Universidade Federal de Santa Catarina

Departamento de Informática e Estatística

Curso de Ciências da Computação

Plug-in para criação semi-automática de rastreabilidade entre artefatos de software no ambiente Enterprise Architect, sob a ótica da Rastreabilidade Indutiva

Florianópolis

Dezembro, 2009

DIOGO DANTAS FONSECA DOS SANTOS

Plug-in para criação semi-automática de rastreabilidade entre artefatos de software no ambiente Enterprise Architect, sob a ótica da Rastreabilidade Indutiva

Trabalho de conclusão de curso apresentado como parte dos requisitos para obtenção do grau de Bacharel em Ciências da Computação.

Florianópolis

Dezembro, 2009

DIOGO DANTAS FONSECA DOS SANTOS

Plug-in para criação semi-automática de rastreabilidade entre artefatos de software no ambiente Enterprise Architect, sob a ótica da Rastreabilidade Indutiva

Este trabalho de conclusão de curso foi aprovado em sua forma final pelo curso de Ciências de Computação da Universidade Federal de Santa Catarina.

Orientador:

_____

Prof. Dr. Raul Sidnei Wazlawick

Banca Examinadora:

_____

Prof. Msc. Antônio Carlos Mariani

_____

Msc. Raquel Nitsche dos Santos

RESUMO

Este trabalho está fundamentado nos conceitos da técnica de Rastreabilidade Indutiva RI. Tal técnica tem por objetivo favorecer a rastreabilidade entre elementos e artefatos de modelagem, em um processo de criação de software, de modo indutivo. Ao final, será implementado um plug-in capaz de oferecer suporte semi-automático de rastreabilidade entre artefatos e elementos, ao longo da modelagem e desenvolvimento de software, no ambiente Enterprise Architect.

Palavras-chave: Modelagem de Software, Rastreabilidade, Técnica de Rastreabilidade Indutiva.

ABSTRACT

This work is based on concepts of the art of Software Engineering Inductive. This technique aims to facilitate the traceability between features and artifacts in a modeling process of creating software, so inductive. Finally, a tool will be implemented a plug-in capable of providing automatic support for traceability between artifacts and elements along the modeling and software development in the Enterprise Architect environment.

Keywords: Design and Modeling Software, Traceability, Software Engineering Technique of Inductive.

LISTA DE ABREVIATURAS E SIGLAS

RI: Rastreabilidade Indutiva

RR: Relações de Rastreabilidade

EA: *Enterprise Architect*

UML 2: *Unified Modeling Language*

CASE: *Computer-Aided Software Engineering*

COM: *Component Object Model*

PROJECT BROWSER: Componente de visualização dos elementos de modelagem em um projeto, no *EA*

TOOLBOX: Componente com todos diferentes tipos de elementos disponíveis para modelagem, no *EA*

LISTA DE FIGURAS

SUMÁRIO

1. INTRODUÇÃO

No âmbito da criação de software, obter um produto final que atenda as necessidades do cliente reflete no sucesso do produto. E isso denota dentre outros aspectos, a prática de um padrão de desenvolvimento (PRESSMAN, 1995) e qualidade, nas atividades de modelagem das informações do domínio do problema e de gerenciamento das fases (comumente: projeto, planejamento e do desenvolvimento) de criação.

Com o tempo, organizações com processos bem definidos e aplicados projetam e planejam com mais qualidade e maior rigor no cumprimento do prazo pré-estabelecido para o desenvolvimento de um produto que atenda as reais necessidades do cliente. Diminuir as chances com gastos extras para refazer o que não reflete a necessidade do cliente é uma maneira de evitar a prorrogação de prazos, aumento de custos e a insatisfação do cliente. Assim, desenvolvedores e empresas de criação de software passam a agregar confiabilidade a seus produtos e reconhecimento mercadológico, quando controlam bem esses fatores (SOMMERVILLE, 2000).

Assim como, a prática de processos, estabelecimentos de prazos, outra importante atividade é a modelagem dos aspectos e funcionalidades levantadas inicialmente e desenvolvidas durante fases posteriores de desenvolvimento. Documentar e organizar essas informações que refletem as decisões de planejamento e projeto é outro objetivo da prática de modelagem, a qual faz uso conjunto de descrição textual com os diversos elementos (artefatos de modelagem disponíveis em uma ferramenta *CASE*) previstos na *UML*.

A forma como são organizados artefatos e elementos, os quais abstraem características desejadas para o futuro software contribui essencialmente para que modificações e operações sob os mesmos possam ocorrer com segurança, sem gerar anomalias ou inconsistências. Dado o caráter evolutivo do desenvolvimento de software, há um gradual crescimento já esperado de especificações e alterações das mesmas (LARMAN, 2007). Assim, uma estrutura que reaja de forma coesa tanto com relação a esse aumento quanto às operações mais comuns podendo envolver elementos (criação, modificação, exclusão) traz benefícios como poderá ser constatado.

