2
Fundamentação Teórica

Neste capítulo são apresentados os conceitos e abordagens utilizadas, que servem como base para o trabalho realizado. Dentre estes conceitos estão duas técnicas da engenharia de requisitos (LAL e cenários) e o Catálogo de Transparência de Software, que é peça fundamental neste estudo. Apresentamos também, neste capítulo, a instanciação dos conceitos do padrão arquitetural MVC, conceito fundamental para o entendimento do trabalho. Por fim, detalhamos a motivação para criação do Catálogo de Transparência de Software, descrevemos o catálogo em si e apontamos sua relação com o trabalho realizado.

2.1.
Léxico Ampliado da Linguagem (LAL)

A técnica LAL (Leite e Franco, 1993) está fundamentada na existência de uma linguagem específica em cada Universo de Informação (UdI). Esta linguagem é composta de símbolos, que possuem um significado particular quando utilizados pelos atores do UdI. Portanto, para compreender o UdI é preciso primeiro entender tal linguagem. O LAL foi criado especificamente para guiar a elicitação e permitir a modelagem desta linguagem. O foco da técnica é especificamente a linguagem, o problema em si não é abordado neste momento da engenharia de requisitos. Leite (Leite et al., 1997), argumenta que o conhecimento desta linguagem deve ser o primeiro passo na fase de elicitação de requisitos.

O processo proposto para construção do LEL é composto de quatro passos: (i) identificar as principais fontes de informação do UdI, (ii) identificar símbolos relevantes no UdI, (iii) elicitar o significado dos símbolos e (iv) validar o LAL resultante. Abaixo detalhamos as atividades compreendidas em cada um destes passos.

- **Identificar fontes de informação.** As fontes de maior importância do UdI são os atores e os documentos. A identificação dos atores corretos, que possam indicar os documentos e outras fontes de informação relevantes, é essencial nesta etapa.

- **Identificar símbolos relevantes.** Nesta etapa é produzida a primeira lista de símbolos. São propostas duas abordagens para construção desta lista: ler documentos e escutar os atores. Em ambos os casos, a principal heurística envolve anotar frases e termos que parecem ter um significado especial. O foco deve ser em listar os símbolos e não em entender como o software irá funcionar. Nesta etapa, os atores podem ser ouvidos através de observação ou entrevistas não estruturadas.

- **Elicitar o significado dos símbolos.** Assim como na etapa anterior, o foco não deve ser a descrição de funcionalidades, e sim a definição da semântica de cada símbolo. A técnica de elicitação indicada para esta etapa é a entrevista estruturada. A lista de símbolos previamente criada deve servir como guia na entrevista, e perguntas diretamente relacionadas ao significado dos símbolos devem ser incluídas no roteiro.

- **Validar o LAL**. Duas técnicas complementares são utilizadas para validação. Uma baseia-se na checagem informal, onde o LAL é validado com a ajuda dos atores do UdI. A outra envolve uma análise das entradas do LAL a fim de detectar símbolos esquecidos.

Como resultado do processo, cada símbolo elicitado será identificado por um nome e descrito através de noções (denotação) e impacto (conotação). Na noção, frases curtas descreverão o significado literal do símbolo no UdI. No impacto, serão descritos os efeitos do uso e ocorrência do símbolo no UdI. Adicionalmente, os símbolos podem possuir sinônimos e devem ser classificados, gramaticalmente, como sujeito, estado, objeto ou verbo, como mostra o modelo ER apresentado na Figura 1. Para exemplificar, na Tabela 1 apresentamos a descrição de um símbolo do LAL pertencente ao UdI da ferramenta C&L (C&L, 2013a).

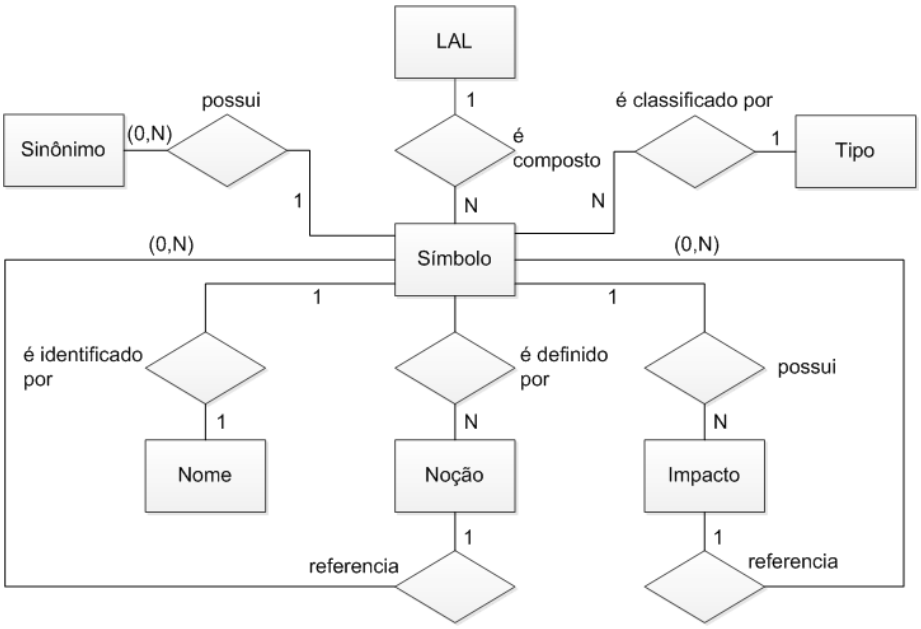

Figura 1. Modelo Entidade Relacionamento do LAL, adaptado de (Leite e Franco, 1993).

Nome	Projeto
Classificação	Objeto
Noção	- Conceito utilizado para representar um UdI específico dentro do software. - Símbolos e cenários podem ser criados no projeto.
Impactos	- O usuário pode criar um projeto. - O usuário pode editar um projeto. - O usuário pode adicionar símbolos e adicionar cenários a um projeto. - O usuário pode exportar um projeto. - O usuário pode gerar o grafo do projeto.

Tabela 1. Exemplo de símbolo do LAL pertencente ao UdI do C&L.

Outra característica do LAL é utilização de dois princípios básicos, que guiam a descrição de seus símbolos. O princípio da circularidade estabelece que, na descrição da noção e impacto de um símbolo, deve-se utilizar ao máximo outros símbolos pertencentes ao LAL. Por exemplo, na descrição do símbolo projeto, apresentada na Tabela 1, o símbolo usuário é utilizado em todas as sentenças que descrevem o impacto de projeto. De forma complementar, o vocabulário mínimo estabelece que, ao descrever a noção e o

impacto de um símbolo, deve-se minimizar o uso de termos externos ao LAL. Caso seja necessário o uso de termos externos, deve-se optar por aqueles mais simples, com significado claro. Na descrição do impacto do símbolo projeto (Tabela 1), por exemplo, o único termo externo utilizado foi a palavra "pode".

