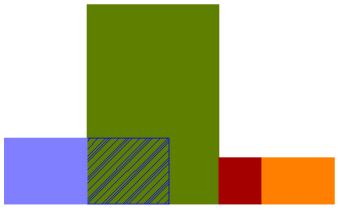
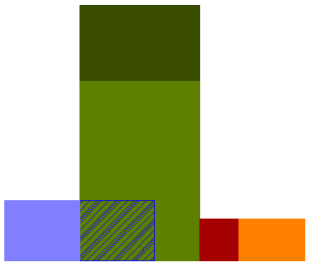
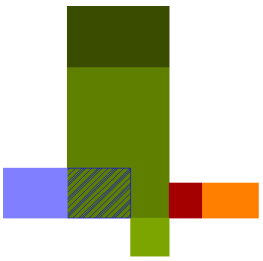
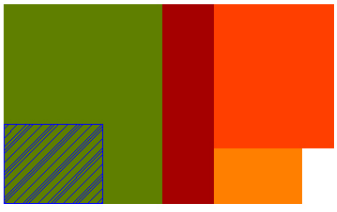
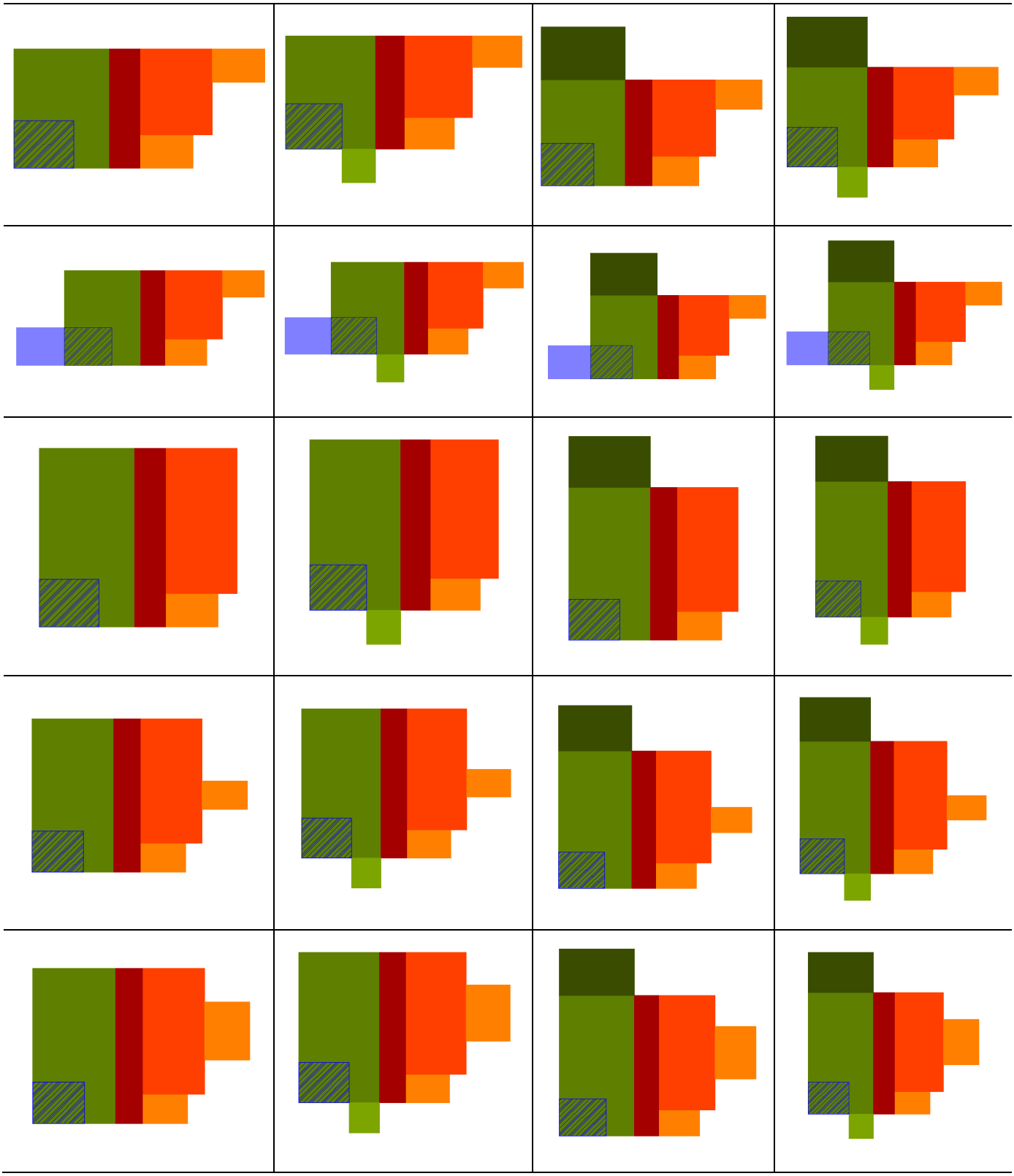


Tabela 6.3. Tipologias residencias com sala e cozinha integrada

TIPOLOGIAS RESIDENCIAS (B)			
			
			
			



É claro que várias outras questões ainda precisariam ter sido discutidas no processo de projeto, tais como: implantação do edifício no terreno, os requisitos de conforto ambiental e outros. No

entanto, por motivos de simplificação optou-se por abordar os requisitos mínimos envolvidos no processo de concepção de um projeto de arquitetura.

OBJETO REAL

Portanto, a última etapa do processo de projeto proposto na pesquisa consistiria na construção da tipologia residencial no mundo real, que se desenvolveu a partir de sua representação projetual.

6.2 A IMPLEMENTAÇÃO COMPUTACIONAL

O processo de implementação do protótipo computacional consistiu basicamente em duas etapas: (1) definição da classe residencial representada por seus atributos e métodos e (2) instanciamento da classe de objeto.

A etapa de definição da classe residencial foi, na verdade, uma extensão da modelagem do processo de projeto descrita inicialmente neste capítulo. Uma vez que as informações relativas ao projeto residencial encontravam-se estruturadas e modeladas, tornou-se possíevel transcrever de maneira clara e intelígel as características do edifício para uma linguagem de programação orientada a objetos (VBA). A partir da definição de todos os seus atributos e métodos a classe pôde ser intanciada através do programa computacional formulado.

A Figura 6.7 ilustra a interface do protótipo desenvolvido no *Visual Basic for Applications* (VBA). O algoritmo da classe implementada pode ser vista no Apêndice B desta dissertação.

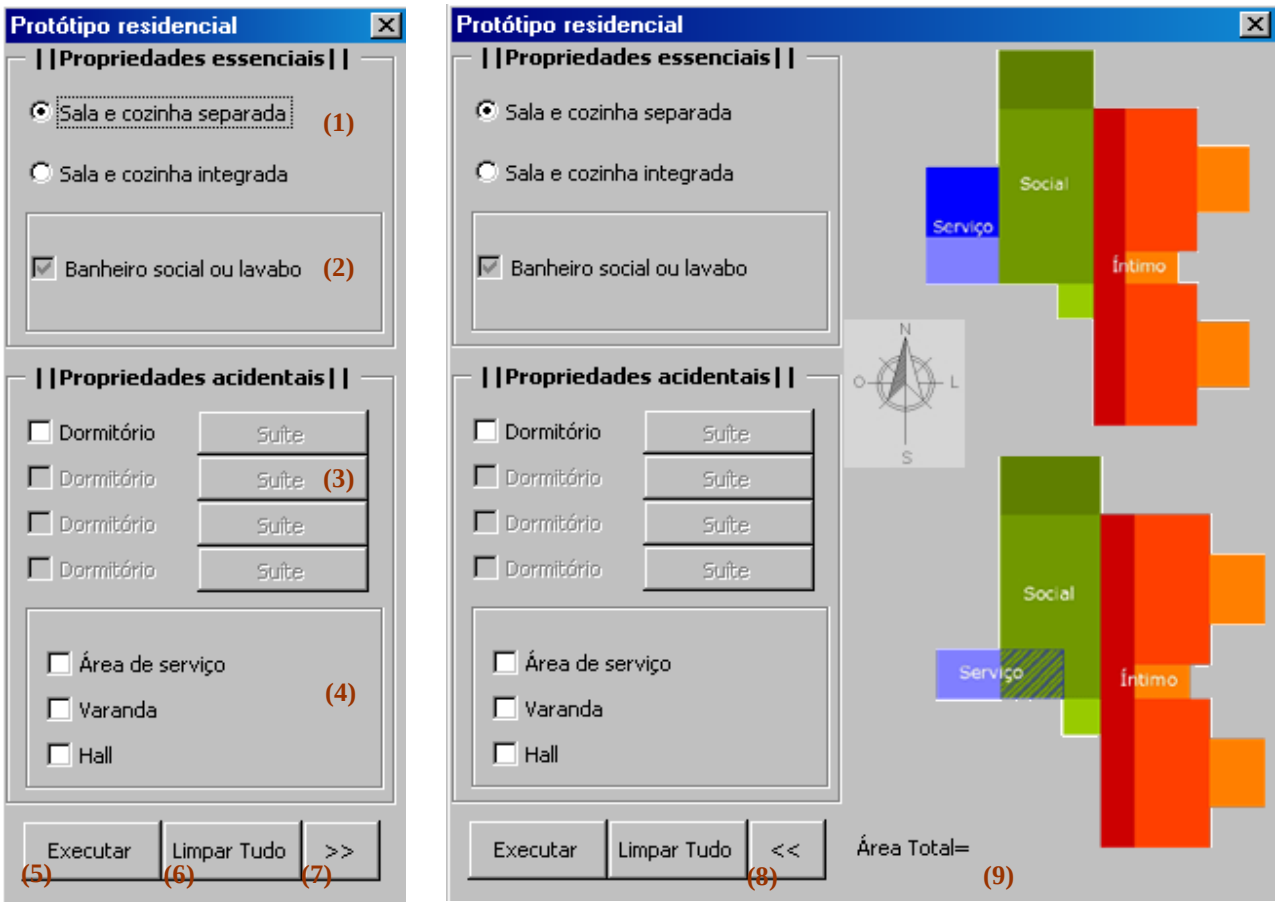


Figura 6.7. Interface do protótipo final desenvolvido no VBA.

Através dos botões e menus acima o usuário pode alterar alguns atributos que correspondem às propriedades essenciais e acidentais da classe residencial, como ilustra a descrição do protótipo abaixo:

(1) Permite selecionar as propriedades essenciais da tipologia residencial dentre duas opções pré-estabelecidas do projeto. (A) Sala e cozinha separada e (B) Sala e cozinha integrada.

(2) Trata-se de uma propriedade comum a qualquer tipologia residencial. Portanto, esta opção na interface do protótipo computacional estará sempre habilitada para o usuário do programa.

(3) Permite selecionar parâmetros para o instanciamento de uma tipologia residencial com um número entre 1 - 4 dormitórios simples ou suítes. A cada dormitório selecionado, a opção suíte e novo dormitório tornam-se habilitados para possível seleção.

(4) Permite selecionar as demais propriedades acidentais consideradas no estudo do projeto computacional, tais como: área de serviço, varanda e hall.

(5) É responsável por realizar o instanciamento da classe residencial criada no VBA.

(6) Permite apagar todos os objetos presentes no Model Space do AutoCAD.

(7) Permite visualizar o esquema das tipologias consideradas no estudo, orientação Norte-Sul e área total do tipo instanciado.

(8) Permite voltar à interface padrão do protótipo residencial.

(9) Define a área total da tipologia residencial calculada em termos de suas propriedades essenciais e acidentais.

Ao avaliar a asistência que este programa confere ao processo de projeto, de acordo com a escala de níveis de pretensão do uso do CAD proposto por Mitchell (1975) em *The theoretical foundation of computer-aided architectural design*, notou-se que este protótipo computacional poderia perfeitamente se enquadrar em um nível médio de ambição do uso do CAD. O protótipo computacional desenvolvido permite a geração e representação automática, porém fica a cargo do arquiteto a responsabilidade de avaliar as soluções obtidas pelo programa. A Tabela 6.4 sintetiza com clareza a relação entre o arquiteto e o computador como participantes do processo de projeto.

Tabela 6.4. Níveis de pretensão no uso de sistemas CAD de acordo com Mitchell (1975).

Grau de Ambição	Funções da Máquina	Funções Humanas	Critérios para geração	Critérios para avaliação	Exemplos
Menor	Representação de alternativas	Geração e avaliação de alternativas	Bem ou mal definidos	Bem ou mal definidos	Sistemas de gerenciamento de banco de dados programáticos e descritivos de edifícios, produção de perspectivas e desenhos executivos, etc.
Médio Baixo	Avaliação de alternativas	Geração de alternativas	Mal definidos	Bem definidos	Sistemas de avaliação térmica, acústica, lumínica, estrutural, etc.
Médio	Geração de alternativas	Avaliação de alternativas	Bem definidos	Mal definidos	Enumeração sistemática de todas as alternativas de organização de uma planta, onde a seleção da melhor depende de conceitos arquitetônicos sutis
Médio Alto	Geração e avaliação de alternativas	-	Bem definidos	Bem definidos	Programas de otimização
Alto	Geração e avaliação de alternativas	-	Mal definidos	Mal definidos	CAD inteligente

Dessa maneira, foi possível concluir que o método de projeto proposto no presente trabalho, além de fortalecer uma visão do mundo construtivo em termos de tipos e objetos, tem o intuito principal de auxiliar o arquiteto na estruturação e formalização dos problemas de projeto, desenvolvendo dessa maneira o raciocínio, a lógica e também a capacidade de descrever projetos computacionais. Assim, todas as informações obtidas a respeito de uma classe (tipo arquitetônico) poderão ser utilizadas em um discurso crítico, influenciar as intenções de projeto e guiar todas as explorações formais estabelecida pelo projetista (MITCHELL, 1990).

7 CONSIDERAÇÕES FINAIS

Esta pesquisa teve como ponto de partida a proposta apresentada por Mitchell (1990) de se pensar o projeto de arquitetura de maneira análoga a uma técnica da área de computação, a definição de classes nas linguagens de programação orientadas a objetos. Para que se pudesse compreender a fundo essa proposta, foi necessário aprofundar os conhecimentos (1) sobre tipologias arquitetônicas, (2) sobre linguagens orientadas a objetos, (3) sobre metodologia e processo de projeto em arquitetura, e finalmente (4) sobre aplicações computacionais na arquitetura.

(1) O estudo de tipologias arquitetônicas foi baseado, sobretudo nas referências apresentadas pelo próprio Mitchell (1990), Colquhoun (1967) e Argan (1962 *in* NESBITT, 2006).

(2) No estudo do paradigma da orientação a objetos procurou-se entender suas origens na filosofia platônica e definir as principais características desse tipo de linguagem. Ao mesmo tempo buscou-se adquirir conhecimentos técnicos visando a posterior implementação computacional de exemplos arquitetônicos.

(3) No estudo sobre o processo de projeto em arquitetura percebeu-se que ele pode ter duas abordagens principais, tal como sugerido por Carrara; Kalay e Novembri (1994):

1. Ele pode ser um processo de resolução de problema, envolvendo as etapas de análise e síntese. Nesse caso a solução emerge do próprio problema, em resposta a ele.
2. Ele pode ser um processo de adaptação de soluções pré-existentes a uma situação específica. Neste caso, é necessário que o arquiteto tenha um conhecimento de tipologias arquitetônicas mais ou menos abstratas, que podem ser adaptadas a casos específicos.

Ainda segundo os autores Carrara; Kalay e Novembri (1994), na maioria dos casos o que ocorre é uma combinação destas duas formas de projetar.

(4) Na última etapa da fundamentação teórica, procurou-se estudar uma das principais referências na área de aplicações computacionais na arquitetura: a obra de Alexander (1964). Percebeu-se que Alexander (1964) propõe a utilização das duas abordagens de projeto descritas acima, porém de maneira isolada, em diferentes obras. A primeira abordagem é apresentada em sua obra *Notes on the synthesis of form*, e a segunda na obra *A pattern language* que se estendeu ao trabalho seguinte *The timeless way of building*.

Tendo em vista a tese dos autores, Carrara; Kalay e Novembri (1994), de que as duas abordagens costumam ocorrer juntas, foi proposto, nesta dissertação, uma representação do processo de projeto que se baseia no esquema de Alexander (1964), porém acrescenta-lhe um nível a mais de abstração, no qual o arquiteto faz uma generalização do problema, tentando encaixá-lo em uma das grandes tipologias da arquitetura (C4).

Finalmente, para ilustrar as idéias aqui desenvolvidas, foram implementados alguns exemplos: primeiramente duas implementações bastante simples, baseadas nos exemplos de Mitchell (1990) da cadeira e da ordem dórica, e finalmente uma implementação mais complexa, de uma classe "residencial". É importante ressaltar que a implementação não tem o objetivo prático de resolver o processo de projeto de uma residência, mas apenas de ilustrar um processo em que se parte de um problema específico, chega-se até um nível de abstração em que se pode pensar sobre a tipologia residencial como uma grande classe de objetos, e em seguida volta-se ao problema específico por meio do instanciamento de uma casa que resolve o problema de projeto inicial.

O exercício de reflexão sobre o projeto arquitetônico aqui desenvolvido permitiu que se chegasse à seguinte conclusão: o projeto de arquitetura envolve um tipo de raciocínio que parte de um problema específico, caminha para uma generalização, e depois retorna ao problema específico, com o objetivo de resolvê-lo. Ao pensarmos deste modo, podemos estabelecer uma interessante comparação entre o processo de projeto em arquitetura e a pesquisa científica. Tipicamente, uma pesquisa científica apresenta o formato de ampulheta (Figura 7.1) com as extremidades largas e a parte central afinada. Em outras palavras, parte-se de um problema geral (A), escolhe-se um problema específico para analisar objetivamente (B), e a partir da análise desse problema específico procura-se concluir como as conclusões obtidas podem ser generalizadas e aplicadas a outras situações semelhantes (C) (exatamente como estamos fazendo nesta conclusão).

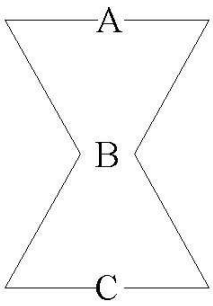


Figura 7.1: A pesquisa científica como uma ampulheta.

No processo de projeto em arquitetura ocorre precisamente o oposto: parte-se de um problema específico (A') - projetar um edifício que atenda a determinados requisitos e que se encaixe em um terreno específico, procura-se transcender esse problema específico pensando-se nos grandes arquétipos arquitetônicos (B') - as tipologias, e finalmente volta-se ao caso específico adaptando-se as soluções arquetípicas ao caso específico (C'). Como resultado, obtém-se um processo que lembra muito mais uma ânfora que uma ampulheta, com a parte mais alargada no meio e as extremidades mais finas, conforme ilustra a Figura 7.2[5].

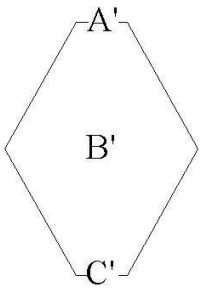


Figura 7.2: O processo de projeto como uma ânfora.

[5] Esta característica particular do projeto de arquitetura foi apontada por Schön em *The reflexive practitioner* (1983), em que ele questiona precisamente como a experiência anterior pode auxiliar o arquiteto a resolver um problema se cada projeto se constitui em um caso totalmente novo e particular.

Os arquétipos arquitetônicos não se restringem às tipologias funcionais como a residencial utilizada nesta pesquisa. É possível pensar em diversos temas da arquitetura como temas arquetípicos que podem contribuir para o processo de projeto, desde que o arquiteto realize o exercício de abstração sugerido nesta dissertação. Quando pensamos em uma escada, por exemplo, podemos transcender a escada convencional e pensar no problema mais geral de se realizar a transição de um plano horizontal a outro, como fez Ubach i Nuet (1994) no livro *La escalera* (Figura 7.3). Dessa forma, passamos a incluir sob o tema "escada" as rampas, as arquibancadas, os anfiteatros, até mesmo as estantes, e nos reportamos a inúmeras novas possibilidades de solução do problema que talvez não tivessem sido pensadas no momento da estruturação do problema de projeto a partir de seus requisitos funcionais. Todas essas escadas possuem a propriedade essencial de uma escada, que é a de permitir a transposição de um plano a outro, mas elas variam em suas propriedades acidentais, como por exemplo, o fato do degrau poder ser utilizado como assento, dele ser horizontal ou inclinado, formando uma rampa e outras possibilidades.

Figura 7.3: Alguns exemplos de "escadas", tal como sugerido por Ubach i Nuet (1994).

Em resumo, é possível concluir que esse exercício de abstração durante o processo de projeto em arquitetura que faz com que o arquiteto transcenda o caso específico tem dois resultados desejáveis:

1. Contribuir para que se chegue a uma melhor solução arquitetônica, uma solução que não leve em conta apenas os requisitos funcionais do problema, mas que proponha novas características com base na consulta às tipologias relacionadas ao problema inicial;

2. Contribuir para que cada novo projeto arquitetônico se constitua em uma oportunidade de reflexão sobre problemas típicos da arquitetura e sobre como resolvê-los.

A metodologia de projeto proposta nesta dissertação, que utilizou a definição de classes de objetos como maneira de definir tipologias arquitetônicas com suas propriedades essenciais e acidentais pode ser uma maneira de se estimular este tipo de raciocínio em estudantes de arquitetura. A aplicação desta metodologia no ensino do projeto se constituirá precisamente no tema de nossos estudos futuros.

REFERÊNCIAS

ALEXANDER, Christopher. **Notes on the Synthesis of Form.** 1ª ed. Londres: Harvard University Press, 1964.

ALEXANDER, Christopher; *et al*. **A Pattern Language:** Towns, Buildings, Construction. 1ª ed. Nova Iorque: Oxford University Press, 1977.

ARGAN, Giulio. Sobre a tipologia em arquitetura in: NESBITT, KATE (Org). **Uma nova agenda para a arquitetura:** antologia teórica (1965-1995). 1ª ed. [S.I]: Cosacnaify, 2006. Cap. 5. p. 268-272.

BITTENCOURT, Guilherme. **Inteligência Artificial.** 2ª ed. Florianópolis: Editora da UFSC, 2001.

BOOCH, Grady; RUMBAUGH, James; JACOBSON, Ivar. **UML:** Guia do usuário. 2ª ed. Rio de janeiro: Elsevier, 2005.

CARRARA, G; KALAY, Y.E.; NOVEMBRI, G. Knowledge-Based Computational Support for Architectural Design. *In:* _____ **Knowledge-Based Computer-Aided Architectural Design.** Nova Iorque: Elsevier, 1994. p. 1-38. 1 arquivo (655 kb); disponível em <http://arch.ced.berkeley.edu/>. Acesso em: 14 dez. 2006.