Para contribuir na estrutura ideal da organização descrita, encontra-se no conceito de rastreabilidade, uma solução viável a ser aplicada durante a evolução de um software quanto ao aspecto de “registrar” e representar possíveis relações entre elementos de modelagem, tornando-as também flexível a futuras alterações.

Mas para que ocorra sua efetiva aplicação no processo de desenvolvimento de software é preciso que haja um modelo de rastreabilidade. Assim, sob a ótica da técnica de Rastreabilidade Indutiva RI é possível a viabilidade de tal modelo, implementado nesse trabalho sob a forma de um plug-in o qual permite a criação semi-automática de relações de rastreabilidade RR entre elementos que são dependentes (de acordo com uma abstração modelada).

1.1 OBJETIVOS

Com base na contextualização apresentada na seção anterior, o objetivo geral deste trabalho é a implementação de um plug-in que permitirá a criação semi-automática de RR, segundo premissas e conceitos previstos no contexto da Técnica de Rastreabilidade Indutiva RI (SANTOS; WAZLAWICK, 2009) sobre os elementos e artefatos de modelagem do ambiente Enterprise Architect *EA*.

Para atender esse objetivo geral emergem os seguintes objetivos específicos:

- Entendimento e posterior aplicação do conteúdo da técnica de RI às RR entre elementos de modelagem que compõe a especificação de um software.

- Estudo do ambiente de modelagem de software *EA*, o qual permite o acoplamento de plug-ins clientes que sejam desenvolvidos na padrão ActiveX *COM*.

1.2 JUSTIFICATIVA

A maior justificativa para a realização do trabalho em questão é disponibilizar outra (o *EA* dispõe da matriz de relacionamento cujo enfoque é diferente da RI) opção de rastreabilidade

para a ferramenta *CASE EA*, entre os elementos que modelam os aspectos de um software, e que sejam rastreáveis (caso relações entre elementos sejam necessárias) nas especificações modeladas.

Sendo assim, a aplicação da técnica de RI, implementada em um *plug-in,* é o tema deste trabalho que pretende contribuir qualitativamente para prática de modelagem e desenvolvimento de software.

2. MODELAGEM DE SOFTWARE

A seguir é feita uma abordagem sobre a importância da modelagem de software, além de ressaltar alguns elementos de modelagem que serão usados em exemplos.

A prática de modelagem busca favorecer a captura e compreensão de requisitos e funcionalidades, descrever aspectos dinâmicos e estáticos de um software, documentar o desenvolvimento e melhorar a comunicação à fase de programação de código sobre o que deve ser desenvolvido. A clareza sobre os requisitos contribui para que o desenvolvimento seja bem sucedido (PRESSMAN, 1995), (SOMMERVILLE, 2000). "Construímos modelos de sistemas complexos porque não é possível compreendê-los em sua totalidade" (RUMBAUGH; JACOBSON; BOOCH, 1999, p. 7).

Os resultados da modelagem podem ser usados também como roteiro de revisão, verificação da consistência e precisão do que foi especificado. Permite que erros sejam visualizados antes da fase de codificação do software e a comunicação de decisões sem ambiguidade de interpretação (RUMBAUGH; JACOBSON; BOOCH, 1999). Outra importância crucial da modelagem é que a qualidade de organização da mesma poderá refletir no trabalho a ser realizado frente a ações de modificações (mesmo depois de finalizado o desenvolvimento, já que mudanças podem ser necessárias).

A modelagem de software é representada pelo uso de notação gráfica ou textual e pode também ser direcionada por modelos (padrão de desenvolvimento) que ajudam a abordar aspectos importantes de um software em construção (modelo de requisitos, modelo de diagrama conceitual, modelo de caso de uso) dando origem a diferentes artefatos compostos por diferentes tipos de elementos (classes, atributos e associações, fluxos, associações, e atores). No capítulo 5 encontram-se as descrições dos elementos de modelagem, os quais serão usados nesse trabalho , disponíveis no *EA.*

Os elementos de modelagem que precisam estar relacionados a outros, seja para especificar um aspecto desejado para o software ou para que se identifique um elemento como a causa para a criação de outro, exige bastante atenção. É possível que durante as constantes operações, comuns em todo processo de desenvolvimento, afetem diretamente tanto o artefato

que esteja sendo trabalhado como indiretamente outros artefatos que tenham alguma relação com este.

Levando em conta os aspectos levantados, há então a necessidade de algo que apóie as operações práticas da modelagem e paralelamente contribua no gerenciamento do conteúdo produzido das atividades que se referem ao planejamento (tudo que é planejado poderá ser modelado ou precisará ser documentado) e desenvolvimento. Nesse sentido, o conceito de rastreabilidade, tratado no capítulo seguinte, é capaz de prover soluções aos problemas e alcançar os benefícios levantados.