Heurísticas foram criadas para ajudar a descrição da noção e impacto dos símbolos de acordo com sua classificação gramatical. Tais heurísticas foram descritas através de perguntas, como mostra a Tabela 2. Um de seus propósitos é enfatizar o uso do princípio da circularidade. De acordo com a heurística para descrição da noção de um objeto, por exemplo, devemos identificar com quais outros objetos ele se relaciona. Desta forma, estamos criando relacionamentos entre símbolos do tipo objeto. De forma semelhante, quando o impacto de um símbolo do tipo sujeito é descrito, devemos enumerar as ações que ele pode executar. Neste caso, estamos estabelecendo relacionamento entre sujeitos e verbos.

Classificação	Noção	Impacto
Sujeito	- Quem é o sujeito?	- Quais ações pode executar?
Verbo	- Quem executa a ação? - Quando ela acontece? - Quais atividades são executadas?	- Quais os reflexas desta ação no ambiente? - Quais estados são alcançados a partir da ação?
Objeto	- Como o objeto pode ser definido? - Com quais outros objetos se relaciona?	- Quais ações podem ser aplicadas ao objeto?
Estado	- O que o estado significa? - Quais ações levaram a este estado?	- Quais outros estados e ações podem ocorrer a partir deste?

Tabela 2. Heurísticas para descrição do LAL.

Como consequência da utilização do princípio da circularidade, obtemos um conjunto de símbolos ligados através de uma rede complexa de

relacionamentos. Esta rede pode ser interpretada como um grafo, onde cada símbolo é representado através de um vértice. Os relacionamentos entre dois símbolos são representados através de arestas ligando dois vértices. A Figura 2 apresenta o grafo de um subconjunto dos termos do LAL referente ao UdI do software C&L.

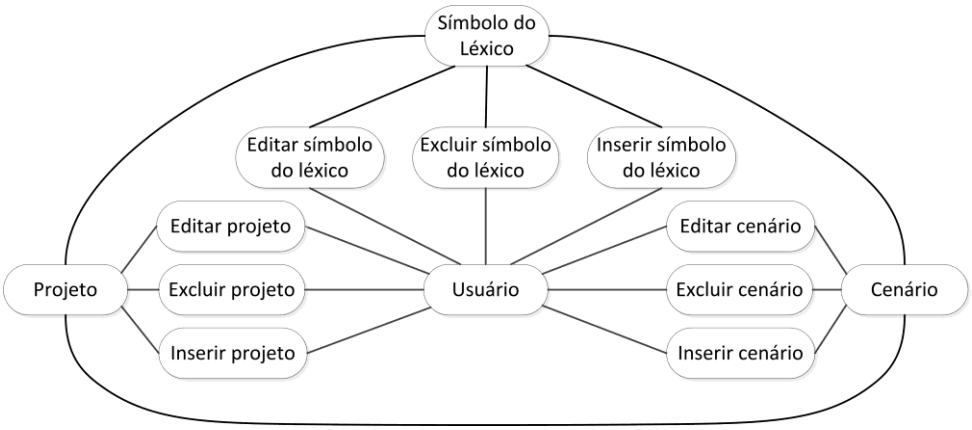

Figura 2. LAL do software C&L representado através de um grafo.

2.2. Cenários

A fim de definir os requisitos, os engenheiros devem compreender e modelar UdI do software. Uma vez modelado o conhecimento, os usuários interagem com engenheiros para verificar se o entendimento foi correto e validar os modelos construídos. Portanto, é essencial que a comunicação entre usuário e engenheiros seja facilitada, de modo que o processo de definição de requisitos seja realizado adequadamente. Esta motivação levou a pesquisa de métodos que ajudassem na interação entre os envolvidos durante o processo de definição dos requisitos. Uma das técnicas criadas através destas pesquisas foi a descrição através de cenários.

Segundo Leite (Leite, 1997), no contexto da engenharia de requisitos, os cenários são entendidos como uma técnica que é focada tanto na descrição do processo quanto centrada no usuário. Está técnica é muito utilizada na engenharia de requisitos, pois ajuda os engenheiros a entender melhor o software e sua interface com o ambiente. A interação com os envolvidos é facilitada pelo uso da linguagem natural na descrição dos cenários. Isto permite que os envolvidos compreendam melhor os modelos produzidos, fazendo com que se sintam mais confortáveis ao interagir com os engenheiros.

No âmbito da engenharia de requisitos, vários modelos de cenários foram propostos, desde os mais formais (Hsia et al., 1994), que utilizam linguagem formal e permitem a geração e validação automáticas, até os informais (Carroll et al., 1994; Rosson e Carroll, 2002), que são descritos sem uma estrutura definida. O modelo adotado em nossa pesquisa pode ser classificado como intermediário, pois visa à descrição de uma situação específica do software, através de linguagem natural semi-estruturada (Leite et al. 1997). Este modelo foi escolhido, pois sua estrutura conduz a um detalhamento mais preciso e maior das situações que são descritas, facilitando o entendimento. Além disto, o modelo propõe a utilização de relacionamentos, a fim de interligar os diversos cenários produzidos, o que também facilita o entendimento, e permite uma análise da interdependência dos cenários. A estrutura estabelecida para este modelo é composta dos seguintes elementos: título, objetivo, contexto, recursos, atores, episódios, exceções e restrições, como mostra a Figura 3.

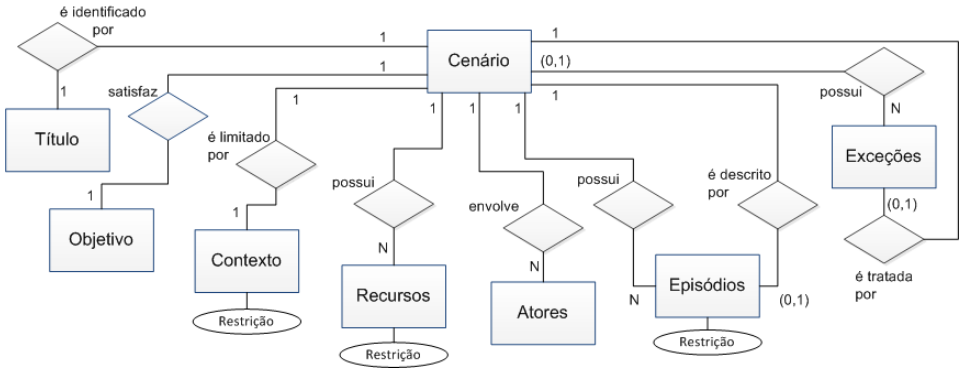

Figura 3. Modelo Entidade Relacionamento da estrutura de cenários, adaptado de (Leite et al., 2000).

O título serve como identificador do cenário dentro do UdI, portanto deve ser único. A finalidade do cenário é descrita, de forma sucinta, através de seu objetivo, que também deve conter breve informação de como a ela é alcançada. O contexto do cenário é descrito através da localização geográfica, localização temporal ou precondições. Pelo menos um dos itens citados anteriormente deve estar presente na descrição. A localização geográfica indica se o cenário requer uma posição geográfica específica para que possa ser executado. A localização temporal pode ser utilizada para delimitar um intervalo de tempo ou informar a periodicidade com que o cenário ocorre. E, por fim, temos as precondições, que servem para estabelecer os requisitos necessários para execução do cenário.