CELANI, Gabriela. **Teaching CAD programming to architecture students.** No prelo, 2007.

CELANI, Gabriela. **Beyond analysis and representation in CAD:** a new computational approach to design education. Estados Unidos, 2002. Tese (Doutorado em Architecture: Design & Computation) - Departamento de Arquitetura, Massachusetts Institute of Technology, MIT.

COATES, P; THUM, R. **Generative Modelling:** Student Workbook**.** Londres: University of East London, 1995 *apud* CELANI, Gabriela. **Teaching CAD programming to architecture**

students. No prelo, 2007.

COLQUHOUN, Alan. **Typology and design method.** Essays in Architectural Criticism: Modern Architecture and Historical Change, Estados Unidos, v. 83, p. 43-50, jun. 1967.

CORREIA, Carlos Henrique; TAFNER, Malcon Anderson. **Análise orientada a objetos.** [S.I]: Visual books, 2001.

DUARTE, JOSÉ. **Customizing Mass Housing: The Grammars of Siza's Malagueira Houses.** Estados Unidos, 2001. Tese (Doutorado em Architecture: Design & Computation) - Departamento de Arquitetura, Massachusetts Institute of Technology, MIT.

GAARDER, JOSTEIN. **O mundo de Sofia:** romance da história da filosofia. São Paulo: Companhia das letras, 1995.

GALLI, Vera. **Mobiliário brasileiro: a cadeira no Brasil.** São Paulo: Empresa de artes, 1988.

GUEDES, Gilleanes T. A. **UML:** Uma abordagem prática. 1º ed. São Paulo: Novatec, 2004.

GUILFORD, J. P .**The Nature of Human Intelligence.** New York: McGraw-Hill, 1967.

JUTLA, Rajinder Singh. **Christopher Alexander's design theory from notes on the synthesis of form to a pattern language.** Design Methods: Theories, Education and Practice, [s.i], v. 27, n. 4, p. 1899-1913, Out-Dez 1993.

KEOGH, Jim; GIANNINI, Mario. **OOP Desmistificado:** Programação orientada a objetos. 1ª ed. Rio de janeiro: Alta books, 2005.

KOWALTOWSKI, D. C. C. K. **Transferência de inovação tecnológica na autoconstrução de moradias.** *In:* FORMOSO, C. T.; INO. A. (ed.), 2003, Porto Alegre. Coletânea Habitare: Gestão da Qualidade & Produtividade e Disseminação do Conhecimento na Construção Habitacional. Porto Alegre: ANTAC, 2003. p. 95-139. Disponível em <http://habitare.infohab.org.br/publicacao_coletanea2.aspx >.

KNELLER, George F. **Arte e ciência da criatividade.** 5ª ed. São Paulo: IBRASA, 1978.

KOESTLER, Arthur. **The act of creation.** Nova Iorque: Macmillan, 1964 *apud* KNELLER, George F. **Arte e ciência da criatividade.** 5ª ed. São Paulo: IBRASA, 1978.

LAWSON, B. **How designers think:** the design process demystified. 3ª Edição, Architectural press, 1997.

MARTINEZ, Alfonso Corona. **Ensaio sobre o projeto.** Brasília: Editora Universidade de Brasília, 2000.

MINSKY, M. **A framework to represent knowledge**. *In:* THE PSYCHOLOGY OF COMPUTER VISION. Nova Iorque: McGraw-Hill, 1975. p. 211-277 *apud* MITCHELL, William. **The Logic of Architecture:** Design, Computation, and Cognition. Estados Unidos: MIT PRESS, 1990.

MITCHELL, William. **The Logic of Architecture:** Design, Computation, and Cognition. Estados Unidos: MIT PRESS, 1990.

MITCHELL, William; LIGGETT, Robin; KVAN, Thomas. **The Art of Computer Graphics Programming.** 1ª ed. Estados Unidos: Van Nostrand Reinhold, 1987.

MITCHELL, William. **The theoretical foundation of computer-aided architectural design.** Environment and Planning B, [s.i], v. 2, p. 127-150, 1975.

MOLES, Abraham A. **A criação científica** 3ª ed. São Paulo: Perspectiva, 1998.

NESBITT, Kate. **Uma nova agenda para a arquitetura:** antologia teórica (1965-1995). 1ª ed. [S.I]: Cosacnaify, 2006. 659 p. v. 1.

PICARELLI, MARLENE. **Habitação:** uma interrogação. São Paulo: [S.n.], 1986.

PINA, Silvia Mikami; *et al*. **Rotinas e orientações para autoconstrução:** em busca da sustentabilidade social. Conferência Latino Americana de Construção Sustentável ENTAC '04 x Encontro Nacional de Tecnologia do Ambiente Construído, v. 1, p.1-15, 2004.

Platão. Disponível em: <http://pt.wikipedia.org> Acesso em: maio 2007.

REIS, A. T. L.; LAY, M.C.D. Tipos arquitetônicos e dimensões dos espaços da habitação social. *In:* AMBIENTE CONSTRUÍDO, 3., 2002, Porto Alegre. **Anais...** Porto Alegre: ANTAC, 2002. p. 7-24. <http://www.antac.org.br/>.

RODRIGUES, A. B. F.; RUSCHEL, R. C. Adaptação da ferramenta Automet para a simulação de projetos arquitetônicos em conjuntos habitacionais. *In:* VI ENCONTRO NACIONAL E III ENCONTRO LATINO-AMERICANO SOBRE CONFORTO NO AMBIENTE CONSTRUÍDO, 2001, São Pedro. **Anais...**Campinas: FEC-UNICAMP, 2001. p. 1-8.

SCHMITT, G. **Microcomputer Aided Design for Architects and Designers.** Nova Iorque: John Wiley & Sons, 1988 *apud* CELANI, Gabriela. **Teaching CAD programming to architecture students.** No prelo, 2007.

SCHON, D. **The Reflexive Practitioner:** How Professionals Think in Practice. Nova Iorque: Basic Books, 1983.

SUMMERSON, J. **The classical language of Architecture.** Cambridge: MIT Press, 1963 *apud* MITCHELL, William. **The Logic of Architecture:** Design, Computation, and Cognition. Estados Unidos: MIT PRESS, 1990.

UBACH I NUET, Antoni. **La escalera**: una perspectiva del siglo XX. Barcelona: Gustavo Gilli, 1994.

TERZIDIS, Kostas. **Algorithmic Architecture.** 1ª ed. Londres: Elsevier, 2006.

VELOSO, Paulo; *et al*. **Estrutura de dados.** 17ª ed. Rio de janeiro: Campus, 1983.

VITRUVIUS, Marcus Pollio. **Ten books on architecture.** Tradução: Ingrid Rowland e Thomas Howe. Cambridge: Cambridge University Press, 1999.

BIBLIOGRAFIAS CONSULTADAS

CHING, F.D.K. **Dicionário visual de arquitetura.** São Paulo: Martins Fontes, 1999.

DUARTE, JOSÉ; BEIRÃO, JOSÉ. **Personalizar a habitação em série:** uma experiência de ensino. Lisboa, 2002. Catálogo.

GIL, Antonio Carlos. **Como elaborar projetos de pesquisa.** 4ª ed. São Paulo: Atlas, 2002.

NEUFERT, PETER; NEFF, LUDWIG. **Casa, apartamento, jardim.** 2ª ed. Barcelona: Gustavo Gili, 2001.

RYBCZYNSKI, Witold. **Casa:** Pequena história de uma idéia. Rio de janeiro: Record, 2002.

STING, HELLMUTH. **Módulo de vivenda y grupo residencial:** Tipologías. Barcelona: Gustavo Gili, [197?]. (Temas de arquitectura actual.) v. 11.

VIEIRA, Sônia. **Como escrever uma tese.** São Paulo: Pioneira Thomson, 2002.

WOLF, RAINER. **La vivienda mínima.** Barcelona: Gustavo Gili, [197?]. (Temas de arquitectura actual.) v. 7.

APÊNDICE A

Este apêndice tem como objetivo documentar o processo de projeto computacional desenvolvido com auxilio do paradigma da orientação a objetos. Os *scripts* apresentados a seguir referem-se aos protótipos “cadeira” e “ordem dórica” desenvolvidos através da linguagem de programação *visual basic for applications* (VBA) do AutoCAD.

CLASSE CADEIRA FIXA

```
'---------------------------------------------------------------
' UNICAMP - Universidade Estadual de Campinas
' FEC - Faculdade de Engenharia Civil, Arquitetura e Urbanismo
' 14/06/2006
'CLASSE CADEIRAFIXA
'---------------------------------------------------------------
Public Function Executar_CadeiraFixa(DoisBracos, UmBraco As Double) As CadeiraFixa
'Deleta qualquer objeto existente no modelspace do AutoCad
      For Each objeto In ThisDrawing.ModelSpace
     objeto.Delete
      Next
'Ponto de insercao da CadeiraFixa
mycenter(0) = 0: mycenter(1) = 0: mycenter(2) = 0
'Variaveis da CadeiraFixa
CompAssento = UserForm1.Slider1.Value * 0.1
LargAssento = UserForm1.Slider2.Value * 0.1
CompEncosto = CompAssento
AltEncosto = UserForm1.Slider3.Value * 0.1
Pintura_Assento_Encosto = UserForm1.Slider4.Value
Pintura_Estrutura = UserForm1.Slider5.Value
'Constantes dos bracos da CadeiraFixa
```

```
dhbraco = 0.2
radiusBrac = 0.02
radiusBrac2 = radiusBrac
'Constantes relacionadas ao encosto da CadeiraFixa
radius = 0.02
dAltura = 0.1
'DEFINICAO DOS OBJETOS DE COMPOSICAO DO TIPO CadeiraFixa
'PROPRIEDADES ESSENCIAIS AO TIPO |ASSENTO|ENCOSTO|PERNAS
' Assento da CadeiraFixa
      center(0) = mycenter(0): center(1) = mycenter(1): center(2) =
mycenter(2)
    length = CompAssento: width = LargAssento: height = AltAssento
      Set Assento = ThisDrawing.ModelSpace.AddBox(center, length, width,
height)
      Assento.color = Pintura_Assento_Encosto
' Sustentacao do assento 1
      center(0) = mycenter(0): center(1) = mycenter(1) + (LargAssento / 2) -
(AltAssento / 2): center(2) = mycenter(2) - AltAssento
      length = CompAssento - RaioPernas: width = AltAssento: height =
AltAssento
      Set Suporte1_Assento = ThisDrawing.ModelSpace.AddBox(center, length,
width, height)
Suporte1_Assento.color = Pintura_Estrutura
' Sustentacao do assento 2
center(0) = mycenter(0) + (CompAssento / 2) - (width / 2) - (RaioPernas * 2):
center(1) = mycenter(1): center(2) = mycenter(2) - AltAssento
length = AltAssento: width = LargAssento: height = AltAssento
Set Suporte2_Assento = ThisDrawing.ModelSpace.AddBox(center, length, width,
height)
Suporte2_Assento.color = Pintura_Estrutura
' Sustentacao do assento 3
center(0) = mycenter(0): center(1) = mycenter(1) - (LargAssento / 2) +
(AltAssento / 2) + (RaioPernas * 2): center(2) = mycenter(2) - AltAssento
length = CompAssento - RaioPernas: width = AltAssento: height = AltAssento
Set Suporte3_Assento = ThisDrawing.ModelSpace.AddBox(center, length, width,
height)
Suporte3_Assento.color = Pintura_Estrutura
' Sustentacao do assento 4
center(0) = mycenter(0) - (CompAssento / 2) + (width / 2) + (RaioPernas * 2):
center(1) = mycenter(1): center(2) = mycenter(2) - AltAssento
length = AltAssento: width = LargAssento: height = AltAssento
Set Suporte4_Assento = ThisDrawing.ModelSpace.AddBox(center, length, width,
height)
 Suporte4_Assento.color = Pintura_Estrutura
```

```
' Encosto da CadeiraFixa
 center(0) = mycenter(0): center(1) = mycenter(1) + (LargAssento / 2) +
LargEncosto: center(2) = mycenter(2) + (AltAssento / 2) + (AltEncosto / 2)
  length = CompEncosto - (RaioPernas * 4): width = LargEncosto: height =
AltEncosto / 2
  Set Encosto = ThisDrawing.ModelSpace.AddBox(center, length, width, height)
  Encosto.color = Pintura_Assento_Encosto
' Suporte do encosto 1
  center(0) = mycenter(0) + (CompAssento / 2) - RaioPernas: center(1) =
mycenter(1) + (LargAssento / 2) + RaioPernas: center(2) = mycenter(2) +
(AltEncosto / 2) + (AltAssento / 2)
  width = RaioPernas: height = AltEncosto
  Set Suporte1_Encosto = ThisDrawing.ModelSpace.AddCylinder(center, width,
height)
  Suporte1_Encosto.color = Pintura_Estrutura
' Suporte do encosto 2
  center(0) = mycenter(0) - (CompAssento / 2) + RaioPernas: center(1) =
mycenter(1) + (LargAssento / 2) + RaioPernas: center(2) = mycenter(2) +
(AltEncosto / 2) + (AltAssento / 2)
  width = RaioPernas: height = AltEncosto
  Set Suporte2_Encosto = ThisDrawing.ModelSpace.AddCylinder(center, width,
height)
  Suporte2_Encosto.color = Pintura_Estrutura
' Arco do encosto da CadeiraFixa
 startTan(0) = 0: startTan(1) = 0: startTan(2) = 1
 endTan(0) = 0: endTan(1) = 0: endTan(2) = -1
 fitPoints(0) = mycenter(0) - (CompAssento / 2) + RaioPernas: fitPoints(1) =
mycenter(1) + (LargAssento / 2) + RaioPernas: fitPoints(2) = AltEncosto +
(AltAssento / 2)
 fitPoints(6) = mycenter(0) + (CompAssento / 2) - RaioPernas: fitPoints(7) =
mycenter(1) + (LargAssento / 2) + RaioPernas: fitPoints(8) = AltEncosto +
(AltAssento / 2)
 fitPoints(3) = (fitPoints(0) + fitPoints(6)) / 2: fitPoints(4) =
(fitPoints(1) + fitPoints(7)) / 2: fitPoints(5) = dAltura + AltEncosto +
(AltAssento / 2)
  Set splineObj = ThisDrawing.ModelSpace.AddSpline(fitPoints, startTan,
endTan)
  centerPoint(0) = fitPoints(0): centerPoint(1) = fitPoints(1): centerPoint(2)
= fitPoints(2)
  Set curves(0) = ThisDrawing.ModelSpace.AddCircle(centerPoint, radius)
  regionObj = ThisDrawing.ModelSpace.AddRegion(curves)
  Set solidObj =
ThisDrawing.ModelSpace.AddExtrudedSolidAlongPath(regionObj(0), splineObj)
  solidObj.color = Pintura_Estrutura
```

```
' Perna 1 da CadeiraFixa
  center(0) = mycenter(0) - (CompAssento / 2) + RaioPernas: center(1) =
mycenter(1) - (LargAssento / 2) + RaioPernas: center(2) = mycenter(2) -
(AltPernas / 2) - (AltAssento / 2)
    width = RaioPernas: height = AltPernas
    Set Perna1 = ThisDrawing.ModelSpace.AddCylinder(center, width, height)
    Perna1.color = Pintura_Estrutura
' Perna 2 da CadeiraFixa
    center(0) = mycenter(0) + (CompAssento / 2) - RaioPernas: center(1) =
mycenter(1) - (LargAssento / 2) + RaioPernas: center(2) = mycenter(2) -
(AltPernas / 2) - (AltAssento / 2)
    width = RaioPernas: height = AltPernas
    Set Perna2 = ThisDrawing.ModelSpace.AddCylinder(center, width, height)
    Perna2.color = Pintura_Estrutura
' Perna 3 da CadeiraFixa
    center(0) = mycenter(0) + (CompAssento / 2) - RaioPernas: center(1) =
mycenter(1) + (LargAssento / 2) + RaioPernas: center(2) = mycenter(2) -
(AltPernas / 2) - (AltAssento / 2)
 width = RaioPernas: height = AltPernas + (AltAssento * 2)
 Set Perna3 = ThisDrawing.ModelSpace.AddCylinder(center, width, height)
 Perna3.color = Pintura_Estrutura
' Perna 4 da CadeiraFixa
 center(0) = mycenter(0) - (CompAssento / 2) + RaioPernas: center(1) =
mycenter(1) + (LargAssento / 2) + RaioPernas: center(2) = mycenter(2) -
(AltPernas / 2) - (AltAssento / 2)
 width = RaioPernas: height = AltPernas + (AltAssento * 2)
 Set Perna4 = ThisDrawing.ModelSpace.AddCylinder(center, width, height)
 Perna4.color = Pintura_Estrutura
'PROPRIEDADES ACIDENTAIS AO TIPO |1 OU 2 BRACOS|
' Define CadeiraFixa com 2 bracos
    If DoisBracos = True Then
'Braco1 da CadeiraFixa
   startTanBrac(0) = 0: startTanBrac(1) = 0: startTanBrac(2) = 1
   endTanBrac(0) = 0: endTanBrac(1) = 0: endTanBrac(2) = 1
      fitPointsBrac(0) = mycenter(0) - (CompAssento / 2) + radiusBrac:
      fitPointsBrac(1) = mycenter(1) - (LargAssento / 2) + radiusBrac:
      fitPointsBrac(2) = (AltAssento / 2) + dhbraco
      fitPointsBrac(6) = mycenter(0) - (CompAssento / 2) + radiusBrac:
fitPointsBrac(7) = mycenter(1) + (LargAssento / 2) + radiusBrac:
fitPointsBrac(8) = (AltAssento / 2) + dhbraco
      fitPointsBrac(3)     =     fitPointsBrac(6):     fitPointsBrac(4)     =
fitPointsBrac(7): fitPointsBrac(5) = dhbraco
      Set splineObjBrac = ThisDrawing.ModelSpace.AddSpline(fitPointsBrac,
startTanBrac, endTanBrac)
centerPointBrac(0) = fitPointsBrac(0): centerPointBrac(1) = fitPointsBrac(1):
centerPointBrac(2) = fitPointsBrac(2)
```

```
Set curvesBrac(0) = ThisDrawing.ModelSpace.AddCircle(centerPointBrac,
radiusBrac)

regionObjBrac = ThisDrawing.ModelSpace.AddRegion(curvesBrac)

Set solidObjBrac =
ThisDrawing.ModelSpace.AddExtrudedSolidAlongPath(regionObjBrac(0),
splineObjBrac)

solidObjBrac.color = Pintura_Estrutura

' Sustentacao do braco1 da CadeiraFixa

center(0) = mycenter(0) - (CompAssento / 2) + RaioPernas: center(1) =
mycenter(1) - (LargAssento / 2) + RaioPernas: center(2) = mycenter(2) +
(AltAssento / 2) + (dhbraco / 2)

 width = RaioPernas: height = dhbraco

 Set Suporte1_Braco = ThisDrawing.ModelSpace.AddCylinder(center, width,
height)

 Suporte1_Braco.color = Pintura_Estrutura

'Braco2 da CadeiraFixa

 startTanBrac2(0) = 0: startTanBrac2(1) = 0: startTanBrac2(2) = 1

 endTanBrac2(0) = 0: endTanBrac2(1) = 0: endTanBrac2(2) = 1

 fitPointsBrac2(0) = mycenter(0) + (CompAssento / 2) - radiusBrac2:
fitPointsBrac2(1) = mycenter(1) - (LargAssento / 2) + radiusBrac2:
fitPointsBrac2(2) = (AltAssento / 2) + dhbraco

 fitPointsBrac2(6) = mycenter(0) + (CompAssento / 2) - radiusBrac2:
fitPointsBrac2(7) = mycenter(1) + (LargAssento / 2) + radiusBrac2:
fitPointsBrac2(8) = (AltAssento / 2) + dhbraco

  fitPointsBrac2(3) = fitPointsBrac2(6): fitPointsBrac2(4) =
fitPointsBrac2(7): fitPointsBrac2(5) = dhbraco

  Set splineObjBrac2 = ThisDrawing.ModelSpace.AddSpline(fitPointsBrac2,
startTanBrac2, endTanBrac2)

  centerPointBrac2(0) = fitPointsBrac2(0): centerPointBrac2(1) =
fitPointsBrac2(1): centerPointBrac2(2) = fitPointsBrac2(2)

   Set curvesBrac2(0) = ThisDrawing.ModelSpace.AddCircle(centerPointBrac2,
radiusBrac2)

   regionObjBrac2 = ThisDrawing.ModelSpace.AddRegion(curvesBrac2)

   Set solidObjBrac2 =
ThisDrawing.ModelSpace.AddExtrudedSolidAlongPath(regionObjBrac2(0),
splineObjBrac2)

    solidObjBrac2.color = Pintura_Estrutura

' Sustentacao do braco2 da CadeiraFixa

    center(0) = mycenter(0) + (CompAssento / 2) - RaioPernas: center(1) =
mycenter(1) - (LargAssento / 2) + RaioPernas: center(2) = mycenter(2) +
(AltAssento / 2) + (dhbraco / 2)

     width = RaioPernas: height = dhbraco

     Set Suporte2_Braco = ThisDrawing.ModelSpace.AddCylinder(center, width,
height)

     Suporte2_Braco.color = Pintura_Estrutura
```

```
End If
' CadeiraFixa com um braco apenas
        If UmBraco = True Then
         startTanBrac2(0) = 0: startTanBrac2(1) = 0: startTanBrac2(2) = 1
         endTanBrac2(0) = 0: endTanBrac2(1) = 0: endTanBrac2(2) = 1
         fitPointsBrac2(0) = mycenter(0) + (CompAssento / 2) - radiusBrac2:
fitPointsBrac2(1) = mycenter(1) - (LargAssento / 2) + radiusBrac2:
fitPointsBrac2(2) = (AltAssento / 2) + dhbraco
          fitPointsBrac2(6) = mycenter(0) + (CompAssento / 2) - radiusBrac2:
fitPointsBrac2(7) = mycenter(1) + (LargAssento / 2) + radiusBrac2:
fitPointsBrac2(8) = (AltAssento / 2) + dhbraco
          fitPointsBrac2(3) = fitPointsBrac2(6): fitPointsBrac2(4) =
fitPointsBrac2(7): fitPointsBrac2(5) = dhbraco
         Set splineObjBrac2 = ThisDrawing.ModelSpace.AddSpline(fitPointsBrac2,
startTanBrac2, endTanBrac2)
         centerPointBrac2(0) = fitPointsBrac2(0): centerPointBrac2(1) =
fitPointsBrac2(1): centerPointBrac2(2) = fitPointsBrac2(2)
         Set curvesBrac2(0) =
ThisDrawing.ModelSpace.AddCircle(centerPointBrac2, radiusBrac2)
         regionObjBrac2 = ThisDrawing.ModelSpace.AddRegion(curvesBrac2)
         Set                          solidObjBrac2                          =
ThisDrawing.ModelSpace.AddExtrudedSolidAlongPath(regionObjBrac2(0),
splineObjBrac2)
         solidObjBrac2.color = Pintura_Estrutura
' Sustentacao do braco2 da CadeiraFixa
         center(0) = mycenter(0) + (CompAssento / 2) - RaioPernas: center(1) =
mycenter(1) - (LargAssento / 2) + RaioPernas: center(2) = mycenter(2) +
(AltAssento / 2) + (dhbraco / 2)
          width = RaioPernas: height = dhbraco
          Set cylinder8 = ThisDrawing.ModelSpace.AddCylinder(center, width,
height)
         cylinder8.color = Pintura_Estrutura
            End If
    Update
        ZoomAll
End Function
Public Function Deletar_CadeiraFixa(desenho) As CadeiraFixa
Dim objeto As AcadObject
      For Each objeto In ThisDrawing.ModelSpace
 objeto.Delete
      Next
Update
End Function
```

