Marcelo Cabral da Silva

Identificação de Estilos de Arquiteturas: Um Processo Dirigido por Conhecimento

Dissertação de Mestrado

Dissertação apresentada ao Departamento de Informática da PUC / RJ como parte dos requisitos para obtenção do título de Mestre em Informática

Orientador: Julio César Sampaio do Prado Leite

Departamento de Informática

Pontifícia Universidade Católica do Rio de Janeiro

Rio de Janeiro, 18 de setembro de 2000

Agradecimentos

Agradeço ao Professor Julio Cesar Sampaio do Prado Leite pela orientação. Não me refiro apenas à orientação desta dissertação, mas também ao exemplo de como ser um Professor.

Ao Professor Sérgio Eduardo Rodrigues Carvalho, por ter acompanhado o início deste trabalho e por ter me apresentado orientação a objetos de uma forma mais abrangente e crítica.

Ao Professor Carlos José Pereira de Lucena pelas chances que tive de apresentar partes deste trabalho no PRONEX, sempre contando com a crítica construtiva de todos os presentes.

Ao colega Márcio Silveira por ter me introduzido ao meio acadêmico de uma forma tão profissional.

Aos colegas Fábio Porto, Paulo Araújo e Sandra Abreu pelo total apoio que me foi dado desde o início do mestrado, certamente foram pessoas determinantes para a realização deste trabalho.

Aos colegas e funcionários deste departamento com os quais convivi durante este período.

Ao meu pai Renan, minha mãe Glória e minha irmã Ana Beatriz por insistirem na minha formação, mesmo quando minha preferência era por jogar bola (há muito tempo atrás), sendo estes fontes inesgotáveis de amor, carinho e compreensão.

À minha esposa Lucianne e ao meu filho Rafael que, com toda certeza, sentiram minha ausência. Destaco o quanto minha esposa foi “mãe e pai”; e quanto meu filho foi compreensivo por todo este período.

Por fim, agradeço e dedico esta dissertação a DEUS. Graças a ELE tenho minha mente iluminada, com muita paz e em condições para cumprir mais esta etapa de minha existência.

Resumo

Um dos assuntos mais recentes na área da engenharia de software, a arquitetura de software, procura sistematizar o processo de desenho de software, fornecendo uma sólida base para o reuso. À medida que sistemas mais complexos vão surgindo, a importância da especificação e do desenho destes sistemas vai aumentando. É neste momento que a arquitetura de software torna-se uma disciplina importante [Shaw+94].

Arquitetura de software diz respeito a macro-estrutura de um sistema. A visão arquitetural é uma visão abstrata dos relacionamentos entre os elementos que formam o sistema, bem como a topologia formada por estes. Não estaremos concentrados no detalhamento de algoritmos ou na representação dos dados. Estudaremos aspectos de como os elementos que formam a arquitetura são compostos e quais são suas respectivas restrições [Bass+98].

Entendemos que arquitetura de software é o conjunto formado por componentes, conectores e restrições sobre o comportamento desses. De outra forma, arquitetura de software é o conjunto das primeiras decisões a serem tomadas na fase de desenho de um sistema. Em geral, estas decisões são as mais difíceis de serem tomadas. Após o desenvolvimento do sistema, as referidas decisões são as mais difíceis de serem modificadas.

Nosso objetivo é prover uma maneira de selecionar arquiteturas com base em uma lista de requisitos. Para tal contaremos com uma base de conhecimento e um método. Para a montagem da base de conhecimento definiremos uma taxonomia e a partir desta identificaremos os relacionamentos entre os elementos. Um conjunto de relacionamentos entre elementos formará um estilos de arquitetura. Em paralelo ao estudo de identificação de estilos de arquiteturas,

utilizaremos também conceitos da Engenharia de Requisitos. Em função disso utilizaremos requisitos como elemento de classificação de arquiteturas e posterior seleção. Faremos uma correlação entre requisitos e elementos, a partir de cada estilo arquitetônico identificado.

Abstract

One of the most recent subjects in the software engineering area, the software architecture, tries to systematize the process of design software, supplying a solid base for the reuse. As more complex systems appear, the importance of system specification and design increases. Therefore software architecture becomes an important discipline [Shaw+94].

Software architecture is related to the system macro-structure. The architectural vision is an abstract vision of the relationships among the elements comprised by a system, as well as the topology they make up. Algorithm detailment or data representation are not the goals of this thesis. Instead, the composition of the elements that form the architectures and their restrictions will be analyzed [Bass+98].

According to our understanding, software architecture is a collection of components, connectors and their behavior restrictions. In other words, software architecture is the collection of the first decisions to be taken during the design phase of a system. In general, these decisions are the most difficult to be taken. After system development, these decisions are the most difficult ones to be modified.

Our objective is to provide a method to select architectures based on a requirement list. For such we will count on a knowledge base, as well as on a method. In order to generate the knowledge base, a taxonomy will be defined and based on it relationships among elements will be identified. The collection of relationships among elements will generate architecture styles. In parallel to the architecture style identification study, we will also use concepts of the Requirements Engineering. Therefore, requirements will be used as elements of architecture classification and

subsequent selection. A correlation between requirements and elements will be made, starting from each of the identified architectural style.

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABELAS

Capítulo 1 – Introdução

Esta dissertação pertence ao contexto da engenharia de software e possui como tema básico o assunto, arquitetura de software. Durante as fases de desenvolvimento de um sistema, podemos falar sobre arquitetura desde o início do projeto. Porém, o momento mais tarde que podemos tratar deste assunto é no início da fase do desenho técnico [Bass+98]. Trataremos este assunto objetivando selecionar estilos de arquiteturas a partir dos requisitos de um sistema em estudo, sob a óptica da reusabilidade. Ou seja, a partir de uma arquitetura indicada, com base nos requisitos de um sistema em estudo, identificaremos outros elementos que serão pertinentes na referida arquitetura. A identificação dos elementos pertinentes torna-se possível a partir de uma base de conhecimento a ser elaborada e implementada.

Esta dissertação é apresentada em seis capítulos a saber.

No capítulo 2, estaremos abordando os elementos básicos da arquitetura de software, como por exemplo, estruturas de classificação, instâncias das estruturas de classificação e respectivos relacionamentos. Para melhor entendimento do caminho utilizado para a escolha dos elementos básicos, falaremos superficialmente sobre LDA's (Linguagens de Descrição de Arquitetura).

Após análise de todos os elementos básicos e das LDA's, poderemos propor uma taxonomia, bem como um modelo de representação dos conhecimentos estáticos e dinâmicos, tendo como base a referida taxonomia. A representação dos conhecimentos estáticos e dinâmicos formarão a base de conhecimento, a qual tornará possível a identificação e catalogação dos estilos de arquiteturas.

O capítulo 3 tratará das heurísticas que nos levarão a determinação de um método de contagem. Este método de contagem atribuirá pontos para cada estilo de arquitetura, conforme requisitos a serem identificados no sistema em estudo[1]. Veremos também que o método de contagem somente se torna possível a partir da taxonomia proposta.

No capítulo 4 falaremos das técnicas de Inteligência Artificial (IA) que utilizamos para implementar a base de conhecimento e o método de contagem. Por último, apresentaremos um software desenvolvido, visando a implementação da base de conhecimento e do método de contagem. A este software demos o nome de SIEA – Sistema Identificador de Estilos de Arquiteturas.

No capítulo 5 iremos descrever 10 (dez) estudos de caso que nos ajudaram no amadurecimento da representação e do método proposto, bem como na implementação do SIEA. Os quatro primeiros estudos de caso serão sobre sistemas implementados pelo autor desta dissertação. Dois estudos de caso são uma auto-análise, ou seja, qual arquitetura deveremos implementar no SIEA. Os quatro últimos estudos de caso falarão sobre sistemas descritos e já classificados em [Buschmann+98]. Faremos um comparativo entre os resultados obtidos em [Buschmann+98] e o nosso método de classificação.

Finalizamos com uma conclusão do assunto e do trabalho proposto. Falaremos sobre as contribuições esperadas e indicaremos os próximos passos que julgamos pertinentes a partir das heurísticas apresentadas. A conclusão é descrita no capítulo 6.

---

[1] *Sistema em estudo* é o sistema que submeteremos ao método de contagem e sob o qual indicaremos estilos de arquiteturas para sua respectiva implementação.

1.1 Motivação

Uma das vantagens da consolidação da área de arquitetura de software é que, além da divulgação de estilos, seria possível construir sistemas de software em bases mais sólidas, inclusive com a possibilidade de reutilização. No entanto, a identificação de um estilo apropriado de arquitetura para um dado problema ainda depende fundamentalmente da experiência dos engenheiros de software que precisam tomar essa decisão. Muitas vezes a comunidade de software parece não lembrar que tornar disponíveis arquiteturas de software não resolve de todo o aspecto do emprego dessas arquiteturas, sob a óptica da reutilização. Para isso, precisamos, além da organização dessas arquiteturas em taxonomias, de mecanismos eficazes para recuperação ou escolha da arquitetura alvo do reuso.

Desta forma, devemos propor uma taxonomia para a representação dos elementos básicos que formam os estilos de arquitetura, bem como uma maneira adequada de armazenar o conhecimento [Klein+99.2]. Quando nos referimos a *taxonomia*, significa uma única taxonomia para todos os estilos de arquitetura apresentados.

Para a proposição da taxonomia e respectiva representação, teremos que estudar todo o vocabulário que envolve um estilo de arquitetura. Teremos que estudar os comportamentos dos componentes e dos conectores, os relacionamentos existentes entre estes e suas restrições.

Após a montagem da base de conhecimento, a partir de uma taxonomia, é que passaremos a propor heurísticas para a formação do método de identificação dos estilos de arquiteturas.

1.2 Visão Geral

Primeiramente vamos definir o que é uma estrutura de classificação. Este será o ponto inicial no sentido de estarmos fortemente apoiados em [Shaw+96.1] e [Bass+98]. Estas duas referências

enfatizam o fato das arquiteturas serem formadas por estruturas, bem como pelos relacionamentos dessas. Faremos uma análise das estruturas de classificação primárias e auxiliares. Abordaremos a questão das estruturas auxiliares serem derivadas de relacionamentos das estruturas primárias. Com base nas conclusões sobre o estado comportamental das estruturas de classificação vamos propor uma representação das mesmas.

Após a definição das estruturas de classificação, vamos detalhar as instâncias destas estruturas. Verificaremos como e quando estas ocorrem. Analisaremos os grupos formados por estas instâncias e a estes grupos chamaremos de *estilos de arquiteturas*.

Vamos verificar que os referidos grupos, primeiramente, formados pelos relacionamentos entre as instâncias das estruturas de classificação, são grupos "harmônicos". Ou seja, são grupos que as instâncias de diferentes estruturas convivem bem entre si, conforme classificação a ser apresentada. A observação destes relacionamentos nos levará a questionamentos de tentar todos os relacionamentos possíveis entre instâncias de estruturas de classificação. A nossa base de conhecimento deverá ser formada por todos esses relacionamentos possíveis das estruturas de classificação e respectivas instâncias [Bass+98].

Com o objetivo de verificar o que influencia a formação da taxonomia a ser proposta, faremos uma análise das LDA's (Linguagens de Descrição de Arquitetura). Esta análise visa observar como uma LDA representa um componente, como representa um conector, como representa um estilo de arquitetura, entre outras características. Com base nos pontos comuns entre as LDA's, transferiremos nossas conclusões para a taxonomia proposta.

A partir dos quatro itens analisados: estruturas de classificação, instâncias, respectivos relacionamentos e LDA's; vamos propor uma taxonomia como resultado desta nossa análise. Utilizaremos na taxonomia a ser apresentada, técnicas de Inteligência Artificial para a representação do conhecimento obtido. Com a taxonomia definida representaremos alguns estilos de arquiteturas, a partir de redes semânticas particionadas.

Agregaremos à nossa base de conhecimento, já iniciada pela formação da taxonomia, um conjunto de requisitos. A estes requisitos daremos o nome de *requisitos de qualidade geral*. Tais requisitos estarão associados às estruturas de classificação, dentro de cada estilo de arquitetura.

Os requisitos de qualidade geral, quando associados aos requisitos de um sistema em estudo, nos dará a condição para início do método de contagem.

O método de contagem atribuirá uma quantidade de pontos para cada estilo de arquitetura existente na base de conhecimento. Esta contagem torna-se possível a partir do momento que os requisitos de qualidade geral estão associados às estruturas de classificação e tais estruturas são relacionadas entre si, com uma pontuação pré-definida - *grau de relacionamento* e *avaliação*.

A Figura 1 apresenta uma macro-visão do método de identificação de estilos de arquiteturas que iremos propor.

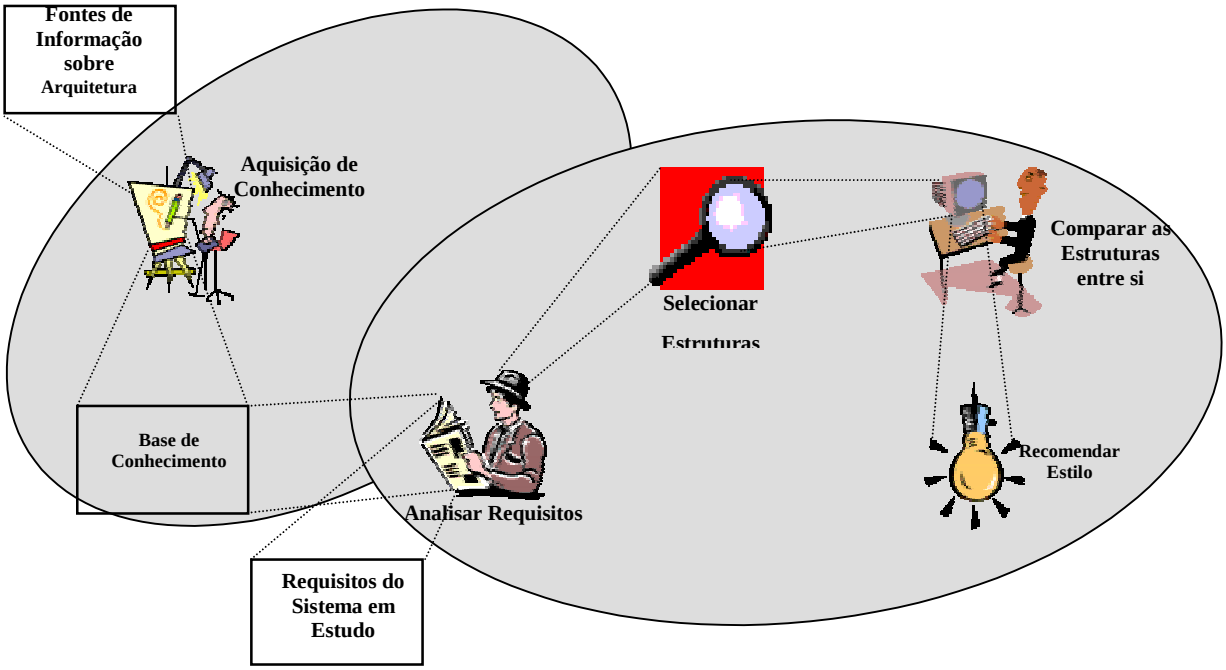

Figura 1 – Macro-visão da formação da base de conhecimento e do método proposto

1.3 Histórico

Em meados dos anos 70 Warnier [Warnier+74], Jackson [Jackson+75], Yourdon [Yourdon+78], Constantine [Yourdon+78], entre outros, reuniram práticas comuns em programação e desenho de sistemas. Estas práticas foram postas em padrões preestabelecidos e respectivos métodos, técnicas e ferramentas foram propostas para apoiar estes padrões. Tais padrões tornaram possível comparar um sistema em desenvolvimento ao padrão proposto. Assim, era verificado o que poderia ser melhorado no produto a ser desenvolvido ao mesmo tempo que se fornecia um "guia inicial" para a construção do sistema (ou programa) em questão.

Em um nível mais alto de abstração que o proposto nos anos 70, os estudiosos de arquitetura de software vêm se preocupando em identificar *estilos* de soluções e suas características comuns.

Estando estes *estilos* e suas características bem identificadas, passa-se a poder compará-los com sistemas em desenvolvimento ou já existentes. Com base na taxonomia e na semântica preexistentes, proveniente dos estilos e características, podemos sugerir melhorias no produto a ser desenvolvido ou já existente.

A idéia arquitetura de software e a forma de pensar sobre esta questão foi fortemente influenciada pela área de arquitetura propriamente dita. Dr. Christopher Alexander propôs determinadas formas de interpretação e resolução de problemas, as quais hoje tentamos praticar na área da engenharia de software. Alexander iniciou uma nova forma de pensar em arquitetura. Ele acreditava que o estudo profundo das leis que governam uma determinada situação nos levaria a uma solução que seria aplicável aquela situação; bem como seria a solução de outras situações governadas pelas mesmas leis [Alexander+77] [Alexander+78].

A comunidade da ciência da computação identificou-se com o contexto onde estava inserido o problema de Alexander, sua busca por leis que vão reger as soluções. Somando a isso o fato de, em um outro problema, ao encontrarmos o mesmo conjunto de leis, indicarmos soluções semelhantes.

A nossa busca incansável por requisitos bem definidos e o longo processo de transformação destes requisitos em solução de software, com possibilidade de reuso, fez os estudiosos deste assunto buscarem inúmeras referências nos trabalhos de Alexander. Inclusive copiando os próprios nomes: arquiteturas e padrões.

Entendemos que o estudo profundo das leis propostas por Alexander deve estar contemplada nos requisitos; e que a solução está associada aos elementos básicos que formam um estilo de arquitetura. Desta forma, um mesmo conjunto de requisitos deve nos levar as mesmas indicações de arquiteturas. Visando práticas de reuso na engenharia de software, passa a ser nosso interesse

armazenar regras e fatos em uma base de conhecimento para a utilização das mesmas soluções em problemas semelhantes.

1.4 Estado Atual

A seguir, descrevemos o atual estado das pesquisas em arquitetura de software, bem como onde está localizado nosso trabalho.

Recentemente, os trabalhos de pesquisa em arquitetura têm tido como núcleo central os resultados apresentados pelo grupo da professora Mary Shaw, na Universidade Carnegie Mellon. Nosso trabalho apoia-se fortemente no proposto por Shaw [Klein+99.2] [Bass+98] [Shaw+96] [Shaw+96.1] [Shaw+94] no propósito de apresentar uma taxonomia (estilos de arquiteturas e respectivas características) em forma de base de conhecimento. Às pesquisas em arquitetura de software tem se concentrado na definição de taxonomias [Klein+99.2] [Shaw+96.1] [ACM+95.2]; definição de linguagens formais e semi-formais [Shaw+94] [CMU01] [Rapide01] para a representação de arquiteturas; construção de *frameworks* de domínios específicos [Bass+98]; e o estudo de modelos e especificações formais da arquitetura de software.

Atualmente conseguimos identificar, de maneira empírica, ou não, paradigmas arquiteturais, como por exemplo: tubos e filtros, camadas, cliente-servidor, repositórios, entre outros. Porém, a utilização de tais arquiteturas, em geral, não ocorre por uma escolha feita pelo desenvolvedor, a partir de um método. Em geral, a arquitetura formada é percebida após a construção de boa parte do software. Como conseqüência desta “escolha” sem critério, o desenvolvedor não consegue explorar todas as formas da arquitetura escolhida, bem como outras possíveis soluções. Outras vezes o desenvolvedor percebe que a arquitetura formada durante o desenvolvimento do software não é adequada à solução que deve ser dada.

Esta dissertação estará abordando como deveremos classificar um software em um estilo de arquitetura, a partir de requisitos, os quais chamaremos de *requisitos de qualidade geral*. Para a realização deste método é necessária a existência de uma taxonomia para todos os estilos arquiteturais. Ou seja, quais são as arquiteturas existentes e quais são suas respectivas características. Esta taxonomia será representada sob a forma de uma *rede semântica* [Rich+83] [Tanimoto01], a ser utilizada no método de contagem. O método de contagem assumirá características de um *sistema de produção* [Brownston+85].

Quando se fala em características aparece uma outra questão, ainda não definida, que é a granularidade do problema a ser tratado. Em arquitetura de software temos, por exemplo, o elemento *componente*, que pode ser um programa, um subsistema, ou até mesmo um sistema completo. Porém, independente dessa granularidade, o que estamos buscando é a estrutura comportamental deste *componente*.

Isso posto, estaremos abordando o estado atual de pesquisa de três grupos. O primeiro grupo é o da Professora Mary Shaw junto com Paul Clements. O segundo grupo sendo formado por Rick Kazman, Paul Clements, Abowd e Len Bass. O terceiro e último grupo a ser analisado é composto por Ahmed Abd-Allah e Barry Boehm. Serão apresentados nesta ordem devido a influência que cada um destes exerceu em nosso trabalho, sendo o grupo de Shaw o de maior influência.

1.4.1 Shaw e Clements

O trabalho realizado por este grupo de pesquisa é a apresentação dos estilos de arquitetura a partir de tipos de componentes e de conectores [Garlan+93] [Shaw+94] [Shaw+96] [Shaw+96.1]. É sugerida a classificação de estilos de arquiteturas baseados em um conjunto de características que

envolvem fluxo de processamento e disposição dos dados. Para cada estilo de arquitetura identificado é descrito de maneira própria o fluxo do processamento e a disposição dos dados. Mesmo assim, não chegam a uma taxonomia, nem a uma granularidade única para todos os estilos de arquiteturas apresentados.

Basicamente a proposta de Shaw caracteriza:

- Tipos de componentes e conectores;
- O controle do processamento;
- Como o dado trafega pelo sistema;
- Como o dado e o controle interagem;

Além disso, o trabalho do grupo aborda características de sincronismo entre os componentes, o que pode nos induzir a uma topologia, como por exemplo, a sequencialidade do estilo de arquitetura tubos e filtros.

Todos os resultados dos seus trabalhos são apresentados na forma de figuras genéricas e tabelas.

Com a proposta de Shaw alguns estilos foram descritos: orientado a objetos, repositórios, baseado em eventos, programa principal / sub-rotinas, entre outros.

1.4.2 Kazman, Clements, Abowd e Bass

O grupo de Kazman enxerga as características comportamentais de um sistema, para posteriormente fazer algum tipo de classificação [Bass+98] [Kazman+99]. Estas características são chamadas de elementos arquiteturais, que podem ser dois: componente e conector. Cada um destes dois elementos deve ser observados sob duas visões distintas: temporal (ao longo do tempo) e estática.

A visão temporal de um elemento arquitetural nada mais é do que a análise do comportamento de tal elemento ao longo do tempo. As seguintes características são utilizadas na visão temporal:

- Controle do fluxo de processamento ao longo do tempo;
- Controle do fluxo de dados ao longo do tempo;
- Transformação dos dados ao longo do tempo.

A visão estática de um elemento arquitetural descreve as características de tais elementos que não variam ao longo do tempo. Buscaram as seguintes características na visão estática:

- O escopo dos dados de um elemento;
- O escopo dos controles de um elemento;
- A capacidade de um elemento transformar os dados;
- Quantidade de elementos que podem ser conectados a porta de um outro determinado elemento;
- Se um elemento aceita retroalimentação (características de *feedback*).

O trabalho deste grupo de pesquisa foi o principal influenciador em duas características de nossa taxonomia. A parte de definição de portas de comunicação de componentes nos levou as seguintes proposições:

- Considerar componente e conector como estruturas primárias e as demais como estruturas compostas destas duas;
- Considerar que conector é uma estrutura e não uma simples parte de um componente.

O trabalho de Kazman apresentou características muito fortes para nosso entendimento dos elementos arquiteturais. Porém, fala muito pouco em como agrupar tais elementos com o objetivo de formar estilos de arquiteturas.

1.4.3 Abd-Allah e Boehn

Abd-Allan descreve estilos de arquiteturas a partir da utilização de Z [Abd-Allah+95] [Gacek+98.2]. Seu grupo construiu um protótipo que focava nos estilos: programa principal / sub-rotinas, tubos e filtros, processos distribuídos e baseado em eventos. Este protótipo tinha o objetivo de detectar possíveis misturas dos referidos estilos arquiteturais, a partir da composição de um sistema em estudo.

Abd-Allan apoia-se fortemente na descrição de elementos que podem ser combinados de formas diferentes para a formação dos estilos de arquitetura. São os seguintes elementos:

- Porta – sempre está associada a um outro componente de controle. Conecta, pelo menos, dois componentes de controle. A porta estará sendo combinada com dois outros elementos: conector de dados e conector de controle.
- Dados – são os dados que refletem a modelagem do estado de cada componente.
- Componente de controle – são os elementos que modelam os dados e que determinam o fluxo de tais dados.
- Objeto – é o encapsulamento dos dados e dos componentes de controle em um único elemento.
- Disparo – a partir do estado assumido pelos dados pode existir um elemento responsável pelo disparo de ações dos componentes de controle. Este elemento está associado principalmente ao estilo de arquitetura repositório.

Durante o trabalho de Abd-Allah, mais especificamente na parte que trata a mistura de estilos, ele cita elementos que influenciam a existência de outros elementos, bem como o caso inverso. Ou seja, a presença de determinados elementos em nada influencia a existência de outros. Em nosso trabalho isso refletiu a formação da parte da base de conhecimento que trata do relacionamento

das estruturas de classificação, bem como o relacionamento dos requisitos de qualidade geral com tais estruturas.

1.5 Arquitetura, Padrão, Framework

As atuais pesquisas sobre reuso de software abordam o assunto com diversas técnicas, bem como em granularidades diferentes. Basicamente três formas de reuso têm tido destaque na engenharia de software, são elas: arquitetura de software, padrões e *framework*. Ao mesmo tempo que as formas de reuso vêm sendo estudadas, essas vêm se confundindo. A seguir apresentamos um breve entendimento do que consideramos ser cada uma destas.

Entendemos por arquitetura de software aquilo proposto em [Shaw+96], ou seja, a macro-estrutura comportamental dos elementos que formam um software, bem como suas restrições.

Padrões, vêm de encontro aquilo proposto em [Gamma+95]. Ou seja, é algo mais físico, que necessariamente envolve codificação. São pedaços de software que funcionam independentemente conforme concebidos em sua forma original. Necessariamente deve passar por uma implementação orientada a objetos.

Entendemos por *framework* aquilo exposto em [Buschmann+98]. São códigos incompletos, os quais devem ser instanciados e completados para que possuam alguma utilidade. Em geral, são de algum domínio específico.

O nosso entendimento sobre arquitetura de software, padrão e *framework*, em nenhum momento tem o objetivo de ser completo ou esgotar o assunto. Continuamos com questões do tipo: *Model / View / Controller* (MVC) [Gamma+95] é um padrão ou é uma arquitetura ? Pode ser codificado como um *framework* ? CORBA (*Common Object Request Broker Architecture*) é um padrão ou uma arquitetura ?

Desta forma, nesta dissertação, continuamos no propósito de entender arquitetura de software como um estudo do comportamento dos relacionamentos entre os componentes que formam um software.

Capítulo 2 – Base de Conhecimento

Esse capítulo objetiva estudar e identificar os elementos que formam um estilo de arquitetura. Os elementos são identificados a partir de observações das representações de arquiteturas de software feitas pelos três grupos de pesquisa anteriormente citados bem como outras fontes da literatura. As arquiteturas, em geral, são apresentadas em alto nível e de maneira informal. Verificamos que as tentativas de tais representações possuem desenhos sem padrão visual e, às vezes, com muita semântica inclusa nas representações gráficas. Tal fato nos levará à busca de mecanismos formais, para a determinação da taxonomia.

Quando falamos em formalismo para arquitetura de software temos dois caminhos a seguir: LDA's (Linguagens de Descrição de Arquitetura) e lógicas formais. Vamos observar a representação de estilos de arquitetura em algumas LDA's, bem como o que existe de comum entre estas. Não é escopo desta dissertação a representação de arquiteturas a partir das lógicas preposicional, de predicados, ou de segunda-ordem. Exemplos deste tipo de formalismo podem ser encontrados no capítulo 6 de [Shaw+96].

Quando observamos a documentação técnica de um sistema, uma das primeiras representações que nos são apresentadas é um diagrama com componentes inter-conectados. Após o diagrama, ou antes, é comum encontramos textos genéricos e que por vezes tentam enfatizar alguma característica maior, como por exemplo, aspectos cliente-servidor do software em estudo [Garlan+93].

Como exemplo genérico, a representação de uma arquitetura de software ocorre conforme a Figura 2. Observamos uma seqüência de três componentes. Entendemos que o processamento

inicia pelo componente mais à esquerda e que tal componente possui um tipo de ligação (tracejado) com um componente de um outro tipo, cujo escopo é externo a nossa análise. O segundo componente interage com um cliente que solicita serviços (satélite), bem como um banco de dados. O segundo componente necessita da ajuda de três estruturas de dados auxiliares para executar seu processamento. O terceiro componente finaliza o processo.

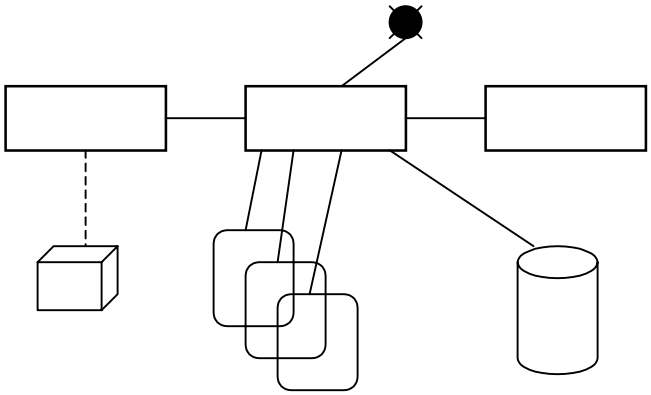

Figura 2 – Representação genérica da arquitetura de um software [Shaw+96]

Visto que estamos falando de um modelo informal, existem muitas variações da representação genérica apresentada na Figura 2, para os diversos tipos de sistemas existentes. Também podemos supor que a diagramação utilizada na representação da Figura 2, pode aparecer com outra semântica em um outro sistema a ser descrito. Porém, algumas características estão sempre presentes: a existência de componentes e a existência de conectores. Anteriormente ao estudo dos estilos arquiteturais, qualquer software podia ser visto como um conjunto de unidades de processamento, ou componentes, que se comunicavam, a partir de conectores, a fim de apresentar um resultado final.

Observamos que o texto que precede a Figura 2 é bastante informal. Podemos citar como exemplos de dúvidas, as seguintes perguntas:

- Os três componentes estão no mesmo nível de abstração ?
- A comunicação entre os componentes ocorre da mesma forma ?
- O que significa a linha contínua ? E a linha tracejada ?
- Quais outros tipos de interações existem entre o sistema representado e o cliente (satélite) ?

Em fim, diversas perguntas surgem em razão da falta de uma taxonomia e por falta de semântica. É neste momento que surge, a necessidade da busca de elementos mais bem definidos para a representação dos estilos arquiteturais. Chamaremos estes elementos de *estruturas de classificação*.

2.1 Estruturas de Classificação

Em um primeiro momento observamos duas estruturas de classificação: componente e conector. A observação de que um componente, ou unidade de processamento, é uma estrutura de classificação, nos parece bastante intuitiva, visto que é a partir destas unidades que sistemas são elaborados. Observamos que a representação de componentes é um ponto em comum nos três grupos de pesquisa citados anteriormente.

A conclusão de que conectores também são estruturas de classificação nos parece um pouco mais sutil, uma vez que podemos interpretar tais conectores como a comunicação entre as interfaces dos próprios componentes. Ou seja, conector é uma parte do próprio componente. Porém, entenderemos por conectores como estruturas próprias que, em geral, não requerem grande quantidade de código, porém são responsáveis pelos mecanismos e formas pelas quais os componentes se comunicam. Esta visão de conector foi exposta pelo grupo de pesquisa de Kazman, conforme citado anteriormente.

Isso posto, e aceitando o fato de que conector não é meramente a interface entre componentes, possuímos as duas *estruturas de classificação primárias*: *componente* e *conector*. A partir dos relacionamentos das estruturas de classificação primárias, chegaremos as *estruturas de classificação auxiliares*.

Novamente fazendo referência aos trabalhos de Alexander [Alexander+77] [Alexander+78], verificamos que para compormos a arquitetura de uma construção, são necessárias as distintas visões de diversos especialistas: engenheiro elétrico, engenheiro hidráulico, projetista de vigas e concretos, decorador, entre outros. Cada um destes especialistas possuem suas visões arquiteturais independentes. Em algum momento, estas arquiteturas devem interagir para formar a arquitetura maior, chamada arquitetura da construção. Entendemos que o resultado de determinadas arquiteturas limitam as demais. Por exemplo, um cálculo estrutural, que provem como resultado a colocação de uma viga de concreto pode alterar, ou limitar a arquitetura hidráulica ou a decoração.

Raciocínio semelhante utilizamos na busca de *estruturas de classificação auxiliares*. Nosso cálculo estrutural e posteriores vigas são as *estruturas de classificação primárias*. A partir do comportamento dos componentes e dos conectores é que vamos observar as *estruturas de classificação auxiliares*. As estruturas de classificação auxiliares somente existem a partir do comportamento dos componentes e dos conectores. Sem a existência destes dois elementos jamais surgiriam as demais estruturas de classificação aqui relacionadas.

A seguir, apresentamos a descrição das estruturas de classificação auxiliares que observamos em nossos estudos. Uma vez que entendemos que estruturas de classificação auxiliares são formadas a partir de estruturas primárias, ao final de cada descrição indicamos qual estrutura primária participa da formação da estrutura auxiliar em questão.

2.1.1 Seqüência de Comunicação

É a ordem em que dois ou mais componentes se comunicam, sendo os conectores responsáveis por esta comunicação. Basicamente existem duas formas de comunicação: a primeira que respeita uma *ordem preestabelecida* para os componentes, ou seja, para um dado passar do componente 1 para o componente 3, antes o mesmo deve passar pelo componente 2. A segunda forma é quando *não existe uma ordem preestabelecida*, ou seja, o dado pode passar do componente 1 para o 3, sem necessariamente haver a interferência do componente 2 [Shaw+96.1].

A *seqüência de comunicação* está preocupada com o caminho a ser percorrido entre o componente inicial e o componente final.

Estrutura (s) primária (s) participante (s): componentes.

2.1.2 Sincronismo

Indica a dependência entre os componentes que interagem para produzir os resultados esperados pelo requisitante do serviço. Pode ser *linear* - o componente precisa aguardar o final do processamento de outro (s) componente (s) para realizar seu processamento e *não-linear*, ou seja, o componente não precisa esperar pelo processamento de outro (s) componente (s) para realizar seu processamento.