Os atores são as entidades ativas do cenário, que utilizam os recursos, entidades passivas, para realizar as ações necessárias a fim de satisfazer o objetivo. Para serem válidos, os atores e recursos devem aparecer em pelo menos um dos episódios do cenário.

Os episódios descrevem ações executadas pelos atores com o uso dos recursos. São apresentados de forma sequencial e podem ser de três tipos distintos: simples, condicionais ou opcionais. Os simples são aqueles indispensáveis para que o objetivo do cenário seja alcançado. Os condicionais dependem da ocorrência de uma condição específica para serem executados. As condições internas de um cenário podem ocorrer devido a precondições, alternativas, restrições de atores ou recursos e episódios anteriores. Por fim, temos os opcionais, que podem ocorrer ou não, dependendo de condições que não podem ser explicitamente detalhadas.

Título	Criar projeto no C&L
Objetivo	Permitir a representação de um UdI específico dentro do software C&L, com a finalidade de agrupar símbolos do léxico e cenários através da ação de inserir projeto.
Contexto	- Usuário deve possuir conta no software. - Executar cenário AUTENTICAR USUÁRIO.
Recursos	Nome do projeto, descrição, formulário para cadastro de projeto, menu principal. **Restrição:** O formulário para cadastro deve ser simples.
Atores	Usuário e software.
Episódios	1. Usuário solicita a inclusão de um novo projeto no software através de comando no menu principal. **Restrição:** o comando para criar projeto deve ser intuitivo. 2. Software exibe formulário com os campos nome e descrição do projeto. 3. Usuário informa nome e descrição do projeto. **Restrição:** os dados devem estar completos. 4. Usuário solicita criação do projeto. 5. Software analisa os dados informados pelo usuário. 6. Software cria novo projeto e exibe mensagem para o usuário, informando que o projeto foi criado com

	sucesso. **Restrição:** mensagem exibida dever ser informativa.
Exceção	- Se o usuário informar um nome para o projeto que já exista na base de projetos, então o software informa que o nome escolhido já existe e um nome diferente precisa ser informado. - Se o usuário deixar algum dos campos do formulário em branco, o software não cria o projeto e emite uma mensagem informando ao usuário que os dados são obrigatórios.

Tabela 3. Exemplo de cenário pertencente ao UdI do C&L.

O atributo restrição pode estar relacionado ao contexto, recursos ou episódios de um cenário. Este atributo é utilizado para descrever aspectos não funcionais que podem interferir na qualidade com que o objetivo de um cenário é alcançado. É importante ressaltar que este atributo não impede a satisfação do objetivo do cenário. No cenário apresentado na Tabela 3, por exemplo, há a restrição "*os dados devem estar completos*" associada ao episódio três. Neste caso, mesmo que o usuário informe parcialmente os dados, o objetivo do cenário será alcançado, isto é, o projeto será criado. Porém, a qualidade da representação do UdI será menor se os dados informados forem muito superficiais.

Uma exceção pode interromper o fluxo normal de execução dos episódios de um cenário. Cada exceção deve ser descrita através de uma sentença simples que deve identificar a causa da interrupção e como tratá-la. O tratamento da interrupção não precisa necessariamente fazer com que o objetivo do cenário seja alcançado.

Uma característica importante dos cenários é a existência de relacionamentos entre eles. Segundo Breitman (Breitman, 2000), cenários estão ligados a outros cenários através de elos, formando uma complexa rede de relacionamentos. Estes elos podem ser de quatro tipos distintos: subcenário, precondição, exceção e restrição.

Um cenário é considerado subcenário de outro quando aparece em pelo menos um de seus episódios. Este relacionamento é útil quando:

- Um comportamento comum é detectado em vários cenários do mesmo UdI. Neste caso, o comportamento é isolado em um novo cenário e este passa a ser referenciado pelos outros que possuíam o comportamento. Desta forma é possível reduzir a redundância de informações e, consequentemente, todos os problemas decorrentes dela.

- Há um curso de ação, que pode ser alternativo ou condicional, dentro de uma situação do UdI. Quando isto ocorre, podemos separar este comportamento em um novo cenário, o que possibilita um melhor detalhamento e, consequentemente, facilita sua compreensão.

- É preciso melhorar a descrição de uma situação que possui um objetivo importante e bem definido dentro de sua descrição. Diante disto, é possível destacar este objetivo e criar um novo cenário para detalhá-lo, facilitando desta forma tanto sua descrição quanto entendimento.

A precondição é um relacionamento definido dentro do componente contexto de um cenário, isto é, um cenário que atua como precondição para outro deve aparecer dentro de seu contexto. Este relacionamento nos permite estabelecer de forma objetiva uma ordem entre os cenários, possibilitando a especificação de estágios que devem ser completados antes da execução de outros. Um exemplo deste relacionamento pode ser visto no cenário descrito na Tabela 3, onde o cenário "*autenticar usuário*" é precondição para "*criar projeto no C&L*". Para destacar o relacionamento, o título do cenário que é referenciado é escrito com letras maiúsculas.

Relações de exceção acontecem quando um cenário aparece no componente exceção de outro. Este relacionamento nos permite detalhar melhor tratamentos de exceção complexos através de outro cenário. Desta forma, contamos com toda a estrutura de um cenário para detalhar o tratamento, ao invés de resumi-lo em uma sentença simples.

O relacionamento de restrição é estabelecido quando utilizamos um novo cenário com o objetivo de detalhar aspectos não funcionais que restringem o funcionamento correto de outro cenário. Este relacionamento ocorre no atributo restrição, que pode estar associado a recursos, contexto e episódios.

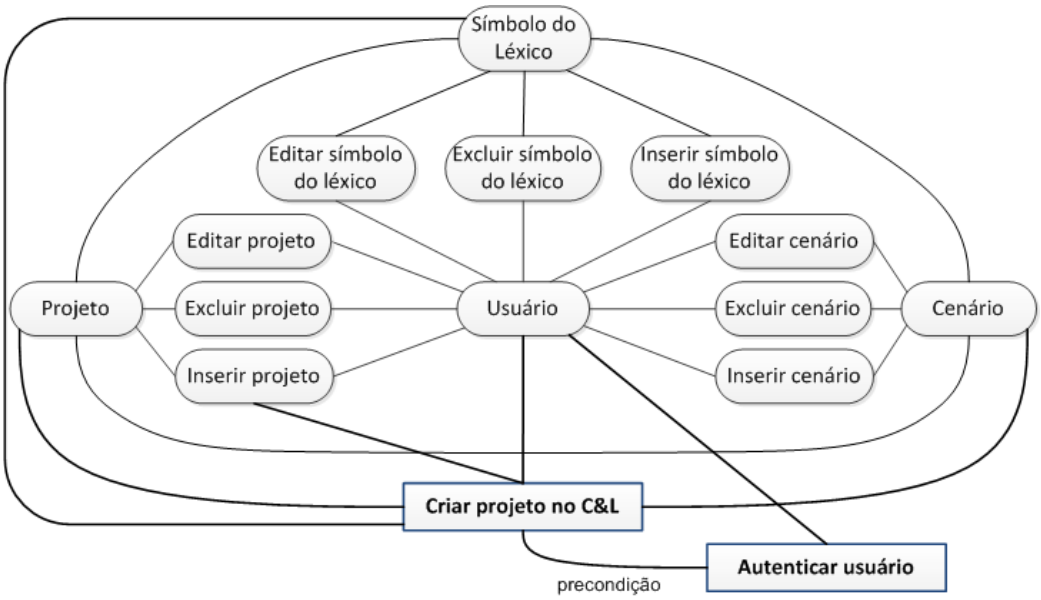

Figura 4. Grafo com símbolos do LAL e cenários.