INSTANCIAMENTO DA CLASSE CADEIRA FIXA

```
'-----------------------------------------------------------------
' UNICAMP - Universidade Estadual de Campinas
' FEC - Faculdade de Engenharia Civil, Arquitetura e Urbanismo
' 14/06/2006
'-----------------------------------------------------------------
Dim classnew As CadeiraFixa
Dim classdel As CadeiraFixa
Private Sub DeletaObjetos_Click()
    Set classdel = New CadeiraFixa
    Call classdel.Deletar_CadeiraFixa(desenho)
End Sub
Private Sub DoisBracos_Click()
    If DoisBracos = True Then
        UserForm1.UmBraco.Enabled = False
            Else
                UserForm1.UmBraco.Enabled = True
    End If
End Sub
Private Sub IntanciaClasse_Click()
    Set classnew = New CadeiraFixa
    Call classnew.Executar_CadeiraFixa(DoisBracos, UmBraco)
Update
End Sub
Private Sub Slider1_Click()
    Set classnew = New CadeiraFixa
    Call classnew.Executar_CadeiraFixa(DoisBracos, UmBraco)
End Sub
Private Sub Slider2_Click()
    Set classnew = New CadeiraFixa
    Call classnew.Executar_CadeiraFixa(DoisBracos, UmBraco)
End Sub
Private Sub Slider3_Click()
    Set classnew = New CadeiraFixa
    Call classnew.Executar_CadeiraFixa(DoisBracos, UmBraco)
End Sub
```

```
Private Sub Slider4_Click()
    Set classnew = New CadeiraFixa
    Call classnew.Executar_CadeiraFixa(DoisBracos, UmBraco)
End Sub
Private Sub Slider5_Click()
    Set classnew = New CadeiraFixa
    Call classnew.Executar_CadeiraFixa(DoisBracos, UmBraco)
End Sub
Private Sub UmBraco_Click()
    If UmBraco = True Then
        UserForm1.DoisBracos.Enabled = False
            Else
                UserForm1.DoisBracos.Enabled = True
    End If
End Sub
```

CLASSE ORDEM DÓRICA DIASTILO

```
'-------------------------------------------------------------
' UNICAMP - Universidade Estadual de Campinas
' FEC - Faculdade de Engenharia Civil, Arquitetura e Urbanismo
' 29/06/2006
'-------------------------------------------------------------
Public Function Executar_Diastilo(ColunasDiastilo, PropDiastilo As Double) As Diastilo
Dim entity As AcadEntity
    For Each entity In ThisDrawing.ModelSpace
        entity.delete
    Next
'PONTO DE INSERCAO DO TEMPLO DIASTILO
myCenter(0) = 0: myCenter(1) = 0: myCenter(2) = 0
'ATRIBUTOS DE COLUNA E ENTABLAMENTO
modulo = PropDiastilo                       'Modulo de proporcao do templo
AlturaFuste = modulo * 13                   'Altura do fuste
AlturaCollarino = modulo / 3                'Altura do Collarino
AlturaEquino = modulo / 3                   'Altura do equino
AlturaAbaco = modulo / 3                    'Altura do abaco
LarguraFuste = modulo * 2                   'Largura do fuste
LarguraCollarino = LarguraFuste             'Largura do collarino
intercolunio = LarguraFuste + (modulo * 5.5)         'Estabele o intercolunio livre entre as colunas
Altura_Coluna = AlturaFuste + AlturaCollarino + AlturaEquino + AlturaAbaco 'Altura total da coluna
AlturaArquitrave = modulo * 1               'Altura da arquitrave
AlturaMetopa = modulo * 1.5                 'Altura das metopas
AlturaTenia = AlturaArquitrave / 7          'Altura da tenia
AlturaCornija = modulo * 0.5                'Altura da cornija
difabaco = modulo / 6
AlturaFrontao = modulo * 3.5                'Altura do frontao
Altura_Total_Templo = Altura_Coluna + AlturaArquitrave + AlturaMetopa + AlturaCornija + AlturaFrontao 'Altura total do tempo
N_Coluna = ColunasDiastilo / 2                'Calcula o numero de colunas do templo
LarguraTriglifo = modulo * 1                'Largua dos triglifos
pontox = LarguraTriglifo / 2
LargMetopa = modulo * 1.5                   'Largura das metopas
'OBJETOS DA COLUNA
```

```
For i = 1 To N_Coluna
'Define a simetria do templo
    meioponto = ((modulo * 5.5 * (N_Coluna - 1)) + (LarguraFuste * N_Coluna) + modulo * 4)
    point1(0) = meioponto: point1(1) = 0: point1(2) = 0
    point2(0) = meioponto: point2(1) = 4: point2(2) = 0
' FUSTE
CoordFuste(0) = myCenter(0): CoordFuste(1) = myCenter(1): CoordFuste(2) = myCenter(2)
    CoordFuste(3) = myCenter(0): CoordFuste(4) = AlturaFuste: CoordFuste(5) = myCenter(2)
    CoordFuste(6) = myCenter(0) + LarguraFuste: CoordFuste(7) = AlturaFuste: CoordFuste(8) = myCenter(2)
    CoordFuste(9) = myCenter(0) + LarguraFuste: CoordFuste(10) = myCenter(1): CoordFuste(11) = myCenter(2)
    CoordFuste(12) = myCenter(0): CoordFuste(13) = myCenter(1): CoordFuste(14) = myCenter(2)
    Set Fuste = ThisDrawing.ModelSpace.AddPolyline(CoordFuste)
    Fuste.color = acByLayer
    Set mirrorObj = Fuste.Mirror(point1, point2)
    Fuste.Update
' COLLARINO
    Collarino(0) = myCenter(0): Collarino(1) = AlturaFuste: Collarino(2) = 0
    Collarino(3) = myCenter(0): Collarino(4) = AlturaFuste + AlturaCollarino: Collarino(5) = 0
    Collarino(6) = myCenter(0) + LarguraCollarino: Collarino(7) = AlturaFuste + AlturaCollarino: Collarino(8) = 0
    Collarino(9) = myCenter(0) + LarguraFuste: Collarino(10) = AlturaFuste: Collarino(11) = 0
    Collarino(12) = myCenter(0): Collarino(13) = AlturaFuste: Collarino(14) = 0
    Set CollarinoDiastilo = ThisDrawing.ModelSpace.AddPolyline(Collarino)
    CollarinoDiastilo.color = acByLayer
    Set mirrorObj = CollarinoDiastilo.Mirror(point1, point2)
    CollarinoDiastilo.Update
' EQUINO
    CoordEquino(0) = myCenter(0): CoordEquino(1) = AlturaFuste + AlturaCollarino: CoordEquino(2) = 0
    CoordEquino(3) = myCenter(0) - difabaco: CoordEquino(4) = AlturaFuste + AlturaCollarino + AlturaEquino: CoordEquino(5) = 0
    CoordEquino(6) = myCenter(0) + LarguraFuste + difabaco: CoordEquino(7) = AlturaFuste + AlturaCollarino + AlturaEquino: CoordEquino(8) = 0
    CoordEquino(9) = myCenter(0) + LarguraFuste: CoordEquino(10) = AlturaFuste + AlturaCollarino: CoordEquino(11) = 0
    CoordEquino(12) = myCenter(0): CoordEquino(13) = AlturaFuste + AlturaCollarino: CoordEquino(14) = 0
```

```
    Set Equino = ThisDrawing.ModelSpace.AddPolyline(CoordEquino)
    Equino.color = acByLayer
    Set mirrorObj = Equino.Mirror(point1, point2)
    Equino.Update
' ABACO
    CoordAbaco(0) = myCenter(0) - difabaco: CoordAbaco(1) = AlturaFuste + AlturaCollarino + AlturaEquino: CoordAbaco(2) = 0
    CoordAbaco(3) = myCenter(0) - difabaco: CoordAbaco(4) = Altura_Coluna: CoordAbaco(5) = 0
    CoordAbaco(6) = myCenter(0) + LarguraFuste + difabaco: CoordAbaco(7) = Altura_Coluna: CoordAbaco(8) = 0
    CoordAbaco(9) = myCenter(0) + LarguraFuste + difabaco: CoordAbaco(10) = AlturaFuste + AlturaCollarino + AlturaEquino: CoordAbaco(11) = 0
    CoordAbaco(12) = myCenter(0) - difabaco: CoordAbaco(13) = AlturaFuste + AlturaCollarino + AlturaEquino: CoordAbaco(14) = 0
    Set Abaco = ThisDrawing.ModelSpace.AddPolyline(CoordAbaco)
    Abaco.color = acByLayer
    Set mirrorObj = Abaco.Mirror(point1, point2)
    Abaco.Update
    myCenter(0) = myCenter(0) + intercolunio
            Set hachura(0) = Fuste
            Set                              objHatch                              = ThisDrawing.ModelSpace.AddHatch(acHatchPatternTypePreDefined, "ANSI31", True)
            objHatch.PatternAngle = 0.7853981 '45 graus
            objHatch.PatternScale = 0.1
            objHatch.AppendOuterLoop (hachura)
            objHatch.Evaluate
            objHatch.color = acByLayer
            Set mirrorObj2 = objHatch.Mirror(point1, point2)
            objHatch.Update
Next i
'OBJETOS DA CREPIDOMA
' ESTILOBATO
    CoordEstilobato(0) = 0 - modulo: CoordEstilobato(1) = 0: CoordEstilobato(2) = 0
    CoordEstilobato(3) = meioponto: CoordEstilobato(4) = 0: CoordEstilobato(5) = 0
    Set Estilobato = ThisDrawing.ModelSpace.AddPolyline(CoordEstilobato)
    Estilobato.color = acByLayer
    Estilobato.Update
'OBJETOS DO ENTABLAMENTO
```

```
' ARQUITRAVE
    CoordArquitrave(0) = meioponto: CoordArquitrave(1) = Altura_Coluna: CoordArquitrave(2) = 0
    CoordArquitrave(3) = 0: CoordArquitrave(4) = Altura_Coluna: CoordArquitrave(5) = 0
    CoordArquitrave(6) = 0: CoordArquitrave(7) = Altura_Coluna + AlturaArquitrave: CoordArquitrave(8) = 0
    CoordArquitrave(9) = meioponto: CoordArquitrave(10) = Altura_Coluna + AlturaArquitrave: CoordArquitrave(11) = 0
    Set Arquitrave = ThisDrawing.ModelSpace.AddPolyline(CoordArquitrave)
    Arquitrave.color = acByLayer
    Arquitrave.Update
' TENIA
    CoordTenia(0) = 0: CoordTenia(1) = Altura_Coluna + AlturaArquitrave - AlturaTenia: CoordTenia(2) = 0
    Coord_Final_Tenia(0) = meioponto: Coord_Final_Tenia(1) = Altura_Coluna + AlturaArquitrave - AlturaTenia: Coord_Final_Tenia(2) = 0
    Set Tenia = ThisDrawing.ModelSpace.AddLine(CoordTenia, Coord_Final_Tenia)
    Tenia.color = acByLayer
' METOPAS
    CoordMetopa(0) = meioponto: CoordMetopa(1) = Altura_Coluna + AlturaArquitrave: CoordMetopa(2) = 0
    CoordMetopa(3) = 0: CoordMetopa(4) = Altura_Coluna + AlturaArquitrave: CoordMetopa(5) = 0
    CoordMetopa(6) = 0: CoordMetopa(7) = Altura_Coluna + AlturaArquitrave + AlturaMetopa: CoordMetopa(8) = 0
    CoordMetopa(9) = meioponto: CoordMetopa(10) = Altura_Coluna + AlturaArquitrave + AlturaMetopa: CoordMetopa(11) = 0
    Set Metopa = ThisDrawing.ModelSpace.AddPolyline(CoordMetopa)
    Metopa.color = acByLayer
    Metopa.Update
' CORNIJA
    CoordCornija(0) = meioponto: CoordCornija(1) = Altura_Coluna + AlturaArquitrave + AlturaMetopa: CoordCornija(2) = 0
    CoordCornija(3) = 0 - modulo: CoordCornija(4) = Altura_Coluna + AlturaArquitrave + AlturaMetopa: CoordCornija(5) = 0
    CoordCornija(6) = 0 - modulo: CoordCornija(7) = Altura_Coluna + AlturaArquitrave + AlturaMetopa + AlturaCornija: CoordCornija(8) = 0
    CoordCornija(9) = meioponto: CoordCornija(10) = Altura_Coluna + AlturaArquitrave + AlturaMetopa + AlturaCornija: CoordCornija(11) = 0
    Set Cornija = ThisDrawing.ModelSpace.AddPolyline(CoordCornija)
    Cornija.color = acByLayer
    Cornija.Update
' FRONTAO
    CoordFrontao(0) = meioponto: CoordFrontao(1) = Altura_Total_Templo: CoordFrontao(2) = 0
```

```
    CoordFrontao(3) = 0 - modulo: CoordFrontao(4) = Altura_Coluna + AlturaArquitrave + AlturaMetopa + AlturaCornija: CoordFrontao(5) = 0
    CoordFrontao(6) = 0 - modulo: CoordFrontao(7) = Altura_Coluna + AlturaArquitrave + AlturaMetopa + AlturaCornija + modulo / 2: CoordFrontao(8) = 0
    CoordFrontao(9) = meioponto: CoordFrontao(10) = Altura_Total_Templo + modulo / 2: CoordFrontao(11) = 0
    Set Frontao = ThisDrawing.ModelSpace.AddPolyline(CoordFrontao)
    Frontao.color = acByLayer
    Frontao.Update
'Espelha os objetos de composicao do templo para estabelecer sua simetria
    Set mirrorObj = Estilobato.Mirror(point1, point2)
    Set mirrorObj = Frontao.Mirror(point1, point2)
    Set mirrorObj = Arquitrave.Mirror(point1, point2)
    Set mirrorObj = Metopa.Mirror(point1, point2)
    Set mirrorObj = Cornija.Mirror(point1, point2)
    Set mirrorObj3 = Tenia.Mirror(point1, point2)
If ColunasDiastilo = 4 Then
    For j = 1 To 11
'TRIGLIFOS
            CoordTriglifos(0) = pontox: CoordTriglifos(1) = Altura_Coluna + AlturaArquitrave: CoordTriglifos(2) = 0
            CoordTriglifos(3) = pontox: CoordTriglifos(4) = Altura_Coluna + AlturaArquitrave + AlturaMetopa: CoordTriglifos(5) = 0
            CoordTriglifos(6) = pontox + LarguraTriglifo: CoordTriglifos(7) = Altura_Coluna + AlturaArquitrave + AlturaMetopa: CoordTriglifos(8) = 0
            CoordTriglifos(9) = pontox + LarguraTriglifo: CoordTriglifos(10) = Altura_Coluna + AlturaArquitrave: CoordTriglifos(11) = 0
            CoordTriglifos(12) = pontox: CoordTriglifos(13) = Altura_Coluna + AlturaArquitrave: CoordTriglifos(14) = 0
            Set Triglifos = ThisDrawing.ModelSpace.AddPolyline(CoordTriglifos)
            Triglifos.color = acByLayer
            Triglifos.Update
                pontox = pontox + LarguraTriglifo + LargMetopa
                Set hachura(0) = Triglifos
                Set objHatch = ThisDrawing.ModelSpace.AddHatch(acHatchPatternTypePreDefined, "ANSI31", True)
                objHatch.PatternAngle = 0.7853981 '45 graus
                objHatch.PatternScale = 0.1
                objHatch.AppendOuterLoop (hachura)
                objHatch.Evaluate
                objHatch.color = acByLayer
```

```
                objHatch.Update
Next j
        ElseIf ColunasDiastilo = 6 Then
           For w = 1 To 17
' TRIGLIFOS
                CoordTriglifos(0) = pontox: CoordTriglifos(1) = Altura_Coluna
+ AlturaArquitrave: CoordTriglifos(2) = 0
                CoordTriglifos(3) = pontox: CoordTriglifos(4) = Altura_Coluna
+ AlturaArquitrave + AlturaMetopa: CoordTriglifos(5) = 0
                CoordTriglifos(6)     =     pontox     +     LarguraTriglifo:
CoordTriglifos(7)   =   Altura_Coluna   +   AlturaArquitrave   +   AlturaMetopa:
CoordTriglifos(8) = 0
                CoordTriglifos(9)     =     pontox     +     LarguraTriglifo:
CoordTriglifos(10) = Altura_Coluna + AlturaArquitrave: CoordTriglifos(11) = 0
                CoordTriglifos(12)    =    pontox:    CoordTriglifos(13)    =
Altura_Coluna + AlturaArquitrave: CoordTriglifos(14) = 0
                Set                          Triglifos                        =
ThisDrawing.ModelSpace.AddPolyline(CoordTriglifos)
                Triglifos.color = acByLayer
                Triglifos.Update
                pontox = pontox + LarguraTriglifo + LargMetopa
                Set hachura(0) = Triglifos
                Set                          objHatch                         =
ThisDrawing.ModelSpace.AddHatch(acHatchPatternTypePreDefined, "ANSI31", True)
                objHatch.PatternAngle = 0.7853981 '45 graus
                objHatch.PatternScale = 0.1
                objHatch.AppendOuterLoop (hachura)
                objHatch.Evaluate
                objHatch.color = acByLayer
                objHatch.Update
            Next w
End If
    ThisDrawing.Application.ZoomExtents
End Function
Public Function Deletar_Diastilo() As Diastilo
   For Each objeto In ThisDrawing.ModelSpace
             objeto.delete
    Next
Update
End Function
'CLASSE ORDEM DÓRICA SISTILO
Public  Function  Executar_Sistilo(ColunasSistilo,  PropSistilo  As  Double)  As
Sistilo
Dim entity As AcadEntity
```

```
    For Each entity In ThisDrawing.ModelSpace
        entity.delete
    Next
modulo = PropSistilo
'PONTO DE INSERCAO DO TEMPLO SISTILO
myCenter(0) = 0: myCenter(1) = 0: myCenter(2) = 0
'ATRIBUTOS DE COLUNA E ENTABLAMENTO
AlturaFuste = modulo * 13                    'Altura do fuste
AlturaCollarino = modulo / 3                 'Altura do Collarino
AlturaEquino = modulo / 3                    'Altura do equino
AlturaAbaco = modulo / 3                     'Altura do abaco
LarguraFuste = modulo * 2                    'Largura do fuste
LarguraCollarino = LarguraFuste              'Largura do collarino
intercolunio = LarguraFuste + (modulo * 3)           'Estabele o intercolunio
livre entre as colunas
Altura_Coluna = AlturaFuste + AlturaCollarino + AlturaEquino + AlturaAbaco
AlturaArquitrave = modulo * 1                'Altura da arquitrave
AlturaMetopa = modulo * 1.5                  'Altura metopa
AlturaTenia = AlturaArquitrave / 7           'Altura tenia
AlturaCornija = modulo * 0.5                 'Altura hcornija
difabaco = modulo / 6
AlturaFrontao = modulo * 3.5                 'Altura do frontao
Altura_Total_Templo  =  Altura_Coluna  +  AlturaArquitrave  +  AlturaMetopa  +
AlturaCornija + AlturaFrontao
N_Coluna = ColunasSistilo / 2                  'Calcula o numero de colunas do
templo
LarguraTriglifo = modulo * 1                 'Largua dos triglifos
pontox = LarguraTriglifo / 2
LargMetopa = modulo * 1.5                    'Largura das metopas
'OBJETOS DA COLUNA
For i = 1 To N_Coluna
'Define a simetria do templo sistilo
    meioponto = ((modulo * 3 * (N_Coluna - 1)) + (LarguraFuste * N_Coluna) +
modulo * 2.75)
    point1(0) = meioponto: point1(1) = 0: point1(2) = 0
    point2(0) = meioponto: point2(1) = 4: point2(2) = 0
' FUSTE
    CoordFuste(0) = myCenter(0): CoordFuste(1) = myCenter(1): CoordFuste(2) =
myCenter(2)
```

```
    CoordFuste(3) = myCenter(0): CoordFuste(4) = AlturaFuste: CoordFuste(5) =
myCenter(2)
    CoordFuste(6) = myCenter(0) + LarguraFuste: CoordFuste(7) = AlturaFuste:
CoordFuste(8) = myCenter(2)
    CoordFuste(9) = myCenter(0) + LarguraFuste: CoordFuste(10) = myCenter(1):
CoordFuste(11) = myCenter(2)
    CoordFuste(12) = myCenter(0): CoordFuste(13) = myCenter(1): CoordFuste(14)
= myCenter(2)
    Set Fuste = ThisDrawing.ModelSpace.AddPolyline(CoordFuste)
    Fuste.color = acByLayer
    Set mirrorObj = Fuste.Mirror(point1, point2)
    Fuste.Update
' COLLARINO
    Collarino(0) = myCenter(0): Collarino(1) = AlturaFuste: Collarino(2) = 0
    Collarino(3) = myCenter(0): Collarino(4) = AlturaFuste + AlturaCollarino:
Collarino(5) = 0
    Collarino(6) = myCenter(0) + LarguraCollarino: Collarino(7) = AlturaFuste
+ AlturaCollarino: Collarino(8) = 0
    Collarino(9) = myCenter(0) + LarguraFuste: Collarino(10) = AlturaFuste:
Collarino(11) = 0
    Collarino(12) = myCenter(0): Collarino(13) = AlturaFuste: Collarino(14) =
0
    Set CollarinoDiastilo = ThisDrawing.ModelSpace.AddPolyline(Collarino)
    CollarinoDiastilo.color = acByLayer
    Set mirrorObj = CollarinoDiastilo.Mirror(point1, point2)
    CollarinoDiastilo.Update
'EQUINO
    CoordEquino(0) = myCenter(0): CoordEquino(1) = AlturaFuste +
AlturaCollarino: CoordEquino(2) = 0
    CoordEquino(3) = myCenter(0) - difabaco: CoordEquino(4) = AlturaFuste +
AlturaCollarino + AlturaEquino: CoordEquino(5) = 0
    CoordEquino(6) = myCenter(0) + LarguraFuste + difabaco: CoordEquino(7) =
AlturaFuste + AlturaCollarino + AlturaEquino: CoordEquino(8) = 0
    CoordEquino(9) = myCenter(0) + LarguraFuste: CoordEquino(10) = AlturaFuste
+ AlturaCollarino: CoordEquino(11) = 0
    CoordEquino(12) = myCenter(0): CoordEquino(13) = AlturaFuste +
AlturaCollarino: CoordEquino(14) = 0
    Set Equino = ThisDrawing.ModelSpace.AddPolyline(CoordEquino)
    Equino.color = acByLayer
    Set mirrorObj = Equino.Mirror(point1, point2)
    Equino.Update
'ABACO
    CoordAbaco(0) = myCenter(0) - difabaco: CoordAbaco(1) = AlturaFuste +
AlturaCollarino + AlturaEquino: CoordAbaco(2) = 0
    CoordAbaco(3) = myCenter(0) - difabaco: CoordAbaco(4) = Altura_Coluna:
CoordAbaco(5) = 0
```

```
    CoordAbaco(6) = myCenter(0) + LarguraFuste + difabaco: CoordAbaco(7) =
Altura_Coluna: CoordAbaco(8) = 0
    CoordAbaco(9) = myCenter(0) + LarguraFuste + difabaco: CoordAbaco(10) =
AlturaFuste + AlturaCollarino + AlturaEquino: CoordAbaco(11) = 0
    CoordAbaco(12) = myCenter(0) - difabaco: CoordAbaco(13) = AlturaFuste +
AlturaCollarino + AlturaEquino: CoordAbaco(14) = 0
    Set Abaco = ThisDrawing.ModelSpace.AddPolyline(CoordAbaco)
    Abaco.color = acByLayer
    Set mirrorObj = Abaco.Mirror(point1, point2)
    Abaco.Update
    myCenter(0) = myCenter(0) + intercolunio
            Set hachura(0) = Fuste
            Set                          objHatch                          =
ThisDrawing.ModelSpace.AddHatch(acHatchPatternTypePreDefined, "ANSI31", True)
            objHatch.PatternAngle = 0.7853981 '45 graus
            objHatch.PatternScale = 0.1
            objHatch.AppendOuterLoop (hachura)
            objHatch.Evaluate
            objHatch.color = acByLayer
            Set mirrorObj2 = objHatch.Mirror(point1, point2)
            objHatch.Update
Next i
'OBJETOS DA CREPIDOMA
' ESTILOBATO
    CoordEstilobato(0)   =   0   -   modulo:   CoordEstilobato(1)   =   0:
CoordEstilobato(2) = 0
    CoordEstilobato(3) = meioponto: CoordEstilobato(4) = 0: CoordEstilobato(5)
= 0
    Set Estilobato = ThisDrawing.ModelSpace.AddPolyline(CoordEstilobato)
    Estilobato.color = acByLayer
    Estilobato.Update
'OBJETOS DO ENTABLAMENTO
' ARQUITRAVE
    CoordArquitrave(0)  =  meioponto:  CoordArquitrave(1)  =  Altura_Coluna:
CoordArquitrave(2) = 0
    CoordArquitrave(3)   =   0:   CoordArquitrave(4)   =   Altura_Coluna:
CoordArquitrave(5) = 0
    CoordArquitrave(6)   =   0:   CoordArquitrave(7)   =   Altura_Coluna   +
AlturaArquitrave: CoordArquitrave(8) = 0
    CoordArquitrave(9)  =  meioponto:  CoordArquitrave(10)  =  Altura_Coluna  +
AlturaArquitrave: CoordArquitrave(11) = 0
    Set Arquitrave = ThisDrawing.ModelSpace.AddPolyline(CoordArquitrave)
```

```
    Arquitrave.color = acByLayer
    Arquitrave.Update
' TENIA
    CoordTenia(0) = 0: CoordTenia(1) = Altura_Coluna + AlturaArquitrave - AlturaTenia: CoordTenia(2) = 0
    Coord_Final_Tenia(0) = meioponto: Coord_Final_Tenia(1) = Altura_Coluna + AlturaArquitrave - AlturaTenia: Coord_Final_Tenia(2) = 0
    Set Tenia = ThisDrawing.ModelSpace.AddLine(CoordTenia, Coord_Final_Tenia)
    Tenia.color = acByLayer
'METOPA
    CoordMetopa(0) = meioponto: CoordMetopa(1) = Altura_Coluna + AlturaArquitrave: CoordMetopa(2) = 0
    CoordMetopa(3) = 0: CoordMetopa(4) = Altura_Coluna + AlturaArquitrave: CoordMetopa(5) = 0
    CoordMetopa(6) = 0: CoordMetopa(7) = Altura_Coluna + AlturaArquitrave + AlturaMetopa: CoordMetopa(8) = 0
    CoordMetopa(9) = meioponto: CoordMetopa(10) = Altura_Coluna + AlturaArquitrave + AlturaMetopa: CoordMetopa(11) = 0
    Set Metopa = ThisDrawing.ModelSpace.AddPolyline(CoordMetopa)
    Metopa.color = acByLayer
    Metopa.Update
'CORNIJA
    CoordCornija(0) = meioponto: CoordCornija(1) = Altura_Coluna + AlturaArquitrave + AlturaMetopa: CoordCornija(2) = 0
    CoordCornija(3) = 0 - modulo: CoordCornija(4) = Altura_Coluna + AlturaArquitrave + AlturaMetopa: CoordCornija(5) = 0
    CoordCornija(6) = 0 - modulo: CoordCornija(7) = Altura_Coluna + AlturaArquitrave + AlturaMetopa + AlturaCornija: CoordCornija(8) = 0
    CoordCornija(9) = meioponto: CoordCornija(10) = Altura_Coluna + AlturaArquitrave + AlturaMetopa + AlturaCornija: CoordCornija(11) = 0
    Set Cornija = ThisDrawing.ModelSpace.AddPolyline(CoordCornija)
    Cornija.color = acByLayer
    Cornija.Update
' FRONTAO
    CoordFrontao(0) = meioponto: CoordFrontao(1) = Altura_Total_Templo: CoordFrontao(2) = 0
    CoordFrontao(3) = 0 - modulo: CoordFrontao(4) = Altura_Coluna + AlturaArquitrave + AlturaMetopa + AlturaCornija: CoordFrontao(5) = 0
    CoordFrontao(6) = 0 - modulo: CoordFrontao(7) = Altura_Coluna + AlturaArquitrave + AlturaMetopa + AlturaCornija + modulo / 2: CoordFrontao(8) = 0
    CoordFrontao(9) = meioponto: CoordFrontao(10) = Altura_Total_Templo + modulo / 2: CoordFrontao(11) = 0
    Set Frontao = ThisDrawing.ModelSpace.AddPolyline(CoordFrontao)
    Frontao.color = acByLayer
    Frontao.Update
```

```
'Espelha os objetos de composicao do templo para estabelecer sua simetria
        Set mirrorObj = Estilobato.Mirror(point1, point2)
        Set mirrorObj = Frontao.Mirror(point1, point2)
        Set mirrorObj = Arquitrave.Mirror(point1, point2)
        Set mirrorObj = Metopa.Mirror(point1, point2)
        Set mirrorObj = Cornija.Mirror(point1, point2)
        Set mirrorObj3 = Tenia.Mirror(point1, point2)
If ColunasSistilo = 4 Then
            For j = 1 To 8
' TRIGLIFOS
                CoordTriglifos(0) = pontox: CoordTriglifos(1) = Altura_Coluna
+ AlturaArquitrave: CoordTriglifos(2) = 0
                CoordTriglifos(3) = pontox: CoordTriglifos(4) = Altura_Coluna
+ AlturaArquitrave + AlturaMetopa: CoordTriglifos(5) = 0
                CoordTriglifos(6)     =     pontox     +     LarguraTriglifo:
CoordTriglifos(7)   =   Altura_Coluna   +   AlturaArquitrave   +   AlturaMetopa:
CoordTriglifos(8) = 0
                CoordTriglifos(9)     =     pontox     +     LarguraTriglifo:
CoordTriglifos(10) = Altura_Coluna + AlturaArquitrave: CoordTriglifos(11) = 0
                CoordTriglifos(12)    =    pontox:    CoordTriglifos(13)    =
Altura_Coluna + AlturaArquitrave: CoordTriglifos(14) = 0
                Set                         Triglifos                         =
ThisDrawing.ModelSpace.AddPolyline(CoordTriglifos)
                Triglifos.color = acByLayer
                Triglifos.Update
                pontox = pontox + LarguraTriglifo + LargMetopa
                        Set hachura(0) = Triglifos
                        Set                      objHatch                      =
ThisDrawing.ModelSpace.AddHatch(acHatchPatternTypePreDefined, "ANSI31", True)
                        objHatch.PatternAngle = 0.7853981 '45 graus
                        objHatch.PatternScale = 0.1
                        objHatch.AppendOuterLoop (hachura)
                        objHatch.Evaluate
                        objHatch.color = acByLayer
                        objHatch.Update
            Next j
                ElseIf ColunasSistilo = 6 Then
                    For w = 1 To 12
                        ' TRIGLIFOS
                            CoordTriglifos(0)  =  pontox:  CoordTriglifos(1)  =
Altura_Coluna + AlturaArquitrave: CoordTriglifos(2) = 0
```

```
                            CoordTriglifos(3) = pontox: CoordTriglifos(4) =
Altura_Coluna + AlturaArquitrave + AlturaMetopa: CoordTriglifos(5) = 0
                            CoordTriglifos(6) = pontox + LarguraTriglifo:
CoordTriglifos(7) = Altura_Coluna + AlturaArquitrave + AlturaMetopa:
CoordTriglifos(8) = 0
                            CoordTriglifos(9) = pontox + LarguraTriglifo:
CoordTriglifos(10) = Altura_Coluna + AlturaArquitrave: CoordTriglifos(11) = 0
                            CoordTriglifos(12) = pontox: CoordTriglifos(13) =
Altura_Coluna + AlturaArquitrave: CoordTriglifos(14) = 0
                            Set                  Triglifos                  =
ThisDrawing.ModelSpace.AddPolyline(CoordTriglifos)
                            Triglifos.color = acByLayer
                            Triglifos.Update
                            pontox = pontox + LarguraTriglifo + LargMetopa
                            Set hachura(0) = Triglifos
                            Set                   objHatch                   =
ThisDrawing.ModelSpace.AddHatch(acHatchPatternTypePreDefined, "ANSI31", True)
                            objHatch.PatternAngle = 0.7853981 '45 graus
                            objHatch.PatternScale = 0.1
                            objHatch.AppendOuterLoop (hachura)
                            objHatch.Evaluate
                            objHatch.color = acByLayer
                            objHatch.Update
                    Next w
End If
    ThisDrawing.Application.ZoomExtents
End Function

Public Function Deletar_Sistilo() As Sistilo
    For Each objeto In ThisDrawing.ModelSpace
             objeto.delete
    Next
Update
End Function
```