O *sincronismo* está preocupado em validar a condição de início de execução de um único componente em relação ao (s) seu (s) predecessor (es).

Estrutura (s) primária (s) participante (s): componentes.

2.1.3 Prioridade de Execução entre Componentes

É a escolha da ordem em que os componentes serão executados, caso em uma dada situação haja a possibilidade da execução de mais de um componente.

A situação de não existir prioridade de execução entre componentes pode ocorrer basicamente por três fatores: ausência de requisito para tal facilidade, ou por restrições de hardware ou software. Também nos confrontamos com situações onde é fundamental a existência de prioridade de execução para que o software produza o resultado esperado.

A *prioridade de execução* está preocupada em saber qual componente deve ser executado no momento presente, a partir de um estado gerado no passado. Esta estrutura disponibiliza a CPU para um componente ser executado. Imediatamente depois é que a estrutura *sincronismo* verifica se o componente está, ou não, em condição de início de execução.

Estrutura (s) primária (s) participante (s): componentes.

2.1.4 Modo de Execução

Indica a possibilidade de haver mais de um componente executando por vez, para realizar os resultados esperados pelo requisitante do serviço. Pode ser *seqüencial* (quando apenas um componente pode ser executado por vez) ou *paralelo* (quando dois ou mais componentes podem ser executados ao mesmo tempo).

O *modo de execução* limita as possibilidades da *prioridade de execução* e *sincronismo*.

Estrutura (s) primária (s) participante (s): componentes.

2.1.5 Reuso

Dado um componente típico de um determinado estilo de arquitetura, são verificadas suas características com o intuito de avaliar a probabilidade de reuso do mesmo. Em geral, quanto maior o grau de *encapsulamento* e uma interface bem definida, maior será a chance de reuso dos

componentes. Quando analisamos a estrutura de reuso, procuramos identificar a existência, ou não, de IDL's (*Interface Definition Language*) para a comunicação de um componente em diversas situações / ambientes.

Estrutura (s) primária (s) participante (s): componentes e conectores.

2.1.6 Encapsulamento de Componentes

É a capacidade da função de um componente ser executada independente de demais componentes, bem como a capacidade de limitar a sua visibilidade e sua interface estar bem definida para diversas situações.

Estrutura (s) primária (s) participante (s): componentes e conectores.

2.1.7 Comunicação entre Componentes

É a forma como os conectores propiciam o fluxo dos dados entre componentes. É a forma como são tratados os pacotes de comunicação entre os componentes. A troca ou passagem de dados entre componentes pode ocorrer de diversas maneiras. Observamos algumas formas de comunicação em nossos estudos: passagem de parâmetro por valor e por referência, envio de mensagens, mecanismos de *callback*, entre outras.

Estrutura (s) primária (s) participante (s): conectores.

2.1.8 Modo de Dados

Indica como os dados são disponibilizados no sistema. Em geral, podem assumir dois valores: *passados* (dados são enviados de componente para componente) ou *compartilhados* (dados são disponibilizados através de uma estrutura de dados central).

Os trabalhos do grupo de Kazman e o modo de dados compartilhado foram os maiores influenciadores para determinarmos que conector é uma estrutura de classificação, e não simplesmente uma parte do componente. No modo de dados compartilhado tais estruturas de

dados assumem características próprias, independente dos conectores. Como por exemplo podemos citar o *blackboard* descrito em [Shaw+96] [Shaw+96.1].

Estrutura (s) primária (s) participante (s): conectores.

2.1.9 Mecanismo de Feedback

Identifica se um componente pode chamar um outro que já o chamou, mesmo que indiretamente (através de outros componentes), formando desta forma um ciclo. Este mecanismo normalmente é utilizado para avaliação e correção de saídas esperadas, podendo também interagir com o meio externo (sendo esse mais um componente).

Estrutura (s) primária (s) participante (s): componentes e conectores.

2.1.10 Topologia

É a forma geométrica que as conexões entre os componentes e o fluxo de dados podem assumir em um determinado estilo de arquitetura [Paula+98.2]. Está diretamente relacionada com as estruturas de classificação *seqüência de comunicação* e *modo de execução*. A forma geométrica assume características específicas que determinam de maneira marcante uma arquitetura. Podemos citar como exemplo a topologia estrelar, quando todos os componentes se comunicam a partir de uma base de dados única, sendo este o único ponto de comunicação entre tais componentes. Outro exemplo marcante de topologia é a em camadas, cuja forma de atuação dos componentes dá o nome à própria arquitetura.

Estrutura (s) primária (s) participante (s): componentes e conectores.

2.1.11 Organização de Processamento

Indica se o processamento do sistema é *batch* ou *interativo*. Entendemos por processamento *batch* aquele que não sofre a interferência do usuário para algum processo decisório. Já o

processamento interativo, disponibiliza, em um dado momento, a opção do usuário interferir no fluxo do processamento.

Estrutura (s) primária (s) participante (s): componentes e conectores.

Os itens 2.1.1, 2.1.2, 2.1.3 e 2.1.4 abordam enfoques diferentes sobre o mesmo assunto: ordem de execução de componentes, com possibilidade de processamento concorrente e / ou paralelo. Entendemos que a implementação prática dos quatro itens passa por uma solução única: hardware e sistemas operacionais que tratem de priorização e processamento multitarefa. Porém, as quatro visões são necessárias, no sentido de propormos a definição do método de contagem.

As definições das estruturas de classificação nos levam a observar que para identificarmos um estilo arquitetural são necessárias análises a partir de diversos aspectos. Tais aspectos nunca são ambíguos, porém estão fortemente relacionados no propósito da definição de um estilo de arquitetura. Também observamos que as estruturas de classificação, embora não sejam ambíguas, também não são totalmente independentes. Ou seja, dada a ocorrência de uma estrutura de classificação em uma forma, isso pode influenciar fortemente a ocorrência de uma outra estrutura de classificação com uma forma específica. Veremos este assunto com mais detalhes quando abordarmos os relacionamentos entre estruturas de classificação. Com isso, podemos novamente observar características semelhantes entre os modelos arquiteturais propostos por Alexander [Alexander+77] [Alexander+78] e os propostos em arquitetura de software.

2.2 Instâncias das Estruturas de Classificação

Durante as definições das estruturas de classificação primárias e auxiliares, foram observadas as formas como cada uma dessas ocorriam em seus respectivos estilos de arquitetura.

A seguir, apresentamos algumas formas para cada uma das estruturas. A estas formas demos o nome de *instâncias das estruturas de classificação*.

2.2.1 Componentes

Instância	Descrição
Filtros	São componentes caracterizados pelo fato de receberem dados a partir de tubos de dados, fazer uma transformação local com tais dados, e enviá-los através de outro tubo para o componente seguinte ou para o meio externo. Em geral, os filtros são componentes independentes, que não compartilham seu estado com os demais componentes do sistema. Esta característica faz com que um filtro não saiba quais outros filtros se comunicam com este. Tudo ocorre a partir dos tubos.
Camadas	São três tipos de componentes caracterizados por três fatos básicos, respectivamente. O primeiro, de sempre haver uma, e apenas uma, camada que se comunica com o meio externo (camada mais externa). Esta primeira camada também se comunica com uma, e apenas uma camada do sistema em questão. O segundo, de sempre haver uma, e apenas uma, camada que faz a interface com o middleware ou hardware (camada mais interna). Esta última camada também se comunica com uma, e apenas uma camada do sistema em questão. O terceiro, é o fato das demais camadas medianas se comunicarem com duas, e apenas duas, camadas vizinhas.

Fontes de Informação[2]	São componentes que em geral são implementados em sistemas especialistas, que visam a representação do conhecimento. Caracterizam-se pela sua comunicação ser feita a partir de uma base de dados central chamada de *blackboard*. Estes componentes possuem seu processamento determinado a partir dos estados assumidos pela base de dados central.
Objetos	São componentes caracterizados pelo forte encapsulamento, comunicando-se com os demais componentes através de mensagens. Ou seja, toda a parte de dados e como este componente obtém seus resultados fica transparente para o componente que o solicitou serviços. Devido ao forte encapsulamento apresentado pelos objetos, estes podem ser usados como componentes em outras arquiteturas. Por exemplo, uma camada, da arquitetura camada, pode ser construída como um objeto.
Programa principal e subrotinas	Tipo de componente bastante utilizado nas décadas de 70 e 80. Caracteriza-se pelo fato de haver um componente, programa principal, que controla o fluxo dos outros demais componentes, subrotinas.

Tabela 1 – Instâncias de componentes

2.2.2 Conectores

Instância	Descrição
Tubo de dados	São os conectores do tipo tubo de dados que permitem aos filtros a característica de não compartilhamento de estado. São estes os responsáveis pela entrada e saída de dados em um filtro.
Interface entre	Em geral, representada por um conjunto de variáveis comuns as duas camadas.

[2] *Knowledge sources (ks)*

camadas	Também pode ser um conjunto de variáveis para cada camada vizinha, onde o conector terá a função de transferir os valores de um conjunto para outro. É o mecanismo responsável por transmitir o resultado gerado de uma camada para outra.
Blackboard	É uma estrutura central de base de dados que funciona como a única forma de comunicação / conexão entre os componentes. Este conector tem a função de, a partir do estado de seus dados, disparar processos que ativarão as *fontes de informação*. As *fontes de informação* têm como parte de suas funções, a alteração no conteúdo dos dados do *blackboard*, desta forma, alterando seu estado.
Mensagens entre objetos	É o tipo de conector responsável pela comunicação entre objetos. Dá-se a partir de métodos que possuem parâmetros. É este conector que permite a característica de encapsulamento nos objetos. Este conector diferencia do tubo de dados pelo fato de um objeto ter que saber a identidade de outro para haver comunicação entre esses. Ou seja, esta tarefa não cabe ao conector, ao contrário do tubo que permite que um tubo não saiba o estado e a identidade de outro. Alguns autores consideram esta característica como uma desvantagem na orientação a objetos [Shaw+96].
Passagem de variáveis	Conector bastante comum na utilização de componentes do tipo programa principal / subrotinas. A passagem de variável ocorre basicamente por dois tipos: por valor e por referência.

Tabela 2 – Instâncias de conectores

2.2.3 Seqüência de Comunicação

Instância	Descrição
Seqüencial por componente	É a seqüência de comunicação na qual existe uma ordem preestabelecida de processamento. Ou seja, ao final da execução de um componente não teremos todos os outros demais componentes como potenciais unidades de processamento. Teremos apenas um componente, conforme seqüência preestabelecida.
Randômica entre componentes	Indica que após a execução de um componente, qualquer outro componente é uma potencial unidade de processamento a ser executada.

Tabela 3 – Instâncias de seqüência de comunicação

2.2.4 Sincronismo

Instância	Descrição
Linear	Indica que um componente necessita esperar o final do processamento de outro componente para iniciar sua execução.
Não-linear	Indica que um determinado componente não precisa esperar o final do processamento de um outro componente para iniciar sua execução. Esta instância de estrutura de classificação, pré-supõem a existência de *modo de execução paralelo*.

Tabela 4 – Instâncias de sincronismo

2.2.5 Prioridade de execução entre componentes

Instância	Descrição
Não existe	É quando o ambiente no qual o sistema está inserido não possui mecanismos para determinar prioridade de execução entre componentes. O processamento ocorre a partir de uma fila onde os componentes para serem executados vão sendo postos. Alguns autores consideram esta fila a própria prioridade, porém sem a opção do processamento paralelo ou concorrente [Kazman+99].
Pode existir	É quando o ambiente no qual o sistema está inserido permite a priorizarão de execução, independente da ordem de chegada dos mesmos na memória real.
Existe sempre	É quando o ambiente no qual o sistema está inserido obriga a existência de uma priorização de execução entre os componentes.

Tabela 5 – Instâncias da prioridade de execução entre componentes

2.2.6 Modo de Execução

Instância	Descrição
Seqüencial	É quando apenas um componente pode ser executado por vez. Esta situação pode ocorrer por requisitos do sistema, por limitações de software e / ou limitações de hardware.
Paralelo	É quando dois ou mais componentes podem ser executados por vez. Para que ocorra tal situação, deveremos ter requisitos que indiquem tal necessidade, recursos de software e hardware. A *prioridade de execução entre componentes* deve ser: *pode existir* ou *existe sempre*.

Tabela 6 – Instâncias do modo de execução

2.2.7 Reuso

Instância	Descrição
Forte por componente	É quando existe a possibilidade da reutilização de componentes. Para viabilizar tal fato, supomos a existência de diversas pré-condições. Citamos como exemplo de tais condições: encapsulamento, interfaces bem definidas, hardware e software com arquitetura aberta que facilitem a prática do reuso, existência de uma IDL, entre outras condições.
Ausente	É quando não existe a prática do reuso. É quando os componentes que formam um sistema não podem ser aproveitados, mesmo que parcialmente, em outro sistema. Alguns fatores nos levam a este estado: ausência de facilidade de hardware e software, desenho e programação inadequados, entre outros.

Tabela 7 – Instâncias de reuso

2.2.8 Encapsulamento de Componentes

Instância	Descrição
Existe encapsulamento (sem conhecimento das interfaces)	É o tipo de encapsulamento que um componente não possui nenhum conhecimento sobre outros componentes ou o meio externo ao qual está prestando um serviço. Este tipo de encapsulamento é bem caracterizado pela falta de conhecimento dos estados entre os filtros no estilo de arquitetura tubos e filtros.
Existe encapsulamento (com conhecimento	É o tipo de encapsulamento no qual um componente tem ciência dos atributos e métodos públicos dos outros componentes (identidade e estado). Este tipo de encapsulamento caracteriza, na maior parte das vezes, o estilo orientação a objetos.

das interfaces)	
Não existe encapsulamento	Caracteriza a total falta de encapsulamento, tornando bastante difícil a prática do reuso. Em geral, ocorre quando um componente interfere no estado e no conteúdo dos outros componentes de maneira explícita.

Tabela 8 – Instâncias de encapsulamento de componentes

2.2.9 Comunicação entre Componentes

Instância	Descrição
Explicita (envio de dados)	É quando um conector une dois componentes distintos, que se comunicam a partir do envio de dados das variáveis do componente origem para outras variáveis do componente destino.
Explicita (envio de mensagens)	É quando um componente envia mensagens para outros componentes, a partir de um conector e faz com que tais mensagens estimulem ações no componente de destino. Em geral, estas mensagens são métodos públicos no componente de destino.
Chamada de funções	É quando a partir de um componente origem, um conector chama uma função responsável pelo início do processamento de um componente destino. Esta função pode pertencer ao componente origem, ou ao destino, ou pode ser um mecanismo independente (depende do *grau de encapsulamento* que está sendo utilizado).
Mecanismos de *callback*	É quando o componente 1 inicia um processo que estimula o componente 2. A função do componente 2 é chamar o componente 1. Utilizado em sistemas de telefonia quando não existe a facilidade de tarifa invertida.

Compartilhamento de dados	A comunicação entre os componentes ocorre a partir de uma base de dados central, como por exemplo, *blackboard*. O *blackboard* assume estados que estimulam a ação de componentes.

Tabela 9 – Instâncias de comunicação entre componentes

2.2.10 Modo de Dados

Instância	Descrição
Passados	É quando um componente passa seus dados de saída para um outro componente, a partir de um conector exclusivo para tal tarefa. O referido conector passa os dados diretamente para o componente de destino.
Compartilhados	É quando um componente passa seus dados de saída através de um conector único, porém tais dados vão para uma estrutura central de compartilhamento. Tal estrutura possui um ou mais conectores ligados a cada um dos componentes que formam o sistema. O modo de dados *compartilhado* não necessariamente está diretamente relacionado a comunicação entre componentes do tipo *compartilhamento de dados*. Podemos, por exemplo, ter um modo de dados *compartilhado*, implementando um *mecanismo de callback*.

Tabela 10 – Instâncias do modo de dados

2.2.11 Mecanismo de Feedback

Instância	Descrição
Presente (cíclico)	É quando existe a possibilidade de um componente chamar o outro que já o chamou, formando um ciclo, mesmo que indiretamente, a partir de outros componentes.
Ausente (acíclico)	É quando não existe a possibilidade de formação do ciclo citado anteriormente.

Tabela 11 – Instâncias do mecanismo de feedback

2.2.12 Topologia

Instância	Descrição
Linear	Topologia característica do estilo de arquitetura tubos e filtros, onde cada componente tem sua função isolada, não se preocupando com o resultado final formado pelo conjunto de componentes.
Hierárquica	Topologia característica do estilo camadas, onde uma camada depende de sua camada vizinha-superior para que seu processamento possa iniciar e fazer sentido para o todo.
Estrelar	Topologia característica do estilo repositórios, onde uma base de dados central (*blackboard*) une todos os componentes, tendo sua forma final assemelhada a uma estrela.
Árvore	Topologia característica de estilo programa principal / subrotinas. O programa principal assume as características da raiz e as subrotinas são os ramos e as folhas da árvore.
Grafo	Topologia característica do estilo orientação a objetos. As características dos

	objetos de alto encapsulamento e troca de mensagens faz com que, dependendo da situação, um objeto possa chamar qualquer outro. Desta forma, não podemos organizar os objetos em uma estrutura linear, ou hierárquica, ou qualquer outra. Conforme citado por Buschmann em [Buschmann+98], temos um "mar de objetos". O próprio Buschmann sugere a implementação de um *mediador* [Gamma+95] para organizar tal fato.

Tabela 12 – Instâncias da topologia

2.2.13 Organização de Processamento

Instância	Descrição
Batch	É quando, a partir do início da rotina, o usuário não pode interagir ou interferir no fluxo do processamento até o resultado final.
Interativo	É quando, em determinados pontos, o usuário pode interagir ou interferir no fluxo do processamento até o resultado final.

Tabela 13 – Instâncias da organização de processamento

Durante nossos estudos sobre instâncias das estruturas de classificação, observamos que estas podem ser subdivididas em outros níveis. Citamos como exemplo:

- O conector *passagem de variáveis* pode ocorrer por *parâmetro* ou por *referência*.
- O modo de execução *paralelo* pode ser *síncrono* ou *assíncrono* em relação ao programa chamador.
- O sincronismo do tipo *linear* pode ser subdividido nos mesmos tipos que aqueles utilizados em atividades de gerência de projetos. Ou seja: *iniciar-para-iniciar*, *acabar-para-iniciar*, entre outros.

Também observamos que para determinadas instâncias ocorrerem, a preexistência de outras se fazem obrigatórias. Chamaremos esta dependência de *relacionamento*. Como por exemplo:

- Para que ocorra sincronismo *não-linear*, necessitamos de um *modo de execução paralelo*.
- Para que ocorra seqüência de comunicação *seqüencial por componente*, necessitamos da existência de *prioridade de execução entre componentes*.
- Para que ocorra o modo de execução *paralelo*, necessitamos da existência de *prioridade de execução entre componentes*.
- Entre outras.

Observamos também que podemos fazer uma classificação de dependência de hardware, software e estilo de desenho / programação. Ou seja, para uma determinada instância da estrutura ocorrer dependemos de facilidades de hardware, software e / ou formas de programação.

Isto posto observamos que existem formas de dependências e relacionamentos das estruturas de classificação, as quais não trataremos nessa dissertação. Para o nosso objetivo final, que é a elaboração de uma taxonomia para sua utilização em um método de contagem, vamos parar no nível de instância apresentado nas tabelas de 1 a 13.

A primeira observação apresentada, que é a questão de uma instância ser dividida em sub-níveis, não será tratada. Ou seja, nossa abordagem contempla o nível de detalhe até instâncias.

A segunda observação, que trata das dependências entre estruturas de classificação e respectivas instâncias, trataremos de forma parcial. Ou seja, o resultado do método de contagem a ser proposto vai ser apresentado na forma de estilos de arquitetura, e não em termos de estruturas de classificação e instâncias. Isto significa que se uma estrutura é dependente de outra, as mesmas já estarão catalogadas em um mesmo estilo. Desta forma, podemos dizer que o problema de tratar as dependências entre estruturas ficará sob a responsabilidade do projetista que formará a base de conhecimento dos estilos de arquiteturas.

2.3 Os Relacionamentos

Conforme dito anteriormente, existe um *grau de relacionamento* entre pares de estruturas de classificação. O objetivo da determinação deste grau é saber quanto uma estrutura de classificação influencia na existência de outra e o quanto suas respectivas instâncias convivem bem ou mal. O que observamos na prática [Bass+98] [Shaw+96] [Silva+99.2-2] é que quando uma instância de uma estrutura ocorre, é quase obrigatório, ou é muito conveniente, que a instância de uma outra estrutura ocorra junto – conforme trabalhos do professor Abd-Allah [Abd-Allah+95]. Porém, para o nosso modelo de representação do conhecimento vamos considerar todas as possibilidades de combinação entre estruturas e instâncias [Bass+98], desta forma fazendo nossa base de conhecimento.

Isso posto, apresentamos duas tabelas quantificadoras dos relacionamentos. A tabela 14 representa os graus possíveis entre relacionamentos de duas estruturas de classificação. A tabela 15 representa as avaliações possíveis entre os relacionamentos das instâncias das estruturas de classificação.

Grau de	Relacionamento
0	nenhum
1	fraco
2	médio
3	forte
4	muito forte

Tabela 14 – Grau de Relacionamento

Avaliação	
-3	não podem estar juntas
-2	quando juntas trabalham muito mal
-1	quando juntas trabalham mal
0	é indiferente
+1	quando juntas trabalham bem
+2	quando juntas trabalham muito bem
+3	devem estar juntas

Tabela 15 - Avaliação

A pontuação escolhida para o *grau de relacionamento* partiu do número zero. Observamos que a existência de determinadas estruturas em nada influenciava a existência e o comportamento de outras estruturas. Considerando que o referido grau entrará em um método de contagem como um fator multiplicador, o número zero é utilizado para anular a pontuação do relacionamento das duas estruturas em questão. As demais graduações (fraco, médio, forte e muito forte) surgem seqüencialmente a partir do número zero. Após observação dos estudos de caso e o estágio atual de conhecimento do método de contagem, o crescimento linear de tais graus, mostra-se satisfatório.

A pontuação escolhida para a *avaliação* possui origem semelhante ao *grau de relacionamento.* Ou seja, dado que determinadas instâncias não influenciam na existência e no comportamento de outras instâncias, a pontuação zero anula a multiplicação feita no método de contagem. A opção por crescimento linear da *avaliação* também passou pelas mesmas observações nos estudos de caso. Porém o crescimento linear deu-se tanto para inteiros positivos quanto para negativos. Os inteiros positivos representam o fato de uma instância conviver bem com outras instâncias. Os

inteiros negativos representam o fato de determinadas instâncias não conviverem bem com outras instâncias.

A seguir, apresentaremos um exemplo com o intuito de mostrar como convivem às duas tabelas e como o método de contagem é aplicado nelas. Posteriormente, na Figura 14, seção 3.2, detalhamos o método de contagem, conforme nossa sugestão de implementação.

Exemplo:

Dada a estrutura de classificação, “componente”, dado um valor que ela pode assumir, “camada” (tabela 17). Dada outra estrutura de classificação, “seqüência de comunicação”, com o valor “randômica entre componentes”.

Supondo a base de conhecimento já formada, com grau de relacionamento *muito forte* (4) destas duas estruturas e *não podem estar juntas* (-3). Isso é verdade pois em uma seqüência de comunicação randômica entre componentes, todos os componentes tem a possibilidade de comunicação entre si. Já no estilo de arquitetura do tipo camadas, isso não pode ocorrer devido as características seqüenciais do processamento em camadas.

Este é um exemplo claro de uma *“característica eliminatória”*. Ou seja, se por algum motivo necessito de uma arquitetura em camadas, é fortemente recomendável não termos uma seqüência de comunicação randômica.

Observe que isso não inviabiliza a existência das duas estruturas juntas, porém torna muito difícil a integração das mesmas.

Para a representação do conhecimento sobre relacionamentos entre estruturas e respectivas instâncias, utilizamos redes semânticas. No anexo A apresentamos esta base de conhecimento, com todos os relacionamentos possíveis, em forma de tabela para facilitar a visualização.

A Figura 3 apresenta a rede semântica que contém os relacionamentos deste exemplo. A "*característica eliminatória*" aparece como uma rede semântica particionada.

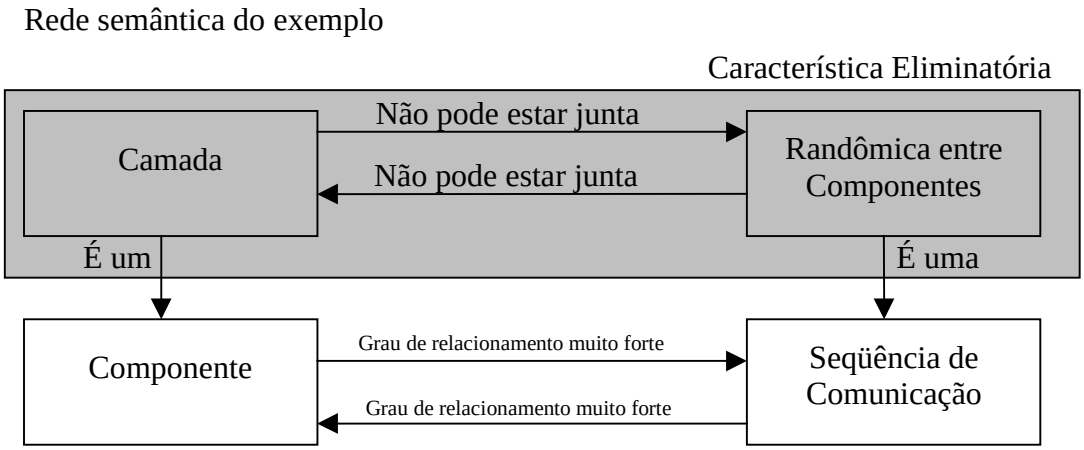

Figura 3 – Rede semântica do exemplo

Com isso temos, com os valores das tabelas 14 e 15, o seguinte cálculo:

(((4) * (-3)) * 2) = -24.

A multiplicação por 2 ao final de nossa contagem deve-se ao fato do relacionamento entre duas instâncias ocorrer de forma bilateral. Ou seja, a instância A se comunica com a instância B, assim como B se comunica com A.

A situação de "*característica eliminatória*" somente ocorre quando é feita a comparação entre duas estruturas de classificação e o grau de relacionamento é *muito forte* (4) e possuem uma avaliação que *não podem estar juntas* (-3). A referida *característica*, até este momento, possui características meramente documentacional. Ou seja, não mais estaremos explorando *"característica eliminatória"* em nosso método de identificação de estilos de arquiteturas.

2.4 Linguagens de Descrição de Arquitetura (LDA)

Até este momento classificamos os elementos que vão formar os estilos de arquiteturas. A estes elementos demos os nomes de estruturas de classificação e instâncias das estruturas de classificação. Também observamos que existem relacionamentos entre as estruturas e suas instâncias, quantificamos tais relacionamentos e experimentamos nossos critérios, a partir de estudos de caso do capítulo 5.

Agora, ainda com o objetivo de validar nossa busca por elementos que formam as estruturas de classificação, vamos descrever de maneira superficial algumas LDA's. Estaremos especialmente interessados nos aspectos semânticos da linguagem, como estas representam cada um dos elementos que formam um estilo de arquitetura.

A partir de textos lidos, falaremos das seguintes linguagens UniCon, Aesop, ACME e Rapide [Allen+97.2] [CMU01] [Rapide01] [Shaw+94].

2.4.1 UniCon

UniCon é qualificada como uma linguagem de descrição de arquiteturas que tem o objetivo de auxiliar os projetistas na definição de arquiteturas de software ao que se refere as abstrações que se mostrem úteis, bem como garantir uma transição mais natural entre o modelo arquitetural e o código. Foi desenvolvida pelo grupo da professora Mary Shaw, Universidade de Carnegie Mellon. O ambiente UniCon disponibiliza uma ferramenta de configuração que permite a construção de executáveis à base de tipos de componentes e conexões entre tais componentes. UniCon suporta conectores simétricos e assimétricos. Os conectores são declarados de tal forma

que definem logicamente a interação entre dois componentes. Cada conector tem um conjunto de papéis indicando quais componentes vão participar da interação a ser definida. As interfaces dos componentes ao invés de providenciarem ou requisitarem serviços, possuem um outro componente associado, chamado de *jogadores*[3]. Estes *jogadores* possuem um tipo que indica a natureza da interação esperada e um conjunto de características que especificam detalhes sobre a interação do componente naquela interface.

2.4.2 Aesop

Aesop é uma ferramenta da família de ambientes para desenho de sistemas, que também foi desenvolvida pelo grupo da professora Mary Shaw, tendo como líder de projeto David Garlan. A esta família é dado o nome de ABLE (*Architecture Based Languages and Environments*). Aesop fornece um vocabulário para a descrição de arquiteturas através de um modelo de subtipos orientados a objetos. Uma configuração de arquitetura é representada como uma coleção de instâncias de objetos interconectados, enquanto que o vocabulário do estilo de arquitetura é descrito pela definição dos tipos básicos arquiteturais: componente, conector, porta, papel / responsabilidade, configuração e ligação.

Uma configuração arquitetônica em Aesop é uma estrutura hierárquica de componentes, conectores e configurações.

Por exemplo, no estilo tubos e filtros define filtro como um subtipo de componente e tubo como um subtipo de conector. Definiríamos uma porta de entrada de um filtro, cujo papel seria apenas de permitir a entrada de dados e nunca a saída. Esta definição caracterizaria a ligação entre os componentes.

---

[3] *players*

Com isso, Aesop fornece mecanismos explícitos para a descrição e representação dos estilos de arquiteturas.

2.4.3 ACME

Também pertencente a mesma família de ambientes para desenho de sistemas da Aesop. A ACME foi desenvolvida com a intenção de ser uma linguagem de intercâmbio entre todas as formas de descrição de arquiteturas até o momento existentes (ano de 1995). Assim como a Aesop, a ACME define um vocabulário básico de componentes, conectores, portas, papéis, ligações e configurações. Estes elementos têm especificações associadas a eles através de uma lista de características. ACME aproveita diversas outras LDA's para especificar a lista de características.

O objetivo maior da ACME é a comparação de modelos arquiteturais escritos em LDA's diferentes, bem como a unificação destes modelos.

2.4.4 Rapide

Rapide foi desenvolvida pela equipe do professor David Luckham, da Universidade de Stanford. É uma linguagem de descrição de arquitetura baseada na modelagem de interações como conjuntos de elementos parcialmente ordenados. É baseada em um novo conjunto de linguagens para descrição de arquiteturas que recebe o nome de EADL's (*Executable Architecture Definition Languages*). Diferentemente da UniCon que define a semântica comportamental dos componentes através de suas implementações, Rapide define tipos de componentes (chamados interfaces) em termos de uma coleção de eventos de comunicação, que podem ser observados ou inicializados. As interfaces do Rapide definem o comportamento computacional de um componente.

As interações entre componentes podem ser definidas de duas maneiras. A primeira indica quais eventos podem ser conectados. A segunda, indica que uma arquitetura pode declarar restrições, que especificam relacionamentos entre eventos em diferentes componentes. Ou seja, dada uma restrição, a inicialização de um evento terá como resultado a geração de um outro evento acompanhado-o em todas as ordens do evento, mas eles não são considerados como um mesmo evento.

Após análise das quatro LDA's apresentadas, alguns pontos foram observados como comuns:

- Todas iniciam sua representação a partir de componentes e conectores.
- Todas representam, de alguma forma, a ligação dos componentes a partir dos conectores.
- Todas possuem uma forma para representar as características e restrições dos estilos de arquiteturas a partir da ligação entre componentes.

Isso posto, confirmamos que as duas considerações, de que: componentes e conectores são estruturas primárias; e que somente a partir da relação destes dois elementos é que chegaremos as outras estruturas de classificação; são considerações que devem ser exploradas para compormos uma taxonomia, para a representação de todos os estilos de arquitetura.

2.5 Taxonomia para todos os Estilos

No propósito da montagem de uma base de conhecimento que servirá para o método de contagem a ser proposto, uma taxonomia para todos os estilos de arquitetura faz-se obrigatória. Somente desta forma é que poderemos comparar e quantificar os diversos estilos de arquitetura.

Objetivando esta taxonomia, é que estudamos as estruturas de classificação, suas instâncias e seus relacionamentos.

Cada uma das estruturas de classificação apresentadas surgiram a partir das observações de elementos que caracterizavam arquiteturas [Shaw+96.1]. Muitas destas observações vieram a ser confirmadas com a análise de algumas LDA's. Todas as quatro LDA's pesquisadas enfatizaram como ponto inicial a definição de componentes e conectores. Estas mesmas linguagens, cada uma em sua forma, fazem representações semânticas dos relacionamentos e das restrições de seus elementos de tal forma a representar um estilo de arquitetura.

Isto posto, a tabela 16 apresenta os elementos que formam a taxonomia. Estes elementos, a partir de suas instâncias, servirão para a formação dos estilos de arquitetura. Os relacionamentos entre os elementos da tabela 16 ocorrem conforme exemplo da Figura 3.

Estrutura de Classificação		**Instância**	
1	Componente	1.1	Filtro
		1.2	Camada
		1.3	*Fontes de informação*
		1.4	Objeto
		1.5	Programa Principal / Subrotina
2	Conector	2.1	Tubo de dados
		2.2	Interface entre camadas
		2.3	*Blackboard*
		2.4	Mensagens entre objetos
		2.5	Passagem de variáveis

3	Seqüência de comunicação	3.1	Seqüencial por componente
		3.2	Randômica entre componentes
4	Modo de execução	4.1	Seqüencial
		4.2.	Paralelo
5	Sincronismo	5.1	Linear
		5.2	Não-linear
6	Encapsulamento de componentes	6.1	Sempre (sem conhecimento das interfaces)
		6.2	Sempre (com conhecimento das interfaces)
		6.3	Não existe encapsulamento
7	Comunicação entre componentes	7.1	Explicita (envio de dados)
		7.2	Chamada de funções
		7.3	Mecanismos de callback
		7.4	Compartilhamento de dados
		7.5	Explicita (envio de mensagens)
8	Reuso	8.1	Forte por componente
		8.2	Ausente
9	Mecanismos de feedback	9.1	Presente (cíclico)
		9.2	Ausente (acíclico)
10	Prioridade de execução entre componentes	10.1	Existe sempre
		10.2	Pode existir
		10.3	Não existe
11	Topologia	11.1	Linear

		11.2	Hierárquica
		11.3	Estrelar
		11.4	Árvore
		11.5	Não há forma
12	Modo de dados	12.1	Passados
		12.2	Compartilhados
13	Organização de processamento	13.1	Batch
		13.2	Interativo

Tabela 16 – Taxonomia das estruturas de classificação e instâncias

2.6 Estilos de Arquitetura

A partir de uma taxonomia, definida anteriormente, vamos classificar cinco estilos de arquitetura com suas estruturas de classificação, e suas respectivas instâncias.