O processo de construção de cenários para descrever situações do UdI está intimamente relacionado ao LAL, técnica descrita na seção 2.1. Esta relação existe, pois, ao descrever os cenários, é natural que os envolvidos utilizem a linguagem do UdI, que é justamente o alvo da técnica LAL. Desta forma, além de elos entre cenários, temos também elos entre cenários e símbolos do LAL, facilitando ainda mais a compreensão dos cenários. Os símbolos do LAL utilizados na descrição do cenário da Tabela 3 estão sublinhados. Desta forma, por exemplo, se quiséssemos compreender melhor o significado do termo "*software*" no UdI do C&L, bastaria olhar a definição destes símbolos no LAL.

Extrapolando a ideia exposta na seção 2.1, de desenhar um grafo utilizando os símbolos como vértices e os relacionamentos como arestas, podemos desenhar um grafo contendo cenários e símbolos do LAL. A Figura 4 representa o grafo da Figura 2 com a adição de dois cenários, que estão destacados juntamente com seus relacionamentos. O relacionamento entre dois cenários também é representado através de uma aresta, que neste caso possui o nome do relacionamento, como podemos ver no caso dos cenários "*autenticar usuário*" e "c*riar projeto no C&L*", onde o primeiro é precondição para o segundo.

2.3.
Arquitetura Model View Controller (MVC)

O padrão arquitetural MVC foi proposto por Reenskaug (Reenskaug, 1979). Sua principal característica é uma organização, que promove a separação da interface, lógica e dados do software. Reenskaug propôs a camada de Controle (*Controller*) como um elo entre o usuário e o software. Este elo permite o envio de informações do usuário para o software através de comandos e dados. Também permite o fluxo inverso, isto é, a saída de informações através da apresentação das interfaces apropriadas. A camada de Modelo (*Model*), por sua vez, é utilizada para representar a base de conhecimento do software e armazenar o seu estado corrente. Por fim, a camada de Visão (*View*) abarca toda a representação visual da camada de modelo. Componentes desta camada podem filtrar os dados obtidos da camada de Modelo, escondendo alguns atributos e destacando outros. Os dados apresentados são obtidos diretamente da camada de Modelo. Componentes da Visão também podem atualizar a camada de Modelo, caso seja necessário. A arquitetura descrita anteriormente está representada na Figura 5.

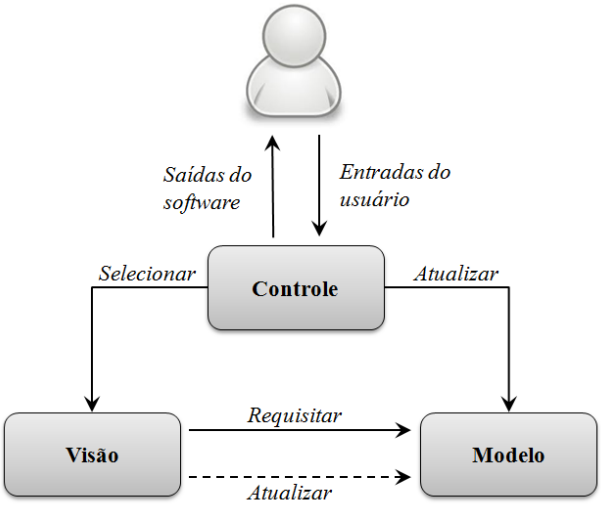

Figura 5. Diagrama da arquitetura MVC adaptado de (Reenskaug, 1979).

Uma das primeiras linguagens a explorar este padrão arquitetural foi smalltalk-80 (Krasner e Pope, 1988). Segundo Krasner, esta divisão implica na separação (i) das partes que representam o modelo do domínio do software (ii) da forma com que ele é apresentado paro o usuário e (iii) do modo com que o usuário interage com o software (Krasner e Pope, 1988). Esta organização é interessante, pois, segundo o próprio autor, a separação destes conceitos facilita

o entendimento e a modificação de um componente, sem que seja necessário conhecer o restante do software.

O padrão arquitetural MVC empregado neste trabalho é diferente do explicado anteriormente, porém, a motivação para sua utilização é a mesma. Como o contexto desta pesquisa é restrito a software Web, adotamos uma variante do MVC que implementa a arquitetura em três camadas, comum neste domínio. A principal diferença para a arquitetura proposta originalmente é a adoção do conceito de camadas. Assim, cada camada da arquitetura só pode se comunicar com aquelas adjacentes, como mostra a Figura 6. Além disto, esta proposta também engloba o acesso ao banco de dados do software. Normalmente, a camada que contempla os serviços relacionados a acesso ao banco de dados é utilizada na forma de uma biblioteca, como representado através da caixa DAL (*Data Access Library*) na Figura 6.

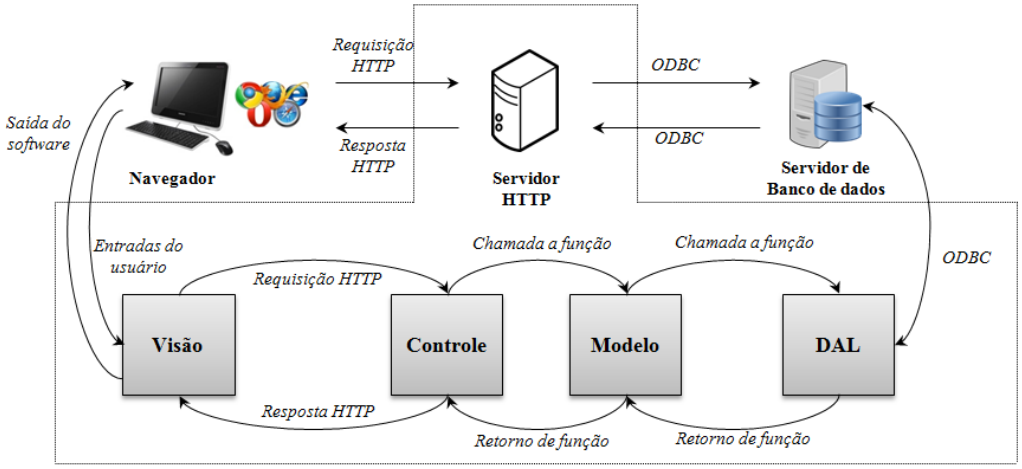

Figura 6. Arquitetura MVC proposta.