INSTANCIAMENTO DA CLASSE ORDEM DÓRICA DIASTILO E SISTILO

```
'-----------------------------------------------------------------
' UNICAMP - Universidade Estadual de Campinas
' FEC - Faculdade de Engenharia Civil, Arquitetura e Urbanismo
' 29/06/2006
'-----------------------------------------------------------------
Dim Diastilo As Diastilo
Dim Sistilo As Sistilo
Dim ClassDel As Diastilo
Dim Del As Sistilo
Private Sub Deletar_Diastilo_Click()
Set ClassDel = New Diastilo
    Call ClassDel.Deletar_Diastilo
    Update
End Sub
Private Sub Deletar_Sistilo_Click()
Set Del = New Sistilo
    Call Del.Deletar_Sistilo
    Update
End Sub
Private Sub Executar_Diastilo_Click()
 If MultiPage1.Page1.Caption = "Diastilo" Then
        If ColunasDiastilo = "" Or PropDiastilo = "" Then
            UserForm1.Hide
            MsgBox   "Por   favor,   selecione   os   valores.",   vbOKOnly   +
vbExclamation, "Alerta!"
            UserForm1.Show
        ElseIf ColunasDiastilo <> 4 And ColunasDiastilo <> 6 Then
                UserForm1.Hide
                MsgBox "Por  favor,  selecione  4  ou  6  colunas.",  vbOKOnly  +
vbExclamation, "Alerta!"
                UserForm1.Show
            Else
                    Set Diastilo = New Diastilo
                    Call           Diastilo.Executar_Diastilo(ColunasDiastilo,
PropDiastilo)
        End If
```

```
End If
End Sub
Private Sub Executar_Sistilo_Click()
      If MultiPage1.Page2.Caption = "Sistilo" Then
        If ColunasSistilo = "" Or PropSistilo = "" Then
            UserForm1.Hide
            MsgBox "Por favor, selecione os valores.", vbOKOnly + vbExclamation, "Alerta!"
            UserForm1.Show
        ElseIf ColunasSistilo <> 4 And ColunasSistilo <> 6 Then
            UserForm1.Hide
            MsgBox "Por favor, selecione 4 ou 6 colunas.", vbOKOnly + vbExclamation, "Alerta!"
            UserForm1.Show
           Else
            Set Sistilo = New Sistilo
            Call Sistilo.Executar_Sistilo(ColunasSistilo, PropSistilo)
        End If
End If
End Sub
Private Sub UserForm_Initialize()
        ColunasDiastilo.AddItem "4"
        ColunasDiastilo.AddItem "6"
        PropDiastilo.AddItem "1"
        PropDiastilo.AddItem "3"
        ColunasSistilo.AddItem "4"
        ColunasSistilo.AddItem "6"
        PropSistilo.AddItem "1"
        PropSistilo.AddItem "3"
End Sub
```

APÊNDICE B

Este apêndice consiste na documentação do processo de projeto computacional desenvolvido com auxilio do paradigma da orientação a objetos. O *script* apresentado a seguir refere-se ao protótipo final de uma tipologia residencial desenvolvido através da linguagem de programação *visual basic for applications* (VBA) do AutoCAD.