Nossa opção em abordar o problema em estilos de arquitetura, e não tratarmos separadamente cada estrutura de classificação com suas instâncias, deve-se ao fato dos desenvolvedores de software terem a tendência de reuso de conjuntos já estabelecidos, ou seja, estilos de arquitetura [Shaw+96].

Os estilos de arquitetura podem ser vistos em dois grandes grupos: "*idiomas e padrões*"; e "*modelos de referência*".

Os idiomas e padrões incluem as estruturas organizacionais globais, sistemas em camadas, tubos e filtros, organização cliente-servidor, *blackboards*, entre outros. Também incluímos neste grupo padrões como o MVC.

Os modelos de referência incluem organizações de sistemas que indicam uma configuração específica de componentes e interações de uma área específica. Podemos citar como exemplo de configurações específicas o modelo ISO OSI 7-camadas, modelos de software de aviação e robótica.

Devido a alta especialização necessária para tratarmos sobre os modelos de referência, esta dissertação falará apenas sobre idiomas e padrões.

Independente da escolha do grupo de estilo de arquitetura a ser tratado, o propósito maior, ao falarmos de estilos de arquitetura passa pelas seguintes questões:

- Definir um vocabulário básico e único entre os desenvolvedores de software.
- Definição de regras de configuração que determinem uma composição pertinente entre os elementos agrupados, a partir dos requisitos de um software.
- Definição de uma semântica, a partir do vocabulário, que será útil no decorrer do desenvolvimento do software.

A seguir, apresentamos uma descrição e um diagrama genérico de cada um dos cinco estilos de arquitetura que fizeram parte de nossos estudos: tubos e filtros, camada, repositórios, orientado a objetos e programa principal / subrotinas.

2.6.1 Tubos e Filtros

No estilo tubos e filtros cada componentes possui um conjunto de entradas e de saídas. Cada componente recebe uma série de dados de entrada, processa estes dados, envia os dados resultantes para os tubos de saída. Em geral, os filtros possuem um grau de encapsulamento

muito forte, fazendo com que estes não tenham conhecimentos das interfaces com os demais filtros. A Figura 4 apresenta um modelo genérico do estilo tubos e filtros.

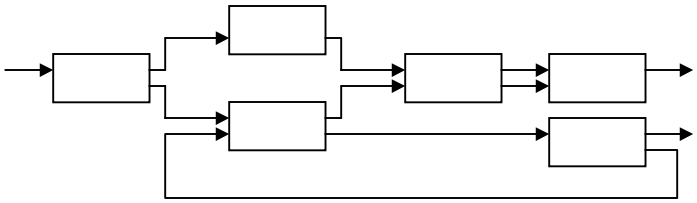

Figura 4 – Tubos e filtros

2.6.2 Camada

O estilo camada é organizado de maneira hierárquica. Cada camada tem o objetivo de prover serviços bem definidos para a camada seguinte, a qual funciona como um cliente para a camada anterior. Cada camada pode possuir vários outros componentes. Cada camada possui seu protocolo de comunicação com suas camadas vizinhas. A alteração de uma camada não pode atingir mais de duas outras camadas, visto que uma camada se comunica, no mínimo, com uma camada e, no máximo, com duas. Esta característica torna este estilo forte para reuso, como por exemplo, máquinas virtuais em linguagens de programação, como o Java. A Figura 5 apresenta um modelo genérico do estilo camada.

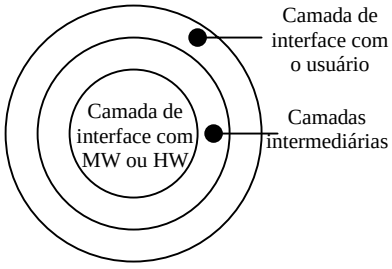

Figura 5 – Camada

2.6.3 Repositórios

No estilo repositórios, existem dois elementos distintos: a base central de dados (*blackboard*) e os componentes que executam operações na referida base, inclusive suas intercomunicações. Os componentes são chamados de *fontes de informação* (*knowledge source - ks*). Os estados assumidos pela base central de dados é o principal motivador de disparo de ações nos componentes que executam tarefas. Um componente não pode se comunicar com outro, exceto via o repositório central. A Figura 6 apresenta um modelo genérico do estilo repositórios.

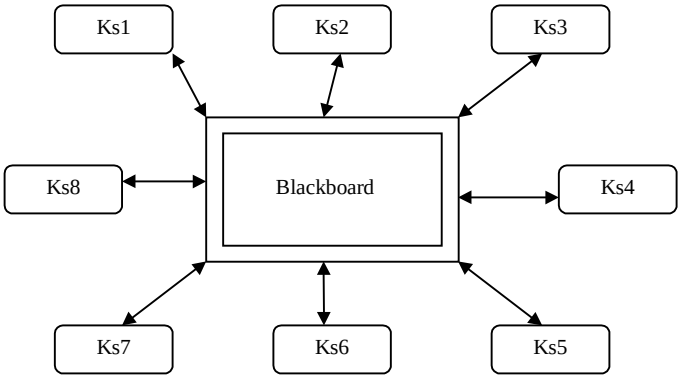

Figura 6 – Repositórios

2.6.4 Orientado a Objetos

O estilo orientado a objetos possui como característica forte o fato dos dados e das operações estarem encapsulados em um componente, o qual chamamos de objeto. A comunicação entre componentes ocorre a partir da troca de mensagens entre esses. Quando os objetos possuem uma interface bem definida e um protocolo de comunicação bem definido, as chances de reuso são grandes. Ao contrário do estilo tubos e filtros, orientação a objetos permite um relaxamento da visibilidade de um objeto em relação a outro, o que diminui o encapsulamento e as chances de reuso. A Figura 7 apresenta um modelo genérico do estilo orientado a objetos.

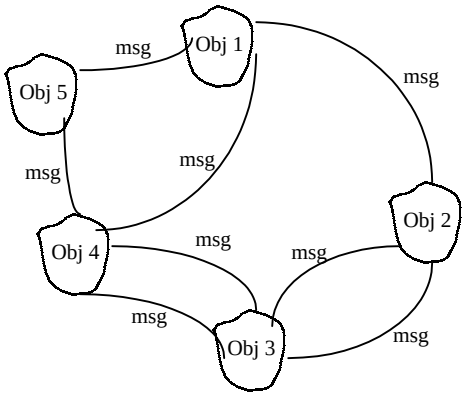

Figura 7 – Orientado a Objetos

2.6.5 Programa Principal / Subrotinas

Este foi o estilo mais utilizado das décadas de 70 e 80. Muitas linguagens de programação possuem seus livros com exemplos construídos a partir deste estilo. Quando as linguagens de programação não possuem um suporte efetivo para o encapsulamento esta é a arquitetura preferencial. É caracterizada pela existência de um programa inicial (programa principal) que chama diversos outros programas (subrotinas) em uma seqüência preestabelecida. Cada um dos programas chamados ao terminar, retorna seu controle ao programa chamador. A Figura 8 apresenta um modelo genérico do estilo programa principal / subrotinas.

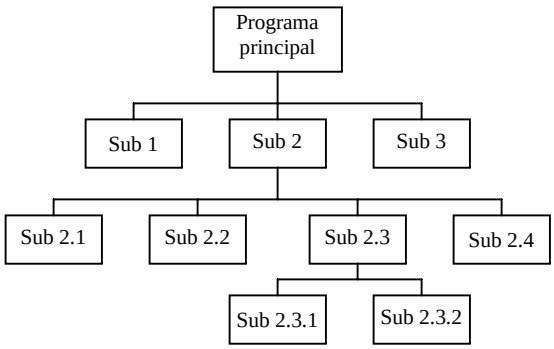

Figura 8 – Programa Principal / Subrotinas

A seguir, apresentamos a tabela 17 com a classificação feita, por estruturas de classificação e respectivas instâncias, para os estilos de arquitetura estudados.

A tabela 17 representa o resultado da classificação, a partir da taxonomia definida.

	ESTILOS DE ARQUITETURA				
Estruturas de Classificação	**Tubos e Filtros**	**Camada**	**Repositório**	**Orientado a Objetos**	**Programa Principal / Subrotinas**
Componente	Filtro	Camada	Fontes de informação	Objeto	Programa Principal, subrotinas
Conector	Tubo de dados	Interface entre camadas	*Blackboard*	Mensagens entre objetos	Passagem de variáveis
Seqüência de Comunicação	Seqüencial por componente	Seqüencial por componente	Randômica entre componentes	Randômica entre componentes	Seqüencial por componente
Modo de Execução	Seqüencial, paralelo	Seqüencial	Seqüencial, paralelo	Seqüencial, paralelo	Seqüencial
Sincronismo	Linear, não linear	Linear	Linear, não-linear	Linear, não-linear	Linear
Encapsulamento de Componentes	Sempre (sem conhecimento das interfaces)	Sempre (com conhecimento das interfaces)	Sempre (sem conhecimento das interfaces)	Sempre (com conhecimento das interfaces)	Pode existir (com conhecimento das interfaces)
Comunicação entre Componentes	Explicita (envio de dados)	Explicita (envio de mensagens), chamada de funções, mecanismos de callback	Compartilhamento de dados	Explicita (envio de mensagens), mecanismos de *callback*	Chamada de funções
Reuso	Forte por componente	Forte por componente	Forte por componente	Forte por componente	Ausente
Mecanismo de *Feedback*	Presente (cíclico)	Ausente (acíclico)	Ausente (acíclico)	Presente (cíclico)	Presente (cíclico)
Prioridade de Execução entre Componentes	Pode existir	Existe sempre	Pode existir	Pode existir	Existe sempre
Topologia	Linear	Hierárquica	Estrelar	Grafo (mar de objetos)	Árvore
Modo de Dados	Passados	Passados	Compartilhados	Passados	Passados
Organização de Processamento	Batch	Batch, interativo	Batch, interativo	Batch, interativo	Batch, interativo

Tabela 17 – Estilos de arquitetura, por instâncias das estruturas de classificação

2.7 Representação do Conhecimento

Durante o estudo que nos levou a taxonomia e posterior representação dos estilos de arquiteturas, utilizados quatro elementos: estruturas de classificação, instâncias das estruturas, relacionamentos entre estruturas e relacionamento entre instâncias.

Todos estes elementos formaram nossa base de conhecimento. Como o nosso principal objetivo é capturar o conhecimento de *projetistas* (*designers*) experientes, a representação deste conhecimento será um *sistema de produção* integrado com uma *rede semântica*. A rede semântica registrará o conhecimento estático, fundamentalmente às taxonomias utilizadas [Brownston+85] [Rich+83] [Tanimoto01]. No capítulo 4 estaremos detalhando a forma como implementamos a representação do sistema de produção e da rede semântica.

O sistema de contagem, que representa o conhecimento dinâmico, assumirá características de um sistema de produção [Brownston+85] [Rich+83] e utilizará a rede semântica em seu funcionamento.

Os objetivos básicos a serem alcançados com a representação escolhida são:

- Fazer com que o trabalho desenvolvido por um *projetista* sem experiência fique muito semelhante ao mesmo trabalho que seria feito por um *projetista* experiente, ao que se refere a arquitetura de software. Ou seja, estar utilizando a base de conhecimento cadastrada através do reuso de conhecimento do especialista.
- Fazer com que a consulta a base de conhecimento seja de fácil manuseio. Este objetivo torna-se difícil de ser alcançado à medida que a base de conhecimento for aumentando de volume. A escolha de representação por rede semântica e sistema de produção tenta minimizar esta dificuldade.

- Ter uma representação do conteúdo da base de conhecimento, de fácil entendimento e que permita a expansão da mesma.

A seguir apresentamos as representações utilizadas para as estruturas de classificação, instâncias, relacionamentos e estilos de arquitetura. Vamos observar que a representação das estruturas de classificação, suas instâncias e seus relacionamentos são feitos a partir de redes semânticas simples. As "*características eliminatórias*" em um relacionamento e os estilos de arquitetura são representados por rede semântica particionada, conforme demonstrado parcialmente em seções anteriores.

2.7.1 Representação das Estruturas de Classificação e Instâncias

A seguir duas estruturas de classificação e suas instâncias, da tabela 16, são representadas em forma de rede semântica. São elas: componente e conector.

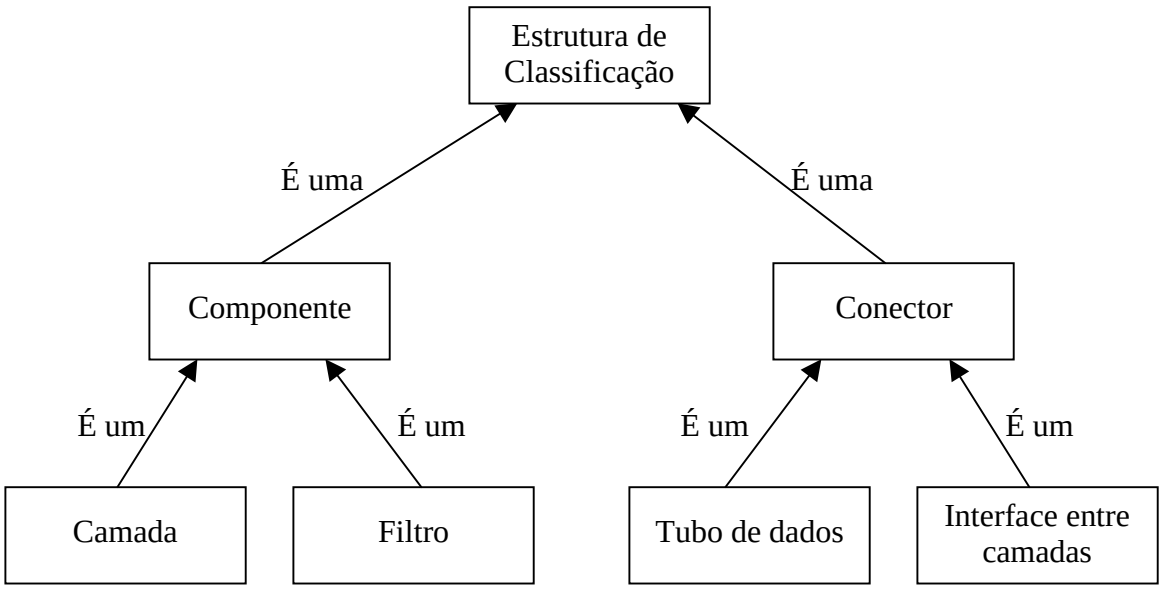

Figura 9 - Rede semântica para a representação das estruturas de classificação e suas instâncias (representação parcial)

2.7.2 Representação dos Relacionamentos

O relacionamento entre as estruturas de classificação e suas respectivas instâncias deverá ser representado na forma de rede semântica. A representação de uma "*característica eliminatória*" será representada por uma rede semântica particionada, conforme sugere a Figura 10. Já demonstramos a "*característica eliminatória*" na Figura 3.

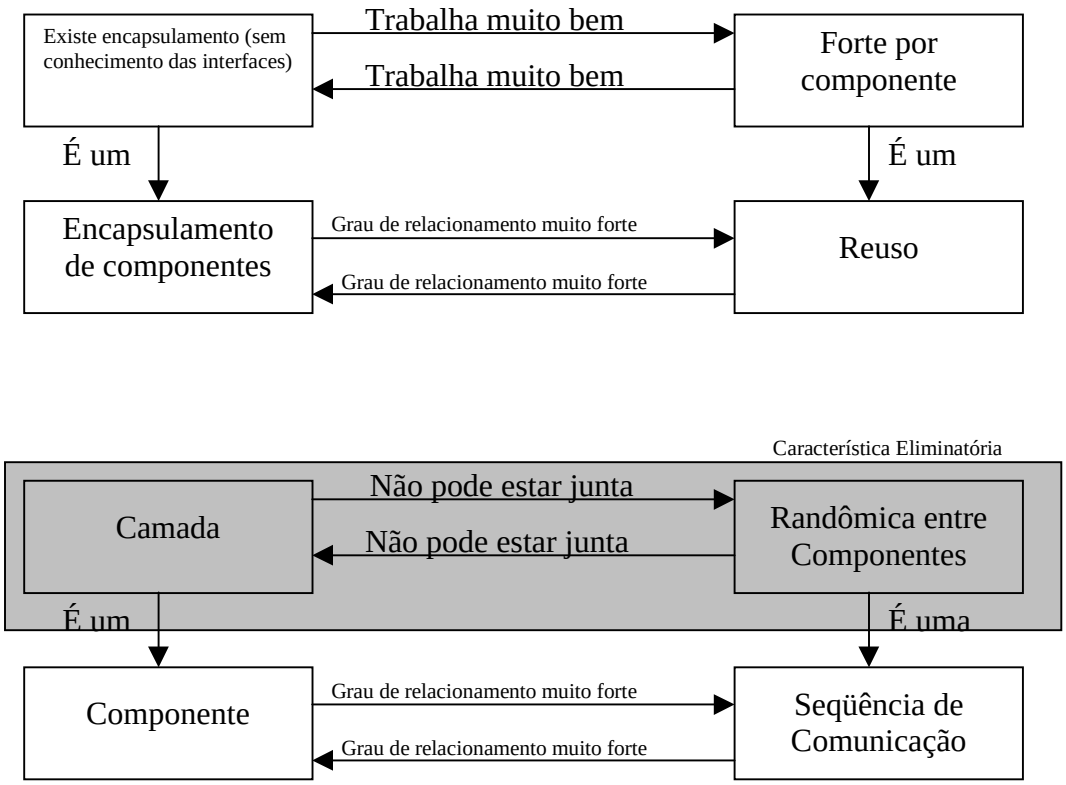

Figura 10 – Rede semântica dos relacionamentos entre estruturas e instâncias (representação parcial)

2.7.3 Representação dos Estilos de Arquitetura

A seguir, apresentamos a representação parcial de dois estilos de arquitetura da tabela 17. Observamos que além da representação da rede semântica simples, utilizada para representar estruturas e instâncias, também utilizamos redes semânticas particionadas, no momento da representação do estilo de arquitetura.

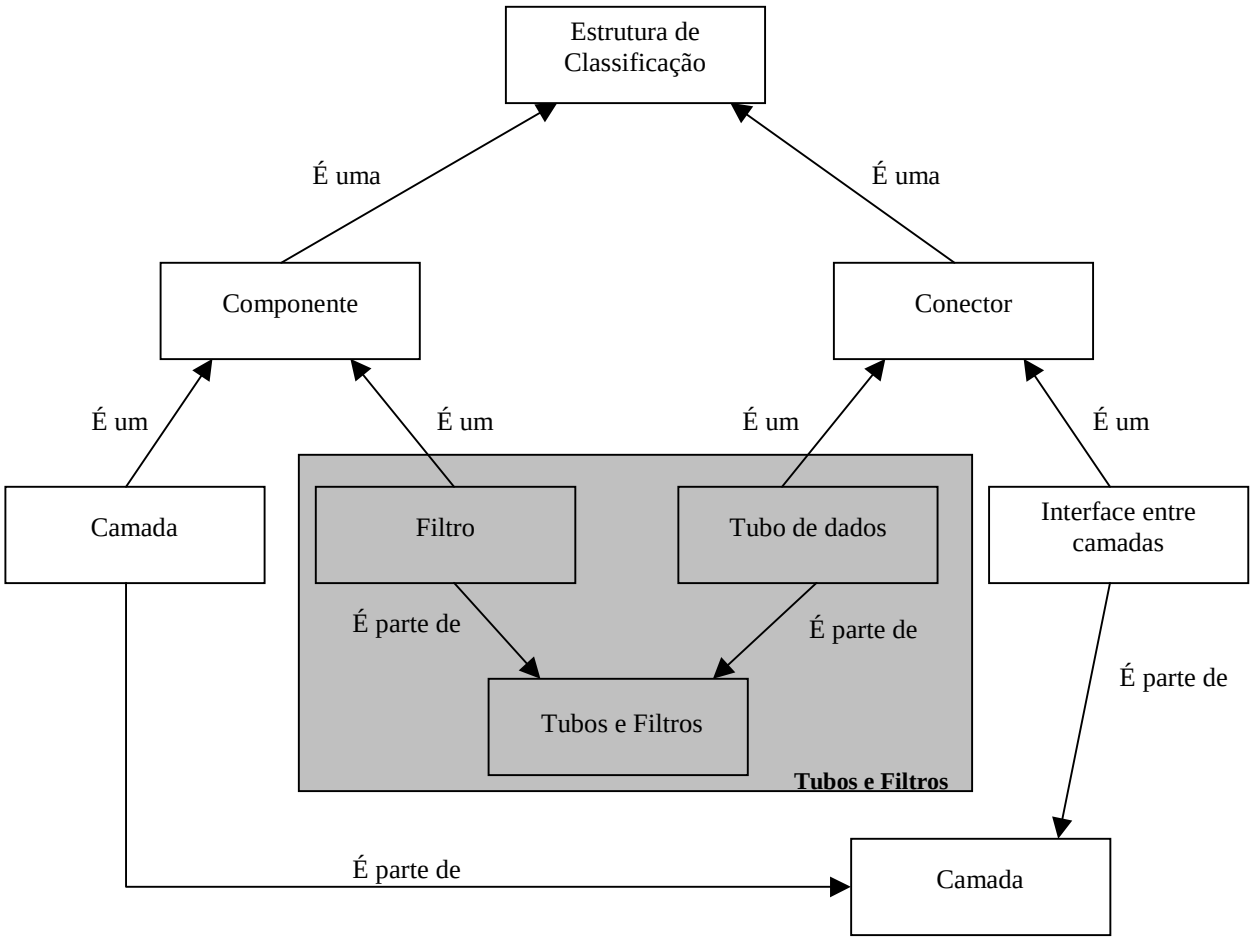

Figura 11 - Rede semântica particionada para a representação dos estilos de arquiteturas (representação parcial)

Com estas representações propostas, entendemos que possuímos o conhecimento estático catalogado em uma taxonomia proposta, para ser utilizado por um método de contagem. Entendemos também que além dos cinco estilos de arquitetura aqui apresentados, conseguiremos estender esta mesma taxonomia para a representação de outros estilos.

Destacamos como positivo a flexibilidade das estruturas de classificação formadas, no intuito de representarmos arquiteturas ainda não exploradas. Acreditamos ser esta a principal característica da taxonomia proposta, bem como do seu modelo de representação do conhecimento.

Capítulo 3 – Método de Identificação de Estilos de Arquiteturas

O método a ser proposto tem o objetivo de indicar um ou mais estilos de arquiteturas para um determinado sistema em estudo. O sistema em estudo, que pode estar implementado ou em desenvolvimento, é representado através de seus requisitos funcionais e não-funcionais. A identificação do (s) estilo (s) de arquitetura (s) se baseia na definição de regras e fatos de um sistema de produção. Ao final do processo também apresentamos as estruturas de classificação faltantes com base na identificação realizada. O modelo SADT (*Structured Analysis and Design Technique*) [Ross01] – Figura 12 - fornece a idéia geral do método proposto.

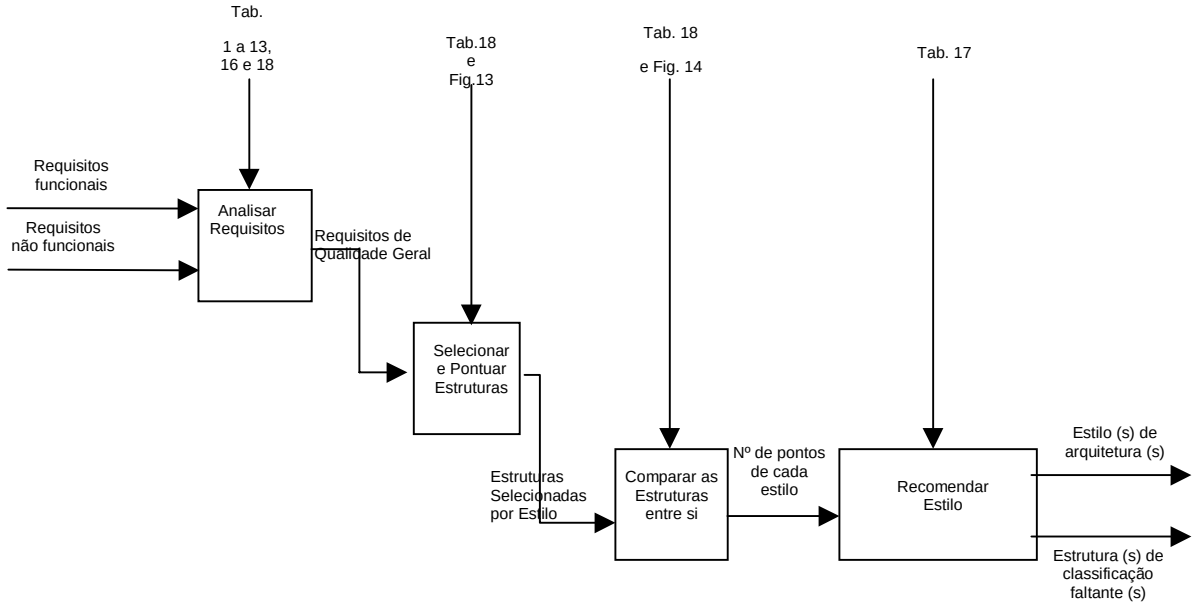

Figura 12 – Modelo SADT do Método Proposto

Nas próximas seções deste capítulo abordaremos todas as partes que compõem o método proposto. Indicaremos a forma como abordamos os requisitos de um sistema em estudo. Esta forma de abordagem dará início ao método de contagem a ser proposto.

3.1 Requisitos Funcionais e Não-funcionais

Para a elaboração de um novo sistema, ou mudança de um já existente, inicialmente devemos estudar os requisitos que provêm das expectativas do cliente quanto ao produto a ser gerado.

A literatura trata requisitos a partir de diversas subdivisões, como por exemplo:

- Requisitos de alto-nível: são requisitos que tratam sobre as necessidades gerais, normalmente utilizados para a definição de escopo nos projetos.
- Requisitos detalhados: são os requisitos derivados dos requisitos de alto-nível. Podemos citar como exemplo a especificação de um cálculo de folha de pagamento para a participação dos funcionários nos lucros de uma empresa.
- Requisitos limitadores: são requisitos obrigatórios e limitadores de soluções. Alguns autores chamam de premissas. Como exemplo, podemos citar a existência de um ambiente, onde o futuro sistema deve, obrigatoriamente, ser executado em um sistema operacional específico.
- Requisitos técnicos: são os requisitos que envolvem toda a parte técnica, como por exemplo soluções de teleprocessamento, integridade de um repositório de dados, entre outros.
- Requisitos funcionais: são os requisitos que atendem as funções de um sistema.
- Requisitos não-funcionais: são requisitos que não estão diretamente ligados as funções fins da área requisitante do sistema. Como exemplo, a possibilidade do sistema em estudo poder ser executado em diversas plataformas operacionais.

Estas são algumas das formas que encontramos para classificar os requisitos. Observamos que muitas delas formam interseções de conjuntos. Por exemplo, um futuro usuário de um sistema, pode declarar como requisito a necessidade de fazer o produto para ser executado em plataforma LINUX. Podemos entender este requisito como um requisito técnico ou, dependendo do caso, como um requisito limitador.

Para a abordagem do método de identificação a ser proposto, trataremos todos os requisitos em dois grupos: requisitos funcionais e não-funcionais. Entendemos que todos os requisitos podem ser representados nestes dois grupos.

Vamos supor que o usuário do método já cumpriu a difícil tarefa, de descobrir e descrever os requisitos de maneira verdadeira, clara e sem ambigüidade. Este usuário tem a tarefa de interpretar todos os seus requisitos, já descritos, no formato dos *requisitos de qualidade geral*, conforme apresentamos na tabela 18, da seção 3.2. Isso é uma tarefa manual do usuário do método e que depende do poder de interpretação deste. Não faz parte do escopo desta dissertação buscar ou justificar a melhor forma de identificar, agrupar e redigir requisitos.

3.2 Requisitos de Qualidade Geral

Ao longo de nosso estudo, percebemos que para determinados requisitos serem satisfeitos algumas estruturas de classificação devem ocorrer de uma forma específica. Percebemos também, o caminho inverso. Ou seja, quando uma estrutura de classificação ocorre de uma maneira específica é muito difícil que o conjunto o qual esta estrutura compõem, seja de um determinado estilo de arquitetura. Por exemplo, se um componente de um software possui comunicação com outros três componentes, podemos supor que tal software não possui o estilo camada, nem o

estilo repositório com a topologia estrelar, vista a quantidade de comunicações entre componentes.

Isso posto, propomos em nosso método, mais uma parte da base de conhecimento. Esta base relaciona requisitos que chamamos de *requisitos de qualidade geral*. Tais requisitos passam a existir em um software desde o momento que determinadas estruturas ocorram de uma forma específica [Klein+99.2].

Desta forma, nosso objetivo com a tabela 18 é identificar um número satisfatório de requisitos que tenham correspondência com requisitos de um sistema em estudo. Cada característica ou requisito da tabela estará relacionado a um ou mais *códigos das estruturas de classificação, do estilo* em questão (tabela 17). Se não houver nenhum código assinalado (célula vazia), significa que aquela característica (ou requisito) não se cumpre para o estilo de arquitetura em questão.

Dada uma célula da tabela 18, todos os elementos devem ser relacionados entre si. Estes relacionamentos ocorrem a partir do que foi definido na seção 2.3, para cada par de estruturas de classificação.

	Códigos das estruturas de classificação que atendem ou favorecem ao requisito				
Requisitos de Qualidade Geral	**Tubos e Filtros**	**Camada**	**Repositórios**	**Orientado a Objetos**	**Programa Principal / Subrotinas**
Os componentes devem possuir uma estrutura hierárquica		1, 2, 3, 4, 5, 10, 11			1, 3, 4, 5, 10, 11
Os componentes não devem possuir uma estrutura hierárquica	1, 2, 5, 10		1, 2, 3, 4, 5, 10, 11	1, 2, 3, 4, 5, 10, 11	
Os componentes devem possuir uma estrutura linear	1, 2, 3, 4, 5, 10, 11	1, 2, 3, 4, 5, 10			1, 2, 3, 4, 5, 10
Os componentes não devem possuir uma estrutura linear			1, 2, 3, 4, 5, 10, 11	1, 2, 3, 4, 5, 10, 11	
Deve existir prioridade de execução entre os componentes	3, 4, 10	3, 4, 10	4, 10	4, 10	3, 4, 10
Não é necessária a existência de prioridade de execução entre os componentes	10		10	10	
Deve existir persistência de dados					
Não deve existir persistência de dados					
Deve compartilhar dados			1, 2, 11, 12	1, 2, 11	12
Não deve compartilhar dados	1, 2, 12	1, 2, 12		1, 2, 12	1, 2, 12
Deve possuir mecanismo de *feedback* com o meio externo	9	9	9	9	9
Deve possuir mecanismo de disparo de ações conforme			1, 2, 11	1, 2, 11	

contexto de dados ou ocorrência de eventos					
Deve ter rapidez de resposta com grande volume de dados			1, 2, 11	1, 2, 11	
Deve ter portabilidade entre plataformas de HW e SW (ex. JAVA beans)	1, 2, 6, 8	1, 2, 6, 8	1, 6, 8	1, 2, 6, 8	
Deve haver a possibilidade de reuso em projetos futuros (funções do negócio fim)	1, 2, 6, 8	1, 2, 6, 7, 8	1, 6, 8	1, 2, 6, 7, 8	
Só pode processar em ambiente de monoprocessamento	3, 4, 10, 11	3, 4, 10, 11			3, 4, 10
Deve tirar proveito do processamento concorrente e paralelo	5, 10		3, 4, 5, 10, 11	3, 4, 5, 10, 11	11
Os componentes terão funcionalidades batch, somente	13	13	13	13	13
Os componentes deverão ter funcionalidades on-line		13	13	13	13

Tabela 18 – Requisitos de qualidade geral, relacionados com códigos das estruturas de classificação, por estilo de arquitetura

Os conhecimentos existentes na tabela 18 são implementados a partir de regras de um sistema de produção. A utilização de tais regras ocorre com a definição de fatos, o que ocasionará, ou não, o disparo das regras.

A definição de fatos e posterior disparo das regras é ilustrado pelo pseudocódigo abaixo relacionado na Figura 13.

Por exemplo, a tabela 18 – primeira linha, seria representada em nossa proposta através da seguinte regra:

Se *"os componentes devem possuir uma estrutura hierárquica"*[4]

Então *Não selecione nenhuma estrutura para os estilos tubos e filtros, repositórios e orientado a objetos.*

Selecione as estruturas 1, 2, 3, 4, 5, 10, 11 para o estilo camada.

Selecione as estruturas 1, 3, 4, 5, 10, 11 para o estilo programa principal / subrotinas.

Conte 1 ponto para cada estrutura selecionada no respectivo estilo.

*Conte os pontos do relacionamento entre as estruturas de classificação, bem como de suas instâncias (*Figura 14*), para a célula em questão, no respectivo estilo.*

Conte o número de pontos encontrados para cada estilo.

Figura 13 – Definição de fatos e conseqüente disparo das regras em um sistema de produção

Com isso, observamos que:

- A seleção dos requisitos de qualidade geral define os fatos de um sistema de produção.

[4] O requisito de qualidade geral é selecionado pelo usuário do método.

- A contagem de pontos ocorre a partir da definição dos fatos que disparam regras.
- As regras utilizam informações de relacionamentos entre estruturas de classificação que constam nas redes semânticas, conforme os valores definidos para o grau de relacionamento e avaliação.

A partir das explicações sobre os requisitos de qualidade geral, podemos detalhar a outra parte da contagem do sistema de produção a ser implementado. Esta segunda parte vai utilizar os conhecimentos armazenados nas redes semânticas, anteriormente explicadas (Figura 10).

A seguir exemplificamos, em pseudocódigo, uma possível definição de fatos de um sistema de produção, para posterior disparo de regras, com base no conhecimento armazenado na rede semântica. A rede semântica em questão é quem trata dos relacionamentos das estruturas de classificação, bem como suas instâncias.

Dentre as estruturas de classificação selecionadas, conforme requisitos de qualidade geral, fazer todos os pares possíveis .....

:

:

Se *Existe "componente" com valor "camada"* ***e***

Existe "seqüência de comunicação" com valor "randômica entre componentes"

Então *Verifique na rede semântica qual é o "grau de relacionamento" entre estas duas estruturas (resposta: valor 4 – tabela 14).*

Verifique na rede semântica qual é a "avaliação" entre os valores assumidos por cada uma das duas estruturas em comparação (resposta: valor -3 – tabela 15).