Devido às mudanças propostas, foi necessário mudar a definição de cada camada, a fim de adaptá-las a nova abordagem. A seguir, descrevemos o propósito de cada camada dentro desta nova abordagem.

- **Visão.** Camada com a qual o usuário irá interagir diretamente. Os dados ou comandos recebidos do usuário são repassados pela camada de visão ao controle através de requisições HTTP. As repostas recebidas através destas requisições são exibidas para o usuário através de arquivos HTML, código Java Script e folhas de estilo (CSS).

- **Controle.** Recebe as requisições da camada de Visão. Contém a lógica do software e, de acordo com a lógica, pode requisitar ou atualizar dados da camada de Modelo. A comunicação entre Controle e Modelo é realizada através de chamadas/retorno de função.

- **Modelo.** Encapsula o modelo conceitual do domínio. É a camada responsável por manter o estado do software. Estabelece comunicação com a base de dados através da biblioteca de acesso a dados (DAL). A comunicação entre Modelo e DAL é realizada através de chamada/retorno de função.

- **DAL (*Data Access Library*).** Contempla todos os serviços para acesso ao banco de dados do software. Comunica-se com o banco de dados através de API específica para linguagem de programação utilizada no desenvolvimento.

2.4. Catálogo de Transparência

É sabido que construir software é uma atividade complexa devido a muitos fatores, como, por exemplo, o envolvimento de um grande número de pessoas com conhecimentos variados. Outro motivo para tal complexidade é que, além de estar de acordo com os requisitos funcionais, o software precisa satisfazer metas de qualidade, que são, normalmente, modeladas como requisitos não funcionais (RNF). A dificuldade em lidar com tais metas está em sua natureza subjetiva e no fato de que possuem um impacto amplo no software.

Com o objetivo de lidar com esta complexidade, muitas técnicas foram propostas para auxiliar no processo de desenvolvimento de software. Estas técnicas abrangem tanto requisitos funcionais quanto os não funcionais (Jackson, 2001; Chung et al., 2000; Bresciani e Perini, 2004; Lamsweerde, 2009). As abordagens orientadas a metas (Yu, 1996; Lamsweerde, 2009) propõem modelar o software através de suas metas e seus relacionamentos. Elas dão suporte à tomada de decisão e permitem a avaliação do impacto destas decisões. Estas técnicas promovem as metas de qualidade a entidades de primeira classe, estimulando os desenvolvedores a perseguir sua satisfação

durante o desenvolvimento do software. Normalmente, estes modelos são produzidos exclusivamente para um software, sem que haja a preocupação de reutilizar o conhecimento que foi adquirido. Entretanto, engenheiros de software se beneficiariam de um conjunto de soluções premontadas para lidar com metas de qualidade. Um caminho importante a ser seguido neste caso é o desenvolvimento de **catálogos**, que listem as possíveis operacionalizações para estas metas de qualidade.

O *framework* para RNF (*NFR framework*), proposto por Chung (Chung et al., 2000) endossa as abordagens orientadas a metas. Em seu trabalho, Chung modela RNFs como metas flexíveis (*softgoals*) que podem entrar em conflito uma com as outras. Portanto, estas devem ser detalhadas e discutidas durante os estágios iniciais do desenvolvimento de software, com o objetivo de se chegar a soluções aceitáveis, que atendam as expectativas dos envolvidos. O principal modelo do *NRF framework* é o *Softgoal Interdependency Graph* (SIG), que é composto de vértices e elos, utilizados para representar as metas flexíveis e seus relacionamentos. Os vértices podem representar três tipos de elementos: metas flexíveis, operacionalizações e afirmações. O primeiro é utilizado para representar os RNFs. O segundo para representar possíveis operacionalizações que satisfazem a contento um RNF. O último é utilizado para justificar as decisões tomadas com base no SIG.

No *NFR framework* as metas flexíveis são descritas por seu tipo e tópico. Seu tipo indica a qualidade que está sendo representada e o tópico o contexto em que é aplicada. No contexto de desenvolvimento de software, por exemplo, poderíamos ter um *NFR softgoal* com tipo Rastreabilidade e o tópico Processo de Desenvolvimento de Software. Os relacionamentos de interdependência dentro do NFR framework podem ser explícitos ou implícitos. O primeiro representa refinamentos ou contribuições, dependendo da direção do relacionamento. Aqueles direcionados para baixo representam uma relação de refinamento, onde uma meta flexível é detalhada por duas ou mais metas flexíveis descendentes. As contribuições são representadas por relacionamentos direcionados para cima e identificam o grau - que pode ser totalmente ou parcialmente e positivo ou negativo - com que o descendente pode contribuir para a satisfação a contento da meta flexível pai.

Uma das principais características do *NFR framework* são os catálogos. O *framework* disponibiliza uma estrutura que permite organizar o conhecimento, fazendo com que os desenvolvedores possam reutilizar o que foi aprendido em experiências anteriores. Inicialmente, o *NFR framework* define três tipos distintos

de catálogo: Tipo, Método e Correlação. Um catálogo Tipo permite a representação estruturada de conhecimento sobre uma meta flexível específica, sem que um tópico seja definido. No SIG de Transparência (Figura 8), por exemplo, a meta flexível transparência é decomposta em Acessibilidade, Usabilidade, Entendimento, Auditabilidade e Informativo. A meta flexível acessibilidade, por sua vez, é decomposta em Portabilidade, Disponibilidade e Publicidade.

O catálogo Método permite a organização de catálogos contendo técnicas de desenvolvimento, obtidas através da decomposição de metas flexíveis, instanciação de operacionalizações e identificação de afirmações. Em seu trabalho, Chung (Chung et al., 2000) exemplifica o uso do catálogo Método através do refinamento "*and"* da meta flexível acurácia aplicada ao tópico conta de banco (Acurácia [Conta]) em Acurácia [ContaRegular], Acurácia [ContaGold] e Acurácia [ContaPremier]. Por fim, o catálogo Correlação permite a construção de catálogos para inferir as possíveis interações entre metas flexíveis, tanto as positivas quanto as negativas. No SIG de Transparência (Figura 8), por exemplo, fica claro que a meta flexível Concisão contribui negativamente para a meta flexível Detalhamento. Um SIG é composto de informações dos três tipos de catálogo descritos. A Figura 8 mostra o SIG de transparência adaptado de (Cappelli, 2009), onde é possível observar a estrutura proposta pelo *NFR framework* para organizar as informações.

O conhecimento a cerca de metas flexíveis compreende, entre outros, o seu refinamento e operacionalização, contribuições entre metas flexíveis, questões que ajudam sua operacionalização e pontos de vista e organização de questões em categorias. Como dito, apenas o SIG não é o suficiente para representar este vasto e complexo conhecimento. Portanto, é necessária a complementação do SIG com outros artefatos de requisitos. Porém, a representação de informações em diferentes artefatos a tornaria mais difícil de utilizar, manter e evoluir. Para lidar com este problema e instanciar o SIG para o tópico [Software], criando desta forma o Catálogo de Transparência de Software (Catálogo de Transparência de Software, 2013) o GrupoER (GrupoER, 2013) propôs organizar o catálogo SIG como uma coleção de padrões RNF.