CLASSE CASA

```
'---------------------------------------------------------------
' UNICAMP - Universidade Estadual de Campinas
' FEC - Faculdade de Engenharia Civil, Arquitetura e Urbanismo
' 14/05/2007
'---------------------------------------------------------------

Public Function DesenhaComodos(SalaCozI, SalaCozS, Dorm1, Dorm2, Dorm3, Dorm4,
Varanda, Aservico, Hall, Suite1, Suite2, Suite3, Suite4) As Casa
On Error Resume Next
'Sala
mycenter(0) = 0: mycenter(1) = 0: mycenter(2) = 0
modulo = 1
CSala = 6 * modulo
LSala = 4 * modulo
CCoz = 3 * modulo
LCoz = 3 * modulo
CVaranda = 2.5 * modulo
LVaranda = LSala
CServico = 2 * modulo
LServico = 3 * modulo
Nquartos = 0
```

```
'Contagem do numero de dormitorios
    If Dorm1.Value = True Then
        Nquartos = Nquartos + 1
    End If
    If Dorm2.Value = True Then
        Nquartos = Nquartos + 1
    End If
    If Dorm3.Value = True Then
        Nquartos = Nquartos + 1
    End If
    If Dorm4.Value = True Then
        Nquartos = Nquartos + 1
    End If
If Nquartos = 0 Then
'Sala e cozinha separada
    If SalaCozS = True Then
    InserirSalaCozI(1) = mycenter(0): InserirSalaCozI(2) = mycenter(1): InserirSalaCozI(3) = mycenter(2)
    InserirSalaCozI(4) = mycenter(0) + LSala: InserirSalaCozI(5) = mycenter(1): InserirSalaCozI(6) = 0
    InserirSalaCozI(7) = mycenter(0) + LSala:: InserirSalaCozI(8) = mycenter(1) + CSala: InserirSalaCozI(9) = 0
    InserirSalaCozI(10) = mycenter(0): InserirSalaCozI(11) = mycenter(1) + CSala: InserirSalaCozI(12) = 0
    InserirSalaCozI(13) = mycenter(0): InserirSalaCozI(14) = mycenter(1): InserirSalaCozI(15) = mycenter(2)
    Set SalaCozI = ThisDrawing.ModelSpace.AddPolyline(InserirSalaCozI)
    SalaCozI.color = 64
    SalaCozI.Update
                Set hachura(0) = SalaCozI
                Set                          objHatch                          = ThisDrawing.ModelSpace.AddHatch(acHatchPatternTypePreDefined, "SOLID", True)
                objHatch.PatternScale = 0.1
                objHatch.AppendOuterLoop (hachura)
                objHatch.Evaluate
                objHatch.color = 64
                objHatch.Update
'Cozinha
    InserirCoz(1) = mycenter(0): InserirCoz(2) = mycenter(1) + CServico: InserirCoz(3) = 0
    InserirCoz(4) = mycenter(0): InserirCoz(5) = mycenter(1) + CServico + CCoz: InserirCoz(6) = 0
    InserirCoz(7) = mycenter(0) - LCoz: InserirCoz(8) = mycenter(1) + CServico + CCoz: InserirCoz(9) = 0
```

```
    InserirCoz(10) = mycenter(0) - LCoz: InserirCoz(11) = mycenter(1) + CServico: InserirCoz(12) = 0
    InserirCoz(13) = mycenter(0): InserirCoz(14) = mycenter(1) + CServico: InserirCoz(15) = 0
    Set Coz = ThisDrawing.ModelSpace.AddPolyline(InserirCoz)
    Coz.color = acBlue
    Coz.Update
                Set hachura(0) = Coz
                Set                          objHatch                          = ThisDrawing.ModelSpace.AddHatch(acHatchPatternTypePreDefined, "SOLID", True)
                objHatch.PatternScale = 0.1
                objHatch.AppendOuterLoop (hachura)
                objHatch.Evaluate
                objHatch.color = acBlue
                objHatch.Update
'Sala e cozinha integrada
    ElseIf SalaCozI = True Then
    CCoz = 2 * modulo
    LCoz = 2.5 * modulo
    InserirSalaCozI(1) = mycenter(0): InserirSalaCozI(2) = mycenter(1): InserirSalaCozI(3) = mycenter(2)
    InserirSalaCozI(4) = mycenter(0) + LSala: InserirSalaCozI(5) = mycenter(1): InserirSalaCozI(6) = 0
    InserirSalaCozI(7) = mycenter(0) + LSala:: InserirSalaCozI(8) = mycenter(1) + CSala: InserirSalaCozI(9) = 0
    InserirSalaCozI(10) = mycenter(0): InserirSalaCozI(11) = mycenter(1) + CSala: InserirSalaCozI(12) = 0
    InserirSalaCozI(13) = mycenter(0): InserirSalaCozI(14) = mycenter(1): InserirSalaCozI(15) = mycenter(2)
    Set SalaCozI = ThisDrawing.ModelSpace.AddPolyline(InserirSalaCozI)
    SalaCozI.color = 64
    SalaCozI.Update
                Set hachura(0) = SalaCozI
                Set                          objHatch                          = ThisDrawing.ModelSpace.AddHatch(acHatchPatternTypePreDefined, "SOLID", True)
                objHatch.PatternScale = 0.1
                objHatch.AppendOuterLoop (hachura)
                objHatch.Evaluate
                objHatch.color = 64
                objHatch.Update
'Cozinha
```

```
    InserirCoz(1) = mycenter(0): InserirCoz(2) = mycenter(1): InserirCoz(3) =
0
    InserirCoz(4)  =  mycenter(0):  InserirCoz(5)  =  mycenter(1)  +  CCoz:
InserirCoz(6) = 0
    InserirCoz(7) = mycenter(0) + LCoz: InserirCoz(8) = mycenter(1) + CCoz:
InserirCoz(9) = 0
    InserirCoz(10)  =  mycenter(0)  +  LCoz:  InserirCoz(11)  =  mycenter(1):
InserirCoz(12) = 0
    InserirCoz(13) = mycenter(0): InserirCoz(14) = mycenter(1): InserirCoz(15)
= 0
    Set Coz = ThisDrawing.ModelSpace.AddPolyline(InserirCoz)
    Coz.color = acBlue
    Coz.Update
                Set hachura(0) = Coz
                Set                          objHatch                          =
ThisDrawing.ModelSpace.AddHatch(acHatchPatternTypePreDefined, "PLASTI", True)
                objHatch.PatternAngle = 0.7853981 '45 graus
                objHatch.PatternScale = 0.07
                objHatch.AppendOuterLoop (hachura)
                objHatch.Evaluate
                objHatch.color = acBlue
                objHatch.Update
    End If
'Varanda
    If Varanda = True Then
    InserirVaranda(1) = mycenter(0): InserirVaranda(2) = mycenter(1) + CSala:
InserirVaranda(3) = mycenter(2)
    InserirVaranda(4) = mycenter(0) + LSala: InserirVaranda(5) = mycenter(1) +
CSala: InserirVaranda(6) = 0
    InserirVaranda(7) = mycenter(0) + LSala: InserirVaranda(8) = mycenter(1) +
CSala + CVaranda: InserirVaranda(9) = 0
    InserirVaranda(10) = mycenter(0): InserirVaranda(11) = mycenter(1) + CSala
+ CVaranda: InserirVaranda(12) = 0
    InserirVaranda(13)  =  mycenter(0):  InserirVaranda(14)  =  mycenter(1)  +
CSala: InserirVaranda(15) = mycenter(2)
    Set Varanda = ThisDrawing.ModelSpace.AddPolyline(InserirVaranda)
    Varanda.color = 66
    Varanda.Update
                Set hachura(0) = Varanda
                Set                          objHatch                          =
ThisDrawing.ModelSpace.AddHatch(acHatchPatternTypePreDefined, "SOLID", True)
                objHatch.PatternScale = 0.1
                objHatch.AppendOuterLoop (hachura)
                objHatch.Evaluate
                objHatch.color = 66
```

```
                objHatch.Update
    End If
'Area de servico
    If Aservico = True Then
    InserirServico(1) = mycenter(0): InserirServico(2) = mycenter(1): InserirServico(3) = mycenter(2)
    InserirServico(4) = mycenter(0): InserirServico(5) = mycenter(1) + CServico: InserirServico(6) = 0
    InserirServico(7) = mycenter(0) - LCoz: InserirServico(8) = mycenter(1) + CServico: InserirServico(9) = 0
    InserirServico(10) = mycenter(0) - LCoz: InserirServico(11) = mycenter(1): InserirServico(12) = 0
    InserirServico(13) = mycenter(0): InserirServico(14) = mycenter(1): InserirServico(15) = mycenter(2)
    Set Servico = ThisDrawing.ModelSpace.AddPolyline(InserirServico)
    Servico.color = 171
    Servico.Update
                Set hachura(0) = Servico
                Set objHatch = ThisDrawing.ModelSpace.AddHatch(acHatchPatternTypePreDefined, "SOLID", True)
                objHatch.PatternScale = 0.1
                objHatch.AppendOuterLoop (hachura)
                objHatch.Evaluate
                objHatch.color = 171
                objHatch.Update
    End If
'Hall
    If Hall = True Then
    CHall = 1.5 * modulo
    LHall = 1.5 * modulo
    InserirHall(1) = mycenter(0) + LSala: InserirHall(2) = mycenter(1): InserirHall(3) = mycenter(2)
    InserirHall(4) = mycenter(0) + LSala - LHall: InserirHall(5) = mycenter(1): InserirHall(6) = 0
    InserirHall(7) = mycenter(0) + LSala - LHall: InserirHall(8) = mycenter(1) - CHall: InserirHall(9) = 0
    InserirHall(10) = mycenter(0) + LSala: InserirHall(11) = mycenter(1) - CHall: InserirHall(12) = 0
    InserirHall(13) = mycenter(0) + LSala: InserirHall(14) = mycenter(1): InserirHall(15) = mycenter(2)
    Set Hall = ThisDrawing.ModelSpace.AddPolyline(InserirHall)
    Hall.color = 62
```

```
    Hall.Update
                Set hachura(0) = Hall
                Set                          objHatch                          =
ThisDrawing.ModelSpace.AddHatch(acHatchPatternTypePreDefined, "SOLID", True)
                objHatch.PatternScale = 0.1
                objHatch.AppendOuterLoop (hachura)
                objHatch.Evaluate
                objHatch.color = 62
                objHatch.Update
    End If
'Acesso ao banheiro
    CCorredor = 1.4 * modulo
    LCorredor = 1.3 * modulo
    InserirCorredor(1)   =   mycenter(0)   +   LSala:   InserirCorredor(2)   =
mycenter(1): InserirCorredor(3) = mycenter(2)
    InserirCorredor(4) = mycenter(0) + LSala + LCorredor: InserirCorredor(5) =
mycenter(1): InserirCorredor(6) = 0
    InserirCorredor(7) = mycenter(0) + LSala + LCorredor:: InserirCorredor(8)
= mycenter(1) + CCorredor: InserirCorredor(9) = 0
    InserirCorredor(10)   =   mycenter(0)   +   LSala:   InserirCorredor(11)   =
mycenter(1) + CCorredor: InserirCorredor(12) = 0
    InserirCorredor(13)   =   mycenter(0)   +   LSala:   InserirCorredor(14)   =
mycenter(1): InserirCorredor(15) = mycenter(2)
    Set Corredor = ThisDrawing.ModelSpace.AddPolyline(InserirCorredor)
    Corredor.color = 12
    Corredor.Update
                Set hachura(0) = Corredor
                Set                          objHatch                          =
ThisDrawing.ModelSpace.AddHatch(acHatchPatternTypePreDefined, "SOLID", True)
                objHatch.PatternScale = 0.1
                objHatch.AppendOuterLoop (hachura)
                objHatch.Evaluate
                objHatch.color = 12
                objHatch.Update
'Banheiro
    CWc = 1.4 * modulo
    LWc = 2.2 * modulo
    InserirWc(1)   =   mycenter(0)   +   LSala   +   LCorredor:   InserirWc(2)   =
mycenter(1): InserirWc(3) = mycenter(2)
    InserirWc(4) = mycenter(0) + LSala + LCorredor + LWc: InserirWc(5) =
mycenter(1): InserirWc(6) = 0
    InserirWc(7) = mycenter(0) + LSala + LCorredor + LWc: InserirWc(8) =
mycenter(1) + CWc: InserirWc(9) = 0
```

```
    InserirWc(10) = mycenter(0) + LSala + LCorredor: InserirWc(11) =
mycenter(1) + CWc: InserirWc(12) = 0
    InserirWc(13) = mycenter(0) + LSala + LCorredor: InserirWc(14) =
mycenter(1): InserirWc(15) = mycenter(2)
    Set Wc = ThisDrawing.ModelSpace.AddPolyline(InserirWc)
    Wc.color = 30
    Wc.Update
                Set hachura(0) = Wc
                Set                          objHatch                          =
ThisDrawing.ModelSpace.AddHatch(acHatchPatternTypePreDefined, "SOLID", True)
                objHatch.PatternScale = 0.1
                objHatch.AppendOuterLoop (hachura)
                objHatch.Evaluate
                objHatch.color = 30
                objHatch.Update

    AreaCompleta = Abs(SalaCozI.Area + Coz.Area + Wc.Area + Corredor.Area)
        If Varanda.Value = True Then
            AreaCompleta = AreaCompleta + Varanda.Area
        End If
        If Aservico.Value = True Then
            AreaCompleta = AreaCompleta + Servico.Area
        End If
        If Hall.Value = True Then
            AreaCompleta = AreaCompleta + Hall.Area
        End If

    UserForm3.Area.Caption = AreaCompleta & " m²"
End If
If Nquartos = 1 Then
CSala = 5 * modulo
'Sala e cozinha separada
    If SalaCozS = True Then
    InserirSalaCozI(1) = mycenter(0): InserirSalaCozI(2) = mycenter(1):
InserirSalaCozI(3) = mycenter(2)
    InserirSalaCozI(4) = mycenter(0) + LSala: InserirSalaCozI(5) =
mycenter(1): InserirSalaCozI(6) = 0
    InserirSalaCozI(7) = mycenter(0) + LSala:: InserirSalaCozI(8) =
mycenter(1) + CSala: InserirSalaCozI(9) = 0
    InserirSalaCozI(10) = mycenter(0): InserirSalaCozI(11) = mycenter(1) +
CSala: InserirSalaCozI(12) = 0
```

```
    InserirSalaCozI(13) = mycenter(0): InserirSalaCozI(14) = mycenter(1): InserirSalaCozI(15) = mycenter(2)
    Set SalaCozI = ThisDrawing.ModelSpace.AddPolyline(InserirSalaCozI)
    SalaCozI.color = 64
    SalaCozI.Update
                Set hachura(0) = SalaCozI
                Set                          objHatch                          = ThisDrawing.ModelSpace.AddHatch(acHatchPatternTypePreDefined, "SOLID", True)
                objHatch.PatternScale = 0.1
                objHatch.AppendOuterLoop (hachura)
                objHatch.Evaluate
                objHatch.color = 64
                objHatch.Update
'Cozinha
    InserirCoz(1) = mycenter(0): InserirCoz(2) = mycenter(1) + CServico: InserirCoz(3) = 0
    InserirCoz(4) = mycenter(0): InserirCoz(5) = mycenter(1) + CServico + CCoz: InserirCoz(6) = 0
    InserirCoz(7) = mycenter(0) - LCoz: InserirCoz(8) = mycenter(1) + CServico + CCoz: InserirCoz(9) = 0
    InserirCoz(10) = mycenter(0) - LCoz: InserirCoz(11) = mycenter(1) + CServico: InserirCoz(12) = 0
    InserirCoz(13) = mycenter(0): InserirCoz(14) = mycenter(1) + CServico: InserirCoz(15) = 0
    Set Coz = ThisDrawing.ModelSpace.AddPolyline(InserirCoz)
    Coz.color = acBlue
    Coz.Update
                Set hachura(0) = Coz
                Set                          objHatch                          = ThisDrawing.ModelSpace.AddHatch(acHatchPatternTypePreDefined, "SOLID", True)
                objHatch.PatternScale = 0.1
                objHatch.AppendOuterLoop (hachura)
                objHatch.Evaluate
                objHatch.color = acBlue
                objHatch.Update
'Sala e cozinha integrada
    ElseIf SalaCozI = True Then
    CCoz = 2 * modulo
    LCoz = 2.5 * modulo
    InserirSalaCozI(1) = mycenter(0): InserirSalaCozI(2) = mycenter(1): InserirSalaCozI(3) = mycenter(2)
    InserirSalaCozI(4) = mycenter(0) + LSala: InserirSalaCozI(5) = mycenter(1): InserirSalaCozI(6) = 0
```

```
    InserirSalaCozI(7) = mycenter(0) + LSala:: InserirSalaCozI(8) =
mycenter(1) + CSala: InserirSalaCozI(9) = 0
    InserirSalaCozI(10) = mycenter(0): InserirSalaCozI(11) = mycenter(1) +
CSala: InserirSalaCozI(12) = 0
    InserirSalaCozI(13) = mycenter(0): InserirSalaCozI(14) = mycenter(1):
InserirSalaCozI(15) = mycenter(2)
    Set SalaCozI = ThisDrawing.ModelSpace.AddPolyline(InserirSalaCozI)
    SalaCozI.color = 64
    SalaCozI.Update
                Set hachura(0) = SalaCozI
                Set                          objHatch                          =
ThisDrawing.ModelSpace.AddHatch(acHatchPatternTypePreDefined, "SOLID", True)
                objHatch.PatternScale = 0.1
                objHatch.AppendOuterLoop (hachura)
                objHatch.Evaluate
                objHatch.color = 64
                objHatch.Update
'Cozinha
    InserirCoz(1) = mycenter(0): InserirCoz(2) = mycenter(1): InserirCoz(3) =
0
    InserirCoz(4) = mycenter(0): InserirCoz(5) = mycenter(1) + CCoz:
InserirCoz(6) = 0
    InserirCoz(7) = mycenter(0) + LCoz: InserirCoz(8) = mycenter(1) + CCoz:
InserirCoz(9) = 0
    InserirCoz(10) = mycenter(0) + LCoz: InserirCoz(11) = mycenter(1):
InserirCoz(12) = 0
    InserirCoz(13) = mycenter(0): InserirCoz(14) = mycenter(1): InserirCoz(15)
= 0
    Set Coz = ThisDrawing.ModelSpace.AddPolyline(InserirCoz)
    Coz.color = acBlue
    Coz.Update
                Set hachura(0) = Coz
                Set                          objHatch                          =
ThisDrawing.ModelSpace.AddHatch(acHatchPatternTypePreDefined, "PLASTI", True)
                objHatch.PatternAngle = 0.7853981 '45 graus
                objHatch.PatternScale = 0.07
                objHatch.AppendOuterLoop (hachura)
                objHatch.Evaluate
                objHatch.color = acBlue
                objHatch.Update
    End If
'Varanda
```

```
    If Varanda = True Then
    InserirVaranda(1) = mycenter(0): InserirVaranda(2) = mycenter(1) + CSala: InserirVaranda(3) = mycenter(2)
    InserirVaranda(4) = mycenter(0) + LSala: InserirVaranda(5) = mycenter(1) + CSala: InserirVaranda(6) = 0
    InserirVaranda(7) = mycenter(0) + LSala: InserirVaranda(8) = mycenter(1) + CSala + CVaranda: InserirVaranda(9) = 0
    InserirVaranda(10) = mycenter(0): InserirVaranda(11) = mycenter(1) + CSala + CVaranda: InserirVaranda(12) = 0
    InserirVaranda(13) = mycenter(0): InserirVaranda(14) = mycenter(1) + CSala: InserirVaranda(15) = mycenter(2)
    Set Varanda = ThisDrawing.ModelSpace.AddPolyline(InserirVaranda)
    Varanda.color = 66
    Varanda.Update
                Set hachura(0) = Varanda
                Set                          objHatch                          = ThisDrawing.ModelSpace.AddHatch(acHatchPatternTypePreDefined, "SOLID", True)
                objHatch.PatternScale = 0.1
                objHatch.AppendOuterLoop (hachura)
                objHatch.Evaluate
                objHatch.color = 66
                objHatch.Update
    End If
'Area de servico
    If Aservico = True Then
    InserirServico(1) = mycenter(0): InserirServico(2) = mycenter(1): InserirServico(3) = mycenter(2)
    InserirServico(4) = mycenter(0): InserirServico(5) = mycenter(1) + CServico: InserirServico(6) = 0
    InserirServico(7) = mycenter(0) - LCoz: InserirServico(8) = mycenter(1) + CServico: InserirServico(9) = 0
    InserirServico(10) = mycenter(0) - LCoz: InserirServico(11) = mycenter(1): InserirServico(12) = 0
    InserirServico(13) = mycenter(0): InserirServico(14) = mycenter(1): InserirServico(15) = mycenter(2)
    Set Servico = ThisDrawing.ModelSpace.AddPolyline(InserirServico)
    Servico.color = 171
    Servico.Update
                Set hachura(0) = Servico
                Set                          objHatch                          = ThisDrawing.ModelSpace.AddHatch(acHatchPatternTypePreDefined, "SOLID", True)
                objHatch.PatternScale = 0.1
                objHatch.AppendOuterLoop (hachura)
                objHatch.Evaluate
                objHatch.color = 171
```

```
                objHatch.Update
    End If
'Hall
    If Hall = True Then
    CHall = 1.5 * modulo
    LHall = 1.5 * modulo
    InserirHall(1) = mycenter(0) + LSala: InserirHall(2) = mycenter(1): InserirHall(3) = mycenter(2)
    InserirHall(4) = mycenter(0) + LSala - LHall: InserirHall(5) = mycenter(1): InserirHall(6) = 0
    InserirHall(7) = mycenter(0) + LSala - LHall: InserirHall(8) = mycenter(1) - CHall: InserirHall(9) = 0
    InserirHall(10) = mycenter(0) + LSala: InserirHall(11) = mycenter(1) - CHall: InserirHall(12) = 0
    InserirHall(13) = mycenter(0) + LSala: InserirHall(14) = mycenter(1): InserirHall(15) = mycenter(2)
    Set Hall = ThisDrawing.ModelSpace.AddPolyline(InserirHall)
    Hall.color = 62
    Hall.Update
                Set hachura(0) = Hall
                Set                          objHatch                          = ThisDrawing.ModelSpace.AddHatch(acHatchPatternTypePreDefined, "SOLID", True)
                objHatch.PatternScale = 0.1
                objHatch.AppendOuterLoop (hachura)
                objHatch.Evaluate
                objHatch.color = 62
                objHatch.Update
    End If
'Acesso ao banheiro
    CCorredor = CSala
    LCorredor = 1.3 * modulo
    InserirCorredor(1) = mycenter(0) + LSala: InserirCorredor(2) = mycenter(1): InserirCorredor(3) = mycenter(2)
    InserirCorredor(4) = mycenter(0) + LSala + LCorredor: InserirCorredor(5) = mycenter(1): InserirCorredor(6) = 0
    InserirCorredor(7) = mycenter(0) + LSala + LCorredor:: InserirCorredor(8) = mycenter(1) + CCorredor: InserirCorredor(9) = 0
    InserirCorredor(10) = mycenter(0) + LSala: InserirCorredor(11) = mycenter(1) + CCorredor: InserirCorredor(12) = 0
    InserirCorredor(13) = mycenter(0) + LSala: InserirCorredor(14) = mycenter(1): InserirCorredor(15) = mycenter(2)
    Set Corredor = ThisDrawing.ModelSpace.AddPolyline(InserirCorredor)
```

```
    Corredor.color = 12
    Corredor.Update
                Set hachura(0) = Corredor
                Set                          objHatch                          =
ThisDrawing.ModelSpace.AddHatch(acHatchPatternTypePreDefined, "SOLID", True)
                objHatch.PatternScale = 0.1
                objHatch.AppendOuterLoop (hachura)
                objHatch.Evaluate
                objHatch.color = 12
                objHatch.Update
'Banheiro
    CWc = 1.4 * modulo
    LWc = 2.2 * modulo
    InserirWc(1)  =  mycenter(0)  +  LSala  +  LCorredor:  InserirWc(2)  =
mycenter(1): InserirWc(3) = mycenter(2)
    InserirWc(4) = mycenter(0) + LSala + LCorredor + LWc: InserirWc(5) =
mycenter(1): InserirWc(6) = 0
    InserirWc(7) = mycenter(0) + LSala + LCorredor + LWc: InserirWc(8) =
mycenter(1) + CWc: InserirWc(9) = 0
    InserirWc(10)  =  mycenter(0)  +  LSala  +  LCorredor:  InserirWc(11)  =
mycenter(1) + CWc: InserirWc(12) = 0
    InserirWc(13)  =  mycenter(0)  +  LSala  +  LCorredor:  InserirWc(14)  =
mycenter(1): InserirWc(15) = mycenter(2)
    Set Wc = ThisDrawing.ModelSpace.AddPolyline(InserirWc)
    Wc.color = 30
    Wc.Update
                Set hachura(0) = Wc
                Set                          objHatch                          =
ThisDrawing.ModelSpace.AddHatch(acHatchPatternTypePreDefined, "SOLID", True)
                objHatch.PatternScale = 0.1
                objHatch.AppendOuterLoop (hachura)
                objHatch.Evaluate
                objHatch.color = 30
                objHatch.Update


'Dormitorio3
    If Dorm3 = True Then
    CDorm3 = (CSala - CWc)
    LDorm3 = (modulo * 3)
    InserirDorm3(1) = mycenter(0) + LSala + LCorredor: InserirDorm3(2) =
mycenter(1) + CWc: InserirDorm3(3) = 0
    InserirDorm3(4)  =  mycenter(0)  +  LSala  +  LCorredor  +  LDorm3:
InserirDorm3(5) = mycenter(1) + CWc: InserirDorm3(6) = 0
```

```
    InserirDorm3(7)  =  mycenter(0)  +  LSala  +  LCorredor  +  LDorm3:
InserirDorm3(8) = mycenter(1) + CWc + CDorm3: InserirDorm3(9) = 0

    InserirDorm3(10) = mycenter(0) + LSala + LCorredor: InserirDorm3(11) =
mycenter(1) + CWc + CDorm3: InserirDorm3(12) = 0

    InserirDorm3(13) = mycenter(0) + LSala + LCorredor: InserirDorm3(14) =
mycenter(1) + CWc: InserirDorm3(15) = 0

    Set Dorm3 = ThisDrawing.ModelSpace.AddPolyline(InserirDorm3)

    Dorm3.color = 20

    Dorm3.Update

                Set hachura(0) = Dorm3

                Set                          objHatch                          =
ThisDrawing.ModelSpace.AddHatch(acHatchPatternTypePreDefined, "SOLID", True)

                objHatch.PatternScale = 0.1

                objHatch.AppendOuterLoop (hachura)

                objHatch.Evaluate

                objHatch.color = 20

                objHatch.Update


    End If

    If Suite3 = True Then

    InserirWc(1) = mycenter(0) + LSala + LCorredor + LDorm3: InserirWc(2) =
mycenter(1) + CDorm3 + CWc: InserirWc(3) = mycenter(2)

    InserirWc(4)  =  mycenter(0)  +  LSala  +  LCorredor  +  LDorm3  +  LWc:
InserirWc(5) = mycenter(1) + CDorm3 + CWc: InserirWc(6) = 0

    InserirWc(7)  =  mycenter(0)  +  LSala  +  LCorredor  +  LDorm3  +  LWc:
InserirWc(8) = mycenter(1) + CDorm3: InserirWc(9) = 0

    InserirWc(10) = mycenter(0) + LSala + LCorredor + LDorm3: InserirWc(11) =
mycenter(1) + CDorm3: InserirWc(12) = 0

    InserirWc(13) = mycenter(0) + LSala + LCorredor + LDorm3: InserirWc(14) =
mycenter(1) + CDorm3 + CWc: InserirWc(15) = mycenter(2)

    Set Suite3 = ThisDrawing.ModelSpace.AddPolyline(InserirWc)

    Suite3.color = 30

    Suite3.Update

                Set hachura(0) = Suite3

                Set                          objHatch                          =
ThisDrawing.ModelSpace.AddHatch(acHatchPatternTypePreDefined, "SOLID", True)

                objHatch.PatternScale = 0.1

                objHatch.AppendOuterLoop (hachura)

                objHatch.Evaluate

                objHatch.color = 30

                objHatch.Update

    End If
```

```
        AreaCompleta = Abs(SalaCozI.Area + Coz.Area + Wc.Area + Corredor.Area)
        If Varanda.Value = True Then
            AreaCompleta = AreaCompleta + Varanda.Area
        End If
        If Aservico.Value = True Then
            AreaCompleta = AreaCompleta + Servico.Area
        End If
        If Hall.Value = True Then
            AreaCompleta = AreaCompleta + Hall.Area
        End If
        If Dorm3.Value = True Then
            AreaCompleta = AreaCompleta + Dorm3.Area
        End If
        If Suite3.Value = True Then
            AreaCompleta = AreaCompleta + Suite3.Area
        End If
    UserForm3.Area.Caption = AreaCompleta & " m²"
End If
If Nquartos = 2 Then
CSala = 7.5 * modulo
'Sala e cozinha separada
    If SalaCozS = True Then
    InserirSalaCozI(1) = mycenter(0): InserirSalaCozI(2) = mycenter(1): InserirSalaCozI(3) = mycenter(2)
    InserirSalaCozI(4) = mycenter(0) + LSala: InserirSalaCozI(5) = mycenter(1): InserirSalaCozI(6) = 0
    InserirSalaCozI(7) = mycenter(0) + LSala:: InserirSalaCozI(8) = mycenter(1) + CSala: InserirSalaCozI(9) = 0
    InserirSalaCozI(10) = mycenter(0): InserirSalaCozI(11) = mycenter(1) + CSala: InserirSalaCozI(12) = 0
    InserirSalaCozI(13) = mycenter(0): InserirSalaCozI(14) = mycenter(1): InserirSalaCozI(15) = mycenter(2)
    Set SalaCozI = ThisDrawing.ModelSpace.AddPolyline(InserirSalaCozI)
    SalaCozI.color = 64
    SalaCozI.Update
                Set hachura(0) = SalaCozI
                Set                          objHatch                          = ThisDrawing.ModelSpace.AddHatch(acHatchPatternTypePreDefined, "SOLID", True)
                objHatch.PatternScale = 0.1
                objHatch.AppendOuterLoop (hachura)
                objHatch.Evaluate
                objHatch.color = 64
```

```
                objHatch.Update
'Cozinha
    InserirCoz(1) = mycenter(0): InserirCoz(2) = mycenter(1) + CServico: InserirCoz(3) = 0
    InserirCoz(4) = mycenter(0): InserirCoz(5) = mycenter(1) + CServico + CCoz: InserirCoz(6) = 0
    InserirCoz(7) = mycenter(0) - LCoz: InserirCoz(8) = mycenter(1) + CServico + CCoz: InserirCoz(9) = 0