*Valor do par = (((4) * (-3)) * 2) = -24.*

Figura 14 – Regras do sistema de produção (método de contagem), conforme fatos existentes

Com os exemplos das Figura 13 e 14 exemplificamos o sistema de produção que implementa o método de contagem. Tal método de contagem ocorreu a partir da análise dos requisitos de qualidade geral e a partir da comparação das estruturas de classificação e instâncias existentes, em cada estilo.

Isso posto, identificamos que o método de contagem é dividido em três partes. Este método ocorre após a identificação dos requisitos de qualidade geral associados ao sistema em estudo, com a pontuação ocorrendo por estilo de arquitetura. São as seguintes partes:

- Contagem unitária para cada estrutura de classificação selecionada no respectivo estilo, por célula da tabela 18;

- Dentre as estruturas de classificação constantes em cada célula da tabela 18, formar todos os pares possíveis dentro de cada célula. Para os pares formados, nas células: multiplicar grau de relacionamento e avaliação, com o resultado sendo multiplicado por 2;
- Somar os dos valores encontrados nos dois itens anteriores e acumular ao valor já existente no respectivo estilo.

3.3 Método de Identificação

A implementação de um sistema de produção, integrado a uma rede semântica, foi a forma escolhida para a execução do método de identificação de estilos de arquiteturas. Descreveremos, a seguir, este método de identificação de forma genérica, apoiados nas explicações já apresentadas. Com o objetivo de simplificarmos a explicação, percorreremos o método apenas para os estilos tubos e filtros e camada. A implementação do referido método é apresentada no capítulo 4.

O método inicia com o desenvolvedor observando quais características ou requisitos (tabela 18) o seu sistema em estudo possui. Isto é, supondo que o desenvolvedor tenha escolhido o requisito "deve tirar proveito do processamento concorrente e paralelo", observamos que duas estruturas de classificação (5 e 10 – tabela 18) ocorrem para o estilo de arquitetura tubos e filtros. Com isso, somamos dois pontos para o estilo tubos e filtros e nenhum ponto para o estilo camada (célula vazia). Porém, o sistema em estudo deve ter "componentes com funcionalidade on-line". Isso somará um ponto para o estilo camada (estrutura 13 - tabela 18) e nenhum ponto para o estilo tubos e filtros. Continuando este processo de contagem unitária, somaremos um ponto para cada

estrutura de classificação que aparecer na tabela 18, no respectivo estilo de arquitetura, conforme sugere o método de contagem já definido – parte 1 do método de contagem.

Ao terminar o processo de seleção, com base nos requisitos de qualidade geral, partiremos para outra etapa. Esta etapa utiliza as informações existentes na *base de conhecimento* ao que se refere a comparação entre estruturas de classificação. Este processo será feito com base no explicado na Figura 14. Este segundo processo de contagem ocorre da seguinte forma: para cada estrutura de classificação selecionada na tabela 18, vamos comparar esta com todas as outras da mesma célula, para o requisito em questão, no respectivo estilo – parte 2 do método de contagem.

Ao final desta pontuação, teremos um número de pontos para cada estilo de arquitetura. O estilo que obtiver o maior número de pontos é o que será indicado. Com as informações contidas na base de conhecimento, podemos verificar quais estruturas de classificação, com suas respectivas instâncias, não foram cumpridas para o estilo selecionado. Esta diferença dará origem às recomendações (estruturas de classificação faltantes – Figura 12) – parte 3 do método de contagem.

O método de identificação segue, em linhas gerais, o seguinte pseudocódigo:

1. Para cada linha da tabela 18:
    1.1. Se o requisito de qualidade geral é pertinente
        1.1.1. Para cada estrutura de classificação da célula em questão:
            1.1.1.1 Selecionar e contar um ponto no respectivo estilo da estrutura de classificação (Figura 13).

2. Para cada estrutura de classificação selecionada no passo 1:
    2.1. Comparar (Figura 14) esta com todas as outras selecionadas no passo 1.1.1.1, na sua respectiva célula, no seu respectivo estilo;
        2.1.1. Executar a pontuação no estilo.

3. Verificar qual estilo possui maior pontuação e recomendá-lo.
    3.1. Verificar quais estruturas de classificação não foram selecionadas no estilo recomendado;
    3.2. Para cada estrutura de classificação não selecionada:
        3.2.1. Recomendá-la, com a respectiva instância.

Figura 15 – Pseudocódigo do método de identificação

O usuário do método deve observar quando os dois estilos mais pontuados, alcançam uma pontuação próxima (150 pontos de diferença, conforme indicaram nossos estudos de caso), pode significar que o sistema em estudo possui mais de um estilo de arquitetura. Porém, este julgamento caberá ao próprio usuário do método. Nos referimos apenas aos dois primeiros

estilos, pois na literatura consultada, não observamos indicações para combinar mais de dois estilos. O usuário do método também pode optar por dividir o sistema em estudo em alguns módulos, e submeter cada um destes ao método classificatório.

Capítulo 4 – SIEA – Uma Ferramenta de Apoio à Identificação de Estilos de Arquiteturas

4.1 Implementação do Método de Identificação de Estilos de Arquiteturas

Nos capítulos 2 e 3 montamos a base de conhecimento e definimos o método de identificação de estilos de arquiteturas a partir dos requisitos de qualidade geral, respectivamente. Com tais propostas, passamos a ter uma base de conhecimento sobre estilos de arquiteturas de software e um método para acessá-la, objetivando uma classificação.

Aqui vamos apresentar um processo para a implementação do método que identifica os estilos de arquiteturas. O processo para a implementação passará seqüencialmente pelas fases que compõem o método:

1. Montagem da base de conhecimento;
2. Montagem dos requisitos do sistema em estudo;
3. Método de contagem.

Para apoiar o método que identifica os estilos de arquiteturas construímos um software. Chamamos de Sistema Identificador de Estilos de Arquiteturas – SIEA, o software que dá apoio ao método. O SIEA, assim como o método proposto, é composto por três módulos que apoiam as três fases citadas anteriormente.

O primeiro módulo do SIEA implementa todas as funcionalidades para formar e manter atualizada a base de conhecimento. É neste módulo que armazenamos informações sobre: estruturas de classificação, instâncias das estruturas de classificação, relacionamento entre estruturas de classificação e instâncias, estilos de arquiteturas, requisitos de qualidade geral e seus relacionamentos com as instâncias das estruturas de classificação, dentre outras informações contidas na base de conhecimento, anteriormente especificadas.

O segundo módulo do SIEA implementa todas as funcionalidades para formar e manter informações sobre o sistema em estudo. São mantidas informações sobre o nome do sistema em estudo, seus requisitos e a quais requisitos de qualidade geral estão associado aos requisitos do sistema em estudo.

O terceiro módulo do SIEA implementa o método de contagem. A partir de um sistema em estudo selecionado, submetemos este ao método de contagem. São mantidas as informações da pontuação alcançada em cada estilo de arquitetura para o sistema em estudo.

A seguir, na Figura 16, ilustramos os três módulos que compõem o SIEA.

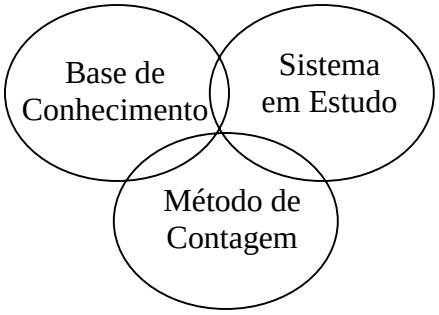

Figura 16 – Módulos que compõem o SIEA

Para a construção do SIEA dois ambientes de programação foram utilizados: Power Builder e CLIPS. Para a persistência de dados utilizamos o MS-Access. A seguir, apresentamos as justificativas de escolha destas ferramentas.

POWER BUILDER 6.0

Este ambiente de programação foi selecionado para a construção de interfaces com o usuário, interfaces com o CLIPS (via arquivo-texto) e interfaces com o MS-Access, via ODBC (*Open Database Connectivity*).

O Power Builder (PB) é um ambiente de ampla utilização comercial para a construção de sistemas cliente-servidor. Este pertencente a Sybase-Powersoft, podendo gerar código executável nas máquinas: Alpha-Digital, HP, Power PC, Macintosh, SUN-Sparc, 486 e Pentium / AMD. Os sistemas operacionais aceitos pelo Power Builder nas respectivas máquinas são: AIX, HP-UX, Macintosh, Solaris 2, Windows 16 bits, Windows 32 bits e Windows NT.

O ambiente de programação Power Builder apresenta uma interface gráfica para a construção de telas no padrão Windows. Toda a programação é a partir da linguagem proprietária, PowerScript. O estilo de programação é todo orientado a eventos, possuindo algumas características de orientação a objetos.

Programas feitos em Power Builder podem acessar dados em arquivos de formato texto e diversos outros formatos, como por exemplo, MS-Word, MS-Excel, todos através de OLE e ActiveX.

Acesso a dados em SGBD's (Sistemas Gerenciadores de Banco de Dados) podem ser feitos de duas formas: por driver nativo do SGBD ou via ODBC. Ambas as formas utilizam a SQL para manipulação das estruturas de banco de dados. Comandos SQL podem ser escritos dentro de códigos feitos em PowerScript. Para acesso aos dados que estarão armazenados em tabelas MS-Access, utilizaremos ODBC.

CLIPS 6.0

Visando apoiar a estruturação e uso da base de conhecimento, escolhemos o ambiente de programação CLIPS (*C Language Integrated Production System*). CLIPS é uma ferramenta para *sistemas especialistas* desenvolvida pelo Software Technology Branch (STB), NASA / Lyndon B. Johnson Space Center. A versão utilizada é a 6.0. Na estrutura da linguagem de programação e a partir dos comandos disponíveis, o CLIPS apresenta conceitos de Inteligência Artificial e orientação a objetos.

Dentre os diversos comandos existentes no CLIPS, utilizamos amplamente dois em nossa solução. São eles: *defrule* e *assert*. O comando *defrule* define regras e o comando *assert* define fatos.

Para a representação das redes semânticas utilizamos o comando *defrule*. Ou seja, representamos redes semânticas a partir da definição de regras. Definimos pares de estruturas de classificação, do lado esquerdo da regra, que ao tornarem se fatos teremos atribuídos, em variáveis, valores do grau de relacionamento e da avaliação – do lado direito da regra. Esta forma de programação em CLIPS será mais detalhada quando formos explicar as partes do programa CLIPS que implementamos – Figura 18 e Figura 19.

O comando *assert* é utilizado para fazer com que os pares de estruturas de classificação tornem-se fatos e então teremos o disparo da atribuição dos valores para o grau de relacionamento e à avaliação – do lado direito da regra. Detalhes serão apresentados posteriormente na Figura 20. A necessidade de tornarmos pares de estruturas de classificação em fatos, ocorre quando o método de identificação está avaliando os requisitos de qualidade geral, selecionados no sistema em estudo. São selecionadas todas as estruturas de classificação, que favorecem aos requisitos em questão, tornando-se fatos. No

momento em que tais estruturas tornaram-se fatos, passamos a ter valores para o grau de relacionamento e avaliação.

MS-ACCESS 97

Para a persistência dos dados, utilizaremos, como banco de dados relacional, o MS-Access. O MS-Access pertence a Microsoft e possui diversas funções além da armazenagem de dados. Esta ferramenta possui uma linguagem própria – VB Script – destinada a manipulação dos dados, cálculos aritméticos, entre outras funções. Para a manipulação de dados o MS-Access também oferece mecanismos de visões, as quais apresentam um linguagem para manipular os dados, bem como um ambiente gráfico para a montagem de telas e relatórios. Não podemos considerar o MS-Access como um SGBDR (Sistema Gerenciador de Banco de Dados Relacional) pois a ele faltam diversos mecanismos para tal: controle de concorrência, backup e recovery, entre outros.

Para a nossa solução, utilizaremos as estruturas de tabelas do MS-Access. Estas tabelas serão implementadas a partir de uma modelagem relacional. Nenhuma outra funcionalidade do MS-Access será utilizada em nossa solução.

A Figura 17 apresenta esquematicamente a proposta para a implementação do SIEA. O esquema procura identificar os pontos de integração das três ferramentas escolhidas: Power Builder, CLIPS e MS-Access.

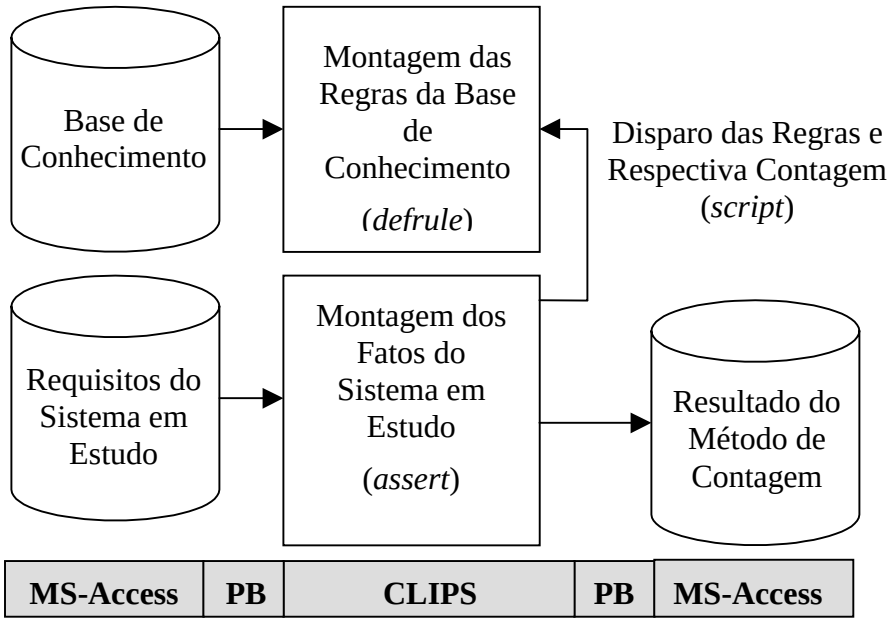

Figura 17 – Esquema para a implementação do SIEA

A seguir, descrevemos as principais características presentes na arquitetura do SIEA e na integração das ferramentas, conforme observado na Figura 17, são elas: persistência de dados, interface do usuário com o SIEA, interface com a persistência de dados e montagem das regras e fatos no programa CLIPS.

PERSISTÊNCIA DE DADOS

Toda a persistência de dados é feita em tabelas do MS-Access. Nestas tabelas armazenamos informações sobre a base de conhecimento, sobre os requisitos do sistema em estudo e sobre os resultados gerados pelo método de contagem.

No módulo de representação da base de conhecimento armazenamos, em tabelas do MS-Access, informações sobre: valores existentes para os graus de relacionamento, valores existentes para as avaliações, estruturas de classificação, instâncias das estruturas de classificação, relacionamentos entre estruturas de classificação, relacionamentos entre as instâncias das estruturas de

classificação, quais instâncias de estruturas de classificação formam quais estilos de arquiteturas e quais instâncias de estruturas de classificação atendem a quais requisitos de qualidade geral.

No módulo de montagem dos requisitos do sistema em estudo armazenamos, em tabelas do MS-Access, informações sobre os requisitos que devem ser atendidos pelo sistema em estudo, bem como estes requisitos se relacionam com os requisitos de qualidade geral. O relacionamento que ocorre entre os requisitos foi explicado anteriormente no capítulo 3, quando falamos sobre sistemas de produção.

No terceiro módulo armazenamos, em tabelas MS-Access, informações sobre os resultados que obtemos ao executar o método de contagem, para cada sistema em estudo.

INTERFACE DO USUÁRIO COM O SIEA

Todas as interfaces de telas e relatórios foram implementadas a partir do ambiente de programação oferecido pelo Power Builder. Tal ambiente permite a construção de interfaces nas quais o usuário interage com o sistema a partir de botões e menus, os quais levarão a execução de alguma ação.

INTERFACE COM A PERSISTÊNCIA DE DADOS

Todas as tabelas do MS-Access são acessadas a partir de programas escritos em PowerScript, via ODBC. No código destes programas existem comandos SQL embutidos, os quais permitem a manipulação dos dados em tabelas do MS-Access. Ou seja, qualquer consulta, ou mudança, na base de conhecimento e nas informações sobre o sistema em estudo, ocorrerão a partir de programas, escritos em PowerScript com SQL embutido, acessando tabelas do MS-Access.

MONTAGEM DAS REGRAS E FATOS

Conforme dito anteriormente, tabelas do MS-Access guardam informações sobre a base de conhecimento e sobre os requisitos dos sistemas em estudo. Um determinado programa, escrito em PowerScript, contido em um evento do botão *Executar* – Figura 36, é o responsável pela

leitura da base de conhecimento, e montagem do programa CLIPS que define regras e fatos. Este programa, escrito em PowerScript, faz acesso as tabelas do MS-Access a partir de comandos SQL embutidos na linguagem PowerScript. Com as informações lidas nas tabelas do MS-Access, o programa PowerScript monta dinamicamente um programa CLIPS. Ou seja, o programa PowerScript, contido em um evento do botão *Executar*, grava um arquivo, com extensão CLP, que contém comandos CLIPS para a definição de regras, fatos e somatórios de valores. A execução do programa CLIPS vai implementar as regras que representarão as redes semânticas, bem como a montagem dos fatos a partir dos requisitos do sistema em estudo. As regras são definidas no programa CLIPS a partir do comando *defrule* e os fatos a partir do comando *assert*. Em seguida, este mesmo programa PowerScript inicializa o programa CLIPS que foi gerado. O programa CLIPS monta as regras e dispara os fatos definidos. A partir do disparo dos fatos é que obtemos os valores dos graus de relacionamento e avaliação, para posteriormente termos a pontuação do sistema em estudo. A pontuação é gravada pelo programa CLIPS em um arquivo-texto. Após a execução do programa CLIPS o controle volta para o programa PowerScript citado anteriormente. Este programa lê o conteúdo do arquivo-texto, gerado pelo programa CLIPS, e então grava o resultado em tabelas do MS-Access.

Com base no que descrevemos, da interface entre o Power Builder e o CLIPS, esquematicamente podemos dividir o programa PowerScript em quatro partes a saber:

1 O programa PowerScript, lê as informações da base de conhecimento, que estão em tabelas MS-Access, monta um programa CLIPS, dinamicamente, em tempo de execução – conforme anexo B.

2 Este mesmo programa PowerScript chama, em modo síncrono, um *engine* do CLIPS, chamado wxCLIPS 1.31, passando como parâmetro o programa CLIPS que acabou de

ser montado – conforme comentário *"Executa o programa CLIPS"* existente no código do anexo B.

3 O programa CLIPS é executado a partir do wxCLIPS 1.31. O programa CLIPS é montado de tal forma que destrua a si próprio e o *engine* após sua execução, inclusive com liberação de memória – conforme comando *"Run("wxclips -clips pgm01.clp -start", Minimized!)"* existente no código do anexo B.

4 Após a execução do programa CLIPS, o controle volta para o programa PowerScript. O programa PowerScript lê os resultados gerados pelo programa CLIPS (em um arquivo-texto) conforme comentário *"aguarda término da execução do CLIPS"* existente no código do anexo B. Posteriormente, o programa PowerScript grava os resultados gerados, em tabelas MS-Access.

O programa CLIPS, montado pelo programa PowerScript, é formado a partir de seis partes que descrevemos a seguir:

1 Regras são definidas retratando os relacionamentos entre estruturas de classificação e a ação a ser executada (dar valor ao *grau de relacionamento*), caso tais regras tornem-se fatos. Com base no exposto na Figura 18, temos uma regra chamada *estr1*. Esta regra define que quando o par de estruturas *1cdestrclas1-eh* e *1cdestrclas2-eh* assumirem os valores 1 e 2, respectivamente, e quando tornarem-se fatos, estaremos atribuindo o valor 4 ao grau de relacionamento, aqui representado pela variável *vlgraurelc*.

A definição da regra aqui apresentada é a forma como representamos, no CLIPS, a rede semântica.

```
(defrule estr1 (1cdestrclas1-eh 1)
               (1cdestrclas2-eh 2)
            => (bind ?*vlgraurelc* 4) )
```

Figura 18 – Código CLIPS que define o relacionamento das estruturas de classificação

2 Regras são definidas retratando os relacionamentos entre as instâncias das estruturas de classificação e a ação a ser executada (dar valor à *avaliação*), caso tais regras tornem-se fatos. Com base no exposto na Figura 19, temos uma regra chamada *inst1*. Esta regra define que quando o par de estruturas *2cdestrclas1-eh* e *2cdestrclas2-eh*, acompanhadas de suas respectivas instâncias *2cdinstestrclas1-eh* e *2cdinstestrclas2-eh*, assumirem os valores 1 e 2, com valores das instâncias 1 e 1, respectivamente, e quando tornarem-se fatos, estaremos atribuindo o valor +2 à avaliação, aqui representada pela variável *vlaval*.

A definição da regra aqui apresentada é a forma como representamos, no CLIPS, a rede semântica.

```
(defrule inst1
         (2cdestrclas1-eh 1)
         (2cdinstestrclas1-eh 1)
         (2cdestrclas2-eh 2)
         (2cdinstestrclas2-eh 1)
      =>
         (bind ?*vlaval* 2)
)
```

Figura 19 – Código CLIPS que define o relacionamento das instâncias das estruturas de classificação

3 Cada requisito de qualidade geral possui a ele associado algumas estruturas de classificação e respectivas instâncias para cada estilo de arquitetura. São estas estruturas de classificação e instâncias que favorecem a ocorrência do requisito. Com base nos requisitos de qualidade geral relacionados com os requisitos do sistema em estudo, são definidos fatos. Estes fatos representam os pares de estruturas de classificação e respectivas instâncias, os quais favorecem a ocorrência dos requisitos. Os fatos são definidos a partir do comando CLIPS *assert* – Figura 20.

4 Cada fato define pares de estruturas de classificação e respectivas instâncias. Estas definições vão causar o disparo das regras onde serão atribuídos valores para grau de relacionamento e avaliação. Para cada fato definido, as regras vão sendo disparadas. Durante a montagem do programa CLIPS, os fatos foram definidos por ordem de estilo de arquitetura, o que possibilita o acúmulo da pontuação para cada um destes. Após a atribuição de valores para grau de relacionamento e avaliação, estes valores são multiplicados, conforme explicado anteriormente durante o detalhamento do método de contagem. Em seguida, o valor da multiplicação é somado a uma variável que acumula

os pontos para o estilo de arquitetura que estamos processando. Na Figura 20 demos o nome de *cont* para esta variável.

```
(bind ?*vlgraurelc* 0)
(bind ?*vlaval* 0)
(assert (1cdestrclas1-eh 1) (1cdestrclas2-eh 2) )
(run)
(assert (2cdestrclas1-eh 1) (2cdinstestrclas1-eh 1)
        (2cdestrclas2-eh 2) (2cdinstestrclas2-eh 1) )
(run)
( bind ?*cont* ( + ( * ( * ?*vlgraurelc* ?*vlaval* ) 2 ) ?*cont* ) )
( retract * )
```

Figura 20 – Código CLIPS que define os fatos, a partir dos requisitos de qualidade geral

5 Um mesmo fato pode ser repetido várias vezes para uma mesma arquitetura, visto que um par de estruturas de classificação aparece em mais de um requisito de qualidade geral, para uma mesma arquitetura (tabela 18). Devido a esta característica e para não haver contagem em duplicidade dos fatos já definidos, temos que eliminar os fatos – comando *retract ** - imediatamente após sua execução.

6 Ao final da definição de todos os fatos de cada estilo de arquitetura, um arquivo-texto é gravado indicando o estilo e a pontuação alcançada por cada um destes.

Após termos apresentado a macro-arquitetura do SIEA, bem como as ferramentas escolhidas e suas formas de interação, vamos detalhar como implementamos a Inteligência Artificial em nossa solução.

4.2 Estratégia de Implementação da Inteligência Artificial

Durante a construção do SIEA, utilizamos duas técnicas de Inteligência Artificial: rede semântica e sistema de produção. A rede semântica foi utilizada como uma forma estruturada de representação do conhecimento [Rich+83]. A definição de regras do CLIPS foi a forma por nós encontrada para a representação das redes semânticas. A esta rede semântica integramos um sistema de produção que é composto pela definição de fatos e execução de *scripts* que somam pontos conforme sugere o método de contagem. O sistema de produção inicia a partir da definição de fatos que disparam regras existentes em nossa base de conhecimento. Estas regras são compostas de pares de estruturas de classificação e respectivas instâncias. Tais regras, quando tornam-se fatos atribuem valores de grau de relacionamento e avaliação pertinentes aos pares de estruturas de classificação e instâncias existentes nas regras. Posteriormente, o sistema de produção multiplica os dois valores mencionados e soma o resultado da multiplicação a uma variável que vai armazenar a pontuação obtida para cada estilo de arquitetura. As operações de definição dos fatos, multiplicação e soma, ocorrem a partir de um *script* do sistema de produção, já mencionado.

Para a escolha de como implementaríamos a rede semântica e o sistema de produção, dois pontos foram observados:

- É a primeira versão do software, desta forma, devemos privilegiar o entendimento do método de identificação do estilo de arquitetura, para depois evoluirmos em representações menos granulares do conhecimento. Esta observação influenciou a escolha das primitivas para a representação do conhecimento.

- O valor do *grau de relacionamento* de um *par de estruturas de classificação* é independente de quaisquer outros valores de pares de estruturas, bem como o valor da *avaliação* entre as respectivas *instâncias*. Isso implica em não podermos fazer deduções de algum relacionamento ao analisarmos um conjunto destes. Ou seja, é necessária a representação individual de cada relacionamento.

Isso posto, detalharemos a seguir a forma pela qual implementamos a rede semântica e o sistema de produção.

REDE SEMÂNTICA

A rede semântica é aplicada em duas situações. A primeira, para identificar quais instâncias são de quais estruturas de classificação - Figura 9. A segunda é para a representação dos valores de grau de relacionamento e avaliação. A representação dos referidos valores ocorre a partir do relacionamento entre estruturas de classificação e entre instâncias destas, respectivamente - Figura 10.

Durante o método de contagem dois valores são obtidos, o grau de relacionamento e a avaliação. Para a obtenção do valor do grau de relacionamento utilizamos a rede semântica dos pares das estruturas de classificação. Para a obtenção do valor da avaliação utilizamos a rede semântica dos pares das instâncias das estruturas de classificação. Isso posto, e para privilegiar o exercício do método de identificação de estilos de arquiteturas, escolhemos dois elementos para o nosso conjunto de primitivas: estruturas de classificação e instâncias das estruturas de classificação. A existência de pares das estruturas de classificação leva a atribuição de valor para o grau de relacionamento. A existência de pares das instâncias das estruturas de classificação leva à atribuição de valor para a avaliação.

Apoiado na afirmação de que “rede semântica é uma das melhores formas para encontrar relações entre objetos [Rich+83]”, tal representação foi escolhida, inclusive conjugando conceitos de orientação a objetos e Inteligência Artificial, conforme sugere o ambiente de programação CLIPS. No CLIPS, implementamos a representação da rede semântica a partir da definição de regras. Ou seja, definimos um par de estruturas de classificação no lado esquerdo da regra e associamos um valor de grau de relacionamento no lado direito da regra – Figura 18. Para a representação da rede semântica das instâncias das estruturas de classificação utilizamos solução semelhante a anteriormente citada, conforme já ilustrado na Figura 19.

SISTEMA DE PRODUÇÃO

Conforme citado anteriormente, implementamos o método de contagem a partir de um sistema de produção, com definição de fatos e execução de *scripts* com operações de multiplicação e soma de valores.

A partir da constatação de que nossos conhecimentos estão codificados em regras de um programa CLIPS, devemos propor um mecanismo que utilize o conhecimento armazenado. Devemos definir fatos para que as regras, anteriormente citadas, possam ter seu lado direito ativado e consequentemente termos as atribuições de valores do grau de relacionamento e avaliação. Os fatos são provenientes da leitura dos requisitos do sistema em estudo, conforme verificamos no capítulo 3. Tais requisitos estão associados aos requisitos de qualidade geral, conforme explicamos anteriormente. Os requisitos de qualidade geral possuem pares de estruturas de classificação, em seus respectivos estilos de arquitetura. Cada um destes pares serão definidos como fatos – comando *assert* - dentro do programa CLIPS. Estes fatos vão ativar o lado direito das regras já definidas e consequentemente teremos as atribuições de valores para o grau de relacionamento e a avaliação.

A identificação unitária de cada relacionamento dos pares das estruturas de classificação e instâncias, e conseqüente escolha das primitivas muito granulares, nos levarão a problemas futuros, tal como o grande volume de dados a ser armazenado em nossa base de conhecimento – na forma de tabelas do MS-Access. Este problema acarretará grande esforço de processamento para converter as primitivas em respostas úteis.

Atualmente, para a forma como codificamos o SIEA, identificamos três problemas descritos na literatura quando escolhemos primitivas muito granulares. São eles: grande volume de armazenamento de dados, esforço de processamento para produzir respostas úteis e a situação do mesmo *fato* ter que ser definido diversas vezes para obtermos o resultado de alto-nível (tendo que ser utilizado em conjunto com um eliminador de *fatos* – comando *retract*). Os dois primeiros problemas não foram considerados como críticos neste momento, pois ainda não temos uma base de dados suficientemente grande para tal. O terceiro problema resultou em uma lógica de definição dos *fatos* mais elaborada para não haver duplicação de pontos no método de contagem.

Porém, para chegarmos as respostas do problema proposto, a definição dos fatos somente nos serviu para a determinação dos valores do *grau de relacionamento* e da *avaliação*. Para manipularmos tais valores e chegarmos aos totais de pontos por estilo de arquitetura, utilizamos um *script*, compondo assim a segunda parte do sistema de produção. A seguir, apresentamos a definição do *script* proposto para cada par de estruturas de classificação das células da tabela 18. Após a Figura 21 apresentaremos explicações mais detalhadas sobre a mesma.

Script: Relação entre pares de Estruturas de Classificação e suas Instâncias

Condições de Entrada: - existir a regra que tenha como par as estruturas E1 e E2 no seu lado esquerdo e o valor do grau de relacionamento (G) em seu lado direito;

- existir a regra que tenha como par a estrutura E1, com sua respectiva instância I1 e a estrutura E2, com sua respectiva instância I2, todos no lado esquerdo da regra; e o valor da avaliação (A) do lado direito;

Resultado: - definição de fatos que disparam o lado direito das regras, atribuindo valores para o grau de relacionamento (G) e avaliação (A);

- multiplicação dos dois valores obtidos a partir do disparo do lado direito das regras.

Propriedade: cada instância de estrutura de classificação pertence ao mesmo estilo de arquitetura.

Regras: Estrutura de classificação 1 = E1

Instância da estrutura de classificação 1 = I1

Estrutura de classificação 2 = E2

Instância da estrutura de classificação 2 = I2

Estilo de Arquitetura ao qual a instância pertence = AR()

Grau de relacionamento entre estruturas de classificação = G

Avaliação entre instâncias das estruturas de classificação = A

Cenário:

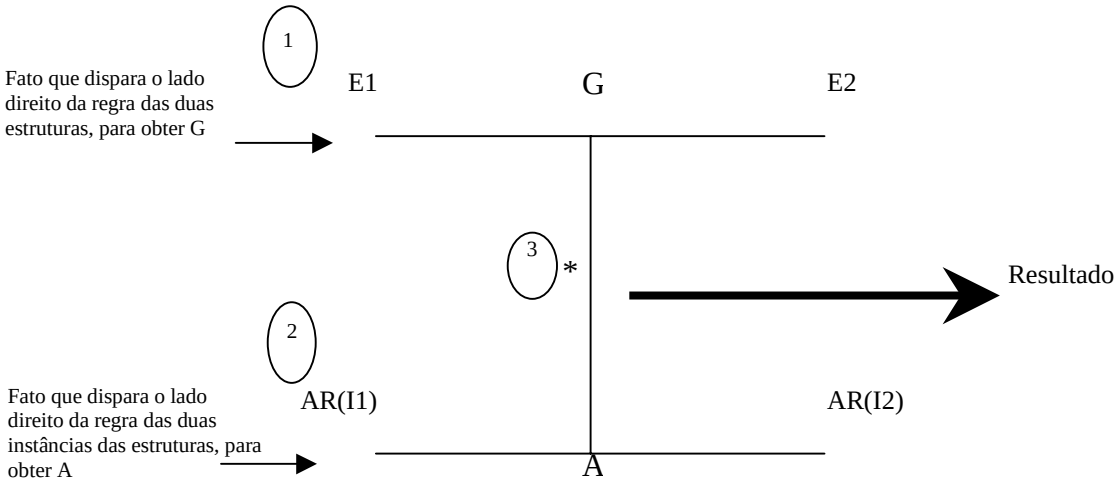

Figura 21 – Script da relação entre duas estruturas de classificação e suas instâncias

Os passos 1 e 2 da Figura 21 são feitos a partir das definições dos fatos. O passo 3 é o *script* de multiplicação dos valores do grau de relacionamento e da avaliação. Para a obtenção do valor de uma célula da tabela 18 o referido *script* é executado várias vezes, uma vez para cada par de estruturas existente na célula. Cada valor resultante de um par de estruturas da célula é somado ao valor já existente para a arquitetura em questão. A soma dos valores de todos os pares resulta na pontuação total da referida arquitetura. Este método de contagem ocorre para cada arquitetura existente em nossa base de conhecimento.

Conforme citado anteriormente, é importante ressaltar que a escolha das primitivas privilegiou o estudo do método de identificação de estilos de arquiteturas e não a representação da base de conhecimento. Conforme o método de identificação de estilos de arquiteturas for sendo solidificado, estaremos modificando as técnicas de representação do conhecimento. Ou seja: o *script* apresentado na Figura 21 poderá ser substituído por uma representação que informa o total de pontos de célula da tabela 18; as primitivas não mais serão estruturas de classificação e instâncias, e sim requisitos de qualidade geral, já pontuados; entre outras formas de elevar a representação do conhecimento armazenado. Ainda no intuito de elevar a representação do conhecimento, podemos prever a existências de grupos pré-definidos de requisitos de qualidade geral, que nos levarão a recomendação de um estilo de arquitetura, não havendo a necessidade do *script* da Figura 21. Ou seja, a seleção de um conjunto de requisitos de qualidade geral nos levarão a recomendação de um estilo de arquitetura já identificado em nossa base de conhecimento.