Os padrões RNF foram propostos por Supakkul (Supakkul et al., 2010), com o propósito de reusar o conhecimento sobre RNFs. Em seu trabalho, Supakkul define três padrões: Objetivo, Problema, Alternativa e Seleção. O padrão Objetivo é utilizado para capturar a definição de RNFs em termos de metas flexíveis. Os problemas e obstáculos para satisfazer a contento as metas

flexíveis são representados através do padrão Problema. Os diferentes meios de satisfazer a contento as metas flexíveis e seus efeitos colaterais são descritos pelo padrão Alternativa**.** Finalmente, temos o padrão Seleção, que dá suporte à escolha de alternativas, utilizando analises qualitativas ou quantitativas. Nos padrões RNF todo o conhecimento é representado através de uma estrutura composta de três partes: Inicial, Resultado e Regras de Refinamento. A parte Inicial captura os conceitos chaves, fornecendo o contexto necessário para o reuso. O Resultado representa o conhecimento disponível sobre o RNF em questão. As Regras de Refinamento capturam os refinamentos de modelos que são feitos manualmente durante uma típica sessão de engenharia de requisitos, onde novos elementos são adicionados ao modelo e relacionados com os já existentes. Além dos padrões e regras, propostos por Supakkul, O GrupoER propôs a criação de um novo padrão, chamado Questão. Duas novas Regras de Refinamento também foram adicionadas àquelas propostas por Supakkul: *GroupIdentification e QuestionIdentification.*

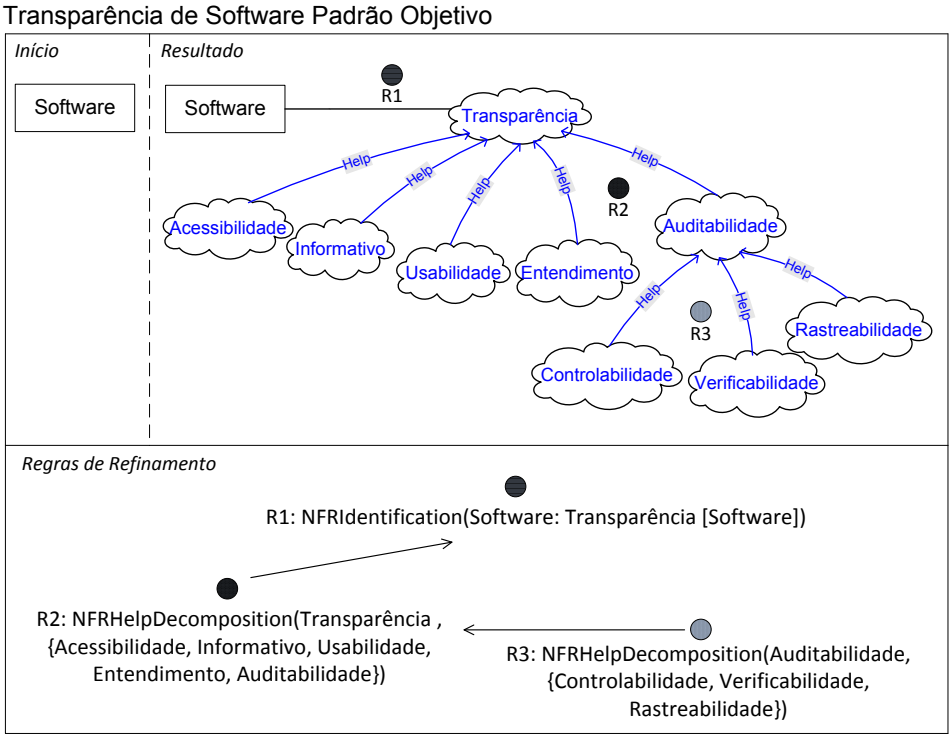

Figura 7. Fragmento do SIG de Transparência representado através do padrão *Objetivo.*

No Catálogo de Transparência de Software, o SIG foi representado através do padrão Objetivo. A seção Inicial do padrão define o tópico, que neste caso é

[Software]. A seção Resultado, por sua vez, identifica a meta flexível principal e sua decomposição. O modelo apresentado nesta seção é alcançado através das Regras de Refinamento: *NFRIdentification* e *NFRDecomposition.* Na Figura 7, o padrão Objetivo identifica a Transparência no contexto de software (Regra de Refinamento R1) e então a meta flexível Transparência é decomposta em Acessibilidade, Informativo, Usabilidade, Entendimento e Auditabilidade (Regra de Refinamento R2). Por último, a meta flexível Auditabilidade é refinada em Controlabilidade, Verificabilidade e Rastreabilidade (Regra de Refinamento R3). Deste modo, fica claro que utilizando as regras *NFRIdentification* e *NFRDecomposition*, é possível descrever todo o SIG de Transparência através do padrão Objetivo.