    InserirCoz(10) = mycenter(0) - LCoz: InserirCoz(11) = mycenter(1) + CServico: InserirCoz(12) = 0
    InserirCoz(13) = mycenter(0): InserirCoz(14) = mycenter(1) + CServico: InserirCoz(15) = 0
    Set Coz = ThisDrawing.ModelSpace.AddPolyline(InserirCoz)
    Coz.color = acBlue
    Coz.Update
                Set hachura(0) = Coz
                Set                          objHatch                          = ThisDrawing.ModelSpace.AddHatch(acHatchPatternTypePreDefined, "SOLID", True)
                objHatch.PatternScale = 0.1
                objHatch.AppendOuterLoop (hachura)
                objHatch.Evaluate
                objHatch.color = acBlue
                objHatch.Update
'Sala e cozinha integrada
    ElseIf SalaCozI = True Then
    CCoz = 2 * modulo
    LCoz = 2.5 * modulo
    InserirSalaCozI(1) = mycenter(0): InserirSalaCozI(2) = mycenter(1): InserirSalaCozI(3) = mycenter(2)
    InserirSalaCozI(4) = mycenter(0) + LSala: InserirSalaCozI(5) = mycenter(1): InserirSalaCozI(6) = 0
    InserirSalaCozI(7) = mycenter(0) + LSala:: InserirSalaCozI(8) = mycenter(1) + CSala: InserirSalaCozI(9) = 0
    InserirSalaCozI(10) = mycenter(0): InserirSalaCozI(11) = mycenter(1) + CSala: InserirSalaCozI(12) = 0
    InserirSalaCozI(13) = mycenter(0): InserirSalaCozI(14) = mycenter(1): InserirSalaCozI(15) = mycenter(2)
    Set SalaCozI = ThisDrawing.ModelSpace.AddPolyline(InserirSalaCozI)
    SalaCozI.color = 64
    SalaCozI.Update
                Set hachura(0) = SalaCozI
```

```
                Set                             objHatch                            =
ThisDrawing.ModelSpace.AddHatch(acHatchPatternTypePreDefined, "SOLID", True)
                objHatch.PatternScale = 0.1
                objHatch.AppendOuterLoop (hachura)
                objHatch.Evaluate
                objHatch.color = 64
                objHatch.Update
'Cozinha
    InserirCoz(1) = mycenter(0): InserirCoz(2) = mycenter(1): InserirCoz(3) =
0
    InserirCoz(4)  =  mycenter(0):  InserirCoz(5)  =  mycenter(1)  +  CCoz:
InserirCoz(6) = 0
    InserirCoz(7) = mycenter(0) + LCoz: InserirCoz(8) = mycenter(1) + CCoz:
InserirCoz(9) = 0
    InserirCoz(10)  =  mycenter(0)  +  LCoz:  InserirCoz(11)  =  mycenter(1):
InserirCoz(12) = 0
    InserirCoz(13) = mycenter(0): InserirCoz(14) = mycenter(1): InserirCoz(15)
= 0
    Set Coz = ThisDrawing.ModelSpace.AddPolyline(InserirCoz)
    Coz.color = acBlue
    Coz.Update
                Set hachura(0) = Coz
                Set                             objHatch                            =
ThisDrawing.ModelSpace.AddHatch(acHatchPatternTypePreDefined, "PLASTI", True)
                objHatch.PatternAngle = 0.7853981 '45 graus
                objHatch.PatternScale = 0.07
                objHatch.AppendOuterLoop (hachura)
                objHatch.Evaluate
                objHatch.color = acBlue
                objHatch.Update
    End If
'Varanda
    If Varanda = True Then
    InserirVaranda(1) = mycenter(0): InserirVaranda(2) = mycenter(1) + CSala:
InserirVaranda(3) = mycenter(2)
    InserirVaranda(4) = mycenter(0) + LSala: InserirVaranda(5) = mycenter(1) +
CSala: InserirVaranda(6) = 0
    InserirVaranda(7) = mycenter(0) + LSala: InserirVaranda(8) = mycenter(1) +
CSala + CVaranda: InserirVaranda(9) = 0
    InserirVaranda(10) = mycenter(0): InserirVaranda(11) = mycenter(1) + CSala
+ CVaranda: InserirVaranda(12) = 0
    InserirVaranda(13)  =  mycenter(0):  InserirVaranda(14)  =  mycenter(1)  +
CSala: InserirVaranda(15) = mycenter(2)
    Set Varanda = ThisDrawing.ModelSpace.AddPolyline(InserirVaranda)
    Varanda.color = 66
```

```
    Varanda.Update
                Set hachura(0) = Varanda
                Set                          objHatch                          =
ThisDrawing.ModelSpace.AddHatch(acHatchPatternTypePreDefined, "SOLID", True)
                objHatch.PatternScale = 0.1
                objHatch.AppendOuterLoop (hachura)
                objHatch.Evaluate
                objHatch.color = 66
                objHatch.Update
    End If
'Area de servico
    If Aservico = True Then
    InserirServico(1)  =  mycenter(0):  InserirServico(2)  =  mycenter(1):
InserirServico(3) = mycenter(2)
    InserirServico(4)  =  mycenter(0):  InserirServico(5)  =  mycenter(1)  +
CServico: InserirServico(6) = 0
    InserirServico(7) = mycenter(0) - LCoz: InserirServico(8) = mycenter(1) +
CServico: InserirServico(9) = 0
    InserirServico(10) = mycenter(0) - LCoz: InserirServico(11) = mycenter(1):
InserirServico(12) = 0
    InserirServico(13)  =  mycenter(0):  InserirServico(14)  =  mycenter(1):
InserirServico(15) = mycenter(2)
    Set Servico = ThisDrawing.ModelSpace.AddPolyline(InserirServico)
    Servico.color = 171
    Servico.Update
                Set hachura(0) = Servico
                Set                          objHatch                          =
ThisDrawing.ModelSpace.AddHatch(acHatchPatternTypePreDefined, "SOLID", True)
                objHatch.PatternScale = 0.1
                objHatch.AppendOuterLoop (hachura)
                objHatch.Evaluate
                objHatch.color = 171
                objHatch.Update
    End If
'Hall
    If Hall = True Then
    CHall = 1.5 * modulo
    LHall = 1.5 * modulo
    InserirHall(1)  =  mycenter(0)  +  LSala:  InserirHall(2)  =  mycenter(1):
InserirHall(3) = mycenter(2)
```

```
    InserirHall(4) = mycenter(0) + LSala - LHall: InserirHall(5) =
mycenter(1): InserirHall(6) = 0
    InserirHall(7) = mycenter(0) + LSala - LHall: InserirHall(8) = mycenter(1)
- CHall: InserirHall(9) = 0
    InserirHall(10) = mycenter(0) + LSala: InserirHall(11) = mycenter(1) -
CHall: InserirHall(12) = 0
    InserirHall(13) = mycenter(0) + LSala: InserirHall(14) = mycenter(1):
InserirHall(15) = mycenter(2)
    Set Hall = ThisDrawing.ModelSpace.AddPolyline(InserirHall)
    Hall.color = 62
    Hall.Update
                Set hachura(0) = Hall
                Set                          objHatch                          =
ThisDrawing.ModelSpace.AddHatch(acHatchPatternTypePreDefined, "SOLID", True)
                objHatch.PatternScale = 0.1
                objHatch.AppendOuterLoop (hachura)
                objHatch.Evaluate
                objHatch.color = 62
                objHatch.Update
    End If
'Acesso ao banheiro
    CCorredor = CSala
    LCorredor = 1.3 * modulo
    InserirCorredor(1) = mycenter(0) + LSala: InserirCorredor(2) =
mycenter(1): InserirCorredor(3) = mycenter(2)
    InserirCorredor(4) = mycenter(0) + LSala + LCorredor: InserirCorredor(5) =
mycenter(1): InserirCorredor(6) = 0
    InserirCorredor(7) = mycenter(0) + LSala + LCorredor:: InserirCorredor(8)
= mycenter(1) + CCorredor: InserirCorredor(9) = 0
    InserirCorredor(10) = mycenter(0) + LSala: InserirCorredor(11) =
mycenter(1) + CCorredor: InserirCorredor(12) = 0
    InserirCorredor(13) = mycenter(0) + LSala: InserirCorredor(14) =
mycenter(1): InserirCorredor(15) = mycenter(2)
    Set Corredor = ThisDrawing.ModelSpace.AddPolyline(InserirCorredor)
    Corredor.color = 12
    Corredor.Update
                Set hachura(0) = Corredor
                Set                          objHatch                          =
ThisDrawing.ModelSpace.AddHatch(acHatchPatternTypePreDefined, "SOLID", True)
                objHatch.PatternScale = 0.1
                objHatch.AppendOuterLoop (hachura)
                objHatch.Evaluate
                objHatch.color = 12
                objHatch.Update
```

```
'Banheiro
    CWc = 1.4 * modulo
    LWc = 2.2 * modulo
    InserirWc(1) = mycenter(0) + LSala + LCorredor: InserirWc(2) = mycenter(1): InserirWc(3) = mycenter(2)
    InserirWc(4) = mycenter(0) + LSala + LCorredor + LWc: InserirWc(5) = mycenter(1): InserirWc(6) = 0
    InserirWc(7) = mycenter(0) + LSala + LCorredor + LWc: InserirWc(8) = mycenter(1) + CWc: InserirWc(9) = 0
    InserirWc(10) = mycenter(0) + LSala + LCorredor: InserirWc(11) = mycenter(1) + CWc: InserirWc(12) = 0
    InserirWc(13) = mycenter(0) + LSala + LCorredor: InserirWc(14) = mycenter(1): InserirWc(15) = mycenter(2)
    Set Wc = ThisDrawing.ModelSpace.AddPolyline(InserirWc)
    Wc.color = 30
    Wc.Update
                Set hachura(0) = Wc
                Set                          objHatch                          =
ThisDrawing.ModelSpace.AddHatch(acHatchPatternTypePreDefined, "SOLID", True)
                objHatch.PatternScale = 0.1
                objHatch.AppendOuterLoop (hachura)
                objHatch.Evaluate
                objHatch.color = 30
                objHatch.Update
'Dormitorio3
    If Dorm3 = True Then
    CDorm3 = (CSala - CWc) / 2
    LDorm3 = (modulo * 3)
    InserirDorm3(1) = mycenter(0) + LSala + LCorredor: InserirDorm3(2) = mycenter(1) + CWc: InserirDorm3(3) = 0
    InserirDorm3(4) = mycenter(0) + LSala + LCorredor + LDorm3: InserirDorm3(5) = mycenter(1) + CWc: InserirDorm3(6) = 0
    InserirDorm3(7) = mycenter(0) + LSala + LCorredor + LDorm3: InserirDorm3(8) = mycenter(1) + CWc + CDorm3: InserirDorm3(9) = 0
    InserirDorm3(10) = mycenter(0) + LSala + LCorredor: InserirDorm3(11) = mycenter(1) + CWc + CDorm3: InserirDorm3(12) = 0
    InserirDorm3(13) = mycenter(0) + LSala + LCorredor: InserirDorm3(14) = mycenter(1) + CWc: InserirDorm3(15) = 0
    Set Dorm3 = ThisDrawing.ModelSpace.AddPolyline(InserirDorm3)
    Dorm3.color = 20
    Dorm3.Update
                Set hachura(0) = Dorm3
```

```
                Set                             objHatch                            =
ThisDrawing.ModelSpace.AddHatch(acHatchPatternTypePreDefined, "SOLID", True)
                objHatch.PatternScale = 0.1
                objHatch.AppendOuterLoop (hachura)
                objHatch.Evaluate
                objHatch.color = 20
                objHatch.Update
    End If
    If Suite3 = True Then
    InserirWc(1) = mycenter(0) + LSala + LCorredor + LDorm3: InserirWc(2) =
mycenter(1) + CDorm3 + CWc: InserirWc(3) = mycenter(2)
    InserirWc(4)  =  mycenter(0)  +  LSala  +  LCorredor  +  LDorm3  +  LWc:
InserirWc(5) = mycenter(1) + CDorm3 + CWc: InserirWc(6) = 0
    InserirWc(7)  =  mycenter(0)  +  LSala  +  LCorredor  +  LDorm3  +  LWc:
InserirWc(8) = mycenter(1) + CDorm3: InserirWc(9) = 0
    InserirWc(10) = mycenter(0) + LSala + LCorredor + LDorm3: InserirWc(11) =
mycenter(1) + CDorm3: InserirWc(12) = 0
    InserirWc(13) = mycenter(0) + LSala + LCorredor + LDorm3: InserirWc(14) =
mycenter(1) + CDorm3 + CWc: InserirWc(15) = mycenter(2)
    Set Suite3 = ThisDrawing.ModelSpace.AddPolyline(InserirWc)
    Suite3.color = 30
    Suite3.Update
                Set hachura(0) = Suite3
                Set                             objHatch                            =
ThisDrawing.ModelSpace.AddHatch(acHatchPatternTypePreDefined, "SOLID", True)
                objHatch.PatternScale = 0.1
                objHatch.AppendOuterLoop (hachura)
                objHatch.Evaluate
                objHatch.color = 30
                objHatch.Update
    End If
'Dormitorio4
    If Dorm4 = True Then
    CDorm4 = (CSala - CWc) / 2
    LDorm4 = (modulo * 3)
    InserirDorm4(1)  =  mycenter(0)  +  LSala  +  LCorredor:  InserirDorm4(2)  =
mycenter(1) + CWc + CDorm4: InserirDorm4(3) = 0
    InserirDorm4(4)  =  mycenter(0)  +  LSala  +  LCorredor  +  LDorm4:
InserirDorm4(5) = mycenter(1) + CWc + CDorm4: InserirDorm4(6) = 0
    InserirDorm4(7)  =  mycenter(0)  +  LSala  +  LCorredor  +  LDorm4:
InserirDorm4(8) = mycenter(1) + CWc + CDorm4 + CDorm4: InserirDorm4(9) = 0
    InserirDorm4(10) = mycenter(0) + LSala + LCorredor: InserirDorm4(11) =
mycenter(1) + CWc + CDorm4 + CDorm4: InserirDorm4(12) = 0
    InserirDorm4(13) = mycenter(0) + LSala + LCorredor: InserirDorm4(14) =
mycenter(1) + CWc + CDorm4: InserirDorm4(15) = 0
```

```
    Set Dorm4 = ThisDrawing.ModelSpace.AddPolyline(InserirDorm4)
    Dorm4.color = 20
    Dorm4.Update
                Set hachura(0) = Dorm4
                Set                          objHatch                          =
ThisDrawing.ModelSpace.AddHatch(acHatchPatternTypePreDefined, "SOLID", True)
                objHatch.PatternScale = 0.1
                objHatch.AppendOuterLoop (hachura)
                objHatch.Evaluate
                objHatch.color = 20
                objHatch.Update
    End If
    If Suite4 = True Then
    InserirWc(1) = mycenter(0) + LSala + LCorredor + LDorm4: InserirWc(2) =
mycenter(1) + CDorm4 + CWc: InserirWc(3) = mycenter(2)
    InserirWc(4)  =  mycenter(0)  +  LSala  +  LCorredor  +  LDorm4  +  LWc:
InserirWc(5) = mycenter(1) + CDorm4 + CWc: InserirWc(6) = 0
    InserirWc(7)  =  mycenter(0)  +  LSala  +  LCorredor  +  LDorm4  +  LWc:
InserirWc(8) = mycenter(1) + CDorm4 + CWc + CWc: InserirWc(9) = 0
    InserirWc(10) = mycenter(0) + LSala + LCorredor + LDorm4: InserirWc(11) =
mycenter(1) + CDorm4 + CWc + CWc: InserirWc(12) = 0
    InserirWc(13) = mycenter(0) + LSala + LCorredor + LDorm4: InserirWc(14) =
CDorm4 + CWc: InserirWc(15) = mycenter(2)
    Set Suite4 = ThisDrawing.ModelSpace.AddPolyline(InserirWc)
    Suite4.color = 30
    Suite4.Update
                Set hachura(0) = Suite4
                Set                          objHatch                          =
ThisDrawing.ModelSpace.AddHatch(acHatchPatternTypePreDefined, "SOLID", True)
                objHatch.PatternScale = 0.1
                objHatch.AppendOuterLoop (hachura)
                objHatch.Evaluate
                objHatch.color = 30
                objHatch.Update
    End If
End If
If Nquartos = 3 Then
CSala = 7.5 * modulo
'Sala e cozinha separada
    If SalaCozS = True Then
```

```
    InserirSalaCozI(1) = mycenter(0): InserirSalaCozI(2) = mycenter(1): InserirSalaCozI(3) = mycenter(2)
    InserirSalaCozI(4) = mycenter(0) + LSala: InserirSalaCozI(5) = mycenter(1): InserirSalaCozI(6) = 0
    InserirSalaCozI(7) = mycenter(0) + LSala:: InserirSalaCozI(8) = mycenter(1) + CSala: InserirSalaCozI(9) = 0
    InserirSalaCozI(10) = mycenter(0): InserirSalaCozI(11) = mycenter(1) + CSala: InserirSalaCozI(12) = 0
    InserirSalaCozI(13) = mycenter(0): InserirSalaCozI(14) = mycenter(1): InserirSalaCozI(15) = mycenter(2)
    Set SalaCozI = ThisDrawing.ModelSpace.AddPolyline(InserirSalaCozI)
    SalaCozI.color = 64
    SalaCozI.Update
                Set hachura(0) = SalaCozI
                Set                          objHatch                          = ThisDrawing.ModelSpace.AddHatch(acHatchPatternTypePreDefined, "SOLID", True)
                objHatch.PatternScale = 0.1
                objHatch.AppendOuterLoop (hachura)
                objHatch.Evaluate
                objHatch.color = 64
                objHatch.Update
'Cozinha
    InserirCoz(1) = mycenter(0): InserirCoz(2) = mycenter(1) + CServico: InserirCoz(3) = 0
    InserirCoz(4) = mycenter(0): InserirCoz(5) = mycenter(1) + CServico + CCoz: InserirCoz(6) = 0
    InserirCoz(7) = mycenter(0) - LCoz: InserirCoz(8) = mycenter(1) + CServico + CCoz: InserirCoz(9) = 0
    InserirCoz(10) = mycenter(0) - LCoz: InserirCoz(11) = mycenter(1) + CServico: InserirCoz(12) = 0
    InserirCoz(13) = mycenter(0): InserirCoz(14) = mycenter(1) + CServico: InserirCoz(15) = 0
    Set Coz = ThisDrawing.ModelSpace.AddPolyline(InserirCoz)
    Coz.color = acBlue
    Coz.Update
                Set hachura(0) = Coz
                Set                          objHatch                          = ThisDrawing.ModelSpace.AddHatch(acHatchPatternTypePreDefined, "SOLID", True)
                objHatch.PatternScale = 0.1
                objHatch.AppendOuterLoop (hachura)
                objHatch.Evaluate
                objHatch.color = acBlue
                objHatch.Update
'Sala e cozinha integrada
    ElseIf SalaCozI = True Then
```

```
    CCoz = 2 * modulo
    LCoz = 2.5 * modulo
    InserirSalaCozI(1) = mycenter(0): InserirSalaCozI(2) = mycenter(1): InserirSalaCozI(3) = mycenter(2)
    InserirSalaCozI(4) = mycenter(0) + LSala: InserirSalaCozI(5) = mycenter(1): InserirSalaCozI(6) = 0
    InserirSalaCozI(7) = mycenter(0) + LSala:: InserirSalaCozI(8) = mycenter(1) + CSala: InserirSalaCozI(9) = 0
    InserirSalaCozI(10) = mycenter(0): InserirSalaCozI(11) = mycenter(1) + CSala: InserirSalaCozI(12) = 0
    InserirSalaCozI(13) = mycenter(0): InserirSalaCozI(14) = mycenter(1): InserirSalaCozI(15) = mycenter(2)
    Set SalaCozI = ThisDrawing.ModelSpace.AddPolyline(InserirSalaCozI)
    SalaCozI.color = 64
    SalaCozI.Update
                Set hachura(0) = SalaCozI
                Set                          objHatch                          = ThisDrawing.ModelSpace.AddHatch(acHatchPatternTypePreDefined, "SOLID", True)
                objHatch.PatternScale = 0.1
                objHatch.AppendOuterLoop (hachura)
                objHatch.Evaluate
                objHatch.color = 64
                objHatch.Update
'Cozinha
    InserirCoz(1) = mycenter(0): InserirCoz(2) = mycenter(1): InserirCoz(3) = 0
    InserirCoz(4) = mycenter(0): InserirCoz(5) = mycenter(1) + CCoz: InserirCoz(6) = 0
    InserirCoz(7) = mycenter(0) + LCoz: InserirCoz(8) = mycenter(1) + CCoz: InserirCoz(9) = 0
    InserirCoz(10) = mycenter(0) + LCoz: InserirCoz(11) = mycenter(1): InserirCoz(12) = 0
    InserirCoz(13) = mycenter(0): InserirCoz(14) = mycenter(1): InserirCoz(15) = 0
    Set Coz = ThisDrawing.ModelSpace.AddPolyline(InserirCoz)
    Coz.color = acBlue
    Coz.Update
                Set hachura(0) = Coz
                Set                          objHatch                          = ThisDrawing.ModelSpace.AddHatch(acHatchPatternTypePreDefined, "PLASTI", True)
                objHatch.PatternAngle = 0.7853981 '45 graus
                objHatch.PatternScale = 0.07
```

```
                objHatch.AppendOuterLoop (hachura)
                objHatch.Evaluate
                objHatch.color = acBlue
                objHatch.Update
    End If
'Varanda
    If Varanda = True Then
    InserirVaranda(1) = mycenter(0): InserirVaranda(2) = mycenter(1) + CSala: InserirVaranda(3) = mycenter(2)
    InserirVaranda(4) = mycenter(0) + LSala: InserirVaranda(5) = mycenter(1) + CSala: InserirVaranda(6) = 0
    InserirVaranda(7) = mycenter(0) + LSala: InserirVaranda(8) = mycenter(1) + CSala + CVaranda: InserirVaranda(9) = 0
    InserirVaranda(10) = mycenter(0): InserirVaranda(11) = mycenter(1) + CSala + CVaranda: InserirVaranda(12) = 0
    InserirVaranda(13) = mycenter(0): InserirVaranda(14) = mycenter(1) + CSala: InserirVaranda(15) = mycenter(2)
    Set Varanda = ThisDrawing.ModelSpace.AddPolyline(InserirVaranda)
    Varanda.color = 66
    Varanda.Update
                Set hachura(0) = Varanda
                Set                          objHatch                          = ThisDrawing.ModelSpace.AddHatch(acHatchPatternTypePreDefined, "SOLID", True)
                objHatch.PatternScale = 0.1
                objHatch.AppendOuterLoop (hachura)
                objHatch.Evaluate
                objHatch.color = 66
                objHatch.Update
    End If
'Area de servico
    If Aservico = True Then
    InserirServico(1) = mycenter(0): InserirServico(2) = mycenter(1): InserirServico(3) = mycenter(2)
    InserirServico(4) = mycenter(0): InserirServico(5) = mycenter(1) + CServico: InserirServico(6) = 0
    InserirServico(7) = mycenter(0) - LCoz: InserirServico(8) = mycenter(1) + CServico: InserirServico(9) = 0
    InserirServico(10) = mycenter(0) - LCoz: InserirServico(11) = mycenter(1): InserirServico(12) = 0
    InserirServico(13) = mycenter(0): InserirServico(14) = mycenter(1): InserirServico(15) = mycenter(2)
    Set Servico = ThisDrawing.ModelSpace.AddPolyline(InserirServico)
    Servico.color = 171
    Servico.Update
```

```
                Set hachura(0) = Servico
                Set                          objHatch                          =
ThisDrawing.ModelSpace.AddHatch(acHatchPatternTypePreDefined, "SOLID", True)
                objHatch.PatternScale = 0.1
                objHatch.AppendOuterLoop (hachura)
                objHatch.Evaluate
                objHatch.color = 171
                objHatch.Update
    End If
'Hall
    If Hall = True Then
    CHall = 1.5 * modulo
    LHall = 1.5 * modulo
    InserirHall(1) = mycenter(0) + LSala: InserirHall(2) = mycenter(1):
InserirHall(3) = mycenter(2)
    InserirHall(4) = mycenter(0) + LSala - LHall: InserirHall(5) =
mycenter(1): InserirHall(6) = 0
    InserirHall(7) = mycenter(0) + LSala - LHall: InserirHall(8) = mycenter(1)
- CHall: InserirHall(9) = 0
    InserirHall(10) = mycenter(0) + LSala: InserirHall(11) = mycenter(1) -
CHall: InserirHall(12) = 0
    InserirHall(13) = mycenter(0) + LSala: InserirHall(14) = mycenter(1):
InserirHall(15) = mycenter(2)
    Set Hall = ThisDrawing.ModelSpace.AddPolyline(InserirHall)
    Hall.color = 62
    Hall.Update
                Set hachura(0) = Hall
                Set                          objHatch                          =
ThisDrawing.ModelSpace.AddHatch(acHatchPatternTypePreDefined, "SOLID", True)
                objHatch.PatternScale = 0.1
                objHatch.AppendOuterLoop (hachura)
                objHatch.Evaluate
                objHatch.color = 62
                objHatch.Update
    End If
'Acesso ao banheiro
    CCorredor = CSala
    LCorredor = 1.3 * modulo
    InserirCorredor(1) = mycenter(0) + LSala: InserirCorredor(2) = mycenter(1)
- CDorm1: InserirCorredor(3) = mycenter(2)
```

```
    InserirCorredor(4) = mycenter(0) + LSala + LCorredor: InserirCorredor(5) =
mycenter(1) - CDorm1: InserirCorredor(6) = 0

    InserirCorredor(7) = mycenter(0) + LSala + LCorredor:: InserirCorredor(8)
= mycenter(1) + CCorredor: InserirCorredor(9) = 0

    InserirCorredor(10)  =  mycenter(0)  +  LSala:  InserirCorredor(11)  =
mycenter(1) + CCorredor: InserirCorredor(12) = 0

    InserirCorredor(13)  =  mycenter(0)  +  LSala:  InserirCorredor(14)  =
mycenter(1) - CDorm1: InserirCorredor(15) = mycenter(2)

    Set Corredor = ThisDrawing.ModelSpace.AddPolyline(InserirCorredor)

    Corredor.color = 12

    Corredor.Update

                Set hachura(0) = Corredor

                Set                          objHatch                          =
ThisDrawing.ModelSpace.AddHatch(acHatchPatternTypePreDefined, "SOLID", True)

                objHatch.PatternScale = 0.1

                objHatch.AppendOuterLoop (hachura)

                objHatch.Evaluate

                objHatch.color = 12

                objHatch.Update

'Banheiro

    CWc = 1.4 * modulo

    LWc = 2.2 * modulo

    InserirWc(1)  =  mycenter(0)  +  LSala  +  LCorredor:  InserirWc(2)  =
mycenter(1): InserirWc(3) = mycenter(2)

    InserirWc(4) = mycenter(0) + LSala + LCorredor + LWc: InserirWc(5) =
mycenter(1): InserirWc(6) = 0

    InserirWc(7) = mycenter(0) + LSala + LCorredor + LWc: InserirWc(8) =
mycenter(1) + CWc: InserirWc(9) = 0

    InserirWc(10)  =  mycenter(0)  +  LSala  +  LCorredor:  InserirWc(11)  =
mycenter(1) + CWc: InserirWc(12) = 0

    InserirWc(13)  =  mycenter(0)  +  LSala  +  LCorredor:  InserirWc(14)  =
mycenter(1): InserirWc(15) = mycenter(2)

    Set Wc = ThisDrawing.ModelSpace.AddPolyline(InserirWc)

    Wc.color = 30

    Wc.Update

                Set hachura(0) = Wc

                Set                          objHatch                          =
ThisDrawing.ModelSpace.AddHatch(acHatchPatternTypePreDefined, "SOLID", True)

                objHatch.PatternScale = 0.1

                objHatch.AppendOuterLoop (hachura)

                objHatch.Evaluate

                objHatch.color = 30

                objHatch.Update

'Dormitorio3
```

```
    If Dorm3 = True Then
    CDorm3 = (CSala - CWc) / 2
    LDorm3 = (modulo * 3)
    InserirDorm3(1) = mycenter(0) + LSala + LCorredor: InserirDorm3(2) = mycenter(1) + CWc: InserirDorm3(3) = 0
    InserirDorm3(4) = mycenter(0) + LSala + LCorredor + LDorm3: InserirDorm3(5) = mycenter(1) + CWc: InserirDorm3(6) = 0
    InserirDorm3(7) = mycenter(0) + LSala + LCorredor + LDorm3: InserirDorm3(8) = mycenter(1) + CWc + CDorm3: InserirDorm3(9) = 0
    InserirDorm3(10) = mycenter(0) + LSala + LCorredor: InserirDorm3(11) = mycenter(1) + CWc + CDorm3: InserirDorm3(12) = 0
    InserirDorm3(13) = mycenter(0) + LSala + LCorredor: InserirDorm3(14) = mycenter(1) + CWc: InserirDorm3(15) = 0
    Set Dorm3 = ThisDrawing.ModelSpace.AddPolyline(InserirDorm3)
    Dorm3.color = 20
    Dorm3.Update
                Set hachura(0) = Dorm3
                Set                          objHatch                          = ThisDrawing.ModelSpace.AddHatch(acHatchPatternTypePreDefined, "SOLID", True)
                objHatch.PatternScale = 0.1
                objHatch.AppendOuterLoop (hachura)
                objHatch.Evaluate
                objHatch.color = 20
                objHatch.Update
    End If
    If Suite3 = True Then
    InserirWc(1) = mycenter(0) + LSala + LCorredor + LDorm3: InserirWc(2) = mycenter(1) + CDorm3 + CWc: InserirWc(3) = mycenter(2)
    InserirWc(4) = mycenter(0) + LSala + LCorredor + LDorm3 + LWc: InserirWc(5) = mycenter(1) + CDorm3 + CWc: InserirWc(6) = 0
    InserirWc(7) = mycenter(0) + LSala + LCorredor + LDorm3 + LWc: InserirWc(8) = mycenter(1) + CDorm3: InserirWc(9) = 0
    InserirWc(10) = mycenter(0) + LSala + LCorredor + LDorm3: InserirWc(11) = mycenter(1) + CDorm3: InserirWc(12) = 0
    InserirWc(13) = mycenter(0) + LSala + LCorredor + LDorm3: InserirWc(14) = mycenter(1) + CDorm3 + CWc: InserirWc(15) = mycenter(2)
    Set Suite3 = ThisDrawing.ModelSpace.AddPolyline(InserirWc)
    Suite3.color = 30
    Suite3.Update
                Set hachura(0) = Suite3
                Set                          objHatch                          = ThisDrawing.ModelSpace.AddHatch(acHatchPatternTypePreDefined, "SOLID", True)
```

```
                objHatch.PatternScale = 0.1
                objHatch.AppendOuterLoop (hachura)
                objHatch.Evaluate
                objHatch.color = 30
                objHatch.Update
    End If
'Dormitorio4
    If Dorm4 = True Then
    CDorm4 = (CSala - CWc) / 2
    LDorm4 = (modulo * 3)
    InserirDorm4(1) = mycenter(0) + LSala + LCorredor: InserirDorm4(2) =