4.3 Utilização do SIEA

A seguir, vamos identificar como implementamos as três fases do método de identificação de estilos de arquiteturas. Falaremos seqüencialmente das fases já citadas anteriormente: montagem da base de conhecimento, montagem dos requisitos do sistema em estudo e o método de contagem. Para cada uma das fases indicaremos quais interfaces são pertinentes, bem como apresentaremos uma breve descrição das telas.

4.3.1 FASE 1 – Montagem da Base de Conhecimento

O processo inicia com o armazenamento do conhecimento existente, segundo a taxonomia proposta. A base de conhecimento é armazenada em tabelas do MS-Access. Os dados destas tabelas são inseridos e mantidos a partir de programas Power Builder. A seguir, na Figura 22, apresentamos a arquitetura proposta para a base de conhecimento. O motivo para selecionarmos o estilo de arquitetura repositório para a representação da base de conhecimento é explicado no estudo de caso 5 do capítulo 5.

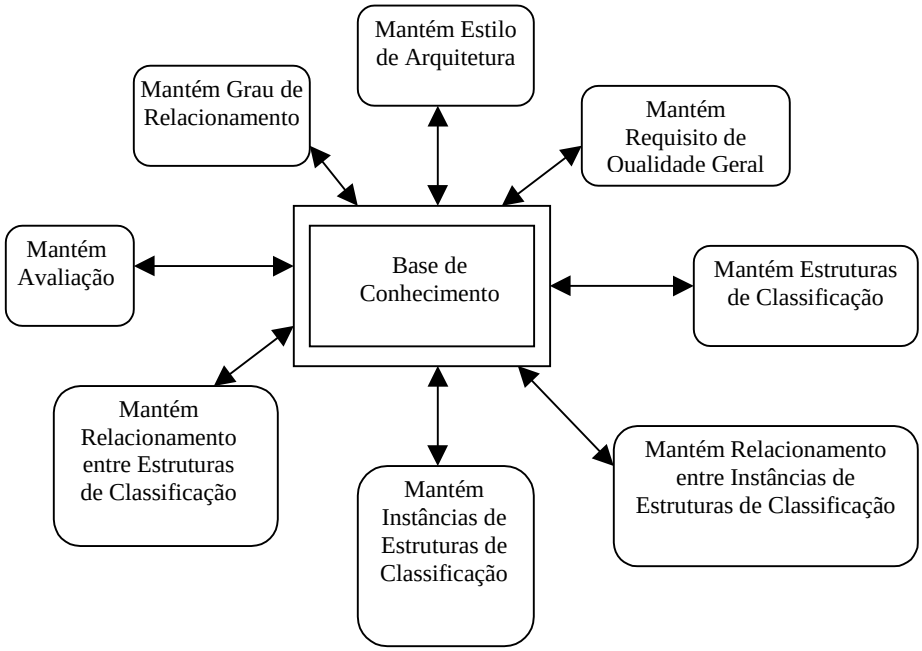

Figura 22 – Arquitetura da base de conhecimento

Na base de conhecimento existem dois grupos de tipos de informações a serem cadastradas. O primeiro grupo contem as informações adquiridas após a definição da taxonomia, são elas: estruturas de classificação, instâncias das estruturas, valores de grau de relacionamento, valores de avaliação, estilos de arquiteturas e requisitos de qualidade geral. O segundo grupo da base de conhecimento contém os relacionamentos encontrados entre os elementos da taxonomia proposta, que são três. O primeiro é o relacionamento dos requisitos de qualidade geral com determinadas estruturas em um estilo de arquitetura - Figura 30. O segundo relacionamento é entre estruturas de classificação, medido pelo grau de relacionamento – Figura 31. O terceiro relacionamento é entre as instâncias das estruturas de classificação, medido pelo valor da avaliação – Figura 32.

A seguir apresentaremos as principais telas que implementam as interfaces que servem para a manutenção da base de conhecimento. Para cada um dos dois grupos da base de conhecimento falaremos das tabelas que as compõem, descrevendo seus campos. Todas as tabelas encontram-se na terceira forma normal. Informações adicionais sobre o Modelo de Entidades e

Relacionamentos - MER, bem como o layout das tabelas da base de conhecimento podem ser consultadas no anexo D.

A Figura 23 apresenta a tela inicial do SIEA. A partir dos menus desta tela é que temos acesso as demais partes do sistema.

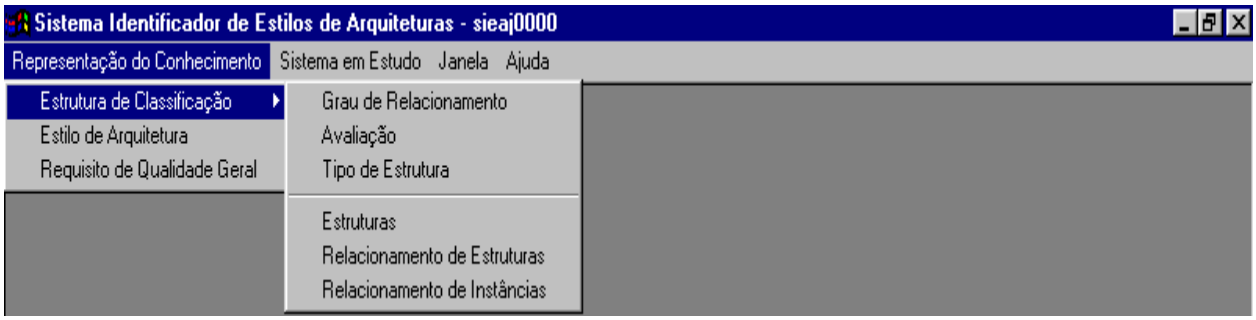

Figura 23 – Tela inicial do SIEA

Na Figura 24 apresentamos a tela, que é ativada a partir do menu *Estruturas* - Figura 23, com dois objetivos semelhantes. O primeiro objetivo é o de cadastrar as estruturas de classificação primárias. Conforme nosso conhecimento atual: componentes e conectores – tabela 16. O segundo objetivo é o cadastramento das estruturas derivantes dos relacionamentos das estruturas de classificação primárias, que são as estruturas de classificação auxiliares. Como por exemplo: o *sincronismo* (código 5 - Figura 24) que ocorre entre dois componentes, a partir de um ou mais conectores – tabela 16.

A tabela que armazena as estruturas de classificação possui os seguintes campos: código, nome, descrição e tipo (primária ou auxiliar). Os tipos possíveis de estruturas de classificação são armazenados em uma tabela contendo: código e descrição.

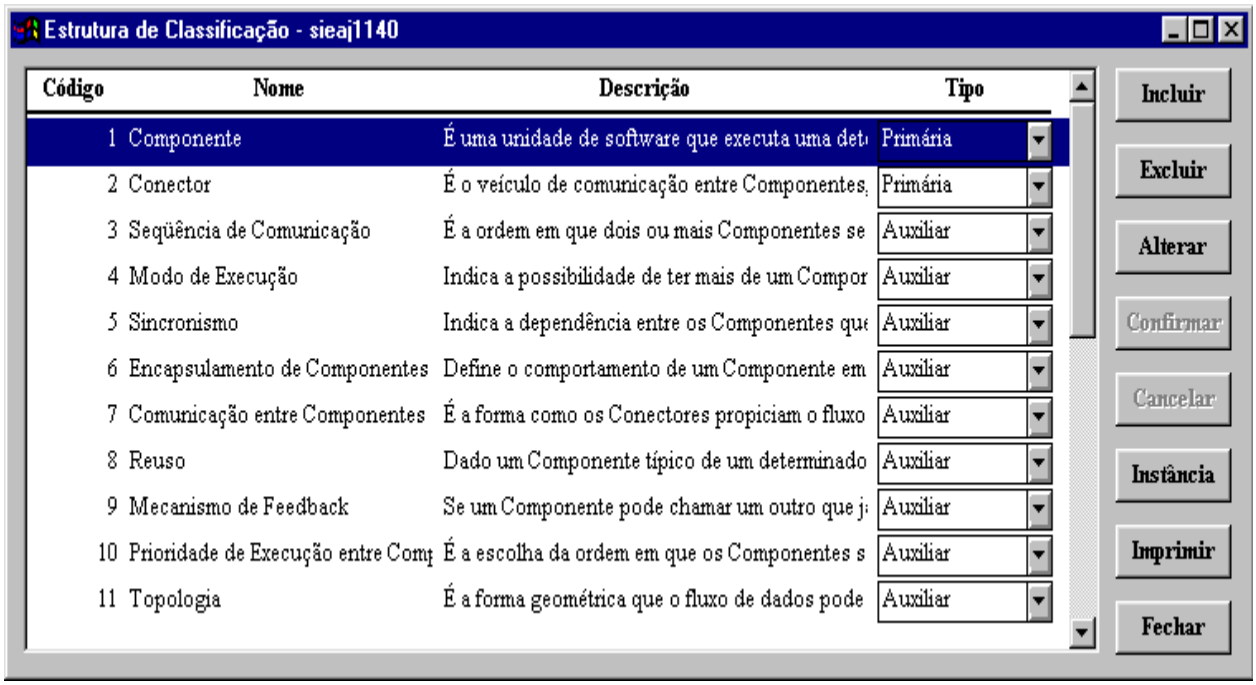

Figura 24 – Tela de cadastramento das estruturas de classificação primárias e auxiliares

Na Figura 25 apresentamos uma única tela com dois objetivos semelhantes. O primeiro objetivo é cadastrar cada uma das instâncias das estruturas de classificação primárias. Como exemplo: filtros, camadas, objetos, tubos de dados, interface entre camadas, troca de mensagens entre objetos, entre outras – tabela 16. O segundo objetivo é cadastrar cada ocorrência das instâncias das estruturas de classificação auxiliares, como por exemplo o *sincronismo* citado na figura anterior, com instâncias: linear e não-linear – tabela 16. A tela da Figura 25 pode ser ativada a partir da tela da Figura 24 – botão “Instância", conforme estrutura de classificação selecionada.

As instâncias das estruturas de classificação são armazenadas em tabelas com o seguinte conteúdo: código da estrutura pai, código da instância, nome e descrição.

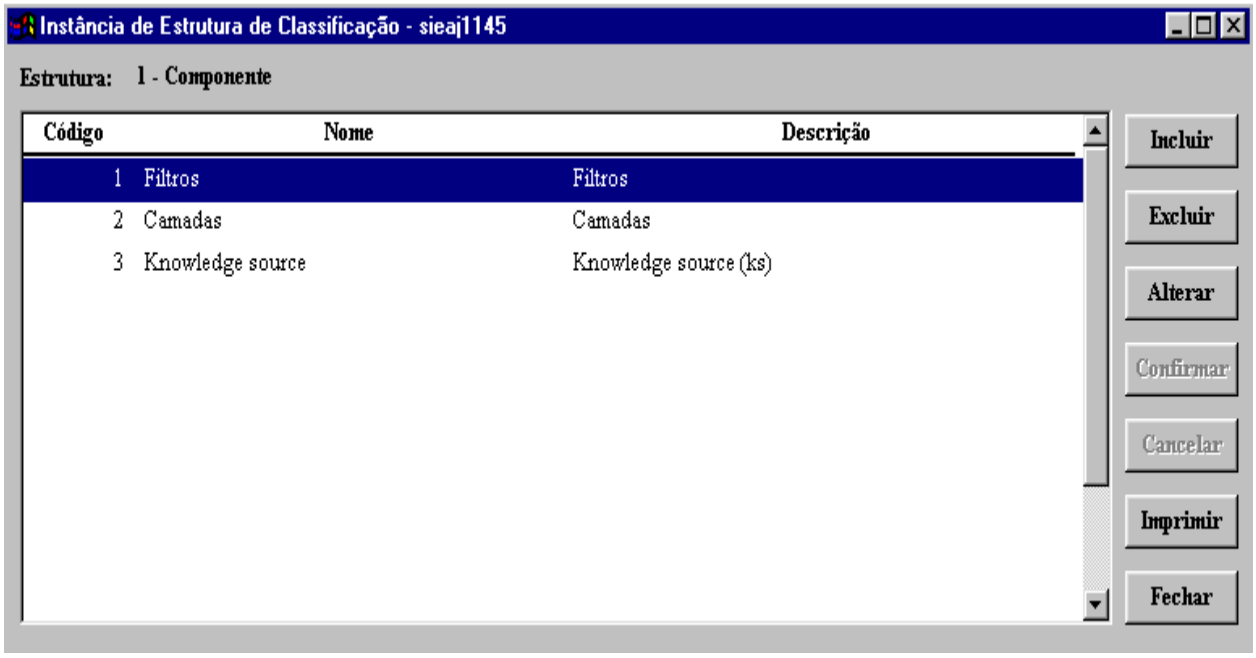

Figura 25 - Tela de cadastramento das instâncias das estruturas primárias e auxiliares

A Figura 26 apresenta a tela que cadastra valores que indicam o *grau de relacionamento* entre pares de estruturas de classificação – tabela 14, tais valores formarão nossa base de conhecimento e participarão do método de contagem. Esta tela é ativada a partir do menu *Grau de Relacionamento* - Figura 23.

Os valores dos graus de relacionamento são armazenados em uma tabela que contém o valor e a descrição de cada grau.

Figura 26 – Tela de cadastramento de valores dos graus de relacionamento

A Figura 27 apresenta a tela que cadastra valores que indicam a *avaliação* feita entre pares de instâncias das estruturas de classificação distintas – tabela 15, tais valores formarão nossa base de conhecimento e participarão do método de contagem. Esta tela é ativada a partir do menu *Avaliação* - Figura 23.

Os valores das avaliações são armazenados em uma tabela que contém o valor e a descrição de cada avaliação.

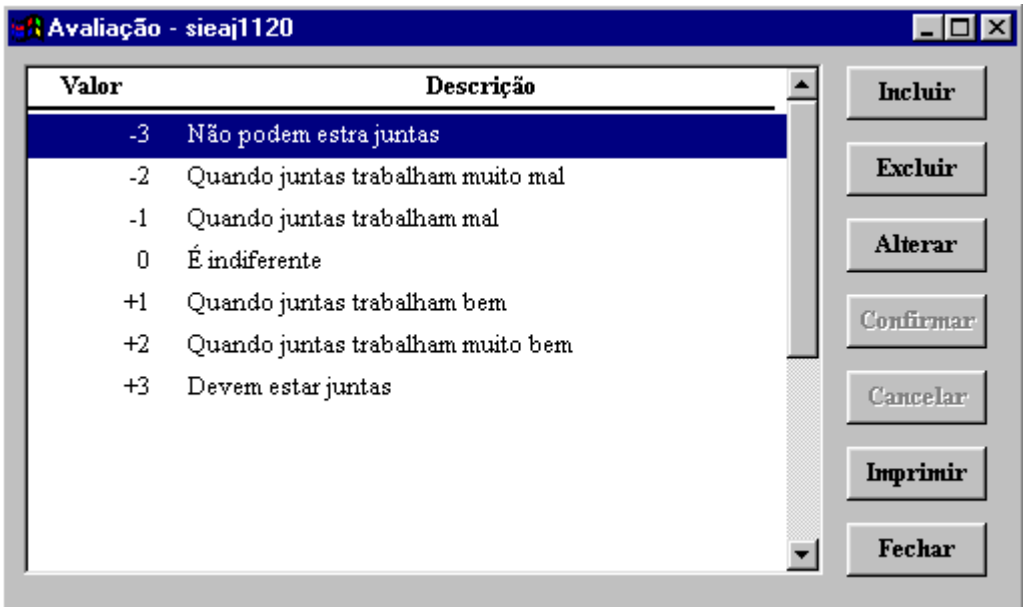

Figura 27 – Tela de cadastramento de valores de avaliação

A Figura 28 apresenta a tela que cadastra os estilos de arquitetura e quais valores das instâncias das estruturas de classificação compõem os respectivos estilos – tabela 17. Esta tela é ativada a partir do menu *Estilo de Arquitetura* - Figura 23.

As informações sobre os estilos de arquitetura são armazenadas em duas tabelas. A primeira tabela contém o código do estilo, o nome e a descrição. A segunda tabela indica quais estruturas de classificação e instâncias formam o referido estilo. Esta é composta de: código do estilo, código da estrutura de classificação e código da instância da estrutura de classificação.

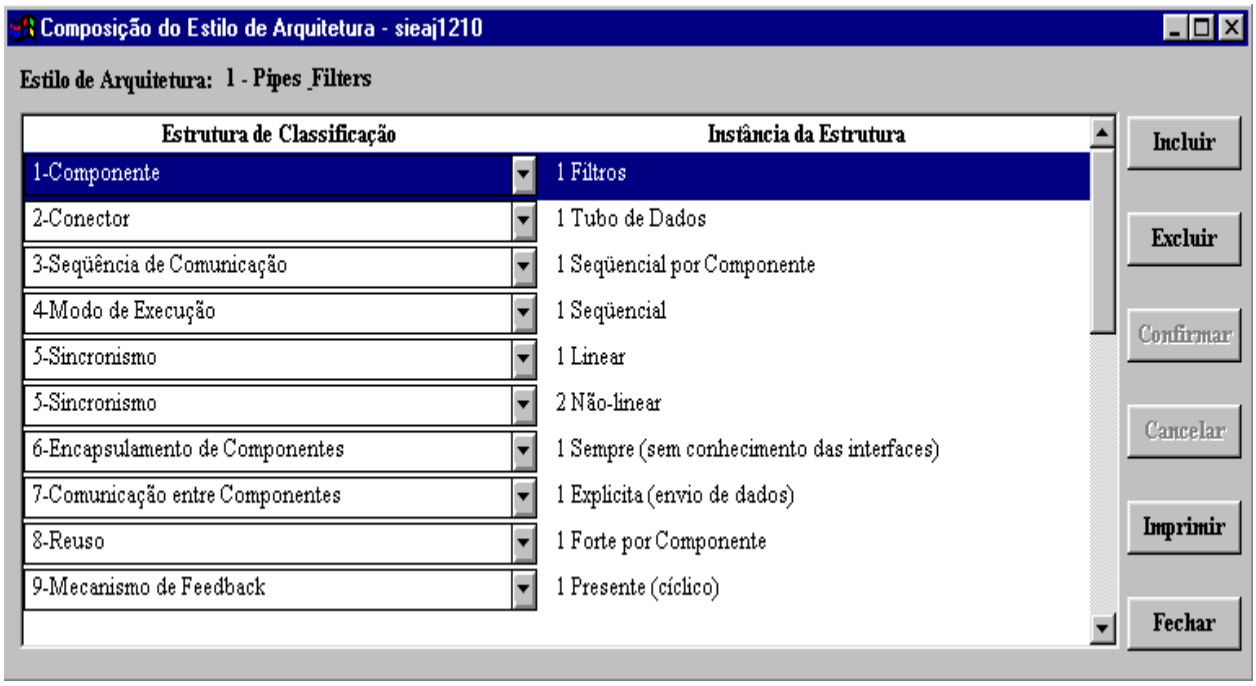

Figura 28 – Tela de cadastramento dos estilos de arquitetura

A Figura 29 apresenta a tela, ativada pelo menu *Requisito de Qualidade Geral* - Figura 23, que cadastra os requisitos de qualidade geral que compõem nossa base de conhecimento.

Tais requisitos são armazenados em uma tabela que contém o código do requisito com sua descrição.

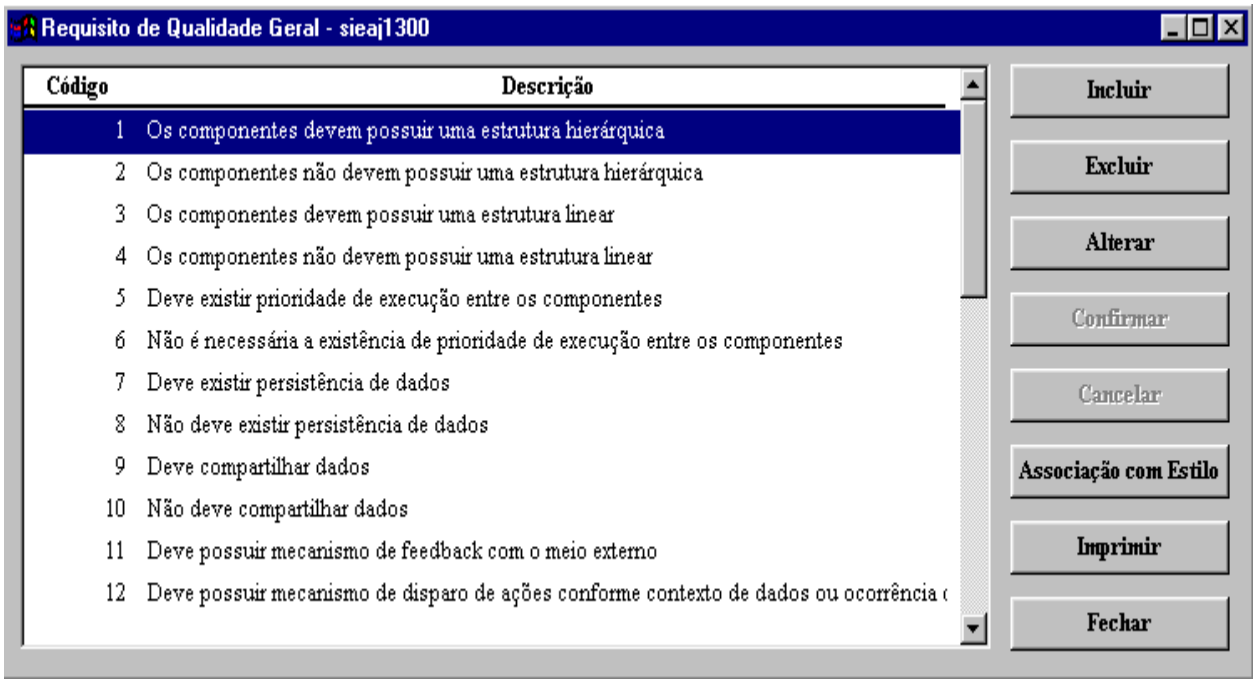

Figura 29 – Tela de cadastramento dos requisitos de qualidade geral

Na Figura 30 apresentamos a tela, ativada pelo botão “Associação com Estilo” - Figura 29, que para cada um dos requisitos cadastrados, relacionamos quais estruturas de classificação são pertinentes, com o estilo de arquitetura em questão – tabela 18.

Esta informação sobre os requisitos é armazenada em uma tabela que contém os campos: código do requisito, código do estilo de arquitetura e código da estrutura de classificação.

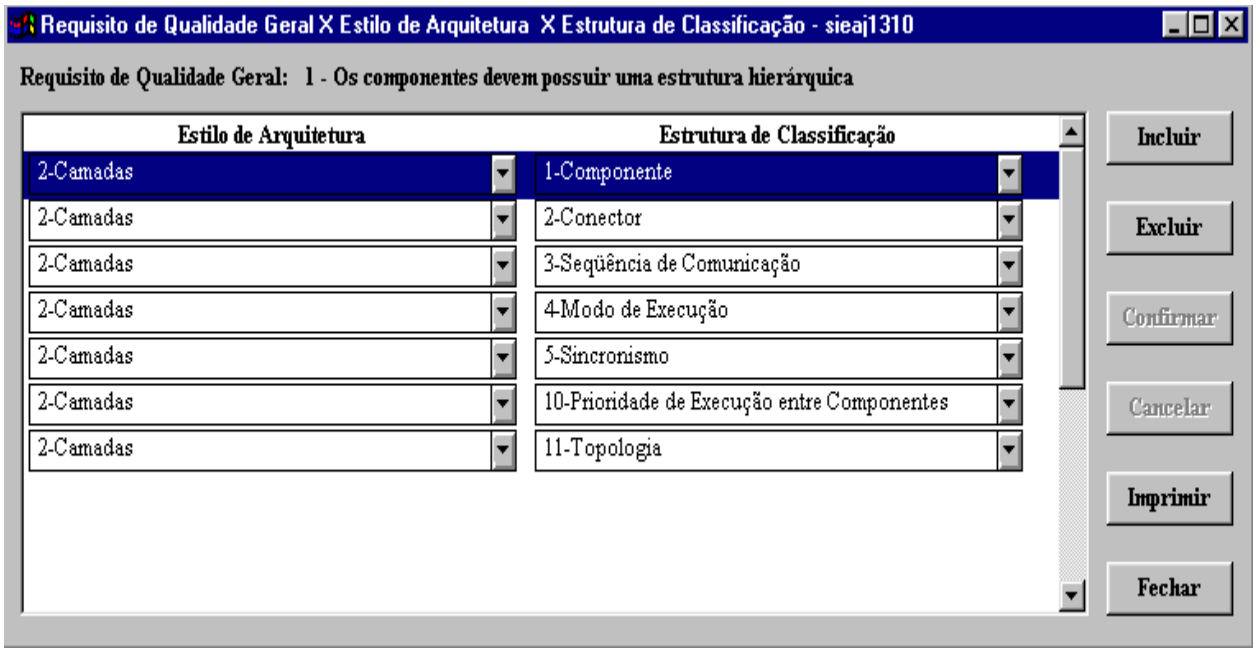

Figura 30 – Tela de relacionamento entre requisitos de qualidade geral e estruturas de classificação

A Figura 31 apresenta a tela, ativada pelo menu *Relacionamento de Estruturas* - Figura 23, que cadastra os graus de relacionamento entre pares de estruturas de classificação. Para armazenar tais informações, utilizamos uma tabela com os seguintes campos: código da primeira estrutura de classificação, código da segunda estrutura de classificação e valor do grau de relacionamento.

Figura 31 - Tela de relacionamento entre estruturas de classificação

A Figura 32 apresenta a tela, ativada pelo menu *Relacionamento de Instâncias* - Figura 23, que cadastra as avaliações entre pares de instâncias de estruturas de classificação. Para armazenar tais informações, utilizamos uma tabela com os seguintes campos: código da primeira estrutura de classificação e respectiva instância, código da segunda estrutura de classificação e respectiva instância e valor da avaliação.

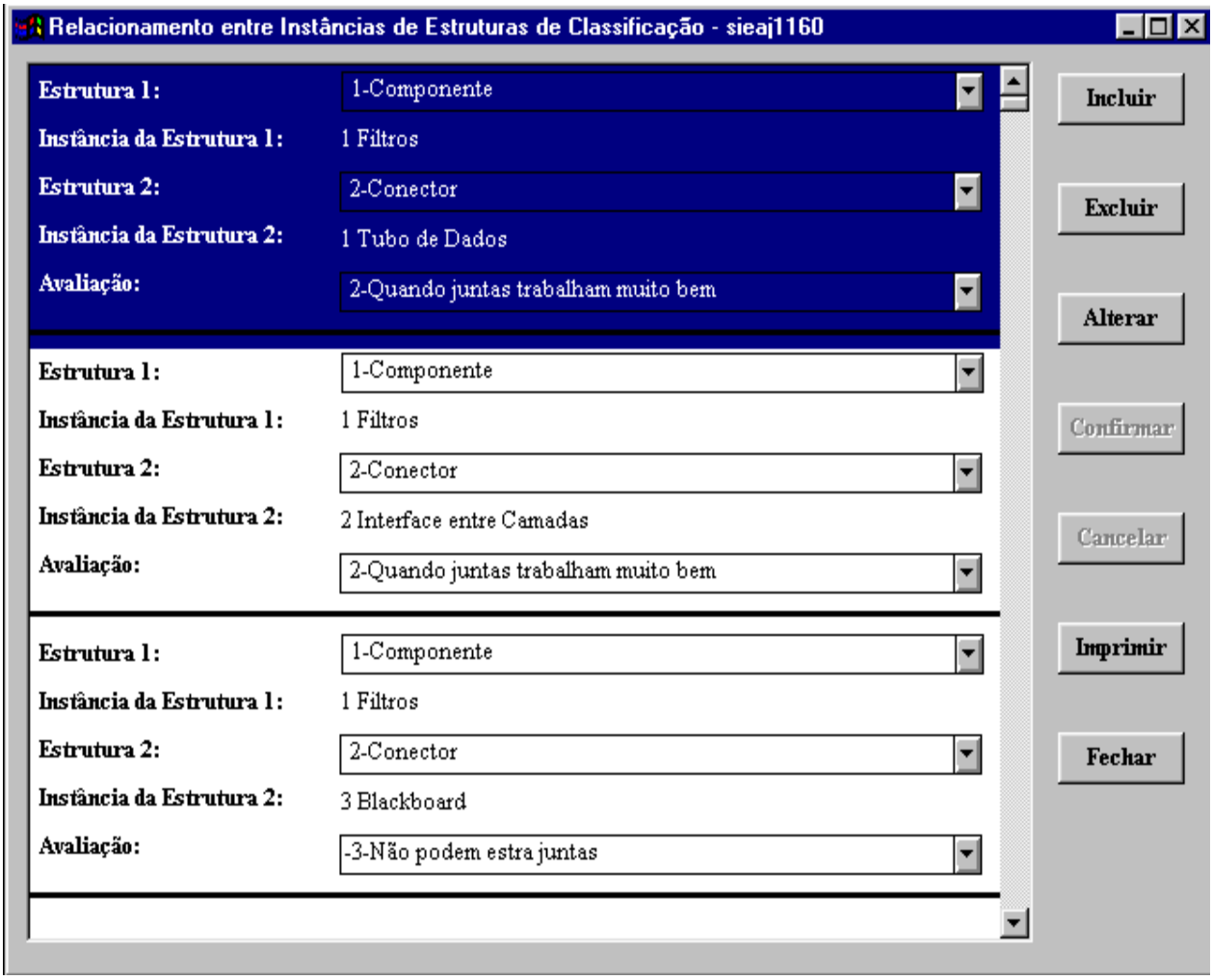

Figura 32 - Tela de relacionamento entre instâncias de estruturas de classificação

4.3.2 FASE 2 – Montagem dos Requisitos do Sistema em Estudo

Após termos toda a base de conhecimento armazenada em tabelas MS-Access, partiremos para a segunda fase do método de identificação de estilos de arquiteturas. Nesta segunda fase estaremos cadastrando um sistema em estudo, com seus respectivos requisitos. Cada requisito do sistema em estudo deve estar associado a um requisito de qualidade geral existente na base de conhecimento.

A tela da Figura 33, ativada pelo menu *Sistema em Estudo* - Figura 23, apresenta o local onde iniciamos o cadastramento do sistema em estudo, a ser classificado. É nesta tela que informamos o código e o nome do sistema em estudo. A tabela que armazena as informações do sistema em estudo possui os seguintes campos: código e descrição do sistema.

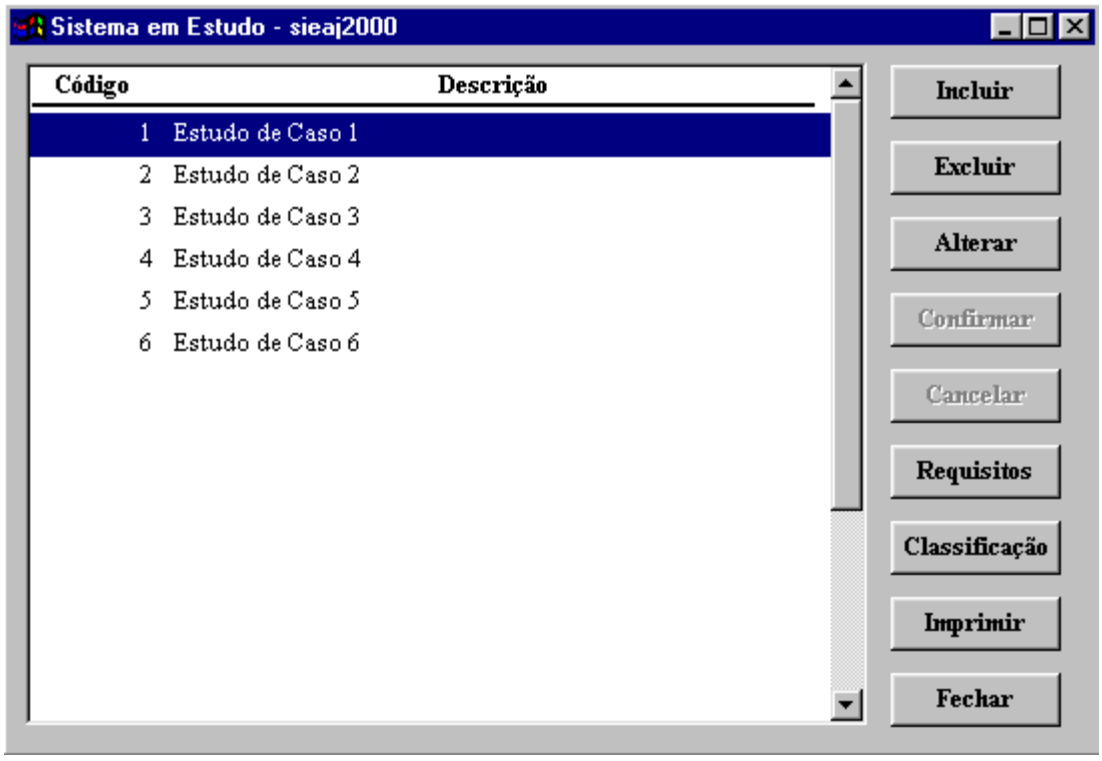

Figura 33 – Tela de cadastramento do sistema em estudo

A Figura 34 apresenta a tela onde cadastramos os requisitos do sistema em estudo. Para cada requisito do sistema em estudo faremos a leitura da lista dos requisitos de qualidade geral e relacionaremos aquele que acharmos pertinente. É feita uma referência cruzada entre os requisitos de qualidade geral e os requisitos do sistema em estudo. Esta tela é ativada a partir do botão “Requisitos” - Figura 33.

A tabela que armazena os requisitos do sistema em estudo, bem como as referências com os requisitos de qualidade geral, possui os seguintes campos: código do sistema, código do requisito do sistema, descrição do requisito do sistema e código do requisito de qualidade geral associado.