2.1 CONCEITOS ABORDADOS NO TRABALHO

Encontra-se na literatura da *UM,* inúmeros elementos específicos que ajudam na modelagem de software. Devido a esse número, a seguir serão abordados conceitos que essencialmente são básicos e usados na maioria de softwares projetados e planejados. Atente-se para o grau e o propósito de abstração de cada um, que podem abstrair aspectos em alto nível dado o momento do desenvolvimento, mas que em outro, poderão estar representando uma especificação em baixo nível de abstração (código).

2.1.1. Requisitos

Requisitos estão relacionados a necessidade do usário. É primordial a compreensão dos requisitos de software para se obter qualidade no desenvolvimento do mesmo. (PRESSMAN, 2002). Através do conjunto de requisitos busca-se garantir uma estrutura para que aplicações possam ser implementadas, de forma a contar com todas as informações que necessite. Segundo (CERRI, 2007) a importância dada aos requisitos reflete no planejamento e desenvolvimento de um projeto de software. A representação gráfica desse elemento no ambiente *EA* (Figura 4).

2.1.2. Casos de Uso

Um caso de uso é a descrição de limites e operações que o sistema deve realizar. Ele ajuda a delimitar as funcionalidades (descreve passos para o início da tarefa até o fim) que o sistema precisa fazer e são originados após a análise de especificação de requisitos.. Por fim eles descrevem requisitos funcionais do sistema sem se tornar um requisito. A representação gráfica desse elemento no ambiente *EA* (Figura 5).

2.1.3. Classes

Uma classe busca refletir um conceito (talvez associado a outros) abstraído do domínio da aplicação. Contribui na definição da estrutura do sistema a ser desenvolvido. O diagrama de classes surge da própria atividade de levantamento de requisitos e descrição de casos de uso. A representação gráfica desse elemento no ambiente *EA* (Figura 6).

3. FUNDAMENTAÇÃO TEÓRICA

Com o intuito de embasar o trabalho sobre as perspectivas de uma técnica nova, a RI (SANTOS; WAZLAWICK, 2009), este capítulo apresenta um conjunto de conceitos que fazem parte diretamente da implementação do trabalho final.

3.1 RASTREABILIDADE

A rastreabilidade consiste em uma maneira de associar elementos de artefatos de modelagem, indicando uma relação de causa-efeito entre os mesmos. Assim, a implementação de conceitos de rastreabilidade fornece meios para tratar efeitos colaterais que possam acontecer em decorrência das operações realizadas durante a modelagem e desenvolvimento de software.

Uma vez que as quantidades de modelos, artefatos, textos e diagramas, usados na modelagem, aumentam conforme a compreensão gradual e a complexidade do software a ser criado possam exigir, não é incomum ocorrerem relações entre esses elementos de naturezas e propósitos distintos. Relações estas, que visam não só retratar decisões previstas em projeto como também podem reaproveitar aspectos já modelados, diminuindo a ocorrência de ambiguidades e redundâncias na modelagem, de modo que não haja riscos na implementação.

As relações mencionadas podem se tornar fontes de erros; ocasionar redundâncias na modelagem ou em código, quando não bem gerenciadas. Mas uma modelagem que as represente, pode-se aplicar uma análise de impactos de modificações que afetem outros elementos ou relações e também fornecer meios que permitem dizer o “histórico, ou motivo” do porque da existência de tal item de modelagem. (CRUZ; JINO e ARGOLLO, 2006) (SAYAO, 2005)

Para lidar com um potencial crescimento na quantidade de elementos especificados, que auxiliam na compreensão, gerenciamento e evolução de um software, a capacidade de uma ferramenta que os gerencie também influencia na qualidade do desenvolvimento de todo processo e consequentemente no produto final.

3.2 RELAÇÕES DE RASTREABILIDADE

Segundo (SANTOS; WAZLAWICK, 2009) um elemento (*e)* é uma unidade de informação que compõe um artefato (*a)*, sendo que (e) representa diferentes tipos de informação de acordo com o nível de abstração desejado. Exemplificam ainda que uma abstração baixa pudesse ser uma palavra, já quando alta, um elemento pudesse ser um caso de uso, uma associação ou uma classe.