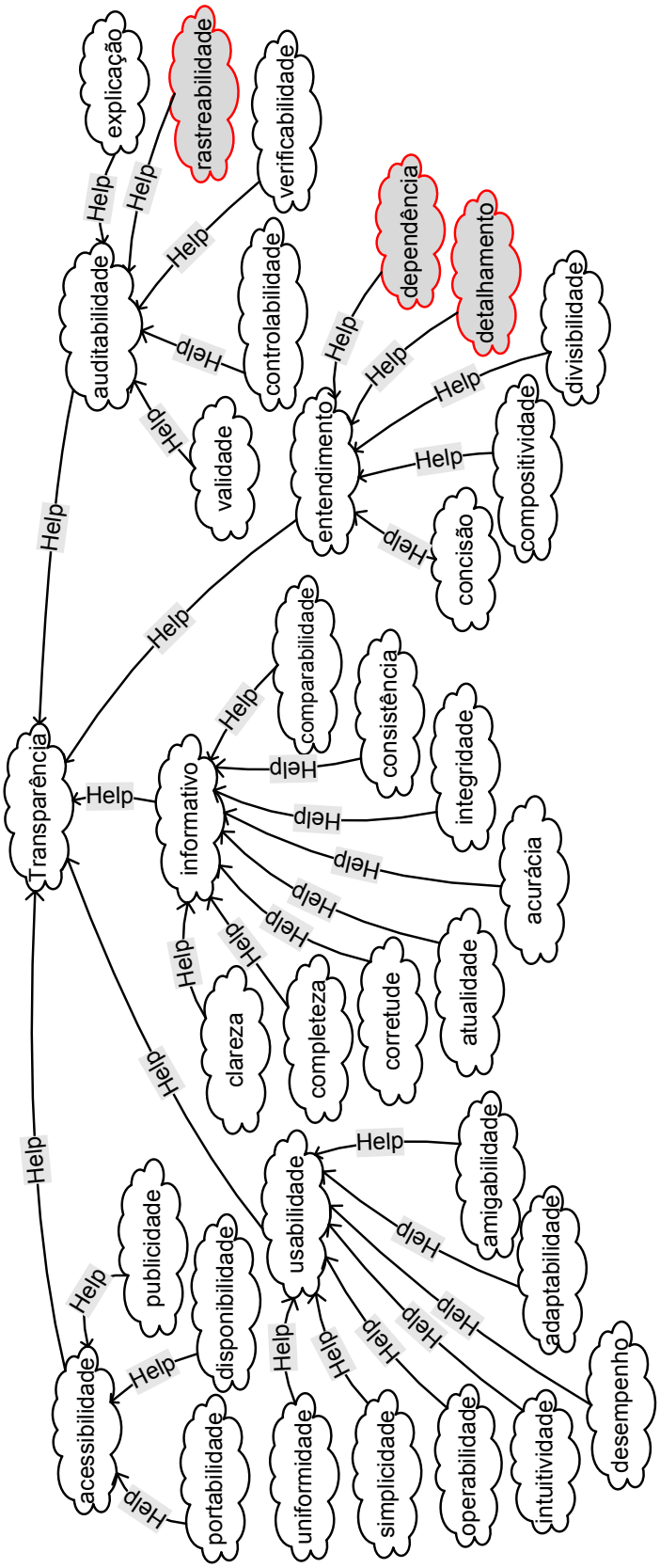

Figura 8. SIG de Transparência, adaptado de (Cappelli, 2009).

É necessário um cuidado especial quando refinamos uma meta flexível através de elos de contribuição do tipo *Help*. Este tipo de elo significa que satisfazer a contento uma meta flexível descendente, implica em uma contribuição positiva a meta flexível pai. Todavia, não satisfazê-la a contento não impede, necessariamente, a satisfação a contento da meta flexível pai. Porém, há alguns casos especiais em que isso não é verdade. Analisando o primeiro nível de metas flexíveis do SIG de Transparência, podemos perceber que a meta flexível Acessibilidade é um destes casos especiais. A Transparência de Software não pode ser alcançada sem que o software esteja acessível. Portanto, neste caso, a impossibilidade de satisfazer a contento a meta flexível Acessibilidade (descendente) deve levar ao impedimento da Transparência de Software (pai). Para descrever este tipo de situação, o padrão Problema foi utilizado. Este padrão foi originalmente criado por Supakkul (Supakkul et al., 2010) para capturar o conhecimento sobre problemas especiais, que poderiam obstruir a satisfação a contento de uma meta flexível ou de uma de suas operacionalizações.

O Catálogo de Transparência de Software (CTS) também inclui uma lista de operacionalizações para metas flexíveis folha (aquelas que não têm descendentes) e mapeia o seu impacto em outras metas flexíveis. A principal contribuição de se representar operacionalizações no SIG é o reuso, já que ajudam a construção e avaliação de novos software. Com o objetivo de definir quais operacionalizações eram importantes para a meta flexível principal, o método Goal-Question-Metric (GQM) (Basili, 1992) foi adaptado para o Goal-Question-Operationalization (GQO) pelo GrupoER (GrupoER, 2013). O método GQM se inicia com metas para o software (ou desenvolvimento do software), define questões para assegurar a satisfação destas metas e cria métricas para permitir que as questões sejam respondidas de forma quantitativa. Em sua adaptação, as métricas foram trocadas por operacionalizações. O GQO segue uma abordagem de boas práticas, onde questões são agrupadas por categoria, usando como base a proximidade de conceitos. Estas questões são respondidas através da identificação de boas práticas da engenharia de software, que operacionalizam as metas flexíveis folha. As questões e operacionalizações fornecem pontos de variabilidade e extensão para o catálogo.

Como visto anteriormente, nosso foco neste trabalho será três metas flexíveis específicas do catálogo, que são: Rastreabilidade, Dependência e Detalhamento. Através de uma análise das questões relacionadas a estas metas flexíveis, presente no CTS foi possível estabelecer sua relação com os

problemas de rastreabilidade entre artefatos de um software, definição de sua organização modular e entendimento do software como um todo. Para uma melhor identificação, destacamos estas metas flexíveis na Figura 8. Através da Figura 9, apresentamos o padrão *Questão* aplicado a Rastreabilidade. Como É possível observar, foram criadas três categorias para organizar as questões relacionadas à meta flexível: pré-rastreabilidade, rastreabilidade em tempo de desenho e rastreabilidade em tempo de execução.

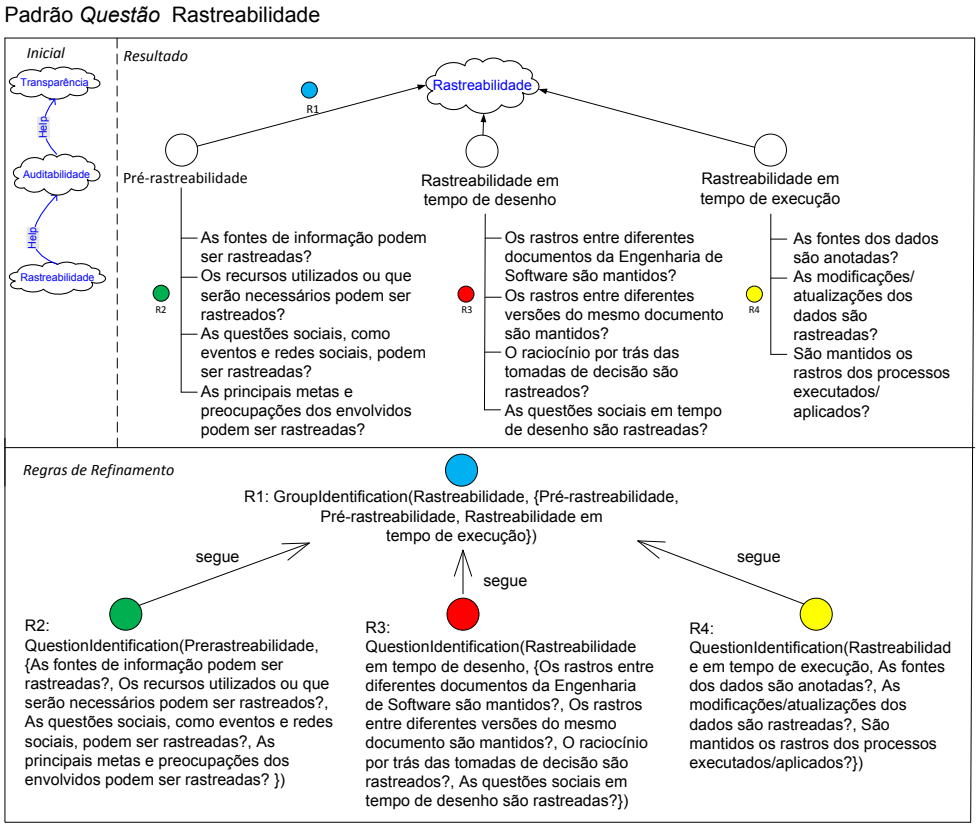

Figura 9. Padrão Questão de Rastreabilidade.