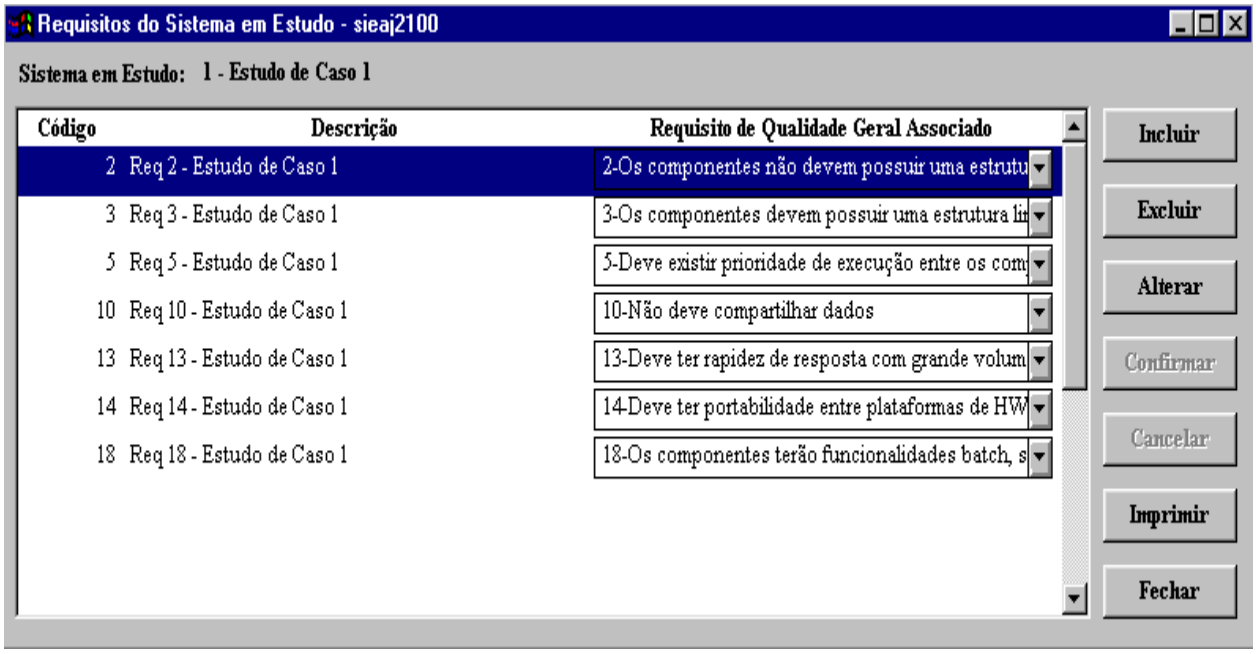

Figura 34 – Tela de cadastramento dos requisitos do sistema em estudo, relacionando com os requisitos de qualidade geral

4.3.3 FASE 3 – Método de Contagem

Após o cadastramento do sistema em estudo e a seleção dos requisitos de qualidade geral, o SIEA está pronto para fazer a contagem dos pontos, a partir do método de contagem definido.

O método de contagem inicia com um programa Power Builder montando o programa CLIPS – conforme descrito anteriormente; e termina com a apresentação do resultado do método de contagem, em tela. Os resultados alcançados por cada estilo de arquitetura são apresentados em ordem decrescente de pontuação. Ou seja, o estilo de arquitetura que consta no topo da lista é o estilo recomendado.

A Figura 35 apresenta o estilo de arquitetura camada para a implementação do método de contagem. A recomendação do estilo de arquitetura camada encontra-se detalhada no estudo de caso 6 do capítulo 5.

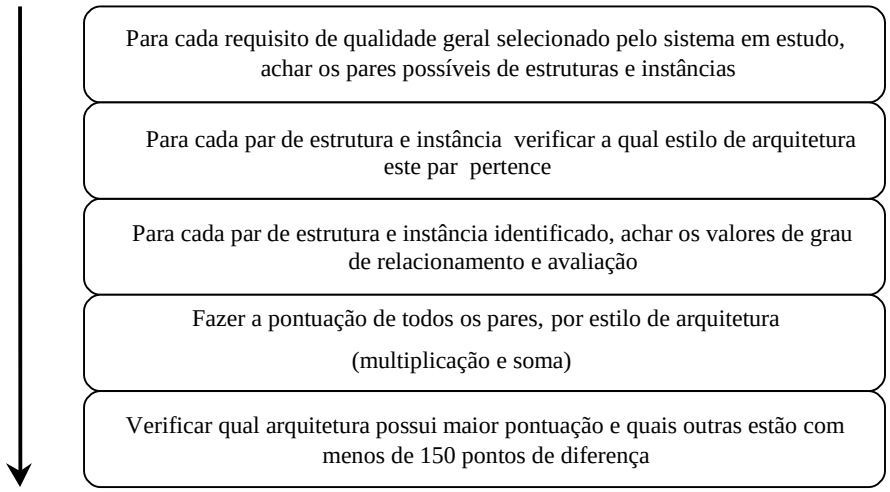

Figura 35 – Arquitetura do método de contagem

O método de contagem é ativado a partir de uma tela específica, conforme ilustra a Figura 36, a qual é ativada a partir da tela da Figura 33 – botão “Classificação”.

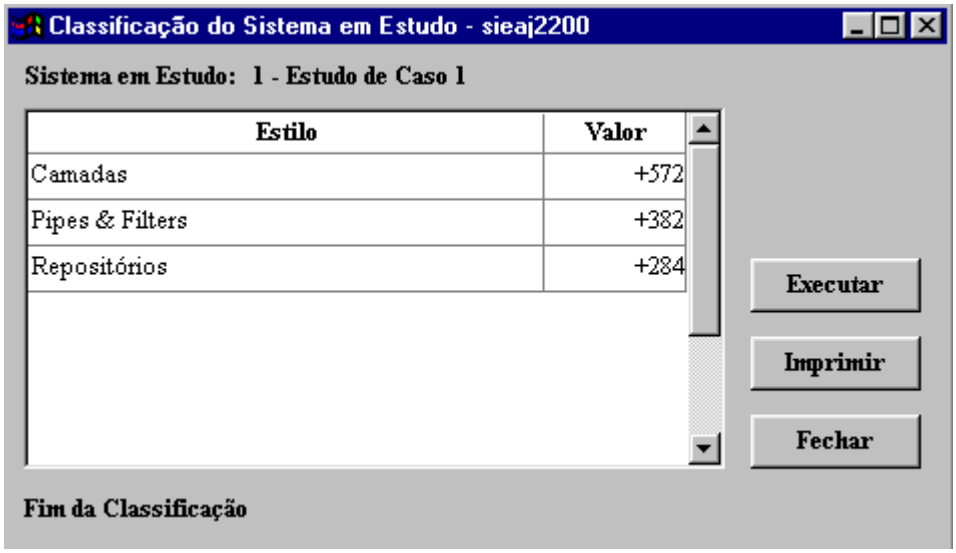

Figura 36 – Tela de contagem de pontos do sistema em avaliação

Isso posto, percorremos a forma de implementar as três fases do método de identificação de estilos de arquiteturas: montamos nossa base de conhecimento, cadastramos um sistema a ser estudado e submetemos este ao método de contagem.

Capítulo 5 – Estudos de Caso

Este capítulo tem o objetivo de apresentar dez estudos de caso desenvolvidos com o propósito de consolidar a base de conhecimento e o método de contagem. Esses mesmos estudos de caso nos ajudaram no teste integrado do SIEA. Neste capítulo apresentamos uma breve descrição dos estudos de caso, o resultado do método de contagem, bem como nossas conclusões. Estes mesmos estudos de caso são apresentados em mais detalhes no anexo C. A seguir, apresentamos uma breve descrição dos dez estudos de caso.

O primeiro estudo de caso retrata um sistema leitor de código de barras em um armazém. O segundo estudo de caso é sobre o mesmo sistema, porém com variações de alguns requisitos, motivados pela baixa performance do sistema implementado no primeiro estudo de caso, o que vai tornar importante a escolha da arquitetura.

O terceiro estudo de caso é sobre a construção de uma biblioteca de componentes reutilizáveis sob a óptica de um ambiente de tecnologia dos anos 80. O quarto estudo de caso é sobre o mesmo sistema, porém baseado nos novos aspectos de reutilização em multiplataformas, JAVA Beans, entre outros.

O quinto e sexto estudos de caso são sobre o SIEA já citado anteriormente. Analisaremos duas partes distintas do SIEA, devido às suas características totalmente independentes: base de conhecimento e método de contagem.

Os demais estudos de caso foram retirados de [Buschmann+98]. Foi feita uma análise comparativa do resultado do método proposto em relação a [Buschmann+98].

Para cada estudo de caso apresentamos: a descrição do sistema em estudo, identificamos os requisitos do sistema em estudo, relacionamos tais requisitos com os requisitos de qualidade

geral, apresentamos os resultado do método de contagem e por fim, faremos algumas observações, explicando os resultados obtidos.

A seguir apresentamos, de forma gráfica, os passos seguidos em cada estudo de caso, bem como referenciamos algumas figuras / telas do capítulo 4, com o intuito de exemplificar a forma como estaremos utilizando o SIEA para a execução dos referidos estudos de caso.

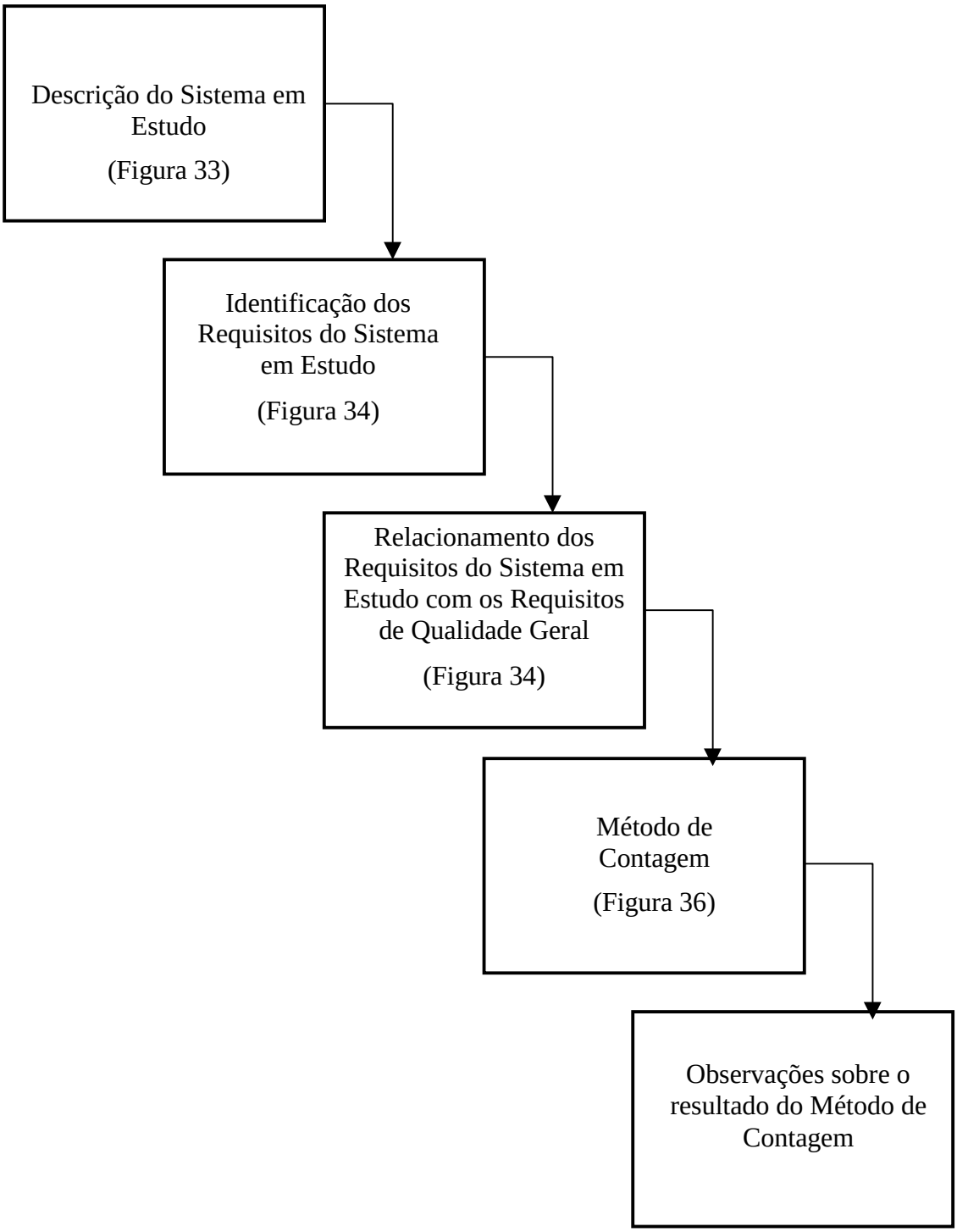

Figura 37 – Análise dos Estudos de Caso

5.1 ESTUDO DE CASO 1

Descrição do Sistema em Estudo:

O sistema do estudo de caso 1 consiste de um leitor móvel de código de barras que lê etiquetas em um armazém. Após leitura, os dados são transferidos para o disco rígido de um microcomputador. Neste momento os dados são criticados de tal forma que um programa lê os dados do disco rígido e após uma série de críticas produz duas saídas. A primeira saída é um relatório de erros de dados inválidos, e a segunda é a gravação dos dados válidos em um banco de dados relacional (Sybase, versão 11).

No referido banco de dados, as informações devem ser gravadas seqüencialmente em tabelas de três sistemas a saber: Estoque, Contabilidade e Livros Fiscais. Por razões operacionais, a ordem da seqüência de gravação é obrigatória.

Submetendo este sistema ao método de identificação de estilos de arquiteturas, teremos:

Resultado do Método de Contagem:

	Tubos e Filtros	**Camada**	**Repositórios**	**Orientada a Objetos**	**Programa Principal / Subrotinas**
Tot	382	571	283	452	445

Tabela 19 – Contagem de pontos do estudo de caso 1

Observações e Resultados:

O estilo camada é o mais indicado devido as suas fortes características de processamento seqüencial. O estilo orientado a objetos também alcançou pontuação significativa, o que nos leva a implementar cada camada como um objeto.

Detalhes sobre este estudo de caso podem ser encontrados no anexo C desta dissertação.

5.2 ESTUDO DE CASO 2

Descrição do Sistema em Estudo:

O estudo de caso 2, é um problema idêntico ao apresentado no estudo de caso 1, porém algumas mudanças devem ser observadas, devido a baixa performance ocorrida após a implementação do sistema do estudo de caso 1.

Não existe mais a exigência da sequencialidade de gravação nas bases de dados de Estoque, Contabilidade e Livros Fiscais. Podemos interpretar o disco rígido do microcomputador como uma base de dados que conforme sua mudança de estado dispara processos para a inclusão de dados nas tabelas de Estoque, Contabilidade e Livros Fiscais. Com esta arquitetura, poderemos tirar proveito do processamento paralelo.

Submetendo estes novos requisitos ao método de identificação de estilos de arquiteturas e mantendo as demais considerações do estudo de caso 1, teremos:

Resultado do Método de Contagem:

	Tubos e Filtros	**Camada**	**Repositórios**	**Orientada a Objetos**	**Programa Principal / Subrotinas**
Tot	198	125	618	760	3

Tabela 20 – Contagem de pontos do estudo de caso 2

Observações e Resultados:

Para a obtenção de maior velocidade no sistema proposto, três características novas surgiram: perda da sequencialidade, processamento paralelo e compartilhamento dos dados. Com isso, os

estilos tubos e filtros, camada e programa principal / subrotinas, tornaram-se totalmente inadequados.

A característica descrita, ou seja, o estado de mudança de uma base de dados causar disparo de ações, levou o método de contagem a pontuar o estilo repositório.

O método recomenda o estilo orientado a objetos, sendo cada objeto uma *fonte de informação* ligada ao repositório (disco rígido do microcomputador).

Detalhes sobre este estudo de caso podem ser encontrados no anexo C desta dissertação.

5.3 ESTUDO DE CASO 3

Descrição do Sistema em Estudo:

Uma firma de desenvolvimento de software, a mais de 20 anos atrás, teve a visão de desenvolver uma biblioteca de componentes reutilizáveis. Tendo como ponto de partida a tecnologia comercial de 20 anos passados.

Basicamente o ambiente de programação era *mainframe* IBM, com COBOL e IMS.

Submetendo este sistema ao método de identificação de estilos de arquiteturas teremos:

Resultado do Método de Contagem:

	Tubos e Filtros	**Camada**	**Repositórios**	**Orientada a Objetos**	**Programa Principal / Subrotinas**
Tot	202	287	374	640	0

Tabela 21 – Contagem de pontos do estudo de caso 3

Observações e Resultados:

Para o problema apresentado, reusabilidade, o método recomenda o estilo orientado a objetos. O referido estilo possui fortes características de reuso, o que não significa haver tecnologia disponível para tais práticas. O estilo predominante da época, programa principal / subrotinas, não apresentou pontuação em um problema típico de reusabilidade. O resultado desta pontuação pode ser refletido na dificuldade de se reaproveitar código em sistemas antigos – não levando em consideração problemas de manutenção que ocorrem no decorre da vida de um sistema.

Detalhes sobre este estudo de caso podem ser encontrados no anexo C desta dissertação.

5.4 ESTUDO DE CASO 4

Descrição do Sistema em Estudo:

Após passada a experiência do estudo de caso 3, a mesma firma propôs no final dos anos 90 um projeto semelhante, porém em multiplataforma, com facilidades de processamento paralelo, desenvolvimento por componentes, frameworks, utilizando o padrão CORBA – IDL, para as práticas de reuso.

Submetendo estes novos requisitos ao método de identificação de estilos de arquiteturas e mantendo as demais características, teremos:

Resultado do Método de Contagem:

	Tubos e Filtros	**Camada**	**Repositórios**	**Orientada a Objetos**	**Programa Principal / Subrotinas**
Tot	321	411	448	767	1

Tabela 22 – Contagem de pontos do estudo de caso 4

Observações e Resultados:

A classificação continuou exatamente a mesma, pelos mesmos motivos. Desta forma, o método sugere o estilo orientado a objetos como solução do problema proposto.

Detalhes sobre este estudo de caso podem ser encontrados no anexo C desta dissertação.

5.5 ESTUDO DE CASO 5

Descrição do Sistema em Estudo:

Fazendo uma auto-análise do método de identificação de estilos de arquiteturas proposto no SIEA, analisaremos duas partes a saber. A primeira é a representação do conhecimento de estilos de arquitetura, o que sugere a forma de repositório, pois possui características de um problema de Inteligência Artificial (IA). A segunda parte do software proposto é o método de contagem de pontos, que será tratado no estudo de caso 6.

Submetendo a parte de representação do conhecimento do SIEA ao método de identificação de estilos de arquiteturas, teremos:

Resultado do Método de Contagem:

	Tubos e Filtros	**Camada**	**Repositórios**	**Orientada a Objetos**	**Programa Principal / Subrotinas**
Tot	202	125	544	690	2

Tabela 23 – Contagem de pontos do estudo de caso 5

Observações e Resultados:

O resultado propõem a construção do referido software com arquitetura mista: orientada a objetos e repositórios. Ou seja, cada *fonte de informação* é um objeto que se comunica com o repositório. O repositório é o local onde temos as redes semânticas que refletem o conhecimento sobre estruturas de classificação, instâncias, relacionamentos e estilos.

Detalhes sobre este estudo de caso podem ser encontrados no anexo C desta dissertação.

5.6 ESTUDO DE CASO 6

Descrição do Sistema em Estudo:

O estudo de caso 6, conforme dito anteriormente, será uma auto-análise do método de contagem do SIEA. Conforme descrito no capítulo 3, o método de contagem possui, basicamente, características de ações seqüenciais.

Submetendo este sistema ao método de identificação de estilos de arquiteturas, teremos:

Resultado do Método de Contagem:

	Tubos e Filtros	**Camada**	**Repositórios**	**Orientada a Objetos**	**Programa Principal / Subrotinas**
Tot	328	510	385	456	371

Tabela 24 – Contagem de pontos do estudo de caso 6

Observações e Resultados:

Observamos que o estilo camada é o mais favorecido. O motivo do resultado desta contagem é semelhante ao ocorrido no estudo de caso 1. Ou seja, a obrigatoriedade de existirem processos seqüenciais favoreceu o estilo camada. Sendo o referido estilo também forte nas características de reuso, assim como o estilo orientada a objetos, que obteve pontuação próxima.

Utilizaremos os estilos camada e orientado a objetos. Cada camada será tratada como um objeto.

Detalhes sobre este estudo de caso podem ser encontrados no anexo C desta dissertação.

5.7 ESTUDO DE CASO 7

Descrição do Sistema em Estudo:

O estudo de caso 7, será uma análise da API (*Application Programming Interface*) sugerida em [Buschmann+98] como camada. As API's devem prover portabilidade entre os sistemas operacionais – páginas: 46 - 47.

Submetendo este sistema ao método de identificação de estilos de arquiteturas, teremos:

Resultado do Método de Contagem:

	Tubos e Filtros	**Camada**	**Repositórios**	**Orientada a Objetos**	**Programa Principal / Subrotinas**
Tot	372	1.154	137	307	645

Tabela 25 – Contagem de pontos do estudo de caso 7

Observações e Resultados:

Assim como em [Buschmann+98], chegamos aos resultados que indicam a arquitetura camada como recomendada. Influenciou no resultado, a favor de camada, a estrutura hierárquica dos componentes, bem como as capacidades de portabilidade e reuso dos mesmos.

Detalhes sobre este estudo de caso podem ser encontrados no anexo C desta dissertação.

5.8 ESTUDO DE CASO 8

Descrição do Sistema em Estudo:

O estudo de caso 8, será uma análise da linguagem / compilador Mocha (*Modular Object Computation with Hypothetical Algorithms*) sugerida em [Buschmann+98] como tubos e filtros – páginas: 53 - 66.

O objetivo é construir um software que “compile”, “linkedite” e gere o executável a partir de um programa fonte em Mocha. Deve ser um compilador com portabilidade entre diversas plataformas de hardware.

Submetendo este sistema ao método de identificação de estilos de arquiteturas, teremos:

Resultado do Método de Contagem:

	Tubos e Filtros	**Camada**	**Repositórios**	**Orientada a Objetos**	**Programa Principal / Subrotinas**
Tot	350	435	221	344	295

Tabela 26 – Contagem de pontos do estudo de caso 8

Observações e Resultados:

Em [Buschmann+98] foi sugerida a arquitetura tubos e filtros. Nosso método propôs uma combinação de camada e tubos e filtros. Tubos e filtros foi indicada principalmente pelas suas características de estrutura linear e portabilidade, características bastante pertinentes ao compilador em questão, conforme seção *implementação* de [Buschmann+98].

A arquitetura camada surgiu com maior pontuação e com as mesmas características de tubos e filtros. Sua alta pontuação destacou-se basicamente na estrutura linear, influenciada pela

linguagem intermediária e pela máquina virtual, citadas em [Buschmann+98]. Desta forma, consideramos nossos resultados compatíveis com [Buschmann+98].

Detalhes sobre este estudo de caso podem ser encontrados no anexo C desta dissertação.

5.9 ESTUDO DE CASO 9

Descrição do Sistema em Estudo:

O estudo de caso 9, será uma análise do modelo OSI[5] – 7 Camadas, conforme apresentado em [Buschmann+98], que o classificou como uma arquitetura camada, típica – páginas: 31 - 46.

Submetendo este sistema ao método de identificação de estilos de arquiteturas, teremos:

Resultado do Método de Contagem:

	Tubos e Filtros	**Camada**	**Repositórios**	**Orientada a Objetos**	**Programa Principal / Subrotinas**
Tot	430	1.215	130	351	719

Tabela 27 – Contagem de pontos do estudo de caso 9

Observações e Resultados:

Em [Buschmann+98] foi sugerida a arquitetura camada. Nosso método indicou o mesmo. Tal pontuação ocorreu pelas mesmas características sugeridas em [Buschmann+98]: sequencialidade, independência das camadas e boa definição de interfaces entre as camadas.

Detalhes sobre este estudo de caso podem ser encontrados no anexo C desta dissertação.

[5] International Standardization Organization

5.10 ESTUDO DE CASO 10

Descrição do Sistema em Estudo:

O estudo de caso 10, será uma análise do software de reconhecimento de palavras sugerido em [Buschmann+98] como repositórios – páginas: 71 - 87. A decisão do livro [Buschmann+98] baseou-se no fato de ser um software típico de Inteligência Artificial (IA) e como tal, indica o estilo repositório como solução.

Submetendo este sistema ao método de identificação de estilos de arquiteturas, teremos:

Resultado do Método de Contagem:

	Tubos e Filtros	**Camada**	**Repositórios**	**Orientada a Objetos**	**Programa Principal / Subrotinas**
Tot	78	0	476	573	1

Tabela 28 – Contagem de pontos do estudo de caso 10

Observações e Resultados:

Em [Buschmann+98] foi sugerida a arquitetura repositório. Nosso método propôs uma combinação de repositório e orientado a objetos.

Como em [Buschmann+98] o estilo orientado a objetos (OO) não existe – não é considerado como estilo – nos limitamos a observar que OO é uma técnica indicada em soluções de Inteligência Artificial (IA), conforme sugere o ambiente CLIPS (*C Language Integrated Production System*).

Desta forma, consideramos nosso resultado compatível com [Buschmann+98].

Detalhes sobre este estudo de caso podem ser encontrados no anexo C desta dissertação.

5.11 Conclusão dos Estudos de Caso

Apresentamos 10 estudos de caso que ajudaram a exercitar e validar o método de identificação de estilos de arquiteturas.

Os estudos de caso 1 e 2 apresentaram uma comparação de troca de requisitos que deve ser observada com atenção. No estudo de caso 1 havia a necessidade de sequencialidade dos processos, o que influenciou fortemente o estilo camada. No estudo de caso 2 apresentamos o mesmo problema, porém todas as necessidades de sequencialidade foram eliminadas e enfatizamos a melhor performance dos processos. Tal mudança nos requisitos nos apresentou uma situação totalmente inadequada para o estilo camada. Com a ênfase no compartilhamento de dados em uma base central e sendo o estado desta base o estimulador das ações do sistema, nos levou a uma boa pontuação do estilo repositório, sendo implementado a partir de características de orientação a objetos.

Os estudos de caso 3 e 4 apresentaram dois problemas em ambientes tecnológicos bastante diferentes, porém exatamente com os mesmos requisitos. Independente do ambiente tecnológico a solução recomendada pelo método foi a mesma, visto que o problema básico não mudou.

Os estudos de caso 5 e 6 apresentaram duas partes do SIEA. Nos capítulos 2, 3 e 4 já havíamos detalhado como seria a implementação da arquitetura. Nestes estudos de caso nos limitamos a apresentar a contagem efetuada em cada um dos requisitos de qualidade geral.

Os estudos de caso 7, 8, 9 e 10 foram fundamentais para a validação do método proposto, visto que [Buschmann+98] é uma referência sólida, que expõem resultados já sedimentados. Nos estudos de caso relativos a [Buschmann+98] não consideramos as arquiteturas orientada a objetos e programa principal / subrotinas. Não faz parte do escopo desta dissertação a validade destas duas arquiteturas serem, ou não, consideradas como tal.

Capítulo 6 – Conclusão

6.1 Conclusões

O trabalho desenvolvido para a elaboração desta dissertação apresentou uma proposta de taxonomia para os diversos estilos de arquitetura. A taxonomia foi elaborada a partir da análise dos componentes, conectores e de elementos de LDA's. Diversos trabalhos de classificação foram consultados, porém nenhum destes apresentou o propósito de uma única taxonomia para todos os estilos [Abd-Allah+95] [ACM+95.2] [Bass+98] [Gacek+98.2] [Garlan+93] [Shaw+96.1]. Representamos a taxonomia em uma base de conhecimento, a partir de redes semânticas. Propomos ainda um grupo de heurísticas para a aplicação desta taxonomia com a intenção de indicar a melhor arquitetura para o grupo de requisitos de um sistema em estudo. Tais heurísticas formaram um método de contagem, implementado por um sistema de produção, integrado a rede semântica. O método proposto se destaca em três pontos a saber:

- Prover uma sugestão de arquitetura nos primeiros momentos de construção de um software, a partir da análise dos requisitos de qualidade geral. Com isso, o desenvolvedor teria a chance de explorar as várias características da arquitetura indicada.
- A partir da sugestão da arquitetura e da base de conhecimento, o método sugere elementos faltantes no sistema em estudo.
- Ter uma base de conhecimento com uma estrutura bastante flexível para a inserção de conhecimentos futuros. Acreditamos que o aumento desta base, com novas arquiteturas e

novos estudos de caso, não irá alterar a taxonomia proposta para todos os estilos. O engenheiro de software responsável pela criação da base de conhecimento pode incluir e excluir *estruturas de classificação*, suas *instâncias*, *requisitos de qualidade geral*, bem como alterar os valores de *grau de relacionamento* e *avaliação*.

Fizemos uma avaliação inicial de nosso trabalho a partir de dez estudos de caso, sendo quatro desses já exercitados e aceitos pela comunidade científica [Buschmann+98]. Os quatro últimos estudos de caso apresentaram as mesmas classificações de [Buschmann+98], conforme as considerações do método. Concentramos nossos estudos de caso nos estilos: tubos e filtros, camadas, repositórios, orientado a objetos e programa principal / subrotinas.

Um software foi construído para automatizar a base de conhecimento e o método de contagem. Este software tem sua arquitetura baseada na recomendação feita pelo próprio método – quinto e sexto estudos de caso. Para a construção do software usamos técnicas de orientação a objeto e Inteligência Artificial. Para conjugar estes dois paradigmas, usamos o ambiente CLIPS (*C Language Integrated Production System*) que prevê a coexistência de ambos.

6.2 Contribuições Esperadas

O processo proposto possibilita o reuso de arquiteturas, como também uma análise dos requisitos, face aos padrões arquiteturais. Desta forma, a presente dissertação pretende ser um passo inicial para unir a análise de requisitos a arquitetura de software. Neste propósito e visando o reuso de arquiteturas, a partir dos requisitos, criamos uma taxonomia para a representação de todos os estilos arquitetônicos aqui estudados.

Com a taxonomia proposta, passamos a disponibilizar uma base de conhecimento. Tal base pode ser expandida e corrigida. Expandida, no sentido que podemos incluir novas estruturas de classificação, bem como todos os seus relacionamentos – *grau de relacionamento* e *avaliação*. Corrigida, no sentido que tudo que existe na referida base de conhecimento pode ser alterado ou excluído.

O método de identificação de estilos de arquiteturas fundamenta-se em uma estratégia numérica baseada em contagem de pontos, semelhante às empregadas na seleção de ferramentas de software. Tal método apresentou resultados satisfatórios, conforme apresentado nos estudos de caso. A medida que novos estudos de caso forem elaborados, o método poderá ser revisto, bem como a base de conhecimento.

A construção e utilização do SIEA possibilitará a formação de mais uma parte da base de conhecimento. Isto é, teremos cada vez mais estudos de caso para observar quais conjuntos de requisitos de qualidade geral, nos levam a determinados estilos de arquitetura.

Isso posto, acreditamos que o processo proposto é original e contribui para a sedimentação das práticas centradas na idéia de arquitetura de software.

6.3 Trabalhos Futuros

Acreditamos que os próximos passos de nossa pesquisa são: a criação de novos estudos de caso e a construção de outro método, o qual analisa os resultados gerados pelo método de contagem proposto.

Os futuros estudos de caso deverão ter complexidade maior que os propostos até o momento e já deverão ter sido classificados por alguma literatura amplamente reconhecida, tipo [Buschmann+98]. Acreditamos que a partir de resultados já aceitos pela comunidade científica, nossos estudos de caso terão boa aceitação.

O novo método a ser desenvolvido deverá analisar, a partir de resultados do SIEA, quais grupos de requisitos de qualidade geral, freqüentemente aparecem em um estilo arquitetônico indicado. Do resultado desta análise nós poderemos aumentar a base de conhecimento, com níveis de primitivas mais elevadas.

Em um passo mais adiante, estaríamos relacionando requisitos de qualidade geral a componentes pré-programados, apenas com a necessidade de instanciação. Os referidos componentes seriam elementos típicos das respectivas arquiteturas.

A base de conhecimento proposta, bem como os resultados dos estudos de caso, pretendem ter papel semelhante aos “bancos de hora” hoje existente para análise de ponto de função. Ou seja, dado um conjunto de requisitos de qualidade geral selecionados, quais componentes serão pertinentes para a resolução do problema em questão.

Isso posto, esperamos ter apresentado um trabalho fundamentado na literatura existente sobre arquitetura de software, bem como termos apresentado um método que ajudará no reuso de software.

ANEXO A – Relacionamentos entre Estruturas de Classificação e respectivas Instâncias

Este anexo destina-se a apresentar as tabelas que armazenam os relacionamentos de todas as estruturas de classificação, bem como suas respectivas instâncias.

O formato de representação em tabelas tem o objetivo de simplificar a visualização, uma vez que seu formato final de implementação é por rede semântica.