Ainda de acordo com os autores mencionados, um artefato é um conjunto de elementos denotado pelo universo (*E)* de todos os elementos possíveis, sendo que um sistema de software pode então ser modelado por um conjunto de artefatos. Afirmam também:

> [...] Um *artefato a* é definido como sendo um conjunto de elementos de *E*. Um sistema de software pode então ser modelado por um conjunto de artefatos $A = \{a_1, a_2, ... a_n\}$ cada qual contendo um conjunto de elementos, ou seja, $A = \{ \{e_{1,1}, e_{1,2}, ...\}, \{e_{2,1}, e_{2,2}, ...\}, ... \{e_{n,1}, e_{n,2}, ...\} \}$. [...] As eventuais associações entre elementos de um artefato (composição, generalização, associação simples, etc.) são também consideradas elementos dos artefatos. Faz-se exceção apenas às relações de rastreabilidade, definidas a seguir, que são consideradas externas aos artefatos, não sendo, portanto, elementos destes. Uma *relação de rastreabilidade* $R \subseteq E \times E$ é uma relação acíclica que estabelece ligações de rastreabilidade entre elementos de artefatos.
>
> A relação representada na Figura 1, por exemplo, que os elementos do artefato A_2, com exceção de $e_{2,2}$ foram criados a partir dos elementos do artefato A_1. A figura também indica que os elementos do artefato A_4 foram criados a partir de elementos do artefato A_2 e A_3. E assim por diante.

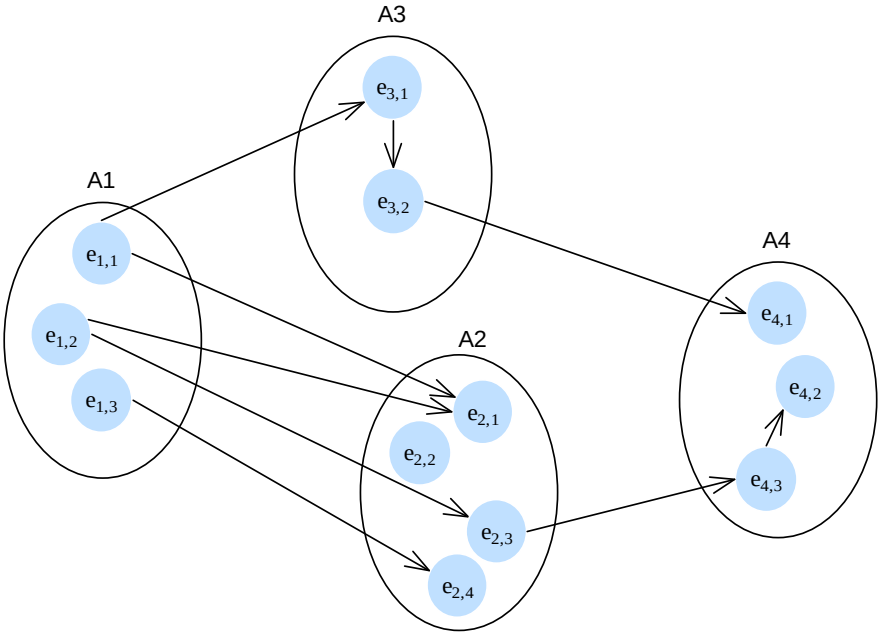


Figura 1: Relação de rastreabilidade.

> Se o artefato A_1 fosse, por exemplo, o diagrama de requisitos e o artefato A_2 fosse o diagrama de casos de uso, então a relação de rastreabilidade estaria definindo quais casos de uso se originam de quais requisitos. As relações de rastreabilidade não ocorrem necessariamente entre elementos de artefatos diferentes. Elas podem ocorrer dentro do mesmo artefato, como no caso dos artefatos A_3 e A_4 na Figura 1.

3.3 TÉCNICA DE RASTREABILIDADE INDUTIVA

As premissas da RI (SANTOS; WAZLAWICK, 2009), são mais bem descritas e compreendidas como é descrito a seguir:

> [...] é possível automatizar a criação e manutenção de ligações de rastreabilidade sob a hipótese de que a inserção de novos elementos nos artefatos não consiste simplesmente em criar um novo elemento no diagrama, mas em uma ação que em muitos casos tem uma causa bem definida a partir de algum outro elemento**.** Por exemplo, uma classe pode estar sendo inserida no modelo conceitual devido à existência de um caso de uso que a menciona. Ou ainda, um caso de uso pode estar sendo inserido no diagrama de casos de uso em função de um ou mais requisitos que lhe dão origem.

> [...] as operações de inserção de elementos nos artefatos sejam *indutivas*. Ou seja, com exceção dos elementos iniciais (base) a inserção de um elemento em um artefato deverá ocorrer sempre a partir de outro elemento, o qual consiste em sua causa. Por exemplo, os elementos iniciais são aqueles que surgem de fontes externas, como os requisitos. Os demais elementos como classes e casos de uso seriam criados em função de elementos já existentes (ou seja, por indução).

> [...] um conjunto de operações para criação e evolução de artefatos do ponto de vista da área que pode ser definida como *Rastreabilidade Indutiva*, mostrando que é possível conceber um processo de desenvolvimento e evolução de software baseado nestas premissas.

3.4 OPERAÇÕES QUE AFETAM A RASTREABILIDADE

No contexto da RI, na busca por manter a coesão das RR é necessário fazer diferenciações importantes entre as operações comuns da modelagem (criação, alteração, exclusão).