Na Figura 10, apresentamos as categorias e questões referentes à meta flexível Dependência, que foi detalhada através de três categorias: informação, acoplamento e coesão. Por fim, na Figura 11, exibimos as questões relacionadas à meta flexível Detalhamento. Estas questões foram agrupadas através de três categorias: usar estruturação de dados (variáveis) / funções (comandos), usar nomes adequados e explicar o software. As questões e categorias associadas a todas as outras metas flexíveis podem ser consultadas em (Catálogo de Transparência de Software, 2013).

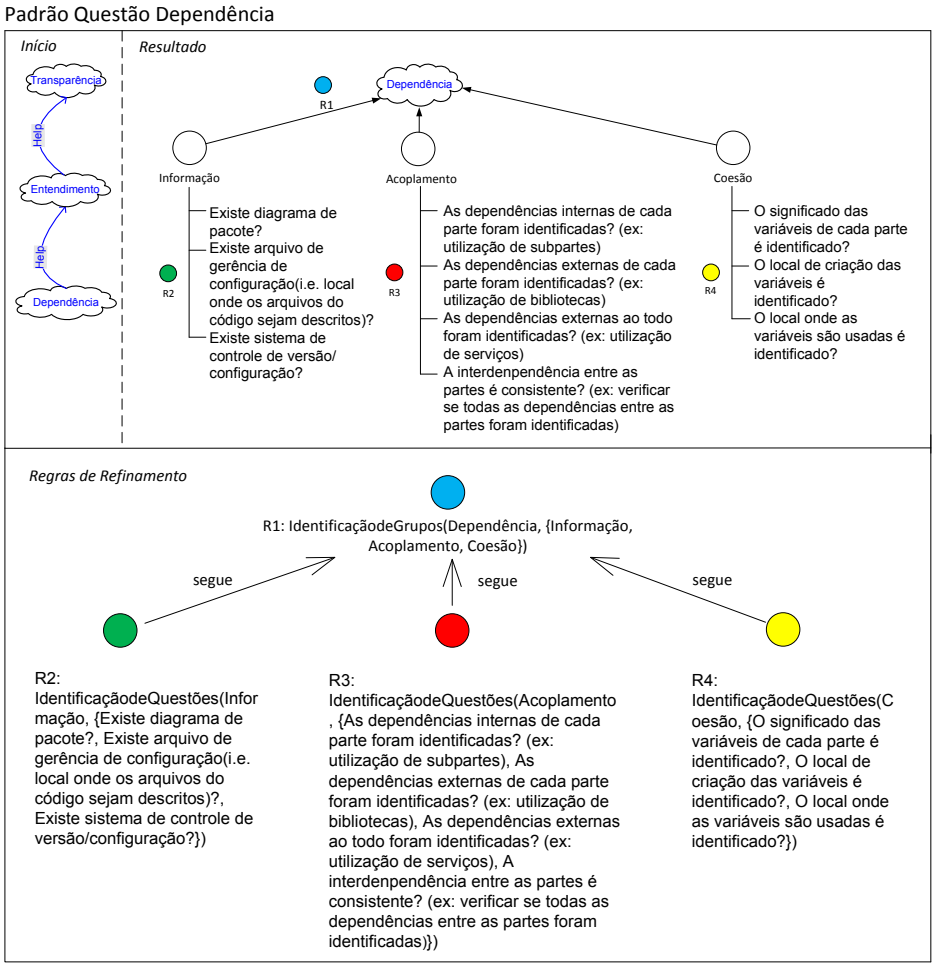

Figura 10. Padrão Questão de Dependência.

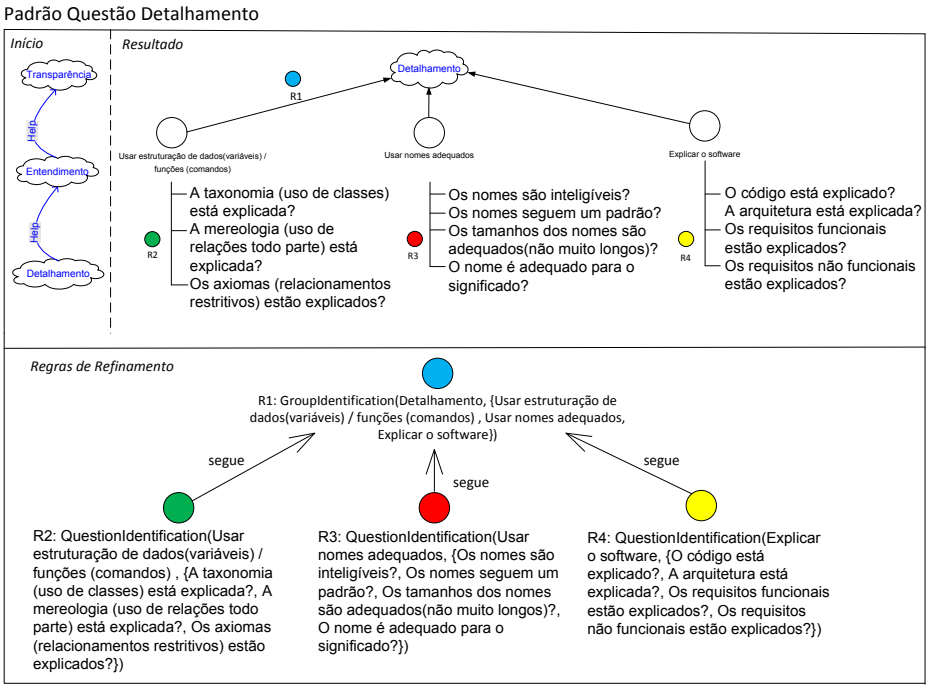

Figura 11. Padrão Questão de Detalhamento.

2.5. Conclusão

Neste capítulo, foram apresentados os conceitos que serviram como base para a construção do processo PDS+T, que é apresentado a seguir. O Léxico Ampliado da Linguagem e os cenários são duas técnicas da engenharia de requisitos que possuem propósitos distintos. A primeira, foca na descrição da linguagem utilizada pelos atores do UdI, sem se preocupar em descrever o problema em si. A segunda tem como objetivo o detalhamento do problema, através da descrição das situações do UdI. Esta descrição é realizada através de uma estrutura bem definida, que permite a identificação dos recursos e atores da situação e seu detalhamento passo a passo. Estas abordagens foram utilizadas em conjunto, pois têm uma ligação muito estreita, já que os cenários são descritos através do uso dos termos identificados no LAL. Também foi apresentado o padrão arquitetural MVC, no qual nos baseamos para criar a arquitetura dos software construídos a partir do processo PDS+T. Este padrão sugere a organização do software em três camadas, isolando a interface para interação com o usuário do modelo conceitual e regras de negócio implementadas pelo software. Por fim, descrevemos como surgiu o Catálogo de Transparência de Software (Catálogo de Transparência de Software, 2013), que tem o objetivo de centralizar as operacionalizações relacionadas às qualidades que decompõem a transparência, de forma que seu reuso seja facilitado.

No próximo capítulo, iremos explicar o processo PDS+T, que utiliza os conceitos explicados anteriormente, a fim de construir software com mais transparência. Com o intuito de organizar e facilitar o reuso do processo, mostraremos como suas operacionalizações foram anexadas ao Catálogo de Transparência de Software.