	Grau de Relacionamento		1 - Componentes									
		Instâncias	Filtros		Camadas		KS		Objetos		Programa / Subrotinas	
Estruturas de Classificação			Avaliação		Avaliação		Avaliação		Avaliação		Avaliação	
2 – Conectores	4	Tubo de dados	2	8	1	4	2	8	-3	-12	2	8
	4	Interface entre camadas	2	8	2	8	2	8	-3	-12	2	8
	4	Blackboard	-3	-12	-3	-12	2	8	-3	-12	2	8
	4	Mensagens entre objetos	1	4	2	8	2	8	3	12	2	8
	4	Passagem de variáveis	1	4	2	8	2	8	-3	-12	2	8
3 – Seqüência de comunicação	4	Seqüencial por componente	2	8	3	12	-2	-8	2	8	3	12
	4	Randômica entre componentes	2	8	-3	-12	2	8	2	8	-3	-12
4 – Modo de execução	4	Seqüencial	2	8	3	12	2	8	2	8	3	12
	4	Paralelo	2	8	-3	-12	2	8	2	8	-2	-8
5 – Sincronismo	4	Linear	2	8	3	12	-2	-8	2	8	3	12
	4	Não-linear	2	8	-3	-12	2	8	2	8	-2	-8
6 – Encapsulamento de componentes	4	Sempre (sem conhecimento das interfaces)	3	12	2	8	2	8	2	8	-2	-8
	4	Sempre (com conhecimento das interfaces)	-3	-12	2	8	-2	-8	2	8	2	8
	4	Não existe encapsulamento	-3	-12	-2	-8	-2	-8	-3	-12	2	8
7 - Comunicação entre componentes	4	Explicita (envio de dados)	2	8	2	8	-1	-4	-3	-12	2	8
	4	Chamada de funções	2	8	2	8	1	4	-3	-12	2	8
	4	Mecanismos de callback	-3	-12	2	8	-2	-8	-3	-12	2	8

	4	Compartilhamento de dados	-3	-12	1	4	2	8	-2	-8	2	8
	4	Explicita (envio de mensagens)	2	8	2	8	1	4	2	8	2	8
8 - Reuso	4	Forte por componente	3	12	3	12	3	12	3	12	-2	-8
	4	Ausente	-3	-12	-3	-12	1	4	-3	-12	2	8
9 - Mecanismo de feedback	4	Presente (cíclico)	2	8	-3	-12	-3	-12	2	8	2	8
	4	Presente (acíclico)	2	8	1	4	1	4	2	8	2	8
10 - Prioridade de execução entre componentes	4	Existe sempre	2	8	3	12	2	8	2	8	3	12
	4	Pode existir	2	8	-3	-12	2	8	2	8	-3	-12
11 - Topologia	4	Linear	2	8	2	8	-3	-12	2	8	1	4
	4	Hierárquica	1	4	2	8	-3	-12	2	8	1	4
	4	Estrelar	-1	-4	-3	-12	3	12	2	8	-1	-4
	4	Árvore	1	4	-3	-12	-3	-12	2	8	2	8
	4	Não há forma	-3	-12	-3	-12	-3	-12	2	8	-3	-12
12 - Modo de dados	3	Passados	3	9	3	9	-3	-9	2	6	2	6
	3	Compartilhados	-3	-9	1	3	3	9	-2	-6	2	6
13 - Organização de processamento	4	Batch	2	8	2	8	2	8	2	8	2	8
	4	Interativo	2	8	2	8	2	8	2	8	2	8

Tabela 29 – Relacionamento da estrutura de classificação componentes com as demais estruturas

	Grau de Relacionamento		2 - Conectores									
		Instâncias	Tubo de dados		Interface entre camadas		Blackboard		Mensagens entre objetos		Passagem de variáveis	
Estruturas de Classificação			Avaliação		Avaliação		Avaliação		Avaliação		Avaliação	
3 – Seqüência de comunicação	4	Seqüencial por componente	2	8	3	12	-2	-8	2	8	2	8
	4	Randômica entre componentes	2	8	-3	-12	2	8	2	8	2	8
4 – Modo de execução	2	Seqüencial	2	4	3	6	-2	-4	2	4	2	4
	2	Paralelo	2	4	-3	-6	2	4	2	4	2	4
5 – Sincronismo	2	Linear	2	4	3	6	-2	-4	2	4	2	4
	2	Não-linear	2	4	-3	-6	2	4	2	4	2	4
6 – Encapsulamento de componentes	4	Sempre (sem conhecimento das interfaces)	2	8	-3	-12	2	8	2	8	-3	-12
	4	Sempre (com conhecimento das interfaces)	2	8	3	12	1	4	1	4	2	8
	4	Não existe encapsulamento	2	8	-3	-12	1	4	-3	-12	2	8
7 - Comunicação entre componentes	4	Explicita (envio de dados)	2	8	1	4	1	4	2	8	2	8
	4	Chamada de funções	2	8	1	4	2	8	-3	-12	2	8
	4	Mecanismos de callback	2	8	1	4	2	8	-3	-12	2	8
	4	Compartilhamento de dados	2	8	2	8	2	8	1	4	2	8
	4	Explicita (envio de mensagens)	2	8	2	8	2	8	2	8	2	8
8 - Reuso	4	Forte por componente	2	8	3	12	2	8	3	12	-2	-8
	4	Ausente	2	8	-2	-8	2	8	-3	-12	2	8
9 - Mecanismo de feedback	4	Presente (cíclico)	2	8	-3	-12	-2	-8	2	8	2	8
	4	Presente (acíclico)	2	8	1	4	1	4	2	8	2	8
10 - Prioridade de execução entre componentes	0	Existe sempre	0	0	0	0	0	0	0	0	0	0
	0	Pode existir	0	0	0	0	0	0	0	0	0	0
11 - Topologia	4	Linear	2	8	2	8	-3	-12	2	8	2	8
	4	Hierárquica	2	8	2	8	-3	-12	2	8	2	8

	4	Estrelar	2	8	-3	-12	3	12	2	8	2	8
	4	Árvore	2	8	-2	-8	-3	-12	2	8	2	8
	4	Não há forma	2	8	-3	-12	-3	-12	2	8	2	8
12 - Modo de dados	4	Passados	2	8	3	12	-2	-8	2	8	2	8
	4	Compartilhados	2	8	2	8	2	8	1	4	2	8
13 - Organização de processamento	0	Batch	0	0	0	0	0	0	0	0	0	0
	0	Interativo	0	0	0	0	0	0	0	0	0	0

Tabela 30 – Relacionamento da estrutura de classificação conectores com as demais estruturas

	Grau de Relacionamento		3 - Seqüência de comunicação			
		Instâncias	Seqüencial por componente		Randômica entre componentes	
Estruturas de Classificação			Avaliação		Avaliação	
4 – Modo de execução	4	Seqüencial	3	12	-2	-8
	4	Paralelo	-3	-12	2	8
5 – Sincronismo	4	Linear	3	12	-3	-12
	4	Não-linear	-3	-12	3	12
6 – Encapsulamento de componentes	0	Sempre (sem conhecimento das interfaces)	0	0	0	0
	0	Sempre (com conhecimento das interfaces)	0	0	0	0
	0	Não existe encapsulamento	0	0	0	0
7 - Comunicação entre componentes	0	Explicita (envio de dados)	0	0	0	0
	0	Chamada de funções	0	0	0	0
	0	Mecanismos de callback	0	0	0	0
	0	Compartilhamento de dados	0	0	0	0
	0	Explicita (envio de mensagens)	0	0	0	0
8 – Reuso	4	Forte por componente	-1	-4	2	8
	4	Ausente	2	8	2	8
9 - Mecanismo de feedback	0	Presente (cíclico)	0	0	0	0
	0	Presente (acíclico)	0	0	0	0
10 - Prioridade de execução entre componentes	4	Existe sempre	3	12	-1	-4
	4	Pode existir	-3	-12	2	8
11 - Topologia	1	Linear	2	2	-3	-3
	1	Hierárquica	2	2	2	2
	1	Estrelar	-3	-3	2	2

	1	Árvore	2	2	2	2
	1	Não há forma	-3	-3	2	2
12 - Modo de dados	0	Passados	0	0	0	0
	0	Compartilhados	0	0	0	0
13 - Organização de processamento	0	Batch	0	0	0	0
	0	Interativo	0	0	0	0

Tabela 31 – Relacionamento da estrutura de classificação seqüência de comunicação com as demais estruturas

	Grau de Relacionamento		4 – Modo de execução			
		Instâncias	Seqüencial		Paralelo	
Estruturas de Classificação			Avaliação		Avaliação	
5 – Sincronismo	4	Linear	3	12	-2	-8
	4	Não-linear	-3	-12	2	8
6 – Encapsulamento de componentes	0	Sempre (sem conhecimento das interfaces)	0	0	0	0
	0	Sempre (com conhecimento das interfaces)	0	0	0	0
	0	Não existe encapsulamento	0	0	0	0
7 - Comunicação entre componentes	0	Explicita (envio de dados)	0	0	0	0
	0	Chamada de funções	0	0	0	0
	0	Mecanismos de callback	0	0	0	0
	0	Compartilhamento de dados	0	0	0	0
	0	Explicita (envio de mensagens)	0	0	0	0
8 – Reuso	4	Forte por componente	-1	-4	2	8
	4	Ausente	2	8	2	8
9 - Mecanismo de feedback	0	Presente (cíclico)	0	0	0	0
	0	Presente (acíclico)	0	0	0	0
10 - Prioridade de execução entre componentes	4	Existe sempre	3	12	-2	-8
	4	Pode existir	-3	-12	2	8
11 - Topologia	0	Linear	0	0	0	0
	0	Hierárquica	0	0	0	0
	0	Estrelar	0	0	0	0
	0	Árvore	0	0	0	0
	0	Não há forma	0	0	0	0

12 - Modo de dados	0	Passados	0	0	0	0
	0	Compartilhados	0	0	0	0
13 - Organização de processamento	0	Batch	0	0	0	0
	0	Interativo	0	0	0	0

Tabela 32 – Relacionamento da estrutura de classificação modo de execução com as demais estruturas

	Grau de Relacionamento		5 – Sincronismo			
		Instâncias	Linear		Não-linear	
Estruturas de Classificação			Avaliação		Avaliação	
6 – Encapsulamento de componentes	0	Sempre (sem conhecimento das interfaces)	0	0	0	0
	0	Sempre (com conhecimento das interfaces)	0	0	0	0
	0	Não existe encapsulamento	0	0	0	0
7 - Comunicação entre componentes	0	Explicita (envio de dados)	0	0	0	0
	0	Chamada de funções	0	0	0	0
	0	Mecanismos de callback	0	0	0	0
	0	Compartilhamento de dados	0	0	0	0
	0	Explicita (envio de mensagens)	0	0	0	0
8 - Reuso	4	Forte por componente	-1	-4	2	8
	4	Ausente	2	8	2	8
9 - Mecanismo de feedback	0	Presente (cíclico)	0	0	0	0
	0	Presente (acíclico)	0	0	0	0
10 – Prioridade de execução entre componentes	4	Existe sempre	3	12	-1	-4
	4	Pode existir	-3	-12	2	8
11 - Topologia	1	Linear	2	2	1	1
	1	Hierárquica	2	2	1	1
	1	Estrelar	-3	-3	2	2
	1	Árvore	2	2	2	2
	1	Não há forma	-3	-3	2	2
12 - Modo de dados	0	Passados	0	0	0	0
	0	Compartilhados	0	0	0	0
13 - Organização de processamento	0	Batch	0	0	0	0
	0	Interativo	0	0	0	0

Tabela 33 – Relacionamento da estrutura de classificação sincronismo com as demais estruturas

	Grau de Relacionamento		6 – Encapsulamento de componentes					
		Instâncias	Sempre (sem conhecimento das interfaces)		Sempre (com conhecimento das interfaces)		Não existe encapsulamento	
Estruturas de Classificação			Avaliação		Avaliação		Avaliação	
7 - Comunicação entre componentes	3	Explicita (envio de dados)	-1	-3	2	6	2	6
	3	Chamada de funções	1	3	1	3	2	6
	3	Mecanismos de callback	1	3	1	3	2	6
	3	Compartilhamento de dados	-3	-9	-3	-9	2	6
	3	Explicita (envio de mensagens)	2	6	2	6	-2	-6
8 - Reuso	4	Forte por componente	3	12	2	8	-3	-12
	4	Ausente	-3	-12	1	4	3	12
9 - Mecanismo de feedback	0	Presente (cíclico)	0	0	0	0	0	0
	0	Presente (acíclico)	0	0	0	0	0	0
10 - Prioridade de execução entre componentes	0	Existe sempre	0	0	0	0	0	0
	0	Pode existir	0	0	0	0	0	0
11 - Topologia	0	Linear	0	0	0	0	0	0
	0	Hierárquica	0	0	0	0	0	0
	0	Estrelar	0	0	0	0	0	0
	0	Árvore	0	0	0	0	0	0
	0	Não há forma	0	0	0	0	0	0
12 - Modo de dados	4	Passados	3	12	2	8	1	4
	4	Compartilhados	-3	-12	-3	-12	2	8
13 - Organização de processamento	0	Batch	0	0	0	0	0	0
	0	Interativo	0	0	0	0	0	0

Tabela 34 – Relacionamento da estrutura de classificação encapsulamento de componentes com as demais estruturas

	Grau de Relacionamento		7 - Comunicação entre componentes									
		Instâncias	Explicita (envio de dados)		Chamada de funções		Mecanismos de callback		Compartilhamento de dados		Explicita (envio de mensagens)	
Estruturas de Classificação			Avaliação		Avaliação		Avaliação		Avaliação		Avaliação	
8 - Reuso	4	Forte por componente	3	12	2	8	2	8	-3	-12	3	12
	4	Ausente	-2	-8	2	8	2	8	3	12	-3	-12
9 - Mecanismo de feedback	0	Presente (cíclico)	0	0	0	0	0	0	0	0	0	0
	0	Presente (acíclico)	0	0	0	0	0	0	0	0	0	0
10 - Prioridade de execução entre componentes	0	Existe sempre	0	0	0	0	0	0	0	0	0	0
	0	Pode existir	0	0	0	0	0	0	0	0	0	0
11 - Topologia	0	Linear	0	0	0	0	0	0	0	0	0	0
	0	Hierárquica	0	0	0	0	0	0	0	0	0	0
	0	Estrelar	0	0	0	0	0	0	0	0	0	0
	0	Árvore	0	0	0	0	0	0	0	0	0	0
	0	Não há forma	0	0	0	0	0	0	0	0	0	0
12 - Modo de dados	4	Passados	3	12	2	8	2	8	-3	-12	3	12
	4	Compartilhados	-3	-12	2	8	2	8	3	12	-3	-12
13 - Organização de processamento	0	Batch	0	0	0	0	0	0	0	0	0	0
	0	Interativo	0	0	0	0	0	0	0	0	0	0

Tabela 35 – Relacionamento da estrutura de classificação comunicação entre componentes com as demais estruturas

	Grau de Relacionamento		8 - Reuso			
		Instâncias	Forte por componente		Ausente	
Estruturas de Classificação			Avaliação		Avaliação	
9 - Mecanismo de feedback	0	Presente (cíclico)	0	0	0	0
	0	Presente (acíclico)	0	0	0	0
10 - Prioridade de execução entre componentes	0	Existe sempre	0	0	0	0
	0	Pode existir	0	0	0	0
11 - Topologia	0	Linear	0	0	0	0
	0	Hierárquica	0	0	0	0
	0	Estrelar	0	0	0	0
	0	Árvore	0	0	0	0
	0	Não há forma	0	0	0	0
12 - Modo de dados	4	Passados	3	12	2	8
	4	Compartilhados	-3	-12	2	8
13 - Organização de processamento	0	Batch	0	0	0	0
	0	Interativo	0	0	0	0

Tabela 36 – Relacionamento da estrutura de classificação reuso com as demais estruturas

	Grau de Relacionamento		9 – Mecanismo de feedback			
		Instâncias	Presente (cíclico)		Presente (acíclico)	
Estruturas de Classificação			Avaliação		Avaliação	
10 - Prioridade de execução entre componentes	0	Existe sempre	0	0	0	0
	0	Pode existir	0	0	0	0
11 - Topologia	2	Linear	2	4	2	4
	2	Hierárquica	-3	-6	2	4
	2	Estrelar	-3	-6	2	4
	2	Árvore	-3	-6	2	4
	2	Não há forma	-2	-4	2	4
12 - Modo de dados	0	Passados	0	0	0	0
	0	Compartilhados	0	0	0	0
13 - Organização de processamento	0	Batch	0	0	0	0
	0	Interativo	0	0	0	0

Tabela 37 – Relacionamento da estrutura de classificação mecanismo de feedback com as demais estruturas

	Grau de Relacionamento	10 - Prioridade de execução entre componentes				
		Instâncias	Existe sempre		Pode existir	
Estruturas de Classificação			Avaliação		Avaliação	
11 - Topologia	1	Linear	3	3	-3	-3
	1	Hierárquica	3	3	-3	-3
	1	Estrelar	-2	-2	-1	-1
	1	Árvore	2	2	2	2
	1	Não há forma	2	2	2	2
12 - Modo de dados	0	Passados	0	0	0	0
	0	Compartilhados	0	0	0	0
13 - Organização de processamento	0	Batch	0	0	0	0
	0	Interativo	0	0	0	0

Tabela 38 – Relacionamento da estrutura de classificação prioridade de execução entre componentes com as demais estruturas

	Grau de Relacionamento		11 - Topologia									
		Instâncias	Linear		Hierárquica		Estrelar		Árvore		Não há forma	
Estruturas de Classificação			Avaliação		Avaliação		Avaliação		Avaliação		Avaliação	
12 - Modo de dados	0	Passados	0	0	0	0	0	0	0	0	0	0
	0	Compartilhados	0	0	0	0	0	0	0	0	0	0
13 - Organização de processamento	0	Batch	0	0	0	0	0	0	0	0	0	0
	0	Interativo	0	0	0	0	0	0	0	0	0	0

Tabela 39 – Relacionamento da estrutura de classificação topologia com as demais estruturas

	Grau de Relacionamento		12 - Modo de dados			
		Instâncias	Passados		Compartilhados	
Estruturas de Classificação			Avaliação		Avaliação	
13 - Organização de processamento	0	Batch	0	0	0	0
	0	Interativo	0	0	0	0

Tabela 40 – Relacionamento da estrutura de classificação modo de dados com as demais estruturas

ANEXO B – Código Power Builder para formar programas CLIPS

Este anexo contém o código Power Builder que monta o programa CLIPS.

```
int      il_vetor [200, 2], il_cdestlarqt_ant, il_ret, il_cdestlarqt
long   ll_num_arq, ll_i, ll_j, ll_k, ll_l, ll_m, ll_cdestrclas1, ll_cdinstestrclas1, &
         ll_cdestrclas2, ll_cdinstestrclas2, ll_vlgraurelc, ll_vlaval
string sl_regra, sl_fato, sl_quebra, sl_quebra_ant, sl_reg, sl_monta_reg, sl_nmestlarqt

SetPointer(Hourglass!)

DELETE FROM classistestd WHERE cdsistestd = :sieaj2000.ii_cdsistestd;
commit;

dw_padrao.Retrieve(sieaj2000.ii_cdsistestd)

FileDelete("result.txt")
FileDelete("pgm01.clp")
ll_num_arq = FileOpen("pgm01.clp", LineMode!, Write!)


// Inicia arquivo CLIPS
st_status.Text = "Iniciando a gravação do programa CLIPS ....."
FileWrite(ll_num_arq, ";;;
============================================================")
FileWrite(ll_num_arq, ";;; ESTE PROGRAMA CLIPS FOI GERADO PELO SIEA 1.0")
```

```
FileWrite(ll_num_arq, ";;;
==========================================================")
FileWrite(ll_num_arq, "(reset)")
FileWrite(ll_num_arq, "(clear)")
FileWrite(ll_num_arq, "(defmodule MAIN)")
FileWrite(ll_num_arq, "(defglobal ")
FileWrite(ll_num_arq, "          ?*cont*       = 0")
FileWrite(ll_num_arq, "          ?*vlgraurelc* = 0")
FileWrite(ll_num_arq, "          ?*vlaval*     = 0")
FileWrite(ll_num_arq, ")")
FileWrite(ll_num_arq, "(open ~"result.txt~" result ~"w~")")


// Carrega as regras de relcestrclas
st_status.Text = "Carregando as regras do relacionamento relcestrclas ....."
dw_trab1.Reset()
dw_trab1.ResetTransObject()
dw_trab1.SetTransObject(sqlca)
dw_trab1.AcceptText()
dw_trab1.Retrieve()

FileWrite(ll_num_arq, "")
FileWrite(ll_num_arq, "")
FileWrite(ll_num_arq, ";;;
==========================================================")
FileWrite(ll_num_arq, ";;; DEFINE AS REGRAS DE relcestrclas")
FileWrite(ll_num_arq, ";;;
==========================================================")

For ll_i = 1 to dw_trab1.RowCount()

        ll_cdestrclas1 = dw_trab1.GetItemNumber(ll_i, "cdestrclas1")
        ll_cdestrclas2 = dw_trab1.GetItemNumber(ll_i, "cdestrclas2")
        ll_vlgraurelc  = dw_trab1.GetItemNumber(ll_i, "vlgraurelc")
```

```
        sl_regra = "(defrule estr" + string(ll_i) + " " + &
                        "(1cdestrclas1-eh " + string(ll_cdestrclas1) + ") "                    + &
                                                "(1cdestrclas2-eh " + string(ll_cdestrclas2) +
                                        ") "                + &
                                        "=> "
                                                                                + &
                                                "(bind ?*vlgraurelc* " +
                                                string(ll_vlgraurelc) + ") " + &
                                ") "

        FileWrite(ll_num_arq, sl_regra)

Next

dw_trab1.Reset() // limpa a dw1 para a janela não ficar "pesada"



// Carrega as regras de relcinstestrclas
st_status.Text = "Carregando as regras do relacionamento relcinstestrclas ....."
dw_trab2.Reset()
dw_trab2.ResetTransObject()
dw_trab2.SetTransObject(sqlca)
dw_trab2.AcceptText()
dw_trab2.Retrieve()

FileWrite(ll_num_arq, "")
FileWrite(ll_num_arq, "")
FileWrite(ll_num_arq, ";;;
==========================================================")
FileWrite(ll_num_arq, ";;; DEFINE AS REGRAS DE relcinstestrclas")
FileWrite(ll_num_arq, ";;;
==========================================================")
```

```
For ll_i = 1 to dw_trab2.RowCount()

        ll_cdestrclas1     = dw_trab2.GetItemNumber(ll_i, "cdestrclas1")
        ll_cdinstestrclas1 = dw_trab2.GetItemNumber(ll_i, "cdinstestrclas1")
        ll_cdestrclas2     = dw_trab2.GetItemNumber(ll_i, "cdestrclas2")
        ll_cdinstestrclas2 = dw_trab2.GetItemNumber(ll_i, "cdinstestrclas2")
        ll_vlaval          = dw_trab2.GetItemNumber(ll_i, "vlaval")

        sl_regra = "(defrule inst" + string(ll_i) + " "
        FileWrite(ll_num_arq, sl_regra)
        sl_regra = "        (2cdestrclas1-eh "     + string(ll_cdestrclas1)     + ") "
        FileWrite(ll_num_arq, sl_regra)
        sl_regra = "        (2cdinstestrclas1-eh " + string(ll_cdinstestrclas1) + ") "
        FileWrite(ll_num_arq, sl_regra)
        sl_regra = "        (2cdestrclas2-eh "     + string(ll_cdestrclas2)     + ") "
        FileWrite(ll_num_arq, sl_regra)
        sl_regra = "        (2cdinstestrclas2-eh " + string(ll_cdinstestrclas2) + ") "
        FileWrite(ll_num_arq, sl_regra)
        sl_regra = "      => "
        FileWrite(ll_num_arq, sl_regra)
        sl_regra = "        (bind ?*vlaval* " + string(ll_vlaval) + ") "
        FileWrite(ll_num_arq, sl_regra)
        sl_regra = ") "
        FileWrite(ll_num_arq, sl_regra)

Next

dw_trab2.Reset() // limpa a dw2 para a janela não ficar "pesada"



// Carrega os fatos de Requisitos X Arquiteturas X Estruturas
```

```
st_status.Text = "Carregando os fatos ....."
dw_trab3.Reset()
dw_trab3.ResetTransObject()
dw_trab3.SetTransObject(sqlca)
dw_trab3.AcceptText()
dw_trab3.Retrieve(sieaj2000.ii_cdsistestd)

FileWrite(ll_num_arq, "")
FileWrite(ll_num_arq, "")
FileWrite(ll_num_arq, ";;;
==========================================================")
FileWrite(ll_num_arq, ";;; DEFINE OS FATOS DE Requisitos X Arquiteturas X Estruturas")
FileWrite(ll_num_arq, ";;;
==========================================================")

If dw_trab3.RowCount() > 0 Then
        il_cdestlarqt_ant = dw_trab3.GetItemNumber(1, "cdestlarqt")
End If

For ll_i = 1 to dw_trab3.RowCount()

        sl_quebra = string(dw_trab3.GetItemNumber(ll_i, "cdestlarqt")) + &
                string(dw_trab3.GetItemNumber(ll_i, "cdreqtchav"))
        sl_quebra_ant = sl_quebra

        ll_j = 0

        Do While sl_quebra = sl_quebra_ant

                ll_j = ll_j + 1

                il_vetor [ll_j, 1] = dw_trab3.GetItemNumber(ll_i, "cdestrclas")
                il_vetor [ll_j, 2] = dw_trab3.GetItemNumber(ll_i, "cdinstestrclas")
```

```
            ll_i = ll_i + 1

            If ll_i > dw_trab3.RowCount() then
                  sl_quebra = "*****"
            Else
                  sl_quebra = string(dw_trab3.GetItemNumber(ll_i, "cdestlarqt")) + &
                    string(dw_trab3.GetItemNumber(ll_i, "cdreqtchav"))
            End If

      Loop

      For ll_k = 1 to ll_j

            ll_m = ll_k + 1

            For ll_l = ll_m to ll_j

                  FileWrite(ll_num_arq, "(bind ?*vlgraurelc* 0)")
                  FileWrite(ll_num_arq, "(bind ?*vlaval* 0)")

                  sl_fato = "(assert (1cdestrclas1-eh " + string(il_vetor[ll_k, 1]) + ") " + &
                              "(1cdestrclas2-eh " + string(il_vetor[ll_l, 1]) + ") )"
                  FileWrite(ll_num_arq, sl_fato)
                  FileWrite(ll_num_arq, "(run)")

                  sl_fato = "(assert (2cdestrclas1-eh "     + string(il_vetor[ll_k, 1]) + ") " + &
                              "(2cdinstestrclas1-eh " + string(il_vetor[ll_k, 2]) + ") " + &
                                    "(2cdestrclas2-eh "     + string(il_vetor[ll_l, 1])
+ ") " + &
                                    "(2cdinstestrclas2-eh " + string(il_vetor[ll_l, 2])
+ ") )"
                  FileWrite(ll_num_arq, sl_fato)
```

```
                    FileWrite(ll_num_arq, "(run)")

                    FileWrite(ll_num_arq, "( bind ?*cont* ( + ( * ( * ?*vlgraurelc* ?*vlaval* )
                        2 ) ?*cont* ) )")

                    FileWrite(ll_num_arq, "( retract * ) ")

                Next

            Next

            FileWrite(ll_num_arq, "( bind ?*cont* ( + " + String(ll_j) + " ?*cont* ) )")

            If ll_i > dw_trab3.RowCount() Then
                FileWrite(ll_num_arq, "(printout result ~"" + string(il_cdestlarqt_ant) + &
                            " *** ~" ?*cont* crlf)")
                FileWrite(ll_num_arq, "(bind ?*cont* 0)")
            Else
                If il_cdestlarqt_ant <> dw_trab3.GetItemNumber(ll_i, "cdestlarqt") then
                    FileWrite(ll_num_arq, "(printout result ~"" + string(il_cdestlarqt_ant) + &
                                " *** ~" ?*cont* crlf)")
                    FileWrite(ll_num_arq, "(bind ?*cont* 0)")

                    il_cdestlarqt_ant = dw_trab3.GetItemNumber(ll_i, "cdestlarqt")
                End If
            End If

            ll_i = ll_i - 1

Next


FileWrite(ll_num_arq, "(close result)")
```

```
FileWrite(ll_num_arq, "(reset)")
FileWrite(ll_num_arq, "(exit)")

FileClose(ll_num_arq)

//Executa o programa CLIPS
st_status.Text = "Executando o programa CLIPS ....."
Run("wxclips -clips pgm01.clp -start", Minimized!)

//Monta a dw da janela

ll_num_arq = -1
Do While ll_num_arq = -1 //aguarda término da execução do CLIPS
        ll_num_arq = FileOpen("result.txt", LineMode!, Read!, LockReadWrite!)
Loop
st_status.Text = "Montando a classificação ....."

il_ret = FileRead (ll_num_arq, sl_reg)

dw_padrao.SetRedraw(false)
Do While il_ret <> -100

        sl_monta_reg = ""
        For ll_i = 1 to il_ret   //busca código do estilo de arquitetura
                If Mid(sl_reg, ll_i, 1) <> " " Then
                sl_monta_reg = sl_monta_reg + Mid(sl_reg, ll_i, 1)
                Else
                        ll_j = ll_i + 1
                        ll_i = il_ret
                End If
        Next
  il_cdestlarqt = Integer(sl_monta_reg)
```

```
        For ll_i = ll_j to il_ret   //passa pelos *
                If Mid(sl_reg, ll_i, 1) <> "*" Then
                ll_j = ll_i + 1
                        ll_i = il_ret
                End If
        Next

        sl_monta_reg = ""   //busca pontuação do respectivo estilo de arquitetura
        For ll_i = ll_j to il_ret
                If Mid(sl_reg, ll_i, 1) <> " " Then
                sl_monta_reg = sl_monta_reg + Mid(sl_reg, ll_i, 1)
                Else
                        ll_i = il_ret
                End If
        Next

  ls_linha_corrente = dw_padrao.InsertRow(0)
        dw_padrao.SetItem (ls_linha_corrente, "cdsistestd", sieaj2000.ii_cdsistestd)
        dw_padrao.SetItem (ls_linha_corrente, "cdestlarqt", il_cdestlarqt)
        dw_padrao.SetItem (ls_linha_corrente, "vlpnto", Long(sl_monta_reg))

        il_ret = FileRead (ll_num_arq, sl_reg)

Loop

if dw_padrao.AcceptText() = -1 then
        return
end if

if dw_padrao.Update() = 1 then
        commit;
else
        rollback;
```

```
end if
dw_padrao.Retrieve(sieaj2000.ii_cdsistestd)
dw_padrao.Sort()
dw_padrao.SetRedraw(true)

FileClose(ll_num_arq)

st_status.Text = "Fim da Classificação"
SetPointer(Arrow!)
```

ANEXO C – Estudos de Caso

Este anexo possui todos os detalhes abordados nos estudos de caso apresentados no capítulo 5. Para cada estudo de caso detalhamos a pontuação alcançada em cada um dos requisitos de qualidade geral. Nos estudos de caso retirados de [Buschmann+98] justificamos a escolha de cada requisito de qualidade geral com base na seção de *implementação* do livro.

AC.1 ESTUDO DE CASO 1

Descrição do Sistema em Estudo:

O estudo de caso 1 apresenta um sistema desenvolvido em 1993 pela Electronic Data Systems do Brasil Ltda. (EDS), para o cliente T. Janér. T. Janér é uma empresa beneficiadora de papel, com sede no Rio de Janeiro e filiais em São Paulo e Porto Alegre. O autor desta dissertação era o líder da equipe que desenvolveu tal sistema – Sistema de Estoque de Bobinas de Papel.

Parte do referido sistema consiste da ação de coleta de dados (produto e natureza da operação[6]) via código de barras, por um leitor laser, o qual possui cinco etapas a saber.

A primeira etapa é a coleta de dados. Esta ocorre em uma pistola laser[7], sendo um processador 386 que executa o sistema operacional DR-DOS e linguagem C ANSI. A linguagem C chama bibliotecas específicas da pistola, como: disparo do laser, acender teclas, travar teclas, entre

[6] Compra, venda, troca, transformação, simples locomoção ou inventário de mercadorias.

[7] Pistola laser da marca Symbol.

outras. A primeira etapa ocorre quando a pistola é disparada e o feixe de laser lê um código de barras que esteja ao seu alcance. Os dados lidos ficam em uma memória EPROM, um tipo de memória eletrônica temporária que não apaga com a falta de bateria – semelhante a de telefones celulares.

A segunda etapa consistente de colocar a pistola em um dispositivo que se conecta a um microcomputador, por uma porta paralela, e transfere os dados da pistola para o disco rígido do microcomputador.

A terceira etapa é um programa que lê os dados do disco rígido do microcomputador e após uma série de críticas produz duas saídas. A primeira saída é um relatório de erros de dados inválidos, e a segunda é a gravação dos dados válidos em um banco de dados relacional (Sybase, versão 11). Este banco de dados disponibiliza as tabelas do Sistema de estoque para gravação dos dados coletados.

A quarta etapa é composta por programas que retiram as informações das tabelas de Estoque e gravam as informações nas tabelas que representam lançamentos contábeis (Sistema de Contabilidade).

Por fim, a quinta etapa, executa um programa que resgata informações das tabelas de Contabilidade e lança as informações nas tabelas de Livros Fiscais.

Considerações:

- Por razões de validações operacionais, o sistema deve obedecer a ordem seqüencial descrita, ou seja: coleta em pistola, disco rígido do microcomputador, tabelas de Estoque, tabelas de Contabilidade e tabelas de Livros Fiscais.
- As transferências de dados que são representadas nos cinco passos descritos devem ocorrer em velocidade considerada alta para não interromper as atividades do armazém que utiliza o

Sistema de Estoque. Em média, a transferência de cada item lido, em cada etapa, deve demorar 0,5 segundo.

- Em termos de sistema operacional, a pistola coletora de dados executa DR-DOS, o microcomputador executa Windows 95 e o banco de dados relacional encontra-se em um servidor UNIX.
- Exceto a leitura de código de barras, todos estes passos ocorrem de maneira batch, sem a interferência do usuário.

Análise dos Requisitos:

- O sistema deve obedecer a uma ordem seqüencial das operações, conforme descrito nas *considerações*. Tal requisito seleciona os requisitos de qualidade geral 2, 3 e 5.
- Conforme forem ocorrendo os processos de maneira seqüencial, novos dados vão sendo gerados para que o respectivo processo seguinte utilize-os. Ou seja, não existe compartilhamento de dados entre processos. Isso nos leva a seleção do requisito de qualidade geral 10.
- O tempo de 0,5 segundo referenciado nas *considerações* nos levou a escolha do requisito de qualidade geral 13.
- Todos os processos descritos neste estudo de caso envolvem três sistemas operacionais distintos, conforme *considerações*, o que nos levou a seleção do requisito de qualidade geral 14.
- A partir do momento que os dados são disponibilizados no disco rígido do microcomputador até a transferência para as tabelas do Sistema de Livros Fiscais, não há a interferência do usuário. Tal fato nos leva a seleção do requisito de qualidade geral 18.

Submetendo este sistema ao método de identificação de estilos de arquiteturas, teremos:

Resultado do Método de Contagem a partir dos Requisitos de Qualidade Geral:

	Tubos e Filtros	**Camada**	**Repositórios**	**Orientada a Objetos**	**Programa Principal / Subrotinas**
1					
2	77		153	227	
3	148	310			294
4					
5	-21	75	-5	-6	75
6					
7					
8					
9					
10	53	61		55	75
11					
12					
13			67	59	
14	124	124	67	116	
15					
16					
17					
18	1	1	1	1	1
19					
Tot	382	571	283	452	445

Tabela 41 – Contagem de pontos do estudo de caso 1 (completa)

Desta forma, podemos optar pela combinação dos estilos de arquiteturas camada e orientado a objetos.

Observações e Resultados:

- A referida parte do sistema em questão foi, na época, implementada em uma arquitetura de tubos e filtros, sem nenhuma característica de orientação a objetos.
- Tubos e filtros: este estilo de arquitetura seria, por intuição, o provável indicado. Não tornou-se a melhor opção, principalmente pelo fato de não manipular tão bem estruturas seqüenciais quando comparado com o estilo camada. Sendo este último um estilo especialista em processos seqüenciais.
- Camada: sendo este um estilo de arquitetura especialista em processos seqüenciais, foi o que melhor se adaptou ao sistema em estudo.
- Repositórios: com sua estrutura totalmente voltada para compartilhamento de dados e disparo de ações conforme determinados estados, em nada se adaptou ao sistema em estudo.
- Orientada a objetos: devido a sua característica de flexibilidade para qualquer natureza de problema, sua pontuação demonstrou porquê vem se tornando uma solução bem aceita em diversos problemas. Em relação ao estilo proposto, teve uma diferença menor que 150 pontos.
- Programa principal / subrotinas: embora este estilo possua uma série de limitações que a orientação a objetos vem demonstrando superar, neste exemplo verificamos porquê este foi, e ainda é, um estilo tão bem aceito, principalmente no processamento de dados comercial.
- O método recomenda o estilo camada, sendo cada camada um objeto, conforme recomenda o estilo orientado a objetos.