Para os autores (SANTOS; WAZLAWICK, 2009), a criação pode ser a inserção de um elemento básico (ex: requisito) ou a criação de um elemento a partir de outro, sendo que essa se subdivide em: com preservação do elemento original e sem preservação do elemento original. A partir dessa subdivisão identificam-se as operações de derivação e divisão respectivamente.

No que se refere à alteração de um elemento a RI prevê duas maneiras de se modificar um elemento: transformação interna e a modificação de suas relações de rastreabilidade, as quais podem ser criadas ou destruídas. Quanto à operação de exclusão, um elemento será removido quando a intenção de remoção simplesmente do mesmo, entretanto, a possibilidade de haver uma RR entre o elemento a ser removido com outro elemento pode exigir um reestruturação de elementos modelados.

No capítulo 5 serão abordados os conceitos referentes às operações a serem posteriormente implementadas.

4. RECURSOS TECNOLÓGICOS

Esta secção tem objetivo de informar sobre as tecnologias necessárias a implementação do plug-in e o porquê de sua utilização.

4.1 ENTERPRISE ARCHITECT

O *EA* é usado para projetar (modelagem e documentação) e construir sistemas de software. É usada a notação *Unified Modeling Language UML* 2 para modelagem das especificações de software. O *EA* é uma ferramenta que suporta todas as etapas de desenvolvimento, favorecendo o gerenciamento desde a fase inicial de concepção até a implantação e manutenção do mesmo. Também suporta testes e mudança de controle. A *UML* 2 passou a ser suportada pelo *EA* a partir da versão 4.5.

O EA oferece a disponibilidade de integração (manipulação elementos, artefatos de modelagem e até funcionalidades do EA) e dispõe de completa documentação (apêndice A) para o desenvolvimento de aplicações externas (plug-in, desenvolvido como um componente *ActiveX COM*) executada dentro do próprio EA.

A seguir é feita uma descrição de elementos (requisitos, casos de uso, classes) e artefatos (diagramas) de modelagem que foram abordados e usados nos exemplos do escopo desse trabalho. A clareza sobre os mesmos contribui na compreensão da funcionalidade do plug-in a ser desenvolvido e na assimilação da criação semi-automática de RR em uma modelagem de software.

É importante lembrar que são vastos os tipos de elementos empregados na ferramenta *EA* e foge do escopo desse trabalho descrever sobre todas as possíveis representações. Porém a aplicabilidade das operações e noções de rastreabilidade, devidamente abordadas até então foi implementada de forma que fosse possível existir RR entre quaisquer elementos.

A seguir se encontram informações sobre itens importantes para o melhor entendimento do trabalho proposto:

Project Browser: (figura 2) Identificam-se os elementos mais comuns como pacotes, diagramas, elementos do tipo: requisito, caso de uso, classe e etc. (apêndice A)

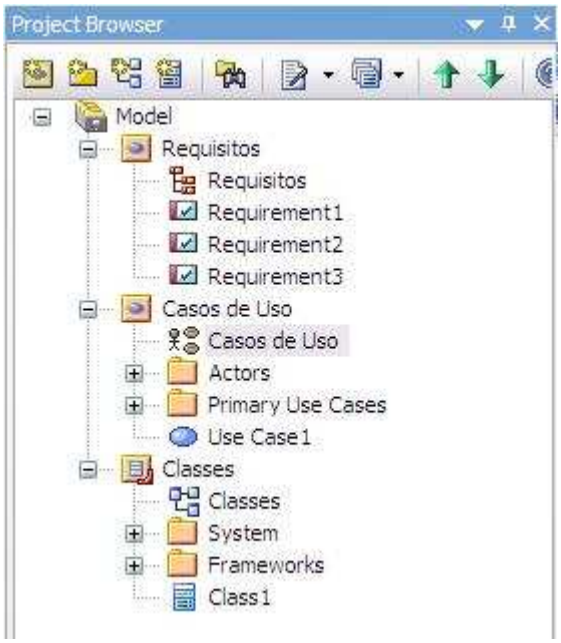


Figura 2: *Project Browser*.

Toolbox: (figura 3) É a ferramenta que possibilita ao usuário selecionar o elemento desejado para o diagrama atual. (apêndice A)

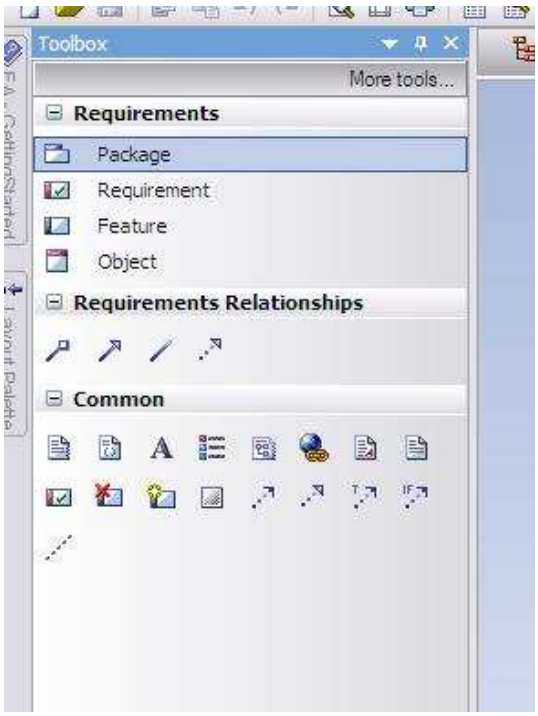


Figura 3: *Toolbox*.