mycenter(1) + CWc + CDorm4: InserirDorm4(3) = 0
    InserirDorm4(4) = mycenter(0) + LSala + LCorredor + LDorm4:
InserirDorm4(5) = mycenter(1) + CWc + CDorm4: InserirDorm4(6) = 0
    InserirDorm4(7) = mycenter(0) + LSala + LCorredor + LDorm4:
InserirDorm4(8) = mycenter(1) + CWc + CDorm4 + CDorm4: InserirDorm4(9) = 0
    InserirDorm4(10) = mycenter(0) + LSala + LCorredor: InserirDorm4(11) =
mycenter(1) + CWc + CDorm4 + CDorm4: InserirDorm4(12) = 0
    InserirDorm4(13) = mycenter(0) + LSala + LCorredor: InserirDorm4(14) =
mycenter(1) + CWc + CDorm4: InserirDorm4(15) = 0
    Set Dorm4 = ThisDrawing.ModelSpace.AddPolyline(InserirDorm4)
    Dorm4.color = 20
    Dorm4.Update
                Set hachura(0) = Dorm4
                Set                          objHatch                          =
ThisDrawing.ModelSpace.AddHatch(acHatchPatternTypePreDefined, "SOLID", True)
                objHatch.PatternScale = 0.1
                objHatch.AppendOuterLoop (hachura)
                objHatch.Evaluate
                objHatch.color = 20
                objHatch.Update
    End If
    If Suite4 = True Then
    InserirWc(1) = mycenter(0) + LSala + LCorredor + LDorm4: InserirWc(2) =
mycenter(1) + CDorm4 + CWc: InserirWc(3) = mycenter(2)
    InserirWc(4) = mycenter(0) + LSala + LCorredor + LDorm4 + LWc:
InserirWc(5) = mycenter(1) + CDorm4 + CWc: InserirWc(6) = 0
    InserirWc(7) = mycenter(0) + LSala + LCorredor + LDorm4 + LWc:
InserirWc(8) = mycenter(1) + CDorm4 + CWc + CWc: InserirWc(9) = 0
    InserirWc(10) = mycenter(0) + LSala + LCorredor + LDorm4: InserirWc(11) =
mycenter(1) + CDorm4 + CWc + CWc: InserirWc(12) = 0
    InserirWc(13) = mycenter(0) + LSala + LCorredor + LDorm4: InserirWc(14) =
CDorm4 + CWc: InserirWc(15) = mycenter(2)
    Set Suite4 = ThisDrawing.ModelSpace.AddPolyline(InserirWc)
```

```
    Suite4.color = 30
    Suite4.Update
                Set hachura(0) = Suite4
                Set                          objHatch                          =
ThisDrawing.ModelSpace.AddHatch(acHatchPatternTypePreDefined, "SOLID", True)
                objHatch.PatternScale = 0.1
                objHatch.AppendOuterLoop (hachura)
                objHatch.Evaluate
                objHatch.color = 30
                objHatch.Update
    End If
'Dormitorio1
    If Dorm1 = True Then
    CDorm1 = (CSala - CWc) / 2
    LDorm1 = (modulo * 3)
    CCorredor = CDorm1
    LCorredor = 1.3 * modulo
    InserirDorm(1)  =  mycenter(0)  +  LSala  +  LCorredor:  InserirDorm(2)  =
mycenter(1) - CDorm1: InserirDorm(3) = 0
    InserirDorm(4) = mycenter(0) + LSala + LCorredor + LDorm1: InserirDorm(5)
= mycenter(1) - CDorm1: InserirDorm(6) = 0
    InserirDorm(7) = mycenter(0) + LSala + LCorredor + LDorm1: InserirDorm(8)
= mycenter(1): InserirDorm(9) = 0
    InserirDorm(10)  =  mycenter(0)  +  LSala  +  LCorredor:  InserirDorm(11)  =
mycenter(1): InserirDorm(12) = 0
    InserirDorm(13)  =  mycenter(0)  +  LSala  +  LCorredor:  InserirDorm(14)  =
mycenter(1) - CDorm1: InserirDorm(15) = 0
    Set Dorm = ThisDrawing.ModelSpace.AddPolyline(InserirDorm)
    Dorm.color = 20
    Dorm.Update
                Set hachura(0) = Dorm
                Set                          objHatch                          =
ThisDrawing.ModelSpace.AddHatch(acHatchPatternTypePreDefined, "SOLID", True)
                objHatch.PatternScale = 0.1
                objHatch.AppendOuterLoop (hachura)
                objHatch.Evaluate
                objHatch.color = 20
                objHatch.Update
    End If
    If Suite1 = True Then
```

```
    InserirWc(1) = mycenter(0) + LSala + LCorredor + LDorm1: InserirWc(2) =
mycenter(1) - CDorm1: InserirWc(3) = mycenter(2)
    InserirWc(4) = mycenter(0) + LSala + LCorredor + LDorm1 + LWc:
InserirWc(5) = mycenter(1) - CDorm1: InserirWc(6) = 0
    InserirWc(7) = mycenter(0) + LSala + LCorredor + LDorm1 + LWc:
InserirWc(8) = mycenter(1) - CDorm1 + CWc: InserirWc(9) = 0
    InserirWc(10) = mycenter(0) + LSala + LCorredor + LDorm1: InserirWc(11) =
mycenter(1) - CDorm1 + CWc: InserirWc(12) = 0
    InserirWc(13) = mycenter(0) + LSala + LCorredor + LDorm1: InserirWc(14) =
mycenter(1) - CDorm1: InserirWc(15) = mycenter(2)
    Set Suite1 = ThisDrawing.ModelSpace.AddPolyline(InserirWc)
    Suite1.color = 30
    Suite1.Update
                Set hachura(0) = Suite1
                Set                          objHatch                          =
ThisDrawing.ModelSpace.AddHatch(acHatchPatternTypePreDefined, "SOLID", True)
                objHatch.PatternScale = 0.1
                objHatch.AppendOuterLoop (hachura)
                objHatch.Evaluate
                objHatch.color = 30
                objHatch.Update
    End If
End If
If Nquartos = 4 Then
CSala = 7.5 * modulo
'Sala e cozinha separada
    If SalaCozS = True Then
    InserirSalaCozI(1) = mycenter(0): InserirSalaCozI(2) = mycenter(1):
InserirSalaCozI(3) = mycenter(2)
    InserirSalaCozI(4) = mycenter(0) + LSala: InserirSalaCozI(5) =
mycenter(1): InserirSalaCozI(6) = 0
    InserirSalaCozI(7) = mycenter(0) + LSala:: InserirSalaCozI(8) =
mycenter(1) + CSala: InserirSalaCozI(9) = 0
    InserirSalaCozI(10) = mycenter(0): InserirSalaCozI(11) = mycenter(1) +
CSala: InserirSalaCozI(12) = 0
    InserirSalaCozI(13) = mycenter(0): InserirSalaCozI(14) = mycenter(1):
InserirSalaCozI(15) = mycenter(2)
    Set SalaCozI = ThisDrawing.ModelSpace.AddPolyline(InserirSalaCozI)
    SalaCozI.color = 64
    SalaCozI.Update
                Set hachura(0) = SalaCozI
                Set                          objHatch                          =
ThisDrawing.ModelSpace.AddHatch(acHatchPatternTypePreDefined, "SOLID", True)
                objHatch.PatternScale = 0.1
                objHatch.AppendOuterLoop (hachura)
```

```
                objHatch.Evaluate
                objHatch.color = 64
                objHatch.Update
'Cozinha
    InserirCoz(1) = mycenter(0): InserirCoz(2) = mycenter(1) + CServico: InserirCoz(3) = 0
    InserirCoz(4) = mycenter(0): InserirCoz(5) = mycenter(1) + CServico + CCoz: InserirCoz(6) = 0
    InserirCoz(7) = mycenter(0) - LCoz: InserirCoz(8) = mycenter(1) + CServico + CCoz: InserirCoz(9) = 0
    InserirCoz(10) = mycenter(0) - LCoz: InserirCoz(11) = mycenter(1) + CServico: InserirCoz(12) = 0
    InserirCoz(13) = mycenter(0): InserirCoz(14) = mycenter(1) + CServico: InserirCoz(15) = 0
    Set Coz = ThisDrawing.ModelSpace.AddPolyline(InserirCoz)
    Coz.color = acBlue
    Coz.Update
                Set hachura(0) = Coz
                Set                          objHatch                          = ThisDrawing.ModelSpace.AddHatch(acHatchPatternTypePreDefined, "SOLID", True)
                objHatch.PatternScale = 0.1
                objHatch.AppendOuterLoop (hachura)
                objHatch.Evaluate
                objHatch.color = acBlue
                objHatch.Update
'Sala e cozinha integrada
    ElseIf SalaCozI = True Then
    CCoz = 2 * modulo
    LCoz = 2.5 * modulo
    InserirSalaCozI(1) = mycenter(0): InserirSalaCozI(2) = mycenter(1): InserirSalaCozI(3) = mycenter(2)
    InserirSalaCozI(4) = mycenter(0) + LSala: InserirSalaCozI(5) = mycenter(1): InserirSalaCozI(6) = 0
    InserirSalaCozI(7) = mycenter(0) + LSala:: InserirSalaCozI(8) = mycenter(1) + CSala: InserirSalaCozI(9) = 0
    InserirSalaCozI(10) = mycenter(0): InserirSalaCozI(11) = mycenter(1) + CSala: InserirSalaCozI(12) = 0
    InserirSalaCozI(13) = mycenter(0): InserirSalaCozI(14) = mycenter(1): InserirSalaCozI(15) = mycenter(2)
    Set SalaCozI = ThisDrawing.ModelSpace.AddPolyline(InserirSalaCozI)
    SalaCozI.color = 64
    SalaCozI.Update
```

```
                Set hachura(0) = SalaCozI
                Set                          objHatch                          =
ThisDrawing.ModelSpace.AddHatch(acHatchPatternTypePreDefined, "SOLID", True)
                objHatch.PatternScale = 0.1
                objHatch.AppendOuterLoop (hachura)
                objHatch.Evaluate
                objHatch.color = 64
                objHatch.Update
'Cozinha
    InserirCoz(1) = mycenter(0): InserirCoz(2) = mycenter(1): InserirCoz(3) =
0
    InserirCoz(4)  =  mycenter(0):  InserirCoz(5)  =  mycenter(1)  +  CCoz:
InserirCoz(6) = 0
    InserirCoz(7) = mycenter(0) + LCoz: InserirCoz(8) = mycenter(1) + CCoz:
InserirCoz(9) = 0
    InserirCoz(10)  =  mycenter(0)  +  LCoz:  InserirCoz(11)  =  mycenter(1):
InserirCoz(12) = 0
    InserirCoz(13) = mycenter(0): InserirCoz(14) = mycenter(1): InserirCoz(15)
= 0
    Set Coz = ThisDrawing.ModelSpace.AddPolyline(InserirCoz)
    Coz.color = acBlue
    Coz.Update
                Set hachura(0) = Coz
                Set                          objHatch                          =
ThisDrawing.ModelSpace.AddHatch(acHatchPatternTypePreDefined, "PLASTI", True)
                objHatch.PatternAngle = 0.7853981 '45 graus
                objHatch.PatternScale = 0.07
                objHatch.AppendOuterLoop (hachura)
                objHatch.Evaluate
                objHatch.color = acBlue
                objHatch.Update
    End If
'Varanda
    If Varanda = True Then
    InserirVaranda(1) = mycenter(0): InserirVaranda(2) = mycenter(1) + CSala:
InserirVaranda(3) = mycenter(2)
    InserirVaranda(4) = mycenter(0) + LSala: InserirVaranda(5) = mycenter(1) +
CSala: InserirVaranda(6) = 0
    InserirVaranda(7) = mycenter(0) + LSala: InserirVaranda(8) = mycenter(1) +
CSala + CVaranda: InserirVaranda(9) = 0
    InserirVaranda(10) = mycenter(0): InserirVaranda(11) = mycenter(1) + CSala
+ CVaranda: InserirVaranda(12) = 0
    InserirVaranda(13)  =  mycenter(0):  InserirVaranda(14)  =  mycenter(1)  +
CSala: InserirVaranda(15) = mycenter(2)
    Set Varanda = ThisDrawing.ModelSpace.AddPolyline(InserirVaranda)
```

```
    Varanda.color = 66
    Varanda.Update
                Set hachura(0) = Varanda
                Set                          objHatch                          =
ThisDrawing.ModelSpace.AddHatch(acHatchPatternTypePreDefined, "SOLID", True)
                objHatch.PatternScale = 0.1
                objHatch.AppendOuterLoop (hachura)
                objHatch.Evaluate
                objHatch.color = 66
                objHatch.Update
    End If
'Area de servico
    If Aservico = True Then
    InserirServico(1)   =   mycenter(0):   InserirServico(2)   =   mycenter(1):
InserirServico(3) = mycenter(2)
    InserirServico(4)  =  mycenter(0):  InserirServico(5)  =  mycenter(1)  +
CServico: InserirServico(6) = 0
    InserirServico(7) = mycenter(0) - LCoz: InserirServico(8) = mycenter(1) +
CServico: InserirServico(9) = 0
    InserirServico(10) = mycenter(0) - LCoz: InserirServico(11) = mycenter(1):
InserirServico(12) = 0
    InserirServico(13)  =  mycenter(0):  InserirServico(14)  =  mycenter(1):
InserirServico(15) = mycenter(2)
    Set Servico = ThisDrawing.ModelSpace.AddPolyline(InserirServico)
    Servico.color = 171
    Servico.Update
                Set hachura(0) = Servico
                Set                          objHatch                          =
ThisDrawing.ModelSpace.AddHatch(acHatchPatternTypePreDefined, "SOLID", True)
                objHatch.PatternScale = 0.1
                objHatch.AppendOuterLoop (hachura)
                objHatch.Evaluate
                objHatch.color = 171
                objHatch.Update
    End If
'Hall
    If Hall = True Then
    CHall = 1.5 * modulo
    LHall = 1.5 * modulo
    InserirHall(1)  =  mycenter(0)  +  LSala:  InserirHall(2)  =  mycenter(1):
InserirHall(3) = mycenter(2)
```

```
    InserirHall(4) = mycenter(0) + LSala - LHall: InserirHall(5) =
mycenter(1): InserirHall(6) = 0
    InserirHall(7) = mycenter(0) + LSala - LHall: InserirHall(8) = mycenter(1)
- CHall: InserirHall(9) = 0
    InserirHall(10) = mycenter(0) + LSala: InserirHall(11) = mycenter(1) -
CHall: InserirHall(12) = 0
    InserirHall(13) = mycenter(0) + LSala: InserirHall(14) = mycenter(1):
InserirHall(15) = mycenter(2)
    Set Hall = ThisDrawing.ModelSpace.AddPolyline(InserirHall)
    Hall.color = 62
    Hall.Update
                Set hachura(0) = Hall
                Set                          objHatch                          =
ThisDrawing.ModelSpace.AddHatch(acHatchPatternTypePreDefined, "SOLID", True)
                objHatch.PatternScale = 0.1
                objHatch.AppendOuterLoop (hachura)
                objHatch.Evaluate
                objHatch.color = 62
                objHatch.Update
    End If
'Acesso ao banheiro
    CCorredor = CSala
    LCorredor = 1.3 * modulo
    InserirCorredor(1) = mycenter(0) + LSala: InserirCorredor(2) = mycenter(1)
- (2 * CDorm1): InserirCorredor(3) = mycenter(2)
    InserirCorredor(4) = mycenter(0) + LSala + LCorredor: InserirCorredor(5) =
mycenter(1) - (2 * CDorm1): InserirCorredor(6) = 0
    InserirCorredor(7) = mycenter(0) + LSala + LCorredor:: InserirCorredor(8)
= mycenter(1) + CCorredor: InserirCorredor(9) = 0
    InserirCorredor(10) = mycenter(0) + LSala: InserirCorredor(11) =
mycenter(1) + CCorredor: InserirCorredor(12) = 0
    InserirCorredor(13) = mycenter(0) + LSala: InserirCorredor(14) =
mycenter(1) - (2 * CDorm1): InserirCorredor(15) = mycenter(2)
    Set Corredor = ThisDrawing.ModelSpace.AddPolyline(InserirCorredor)
    Corredor.color = 12
    Corredor.Update
                Set hachura(0) = Corredor
                Set                          objHatch                          =
ThisDrawing.ModelSpace.AddHatch(acHatchPatternTypePreDefined, "SOLID", True)
                objHatch.PatternScale = 0.1
                objHatch.AppendOuterLoop (hachura)
                objHatch.Evaluate
                objHatch.color = 12
                objHatch.Update
```

```
'Banheiro
    CWc = 1.4 * modulo
    LWc = 2.2 * modulo
    InserirWc(1) = mycenter(0) + LSala + LCorredor: InserirWc(2) = mycenter(1): InserirWc(3) = mycenter(2)
    InserirWc(4) = mycenter(0) + LSala + LCorredor + LWc: InserirWc(5) = mycenter(1): InserirWc(6) = 0
    InserirWc(7) = mycenter(0) + LSala + LCorredor + LWc: InserirWc(8) = mycenter(1) + CWc: InserirWc(9) = 0
    InserirWc(10) = mycenter(0) + LSala + LCorredor: InserirWc(11) = mycenter(1) + CWc: InserirWc(12) = 0
    InserirWc(13) = mycenter(0) + LSala + LCorredor: InserirWc(14) = mycenter(1): InserirWc(15) = mycenter(2)
    Set Wc = ThisDrawing.ModelSpace.AddPolyline(InserirWc)
    Wc.color = 30
    Wc.Update
                Set hachura(0) = Wc
                Set                          objHatch                          =
ThisDrawing.ModelSpace.AddHatch(acHatchPatternTypePreDefined, "SOLID", True)
                objHatch.PatternScale = 0.1
                objHatch.AppendOuterLoop (hachura)
                objHatch.Evaluate
                objHatch.color = 30
                objHatch.Update
'Dormitorio3
    If Dorm3 = True Then
    CDorm3 = (CSala - CWc) / 2
    LDorm3 = (modulo * 3)
    InserirDorm3(1) = mycenter(0) + LSala + LCorredor: InserirDorm3(2) = mycenter(1) + CWc: InserirDorm3(3) = 0
    InserirDorm3(4) = mycenter(0) + LSala + LCorredor + LDorm3: InserirDorm3(5) = mycenter(1) + CWc: InserirDorm3(6) = 0
    InserirDorm3(7) = mycenter(0) + LSala + LCorredor + LDorm3: InserirDorm3(8) = mycenter(1) + CWc + CDorm3: InserirDorm3(9) = 0
    InserirDorm3(10) = mycenter(0) + LSala + LCorredor: InserirDorm3(11) = mycenter(1) + CWc + CDorm3: InserirDorm3(12) = 0
    InserirDorm3(13) = mycenter(0) + LSala + LCorredor: InserirDorm3(14) = mycenter(1) + CWc: InserirDorm3(15) = 0
    Set Dorm3 = ThisDrawing.ModelSpace.AddPolyline(InserirDorm3)
    Dorm3.color = 20
    Dorm3.Update
                Set hachura(0) = Dorm3
```

```
                Set                             objHatch                            =
ThisDrawing.ModelSpace.AddHatch(acHatchPatternTypePreDefined, "SOLID", True)
                objHatch.PatternScale = 0.1
                objHatch.AppendOuterLoop (hachura)
                objHatch.Evaluate
                objHatch.color = 20
                objHatch.Update
    End If
    If Suite3 = True Then
    InserirWc(1) = mycenter(0) + LSala + LCorredor + LDorm3: InserirWc(2) =
mycenter(1) + CDorm3 + CWc: InserirWc(3) = mycenter(2)
    InserirWc(4)  =  mycenter(0)  +  LSala  +  LCorredor  +  LDorm3  +  LWc:
InserirWc(5) = mycenter(1) + CDorm3 + CWc: InserirWc(6) = 0
    InserirWc(7)  =  mycenter(0)  +  LSala  +  LCorredor  +  LDorm3  +  LWc:
InserirWc(8) = mycenter(1) + CDorm3: InserirWc(9) = 0
    InserirWc(10) = mycenter(0) + LSala + LCorredor + LDorm3: InserirWc(11) =
mycenter(1) + CDorm3: InserirWc(12) = 0
    InserirWc(13) = mycenter(0) + LSala + LCorredor + LDorm3: InserirWc(14) =
mycenter(1) + CDorm3 + CWc: InserirWc(15) = mycenter(2)
    Set Suite3 = ThisDrawing.ModelSpace.AddPolyline(InserirWc)
    Suite3.color = 30
    Suite3.Update
                Set hachura(0) = Suite3
                Set                             objHatch                            =
ThisDrawing.ModelSpace.AddHatch(acHatchPatternTypePreDefined, "SOLID", True)
                objHatch.PatternScale = 0.1
                objHatch.AppendOuterLoop (hachura)
                objHatch.Evaluate
                objHatch.color = 30
                objHatch.Update
    End If
'Dormitorio4
    If Dorm4 = True Then
    CDorm4 = (CSala - CWc) / 2
    LDorm4 = (modulo * 3)
    InserirDorm4(1)  =  mycenter(0)  +  LSala  +  LCorredor:  InserirDorm4(2)  =
mycenter(1) + CWc + CDorm4: InserirDorm4(3) = 0
    InserirDorm4(4)  =  mycenter(0)  +  LSala  +  LCorredor  +  LDorm4:
InserirDorm4(5) = mycenter(1) + CWc + CDorm4: InserirDorm4(6) = 0
    InserirDorm4(7)  =  mycenter(0)  +  LSala  +  LCorredor  +  LDorm4:
InserirDorm4(8) = mycenter(1) + CWc + CDorm4 + CDorm4: InserirDorm4(9) = 0
    InserirDorm4(10) = mycenter(0) + LSala + LCorredor: InserirDorm4(11) =
mycenter(1) + CWc + CDorm4 + CDorm4: InserirDorm4(12) = 0
    InserirDorm4(13) = mycenter(0) + LSala + LCorredor: InserirDorm4(14) =
mycenter(1) + CWc + CDorm4: InserirDorm4(15) = 0
```

```
    Set Dorm4 = ThisDrawing.ModelSpace.AddPolyline(InserirDorm4)
    Dorm4.color = 20
    Dorm4.Update
                Set hachura(0) = Dorm4
                Set                          objHatch                          =
ThisDrawing.ModelSpace.AddHatch(acHatchPatternTypePreDefined, "SOLID", True)
                objHatch.PatternScale = 0.1
                objHatch.AppendOuterLoop (hachura)
                objHatch.Evaluate
                objHatch.color = 20
                objHatch.Update
    End If
    If Suite4 = True Then
    InserirWc(1) = mycenter(0) + LSala + LCorredor + LDorm4: InserirWc(2) =
mycenter(1) + CDorm4 + CWc: InserirWc(3) = mycenter(2)
    InserirWc(4)  =  mycenter(0)  +  LSala  +  LCorredor  +  LDorm4  +  LWc:
InserirWc(5) = mycenter(1) + CDorm4 + CWc: InserirWc(6) = 0
    InserirWc(7)  =  mycenter(0)  +  LSala  +  LCorredor  +  LDorm4  +  LWc:
InserirWc(8) = mycenter(1) + CDorm4 + CWc + CWc: InserirWc(9) = 0
    InserirWc(10) = mycenter(0) + LSala + LCorredor + LDorm4: InserirWc(11) =
mycenter(1) + CDorm4 + CWc + CWc: InserirWc(12) = 0
    InserirWc(13) = mycenter(0) + LSala + LCorredor + LDorm4: InserirWc(14) =
CDorm4 + CWc: InserirWc(15) = mycenter(2)
    Set Suite4 = ThisDrawing.ModelSpace.AddPolyline(InserirWc)
    Suite4.color = 30
    Suite4.Update
                Set hachura(0) = Suite4
                Set                          objHatch                          =
ThisDrawing.ModelSpace.AddHatch(acHatchPatternTypePreDefined, "SOLID", True)
                objHatch.PatternScale = 0.1
                objHatch.AppendOuterLoop (hachura)
                objHatch.Evaluate
                objHatch.color = 30
                objHatch.Update
    End If
'Dormitorio1
    If Dorm1 = True Then
    CDorm1 = (CSala - CWc) / 2
    LDorm1 = (modulo * 3)
    CCorredor = CDorm1
    LCorredor = 1.3 * modulo
```

```
    InserirDorm(1) = mycenter(0) + LSala + LCorredor: InserirDorm(2) =
mycenter(1) - CDorm1: InserirDorm(3) = 0

    InserirDorm(4) = mycenter(0) + LSala + LCorredor + LDorm1: InserirDorm(5)
= mycenter(1) - CDorm1: InserirDorm(6) = 0

    InserirDorm(7) = mycenter(0) + LSala + LCorredor + LDorm1: InserirDorm(8)
= mycenter(1): InserirDorm(9) = 0

    InserirDorm(10) = mycenter(0) + LSala + LCorredor: InserirDorm(11) =
mycenter(1): InserirDorm(12) = 0

    InserirDorm(13) = mycenter(0) + LSala + LCorredor: InserirDorm(14) =
mycenter(1) - CDorm1: InserirDorm(15) = 0

    Set Dorm = ThisDrawing.ModelSpace.AddPolyline(InserirDorm)

    Dorm.color = 20

    Dorm.Update

                Set hachura(0) = Dorm

                Set                          objHatch                          =
ThisDrawing.ModelSpace.AddHatch(acHatchPatternTypePreDefined, "SOLID", True)

                objHatch.PatternScale = 0.1

                objHatch.AppendOuterLoop (hachura)

                objHatch.Evaluate

                objHatch.color = 20

                objHatch.Update

    End If

    If Suite1 = True Then

    InserirWc(1) = mycenter(0) + LSala + LCorredor + LDorm1: InserirWc(2) =
mycenter(1) - CDorm1: InserirWc(3) = mycenter(2)

    InserirWc(4) = mycenter(0) + LSala + LCorredor + LDorm1 + LWc:
InserirWc(5) = mycenter(1) - CDorm1: InserirWc(6) = 0

    InserirWc(7) = mycenter(0) + LSala + LCorredor + LDorm1 + LWc:
InserirWc(8) = mycenter(1) - CDorm1 + CWc: InserirWc(9) = 0

    InserirWc(10) = mycenter(0) + LSala + LCorredor + LDorm1: InserirWc(11) =
mycenter(1) - CDorm1 + CWc: InserirWc(12) = 0

    InserirWc(13) = mycenter(0) + LSala + LCorredor + LDorm1: InserirWc(14) =
mycenter(1) - CDorm1: InserirWc(15) = mycenter(2)

    Set Suite1 = ThisDrawing.ModelSpace.AddPolyline(InserirWc)

    Suite1.color = 30

    Suite1.Update

                Set hachura(0) = Suite1

                Set                          objHatch                          =
ThisDrawing.ModelSpace.AddHatch(acHatchPatternTypePreDefined, "SOLID", True)

                objHatch.PatternScale = 0.1

                objHatch.AppendOuterLoop (hachura)

                objHatch.Evaluate

                objHatch.color = 30

                objHatch.Update

    End If
```

```
    'Dormitorio2
    If Dorm2 = True Then
    CDorm2 = (CSala - CWc) / 2
    LDorm2 = (modulo * 3)
    InserirDorm2(1) = mycenter(0) + LSala + LCorredor: InserirDorm2(2) = mycenter(1) - CDorm2: InserirDorm2(3) = 0
    InserirDorm2(4) = mycenter(0) + LSala + LCorredor + LDorm2: InserirDorm2(5) = mycenter(1) - CDorm2: InserirDorm2(6) = 0
    InserirDorm2(7) = mycenter(0) + LSala + LCorredor + LDorm2: InserirDorm2(8) = mycenter(1) - CDorm2 - CDorm2: InserirDorm2(9) = 0
    InserirDorm2(10) = mycenter(0) + LSala + LCorredor: InserirDorm2(11) = mycenter(1) - CDorm2 - CDorm2: InserirDorm(12) = 0
    InserirDorm2(13) = mycenter(0) + LSala + LCorredor: InserirDorm2(14) = mycenter(1) - CDorm2: InserirDorm2(15) = 0
    Set Dorm2 = ThisDrawing.ModelSpace.AddPolyline(InserirDorm2)
    Dorm2.color = 20
    Dorm2.Update
                Set hachura(0) = Dorm2
                Set                          objHatch                          =
ThisDrawing.ModelSpace.AddHatch(acHatchPatternTypePreDefined, "SOLID", True)
                objHatch.PatternScale = 0.1
                objHatch.AppendOuterLoop (hachura)
                objHatch.Evaluate
                objHatch.color = 20
                objHatch.Update
    End If
    If Suite2 = True Then
    InserirWc(1) = mycenter(0) + LSala + LCorredor + LDorm2: InserirWc(2) = mycenter(1) - CDorm2 - CWc: InserirWc(3) = mycenter(2)
    InserirWc(4) = mycenter(0) + LSala + LCorredor + LDorm2 + LWc: InserirWc(5) = mycenter(1) - CDorm2 - CWc: InserirWc(6) = 0
    InserirWc(7) = mycenter(0) + LSala + LCorredor + LDorm2 + LWc: InserirWc(8) = mycenter(1) - CDorm2: InserirWc(9) = 0
    InserirWc(10) = mycenter(0) + LSala + LCorredor + LDorm2: InserirWc(11) = mycenter(1) - CDorm2: InserirWc(12) = 0
    InserirWc(13) = mycenter(0) + LSala + LCorredor + LDorm2: InserirWc(14) = mycenter(1) - CDorm2 - CWc: InserirWc(15) = mycenter(2)
    Set Suite2 = ThisDrawing.ModelSpace.AddPolyline(InserirWc)
    Suite2.color = 30
    Suite2.Update
                Set hachura(0) = Suite2
```

```
                Set                             objHatch                            =
ThisDrawing.ModelSpace.AddHatch(acHatchPatternTypePreDefined, "SOLID", True)
                objHatch.PatternScale = 0.1
                objHatch.AppendOuterLoop (hachura)
                objHatch.Evaluate
                objHatch.color = 30
                objHatch.Update
    End If
End If


'AreaCompleta = Abs(SalaCozI.Area + Coz.Area + Wc.Area + Corredor.Area)
        If Dorm1.Value = True Then
            AreaCompleta = AreaCompleta + Dorm.Area
        End If
        If Dorm2.Value = True Then
            AreaCompleta = AreaCompleta + Dorm2.Area
        End If
        If Dorm3.Value = True Then
           AreaCompleta = AreaCompleta + Dorm3.Area
        End If
        If Dorm4.Value = True Then
            AreaCompleta = AreaCompleta + Dorm4.Area
        End If
        If Varanda.Value = True Then
            AreaCompleta = AreaCompleta + Varanda.Area
        End If
        If Aservico.Value = True Then
            AreaCompleta = AreaCompleta + Servico.Area
        End If
        If Hall.Value = True Then
            AreaCompleta = AreaCompleta + Hall.Area
        End If
        If Suite1.Value = True Then
            AreaCompleta = AreaCompleta + Suite1.Area
        End If
        If Suite2.Value = True Then
            AreaCompleta = AreaCompleta + Suite2.Area
        End If
        If Suite3.Value = True Then
            AreaCompleta = AreaCompleta + Suite3.Area
```

```
        End If
        If Suite4.Value = True Then
            AreaCompleta = AreaCompleta + Suite4.Area
        End If
            UserForm3.Area.Caption = AreaCompleta & " m²"
 ThisDrawing.Application.ZoomExtents
End Function
Public Function Deletar_Casa() As Casa
    For Each objeto In ThisDrawing.ModelSpace
             objeto.Delete
    Next
Update
End Function
```