AC.2 ESTUDO DE CASO 2

Descrição do Sistema em Estudo:

O estudo de caso 2, apresenta um problema idêntico ao apresentado no estudo de caso 1, porém algumas mudanças devem ser observadas. O estudo de caso 2 surgiu, em 1994, a partir do momento que foram identificados problemas de performance no sistema implementado no estudo de caso 1.

Considerações:

- Não existe mais a obrigatoriedade de seqüência a partir do final da segunda etapa, ou seja, após os dados estarem no disco rígido do microcomputador, estes podem ir em seqüências diferentes para as tabelas de Estoque, Contabilidade e Livros Fiscais. A única restrição é que estas operações estejam em uma mesma unidade lógica de trabalho de um banco de dados relacional (Sybase – versão 11).
- Para ganho de performance, a distribuição dos dados por Estoque, Contabilidade e Livros Fiscais deve ocorrer de forma paralela ou concorrente.
- Para compor a modificação da solução descrita, interpretamos que o disco rígido do microcomputador passa a ser uma base de dados compartilhada, de tal forma que seu estado faz disparar processos para a inclusão de dados nas tabelas de Estoque, Contabilidade e Livros Fiscais. Com esta arquitetura, poderemos tirar proveito de processamento paralelo ou concorrente.

Análise dos Requisitos:

- Conforme as *considerações* descritas, houve a total perda de sequencialidade. Tal fato nos leva a escolha dos requisitos de qualidade geral 2, 4 e 6.
- A perda de sequencialidade favorece a existência de uma base de dados única para todos os processos existentes do sistema em estudo. Isso nos leva a escolha dos requisitos de qualidade geral 9 e 12.
- O estudo de caso 2 surgiu a partir da necessidade de maior rapidez do tempo de resposta, conforme sugerem os requisitos de qualidade geral 13 e 17.
- Todos os processos descritos neste estudo de caso envolvem três sistemas operacionais distintos, conforme *considerações*, o que nos levou a seleção do requisito de qualidade geral 14.
- A partir do momento que os dados são disponibilizados no disco rígido do microcomputador até a transferência para as tabelas do Sistema de Livros Fiscais, não há a interferência do usuário. Tal fato nos leva a seleção do requisito de qualidade geral 18.

Submetendo estes novos requisitos ao método de identificação de estilos de arquiteturas e mantendo as demais considerações do estudo de caso 1, teremos:

Resultado do Método de Contagem a partir dos Requisitos de Qualidade Geral:

	Tubos e Filtros	**Camada**	**Repositórios**	**Orientada a Objetos**	**Programa Principal / Subrotinas**
1					
2	77		153	227	
3					
4			153	227	
5					
6	1		1	1	
7					
8					
9			102	59	1
10					
11					
12			67	59	
13			67	59	
14	124	124	67	116	
15					
16					
17	-5		7	11	1
18	1	1	1	1	1
19					
Tot	198	125	618	760	3

Tabela 42 – Contagem de pontos do estudo de caso 2 (completa)

Desta forma, podemos optar pela combinação dos estilos de arquitetura orientado a objetos e repositórios.

Observações e Resultados:

- A referida parte do sistema em questão foi, na época, implementada exatamente como recomendação do método. No momento da implementação da solução, não existia, por parte de nenhum membro da equipe de desenvolvedores, conhecimentos sobre arquitetura de software.
- Para a obtenção de maior velocidade no sistema proposto, três características novas surgiram: perda da sequencialidade, processamento paralelo / concorrente e compartilhamento dos dados. Com isso, os estilos tubos e filtros, camada e programa principal / subrotinas, tornaram-se inadequados.
- Repositórios: ao ler as considerações do problema, intuitivamente, parece ser o estilo mais adequado. As características que predominam neste estilo vão de encontro aos novos requisitos do estudo de caso 2. Foi superado pelo estilo orientada a objetos, porém não foi uma diferença expressiva (menor que 150 pontos).
- Orientada a objetos: sua grande flexibilidade novamente fez este estilo atingir uma boa pontuação, inclusive superando o estilo que intuitivamente seria o indicado (repositórios).
- O método recomenda o estilo orientado a objetos, sendo cada objeto uma *fonte de informação* ligado ao repositório (disco rígido do microcomputador).

AC.3 ESTUDO DE CASO 3

Descrição do Sistema em Estudo:

A EDS, uma firma de desenvolvimento de software, a mais de 20 anos atrás, teve a visão de desenvolver uma biblioteca de componentes reutilizáveis. Tendo como ponto de partida a tecnologia de 20 anos passados. Nesta biblioteca existiriam programas típicos de reuso na época: cálculo de dígito verificador, validade numérica, validade alfanumérica, validade de datas, cálculos com datas, conexão com base de dados de estrutura hierárquica, entre outros. O autor desta dissertação foi usuário desta biblioteca em 1991.

Considerações:

- Os programas são feitos em COBOL e persistência de dados em IMS. Linguagem e base de dados bastante comuns na época.
- Execução no *mainframe* IBM, em VSE ou MVS (sistemas operacionais da IBM).
- Passagem de parâmetros e base de dados como principais formas de comunicação entre os programas.
- Ausência de conceitos de atributos (variáveis) e métodos (programas ou funções) globais ou locais.
- Ausência de conceitos de encapsulamento e proteção de atributos e métodos.

Análise dos Requisitos:

- A referida biblioteca é composta de pequenos programas que podem ser utilizados a partir de qualquer ponto de um sistema em desenvolvimento. Concluímos que não há estrutura

hierárquica para a biblioteca, bem como não existe a necessidade de uma situação precedente para a utilização dos programas desta biblioteca. Ou seja, selecionamos os requisitos de qualidade geral 2, 4 e 6.

- O objetivo principal da biblioteca nos leva a escolha do requisito de qualidade geral 15.

Submetendo este sistema ao método de identificação de estilos de arquiteturas, teremos:

Resultado do Método de Contagem a partir dos Requisitos de Qualidade Geral:

	Tubos e Filtros	**Camada**	**Repositórios**	**Orientada a Objetos**	**Programa Principal / Subrotinas**
1					
2	77		153	227	
3					
4			153	227	
5					
6	1		1	1	
7					
8					
9					
10					
11					
12					
13					
14					
15	124	287	67	185	
16					
17					
18					
19					
Tot	202	287	374	640	0

Tabela 43 – Contagem de pontos do estudo de caso 3 (completa)

Desta forma, podemos optar pelo estilo de arquitetura orientado a objetos.

Observações e Resultados:

- O caminho natural a ser seguido, conforme pensamento vigente naquele momento, seria o estilo de arquitetura programa principal / subrotinas. Vimos no processo classificatório o quanto este estilo tornou-se inadequado quando falamos de reusabilidade de software. Desta forma, o processo classificatório apresenta um exemplo do quanto estes sistemas demonstraram ser de difícil manutenção e reusabilidade nos anos 80.
- A característica de seus componentes com intenção de reuso, tornou os estilos tubos e filtros, camada e repositórios uma boa opção para o problema proposto.
- Orientada a objetos: conforme requisitos apresentados, o estilo orientado a objetos mostrou ser a melhor opção devido as suas características de forte reuso e flexibilidade.

  Cabe ressaltar que os requisitos de qualidade geral que melhor foram associados ao problema proposto foram aqueles que formam o estilo orientado a objetos. Tal fato não significa que, na época, existiam ferramentas comerciais para a prática do referido estilo.
- Programa principal / subrotinas: embora este estilo fosse o mais comum na época em que o problema foi proposto, podemos ver neste exemplo um dos motivos pelos quais os sistemas antigos apresentavam muita dificuldade em sua manutenção. Com zero ponto, não podemos considerar como uma solução viável, embora tenha sido a escolhida na época.
- Para o problema apresentado, reusabilidade, o método recomenda o estilo orientado a objetos. O referido estilo possui fortes características de reuso.

AC.4 ESTUDO DE CASO 4

Descrição do Sistema em Estudo:

Após passada a experiência do estudo de caso 3, a mesma firma propôs no final dos anos 90 um projeto semelhante, porém em multiplataforma, com facilidades de processamento paralelo, desenvolvimento por componentes, frameworks, utilizando o padrão CORBA – IDL, para as práticas de reuso. O autor desta dissertação atuou, de 1997 a 1999, como líder técnico, no Brasil, desta mesma biblioteca.

Considerações:

- Nenhuma consideração nova, exceto as já mencionadas: multiplataforma, com facilidades de processamento paralelo, desenvolvimento por componentes, frameworks, entre outros.

Análise dos Requisitos:

- A referida biblioteca é composta de pequenos programas / objetos que podem ser utilizados a partir de qualquer ponto de um sistema em desenvolvimento. Concluímos que não há estrutura hierárquica para a biblioteca, bem como não existe a necessidade de uma situação precedente para a utilização dos programas / objetos desta biblioteca. Ou seja, selecionamos os requisitos de qualidade geral 2, 4 e 6.
- As características existentes nas novas tecnologia nos levou a seleção dos requisitos de qualidade geral 14 e 17.
- O objetivo principal da biblioteca nos leva a escolha do requisito de qualidade geral 15.

Submetendo estes novos requisitos ao método de identificação de estilos de arquiteturas e mantendo as demais características, teremos:

Resultado do Método de Contagem a partir dos Requisitos de Qualidade Geral:

	Tubos e Filtros	**Camada**	**Repositórios**	**Orientada a Objetos**	**Programa Principal / Subrotinas**
1					
2	77		153	227	
3					
4			153	227	
5					
6	1		1	1	
7					
8					
9					
10					
11					
12					
13					
14	124	124	67	116	
15	124	287	67	185	
16					
17	-5		7	11	1
18					
19					
Tot	321	411	448	767	1

Tabela 44 – Contagem de pontos do estudo de caso 4 (completa)

Observações e Resultados:

- A classificação continua exatamente a mesma, pelos mesmos motivos.
- Podemos observar um maior distanciamento do estilo orientado a objetos, porém ainda não expressivo. Consideramos este comportamento mais um indício do quanto o estilo orientado a objetos se adequa as atuais necessidades de desenvolvimento de software e características de reuso.
- Desta forma, o método sugere o estilo orientado a objetos como solução do problema proposto.

AC.5 ESTUDO DE CASO 5

Descrição do Sistema em Estudo:

Fazendo uma auto-análise do método de identificação de estilos de arquiteturas proposto no SIEA, analisaremos duas partes a saber. A primeira é a representação do conhecimento de estilos de arquitetura, o que sugere a forma de repositório, pois possui características de um problema de Inteligência Artificial (IA). A segunda parte do software proposto é o método de contagem de pontos, que será tratado no estudo de caso 6. Todas as características da representação do conhecimento encontram-se especificadas no capítulo 2.

Análise dos Requisitos:

- Toda a base de conhecimento construída ao longo do capítulo 2 nos levou a uma representação a partir de redes semânticas. A análise das diversas redes semânticas que constituem a base de conhecimento nos leva a seleção dos requisitos de qualidade geral 2 e 4.
- Neste estudo de caso estamos falando da resolução de um problema com características estáticas. Ou seja, conhecimento armazenado em uma rede semântica, não importando a ordem como serão acessados. Isso nos leva a escolha do requisito de qualidade geral 6.
- As redes semânticas vão sendo formadas a partir das informações de redes já existentes. Por exemplo, a rede semântica que armazena o conhecimento sobre *avaliação* e *grau de relacionamento*, depende das redes semânticas das estruturas de classificação e respectivas instâncias. Isso nos leva a seleção do requisito de qualidade geral 9.
- A declaração de fatos sob as regras existentes na base de conhecimento nos leva a escolha do requisito de qualidade geral 12.

- O sistema em questão deve ter a facilidade de ser executado em mais de uma plataforma de hardware e software. Tal fato nos ajuda a disseminar o nosso conhecimento a partir da utilização do SIEA. Isso nos leva a seleção do requisito de qualidade geral 14.
- Conforme especificamos o SIEA, no capítulo 4, o requisito de qualidade geral 19 faz-se necessário.

Submetendo a parte de representação do conhecimento do SIEA ao método de identificação de estilos de arquiteturas, teremos:

Resultado do Método de Contagem a partir dos Requisitos de Qualidade Geral:

	Tubos e Filtros	**Camada**	**Repositórios**	**Orientada a Objetos**	**Programa Principal / Subrotinas**
1					
2	77		153	227	
3					
4			153	227	
5					
6	1		1	1	
7					
8					
9			102	59	1
10					
11					
12			67	59	
13					
14	124	124	67	116	
15					
16					
17					
18					
19		1	1	1	1
Tot	202	125	544	690	2

Tabela 45 – Contagem de pontos do estudo de caso 5 (completa)

Observações e Resultados:

- O método de contagem indicou o estilo orientado a objetos, porém com aproximação do estilo repositórios, sendo este último o estilo intuitivamente sugerido.
- Devido a este resultado o método propõem a construção do referido software com arquitetura mista: orientada a objetos e repositórios. Ou seja, cada *fonte de informação* é um objeto que se comunica com o repositório. O repositório é o local onde temos as redes semânticas que refletem o conhecimento sobre estruturas de classificação, instâncias, relacionamentos e estilos.
- Novamente podemos constatar a boa adaptabilidade do estilo orientado a objetos para vários tipos de situações.

AC.6 ESTUDO DE CASO 6

Descrição do Sistema em Estudo:

O estudo de caso 6, conforme dito anteriormente, será uma auto-análise do método de contagem do software proposto, SIEA. Todo o método de contagem foi especificado no capítulo 3.

Considerações:

- O método de contagem é basicamente seqüencial, de tal forma que possamos fazer a pontuação e posteriores análise e recomendações.
- Demais requisitos, são conforme descrito no capítulo 3, onde detalhamos o método de contagem.

Análise dos Requisitos:

- O método de contagem apresenta características seqüenciais. Isso nos leva a seleção dos requisitos de qualidade geral 2, 3 e 5.
- Todo o método de contagem obtém informações a partir da base de conhecimento, implementada a partir das redes semânticas. Isso nos leva a seleção do requisito de qualidade geral 9.
- A declaração de fatos sob as regras existentes na base de conhecimento nos leva a escolha do requisito de qualidade geral 12.
- A seleção dos software especificados no capítulo 4 – CLIPS, PB e MS-Access - nos leva a seleção do requisito de qualidade geral 14.
- Conforme detalhamos o SIEA no capítulo 4, o requisito de qualidade geral 19 faz-se necessário.

Submetendo este sistema ao método de identificação de estilos de arquiteturas, teremos:

Resultado do Método de Contagem a partir dos Requisitos de Qualidade Geral:

	Tubos e Filtros	**Camada**	**Repositórios**	**Orientada a Objetos**	**Programa Principal / Subrotinas**
1					
2	77		153	227	
3	148	310			294
4					
5	-21	75	-5	-6	75
6					
7					
8					
9			102	59	1
10					
11					
12			67	59	
13					
14	124	124	67	116	
15					
16					
17					
18					
19		1	1	1	1
Tot	328	510	385	456	371

Tabela 46 – Contagem de pontos do estudo de caso 6 (completa)

Observações e Resultados:

- Observamos que o estilo camada é o mais favorecido. O motivo do resultado desta contagem é semelhante ao ocorrido no estudo de caso 1. Ou seja, a obrigatoriedade de existirem processos seqüenciais favoreceu o estilo camada. Sendo o referido estilo também forte nas características de reuso, assim como o estilo orientada a objetos, que obteve pontuação próxima.
- Conforme sugestão do método (150 pontos de diferença), os seguintes estilos são recomendados: camada e orientado a objetos. Desta forma, cada camada a ser construída será um objeto, conforme sugere o estilo orientado a objetos.

AC.7 ESTUDO DE CASO 7

Descrição do Sistema em Estudo:

O estudo de caso 7, será uma análise da API (*Application Programming Interface*) sugerida em [Buschmann+98] como camada - páginas: 46 - 47.

Considerações:

- API é uma camada que encapsula camadas mais baixas de funcionalidades freqüentes, em diversas plataformas de hardware e sistemas operacionais.
- As API's devem prover portabilidade entre os sistemas operacionais. Em geral, a referida portabilidade nos leva a uma baixa performance, o que não constitui um problema para o escopo do nosso estudo.
- Os requisitos serão avaliados com a observação da seção de *implementação* de [Buschmann+98].

Submetendo este sistema ao método de identificação de estilos de arquiteturas, teremos:

Resultado do Método de Contagem a partir dos Requisitos de Qualidade Geral:

	Tubos e Filtros	**Camada**	**Repositórios**	**Orientada a Objetos**	**Programa Principal / Subrotinas**
1		357			274
2					
3	148	310			294
4					
5	-21	75	-5	-6	75
6					
7					
8					
9					
10					
11	2	1	1	1	1
12					
13					
14	124	124	67	116	
15	124	287	67	185	
16					
17	-5		7	11	1
18					
19					
Tot	372	1.154	137	307	645

Tabela 47 – Contagem de pontos do estudo de caso 7 (completa)

Observações e Resultados:

- A escolha do requisito 1, foi devido ao item 3 da seção implementação. Neste item é observada a necessidade de cada uma das camadas ter que atender a sua camada imediatamente superior, até que a camada mais superior possa atender a solicitação do requisitante do serviço. Tal necessidade implementa uma formação hierárquica e linear.
- A escolha do requisito 3, também foi devido ao item 3, e também devido aos itens 6, 8 e 9. Todos estes itens enfatizam a necessidade de termos uma interface bem definida entre as camadas. Quando é falado sobre as interfaces, sempre é citada uma camada J que se comunica apenas com as camadas adjacentes J-1 e J+1, dando a noção de sequencialidade entre as camadas. Para a escolha do requisito 3 não levamos em consideração a variante onde permite um relaxamento na comunicação entre as camadas. Ou seja, a camada J não se comunica apenas com as camadas J-1 e J+1.
- Devido a obrigatoriedade da existência de uma estrutura hierárquica e seqüencial, onde a camada J apenas pode ter comunicação com as camadas J-1 e J+1, também escolhemos o requisito 5. Ou seja, a execução seqüencial das camadas é um tipo de priorização entre componentes.
- A escolha do requisito 11 foi devido aos itens 8 e 10 da seção de implementação. Nestes itens são apresentadas as necessidades de termos mecanismos de *push*, *pull* e tratamento de erros entre camadas. Uma das formas de implementação é uma camada enviar um código de retorno para sua camada superior, existindo assim mecanismos de *feedback* entre camadas vizinhas.
- A escolha dos requisitos 14 e 15 deve-se ao fato de ser uma API, conforme referenciado na seção *usos conhecidos*. Ou seja, conforme dito em [Buschmann+98], um dos principais

motivos da existência de uma API é para encapsularmos funções freqüentemente utilizadas, propiciando portabilidade em diferentes sistemas operacionais.

- Embora tenhamos o processamento linear para uma determinada requisição, isso não significa a ausência de processamento paralelo. Ou seja, enquanto uma camada atende a uma determinada fase de uma requisição, uma outra camada mais inferior está iniciando o atendimento de uma outra requisição em sua fase inicial. Isso posto, escolhemos o requisito 17.
- Assim como em [Buschmann+98], chegamos ao resultado que indica a arquitetura camada como recomendada.
- Influenciou no resultado, a favor de camada, a estrutura hierárquica dos componentes, bem como as capacidades de portabilidade e reuso dos mesmos.

AC.8 ESTUDO DE CASO 8

Descrição do Sistema em Estudo:

O estudo de caso 8, será uma análise da linguagem / compilador Mocha (*Modular Object Computation with Hypothetical Algorithms*) sugerida em [Buschmann+98] como tubos e filtros - páginas: 53 - 66.

Considerações:

- O objetivo é construir um software que “compile”, “linkedite” e gere o executável a partir de um programa fonte em Mocha.
- Deve ser um compilador com portabilidade entre diversas plataformas de hardware.
- Para atingir seus objetivos, os engenheiros de software optaram pela construção de uma linguagem intermediária (AuLait – *Another Universal Language for Intermediate Translation*) rodando sobre uma máquina virtual (Cup – *Concurrent Uniform Processor*).
- Os requisitos serão avaliados com a observação da seção de *implementação* de [Buschmann+98].

Submetendo este sistema ao método de identificação de estilos de arquiteturas, teremos:

Resultado do Método de Contagem a partir dos Requisitos de Qualidade Geral:

	Tubos e Filtros	**Camada**	**Repositórios**	**Orientada a Objetos**	**Programa Principal / Subrotinas**
1					
2	77		153	227	
3	148	310			294
4					
5					
6					
7					
8					
9					
10					
11					
12					
13					
14	124	124	67	116	
15					
16					
17					
18	1	1	1	1	1
19					
Tot	350	435	221	344	295

Tabela 48 – Contagem de pontos do estudo de caso 8 (completa)

Observações e Resultados:

- A característica de sequencialidade é comum na construção de qualquer compilador. Os requisitos 2 e 3 foram escolhidos devido ao item 1 da seção implementação. Este item enfatiza a necessidade de haver um processo com estágios seqüenciais. Ou seja, os dados resultantes do final de um filtro são os mesmos dados necessários para o início do processamento do filtro seguinte. Com a leitura do item 1 entendemos estarmos falando de processos seqüenciais e não hierárquicos.

- Foi selecionado o requisito 2 e não o 1, pois observamos que um filtro produz resultados mesmo antes de consumir toda a entrada. Para tal o filtro conta com a função do tubo para sincronismo dos dados. Esta característica elimina a noção de hierarquia uma vez que o filtro deveria consumir todos os dados de entrada, para depois produzir a saída. O que não ocorre.

- Um compilador deve ter portabilidade entre plataformas de hardware e sistemas operacionais. O requisito 14 foi escolhido devido a esta característica. É citado sobre portabilidade e reaproveitamento de filtros nos itens 2, 3 e 4 da seção de implementação. No item 2 é destacada a opção das interfaces serem sempre via arquivos em formato ASCII, embora seja prejudicial para a performance. No item 3 o motivo de tal representação é visando a substituição de filtros por outros filtros, bem como a diminuição do grau de interdependência entre estes. O item 4 fala da importância de estudarmos bem o comportamento de um filtro, quanto ao seu escopo, para que este tenha boa portabilidade. É citada a importância da boa definição de interfaces entre os filtros, através dos tubos, visto ser esta a única forma de compartilhamento de dados.

- O requisito 18 foi selecionado pois, em geral, compiladores e linkeditores possuem o processamento batch.

- Em [Buschmann+98] foi sugerida a arquitetura tubos e filtros. Nosso método propôs uma combinação de camada e tubos e filtros.
- Houve uma aproximação de pontuação entre: camada, tubos e filtros e orientado a objetos.
- Como em [Buschmann+98] os estilos orientado a objetos e programa principal / subrotinas não existem – não são considerados como estilos – nos limitamos a observar a combinação tubos e filtros e camada.
- Tubos e filtros foi indicada principalmente pelas suas características de estrutura linear e portabilidade, características bastante pertinentes ao compilador em questão, conforme seção *implementação* de [Buschmann+98].
- A arquitetura camada surgiu com maior pontuação e com as mesmas características de tubos e filtros. Sua alta pontuação destacou-se basicamente na estrutura linear, influenciada pela linguagem intermediária e pela máquina virtual descritas em [Buschmann+98].
- Desta forma, consideramos nosso resultado compatível com [Buschmann+98], aproveitando as características lineares do estilo camada.

AC.9 ESTUDO DE CASO 9

Descrição do Sistema em Estudo:

O estudo de caso 9, será uma análise do modelo OSI (International Standardization Organization) – 7 Camadas, conforme apresentado em [Buschmann+98], que o classificou como uma arquitetura camada, típica - páginas: 31 - 46.

Considerações:

- Os requisitos serão avaliados com a observação da seção de *implementação* de [Buschmann+98].

Submetendo este sistema ao método de identificação de estilos de arquiteturas, teremos:

Resultado do Método de Contagem a partir dos Requisitos de Qualidade Geral:

	Tubos e Filtros	**Camada**	**Repositórios**	**Orientada a Objetos**	**Programa Principal / Subrotinas**
1		357			274
2					
3	148	310			294
4					
5	-21	75	-5	-6	75
6					
7					
8					
9					
10	53	61		55	75
11	2	1	1	1	1
12					
13					
14	124	124	67	116	
15	124	287	67	185	
16					
17					
18					
19					
Tot	430	1.215	130	351	719

Tabela 49 – Contagem de pontos do estudo de caso 9 (completa)

Observações e Resultados:

- Em [Buschmann+98] foi sugerida a arquitetura camada. Nosso método indicou o mesmo.
- O software apresentado é um exemplo típico de camada. Conforme podemos observar, quase todos os requisitos que compõem o referido estilo foram escolhidos.
- As justificativas das escolhas dos requisitos são semelhantes a do estudo de caso 7. As únicas diferenças estão nos requisitos 10 e 17.
- O requisito 10 foi selecionado para o estudo de caso 9 pois na arquitetura OSI, em todos os momento fala-se em transferência de dados de uma camada para outra. Nunca é falado em um repositório de dados, pois isso rompe a independência que existe entre as camadas.
- O requisito 17 não foi selecionado pois em nenhum momento foi citada a necessidade de processamento concorrente e paralelo, embora tal facilidade exista na implementação do modelo OSI – 7 camadas. A escolha deste requisito em nada mudaria o resultado e a análise feita para o estudo de caso 9.

AC.10 ESTUDO DE CASO 10

Descrição do Sistema em Estudo:

O estudo de caso 10, será uma análise do software de reconhecimento de palavras sugerido em [Buschmann+98] como repositórios - páginas: 71 - 87.

Considerações:

- A decisão do livro [Buschmann+98] baseou-se no fato de ser um software típico de Inteligência Artificial (IA), e como tal, indica o estilo repositório como solução.
- Os requisitos serão avaliados com a observação da seção de *implementação* de [Buschmann+98].

Submetendo este sistema ao método de identificação de estilos de arquiteturas, teremos:

Resultado do Método de Contagem a partir dos Requisitos de Qualidade Geral:

	Tubos e Filtros	**Camada**	**Repositórios**	**Orientada a Objetos**	**Programa Principal / Subrotinas**
1					
2	77		153	227	
3					
4			153	227	
5					
6	1		1	1	
7					
8					
9			102	59	1
10					
11					
12			67	59	
13					
14					
15					
16					
17					
18					
19					
Tot	78	0	476	573	1

Tabela 50 – Contagem de pontos do estudo de caso 10 (completa)

Observações e Resultados:

- Em [Buschmann+98] foi sugerida a arquitetura repositório. Nosso método propôs uma combinação de repositório e orientada a objetos.
- O item 7 da seção de *implementação* fala da independência das *fontes de informação*, inclusive citando o fato de uma *fonte de informação* não saber da existência da outra. Todas as ordens de ações são determinadas por um programa inerente ao repositório central. Esta independência entre *fontes de informação* nos levou a escolha dos requisitos 2, 4 e 6.
- A figura do repositório central, inclusive sendo este o coordenador de todos as *fontes de informação*, nos levou a escolha do requisito 9. Características de implementação deste repositório podem ser encontradas no item 5 da seção de implementação.
- O requisito 12 foi selecionado a partir da leitura do item 6 da seção de *implementação*. Neste item é abordada a existência de um componente de controle, o qual utiliza a estratégia de resolução de problemas a partir de "*oportunismo"*. Ou seja, o componente de controle verifica o estado atual do repositório. Com base no estado do repositório e o objetivo a ser alcançado, tal componente dispara as melhores *fontes de informação* para resolver o problema existente naquele momento.
- Como em [Buschmann+98] o estilo orientado a objetos (OO) não existe – não é considerado como estilo – nos limitamos a observar que OO é uma técnica indicada em soluções de Inteligência Artificial (IA), conforme sugere o ambiente CLIPS (*C Language Integrated Production System*).
- Desta forma, consideramos nosso resultado compatível com [Buschmann+98].

ANEXO D – Modelo de Entidades e Relacionamentos do SIEA

Neste anexo apresentamos o MER – Modelo de Entidades e Relacionamento – que foi implementado em tabelas do MS-Access, permitindo assim o armazenamento de dados do SIEA.

Referências Bibliográficas

[Abd-Allah+95]

Abd-Allah, Ahmed e Boehm, Barry, "Reasoning about the Composition of Heterogeneous Architectures" – USC Technical Report: USC-CSE-95-503, 1995

[ACM+95.2]

Formalizing Style to Understand Descriptions of Software Architecture.

ACM Transactions on Software Engineering and Methodology, 4(4): 319-364 - October 1995

[Alexander+77]

Alexander, C., Ishikawa, S., Silverstein, M., Jacobson, M., Fiksdahl-King, I., Angel, S.

"A Pattern Language" New York: Oxford University Press - 1977

[Alexander+78]

Alexander, C

"The Timeless Way of Building" New York: Oxford University Press - 1978

[Allen+97.2]

Allen, Robert e Garlan, David: "A Formal Basis for Architetural Connection"

ACM Transactions on Software Engineering and Methodology, 6(3): 213-249 - July 1997

[Bass+98]

Bass, Len, Clements, Paul, Kazman, Rick, "Software Architecture in Practice" – 1998 – ISBN 0-201-19930-0

[Brownston+85]

Brownston, Lee, Farrell, Robert, Kant, Elaine, Martin, Nancy, "Programming Expert Systems in OPS5 – An Introduction to Rule-Based Programming" – ISBN 0-201-10647-7 – Addison-Wesley Publishing Company, Inc. - 1985

[Buschmann+98]

Buschmann, Frank, Meunier, Regine, Rohnert, Hans, Sommerlad, Peter, Stal, Michael, "Pattern – Oriented – Software Architecture – A System of Patterns" – 1998 – ISBN 0-471-95869-7

[CMU01]

Wright, http://www.cs.cmu.edu/afs/cs/project/able/www/wright/index.html

[Gacek+98.2]

Gacek, Cristina, "Detecting Architectural Mismatches During Systems Composition" – December 1998 – A Dissertation Presented to the Faculty of the Graduate School University of Southern California

[Gamma+95]

Gamma, Erich, Helm, Richard, Johnson, Ralph, Vlissides, John, "Design Patterns – Elements of Reusable Object-Oriented Software" – 1995 – ISBN 0-201-63361-2

[Garlan+93]

Garlan, David, Shaw Mary, "An Introduction to Software Architecture" – V Ambriola and G. Tortora, eds. Advances in Software Engineering and Knowledge Engineering, pp. 1-39, 1993

[Jackson+75]

Jackson Michael A "Principles of Program Desing". Londres: Academic Press, 1975

[Kazman+99]

Kazman, Rick, "An Introduction to Software Architecture – class notes" – Departament of Computer Science University of Waterloo – Institute for Computer Research - 1999

[Klein+99.2]

Klein, Mark, Kazman, Rick, "Attribute-Based Architectural Styles" – October 1999 – Technical Report – CMU / SEI – 99 – TR – 022 - ESC – TR – 99 - 022

[Leite+97-1]

Leite, Julio C. S. P., Leonardi, Maria C., Rossi, Gustavo, Antonelli, Leandro, "Deriving Object-Oriented Specifications from a Requirements Baseline" – work material - 1997

[Leite+97-2]

Leite, Julio C. S. P., Rossi, Gustavo, Balaguer, F., Maiorana, V., Kaplan, G., Hadad, G., Oliveros, A., "Enhancing a Requirements Baseline with Scenarios" – Third IEEE International Symposium on Requirements Engineering – Volume 2 Number 4 – 1997 – ISBN 0947 3602

[Paula+98.2]

Paula, Virginia C. de; Justo, G. R. Ribeiro e Cunha, P. R. F., "Specifying Dynamic Distributed Software Architectures", outubro 1998.

XII SBES – Simpósio Brasileiro de Engenharia de Software, Maringá – Paraná – Brasil

[Rapide01]

Rapide, http://pavg.stanford.edu/rapide/rapide.html

[Rich+83]

Rich, Elaine: "Artificial Intelligence" – ISBN 0-07-052261-8 – McGraw-Hill - 1983

[Rofrano+99.1]

Rofrano, John J., "Java Portability by Design" – Dr. Dobb's Journal – June 1999 – Miller Freeman Publicatio

[Ross01]

Ross, Douglas T., Marca, David A., McGowan, Clement L., "SADT – Structured Analysys and Design Technique" – ISBN 0-07-040235-3 - McGraw-Hill

[SATG01]

Software Architecture Technology Guide

http://www-ast.tds-gn.lmco.com/arch/guide.html

[SEI01]

Software Engineering Institute, Software Architecture

http://www.sei.cmu.edu/architecture/sw_architecture.html

[SEI02]

Selected Software Architecture Papers

http://www.sei.cmu.edu/activities/architecture/bibpart1.html

[Shaw+94]

Shaw, Mary e Garlan, David: “Characteristics for High-level Languages for Software Architectures, Technical Report CMU-CS-94-210 - 1994

[Shaw+96]

Shaw, Mary e Garlan, David : “Software Architecture – Perspectives on a Emerging Discipline” – 1996 – ISBN 0-13-182957-2

[Shaw+96.1]

Shaw, Mary e Clements, Paul : “A Field Guide to Boxology: Preliminary Classification of Architectural Styles for Software Systems”, April 1996

[Silva+99.2-1]

Silva, Marcelo Cabral, Leite, Julio C. S. P., “Identificação de Estilos de Arquiteturas: Um Processo Dirigido por Conhecimento” – Setembro 1999 – Pronex 99 – PUC-Rio

[Silva+99.2-2]

Silva, Marcelo Cabral, “Estudos de Caso em Classificação de Arquiteturas de Software” – Dezembro 1999 – Trabalhos Individuais em Engenharia de Software III - PUC-Rio

[Tanimoto01]

Tanimoto, Steven L., “The Elements of Artificial Intelligence – Na Introduction Using LISP” – ISBN 0-88175-113-8 – Computer Science Press, Inc.

[Yourdon+78]

Yourdon, Ed. e Constantine, L. L.: “Structure Design: Fundamentals of a Discipline of Computer Programming and System Design”, 2ª edição. New York: Yourdon Press, 1978

[Warnier+74]

Warnier, Jean-Dominique. “Logic Construction of Programs”. Leiden: Holanda, H.E. Stenfert Kroese BV, 1974

[Wiederhold 01]

Wiederhold, Gio “File Organization for Database Design” – ISBN 0-07-070133-4 - McGraw-Hill