Elemento Requisito: (figura 4) Usualmente é a representação com alta abstração de uma característica ou funcionalidade preterida na construção de um software. (apêndice A)

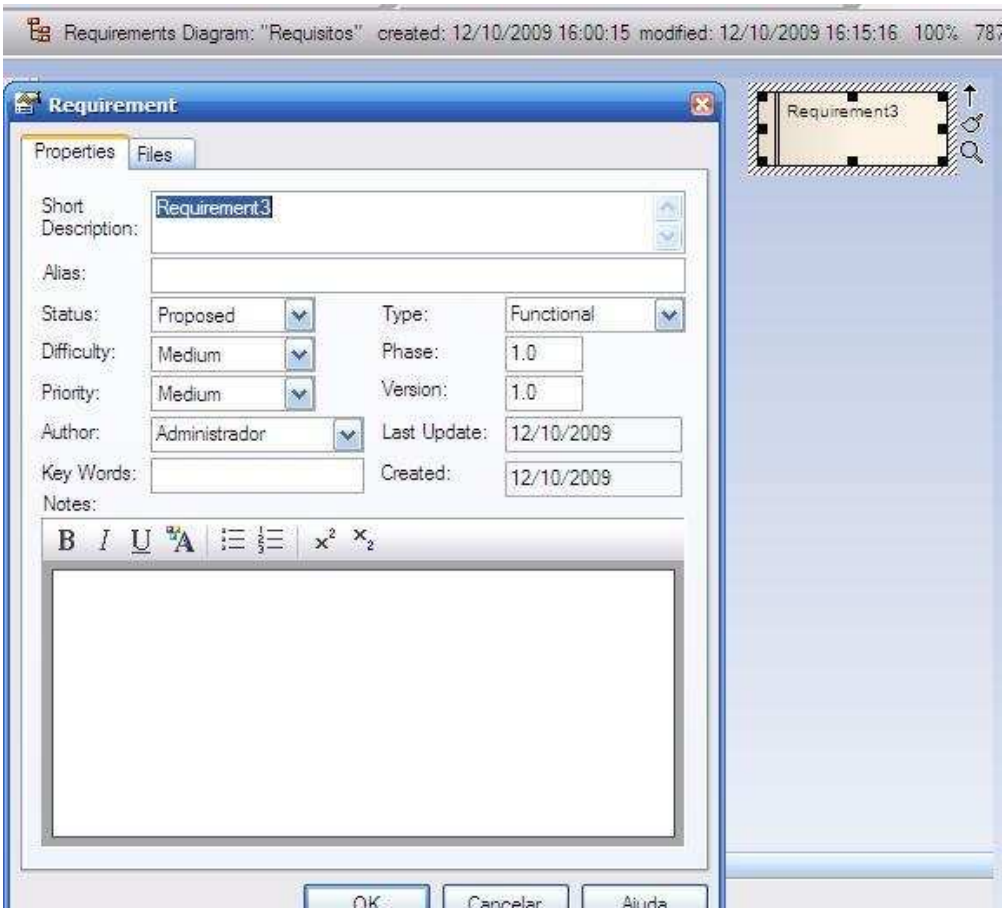


Figura 4: Elemento requisito.

Elemento Classe: (figura 5) É a representação em alto nível do que possivelmente possa na fase de codificação se tornar uma classe com métodos e atributos (representação em baixo nível). (apêndice A)