INSTANCIAMENTO DA CLASSE CASA

```
Public nCasa As Casa
Public Deleta As Casa

Private Sub Aservico_Click()
If UserForm2.Aservico.Value = True Then
    UserForm3.Aservico.Value = True
    Else
    UserForm3.Aservico.Value = False
End If
End Sub

Private Sub CommandButton1_Click()
Me.Hide
    Set nCasa = New Casa
    Call nCasa.DesenhaComodos(SalaCozI, SalaCozS, Dorm1, Dorm2, Dorm3, Dorm4,
Varanda, Aservico, Hall, Suite1, Suite2, Suite3, Suite4)
Me.Show
End Sub
Private Sub CommandButton2_Click()
UserForm2.Hide
UserForm3.Show
End Sub
```

```
Private Sub CommandButton3_Click()
    Set Deleta = New Casa
    Call Deleta.Deletar_Casa
End Sub

Private Sub Dorm1_Click()
If UserForm2.Dorm1.Value = True Then
    UserForm3.Dorm1.Value = True
    UserForm3.Suite1.Enabled = True
    UserForm2.Dorm2.Enabled = True
    Else
    UserForm3.Dorm1.Value = False
    UserForm3.Suite1.Enabled = False
    UserForm3.Suite1.Value = False
    UserForm2.Dorm2.Enabled = False
    UserForm2.Dorm2.Value = False
End If
End Sub

Private Sub Dorm2_Click()
If UserForm2.Dorm2.Value = True Then
    UserForm3.Dorm2.Value = True
    UserForm3.Suite2.Enabled = True
    Else
    UserForm3.Dorm2.Value = False
    UserForm3.Suite2.Enabled = False
    UserForm3.Suite2.Value = False
End If
End Sub

Private Sub Dorm3_Click()
If UserForm2.Dorm3.Value = True Then
    UserForm3.Dorm3.Value = True
    UserForm3.Suite3.Enabled = True
    UserForm2.Dorm4.Enabled = True
    Else
    UserForm3.Dorm3.Value = False
    UserForm3.Suite3.Enabled = False
    UserForm3.Suite3.Value = False
    UserForm2.Dorm4.Enabled = False
```

```
    UserForm2.Dorm4.Value = False
End If
End Sub

Private Sub Dorm4_Click()
If UserForm2.Dorm4.Value = True Then
    UserForm3.Dorm4.Value = True
    UserForm3.Suite4.Enabled = True
    UserForm2.Dorm1.Enabled = True

    Else
    UserForm3.Dorm4.Value = False
    UserForm3.Suite4.Enabled = False
    UserForm3.Suite4.Value = False
    UserForm2.Dorm1.Enabled = False
    UserForm2.Dorm1.Value = False
End If
End Sub

Private Sub Hall_Click()
If UserForm2.Hall.Value = True Then
    UserForm3.Hall.Value = True
    Else
    UserForm3.Hall.Value = False
End If
End Sub

Private Sub SalaCozI_Click()
If UserForm2.SalaCozI.Value = True Then
    UserForm3.SalaCozI.Value = True
    Else
    UserForm3.SalaCozI.Value = False
End If
End Sub

Private Sub SalaCozS_Click()
If UserForm2.SalaCozS.Value = True Then
    UserForm3.SalaCozS.Value = True
```

```
    Else
    UserForm3.SalaCozS.Value = False
End If
End Sub

Private Sub Suite1_Click()
If UserForm2.Suite1.Value = True Then
    UserForm3.Suite1.Value = True
    Else
    UserForm3.Suite1.Value = False
End If
End Sub

Private Sub Suite2_Click()
If UserForm2.Suite2.Value = True Then
    UserForm3.Suite2.Value = True
    Else
    UserForm3.Suite2.Value = False
End If
End Sub

Private Sub Suite3_Click()
If UserForm2.Suite3.Value = True Then
    UserForm3.Suite3.Value = True
    Else
    UserForm3.Suite3.Value = False
End If
End Sub

Private Sub Suite4_Click()
If UserForm2.Suite4.Value = True Then
    UserForm3.Suite4.Value = True
    Else
    UserForm3.Suite4.Value = False
End If
End Sub

Private Sub Varanda_Click()
If UserForm2.Varanda.Value = True Then
    UserForm3.Varanda.Value = True
```

```
    Else
    UserForm3.Varanda.Value = False
End If
End Sub
```

Livros Grátis

(http://www.livrosgratis.com.br)

Milhares de Livros para Download:

Baixar livros de Literatura
Baixar livros de Literatura de Cordel
Baixar livros de Literatura Infantil
Baixar livros de Matemática
Baixar livros de Medicina
Baixar livros de Medicina Veterinária
Baixar livros de Meio Ambiente
Baixar livros de Meteorologia
Baixar Monografias e TCC
Baixar livros Multidisciplinar
Baixar livros de Música
Baixar livros de Psicologia
Baixar livros de Química
Baixar livros de Saúde Coletiva
Baixar livros de Serviço Social
Baixar livros de Sociologia
Baixar livros de Teologia
Baixar livros de Trabalho
Baixar livros de Turismo