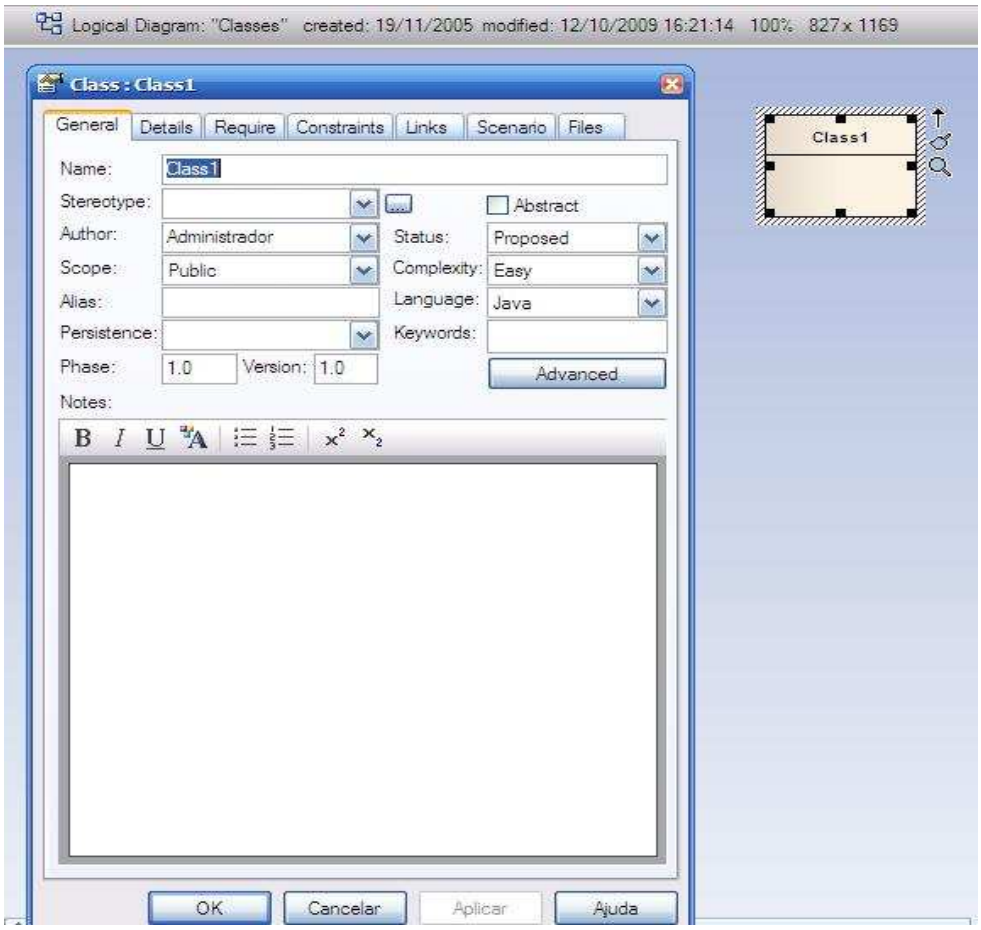


Figura 5: Elemento classe.

Elemento Caso de Uso: (figura 6) Também uma representação em alto nível que aborda comportamentos modelados desejados para um produto final. (apêndice A)

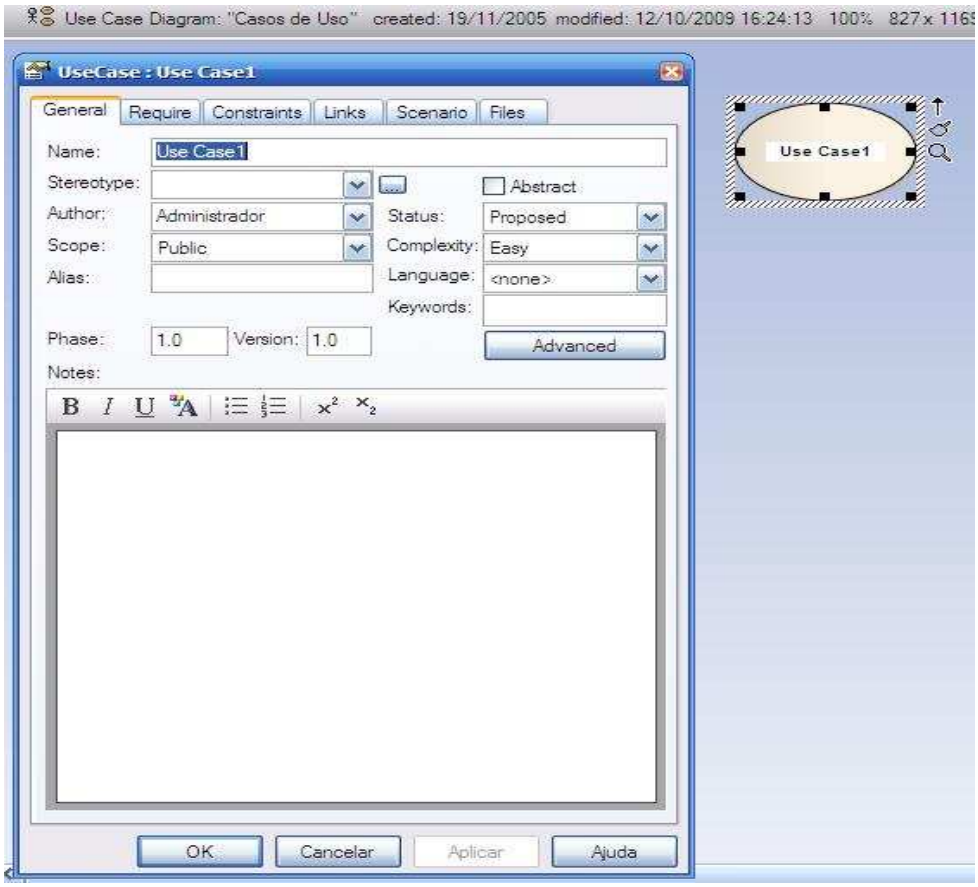


Figura 6: Elemento caso de uso.

Diagrama Atual: (figura 7) Usualmente indica um foco da modelagem sendo estruturado em determinado momento. (apêndice A)

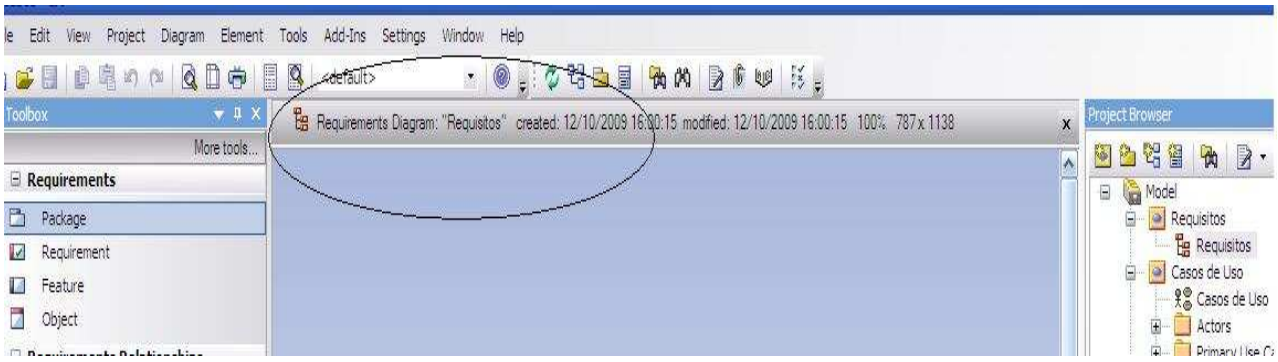


Figura 7: Diagrama Atual.

4.2 ACTIVEX

O termo *ActiveX* é um conjunto de tecnologias com objetivo de facilitar a integração de objetos desenvolvidos em qualquer linguagem. O suporte desta tecnologia em várias plataformas que não apenas a plataforma *Windows*. A utilização de ferramentas de programação familiares e largamente divulgadas para desenvolver os controles *ActiveX*.

4.3 MICROSOFT COMPONENT OBJECT MODEL

O padrão *COM* é uma tecnologia que permite a comunicação entre componentes de software. O *COM* é útil na construção de componentes re-usáveis de software, permite também que diferentes componentes se unam (*ActiveX Controls*) em uma única aplicação, tendo disponíveis as funcionalidades da plataforma Windows.

É independente de linguagens de programação, entretanto algumas linguagens orientadas a objeto oferecem mecanismos mais acessíveis à programação desse tipo de tecnologia.

4.4 MICROSOFT VISUAL BASIC 6

O *Microsoft Visual Basic* 6 VB6 utiliza uma linguagem de programação orientada à eventos (todas as ações que ocorrem durante a execução do programa são estruturadas nos eventos dos objetos) e serve para gerar aplicações voltadas a plataforma *Windows*. O uso desse ambiente foi determinado pela facilidade em se programar aplicações clientes *ActiveX COM* e pela documentação mais consistente sobre a interação entre o *EA* e esse facilitador do VB6.

5. IMPLEMENTAÇÃO

Como pode ser inicialmente abordado no capítulo anterior, o motivo do uso das ferramentas descritas se deve a alta flexibilidade e documentações de uso disponíveis das mesmas, no que tange a intenção desse trabalho.

Nesse capítulo será descrita as operações implementadas e previstas pela técnica de RI, o código implementado de cada uma e o modo de uso do plug-in desenvolvido. Por fim, será falado sobre um estudo de caso de uso prático do plug-in implementado (técnica de RI) em comparação ao que seria necessário para se atingir o mesmo objetivo (determinada tarefa) usando-se outras técnicas de rastreabilidade.

O ambiente *EA* oferece um caminho para que outras aplicações tenham acesso aos seus artefatos de modelagem que estejam sendo modelados. Essas aplicações externas devem ser clientes *ActiveX COM* desenvolvidas em qualquer linguagem de programação que melhor suporte a criação desse tipo de componente de software. As aplicações clientes podem reutilizar funcionalidades e elementos (artefatos de modelagem e funcionalidades) disponiveis no *EA* e manipulá-las da maneira desejada Assim, essa interação acaba acrescentando funcionalidades extras, desenvolvidas por terceiros, ao próprio *EA*.

Ao longo do processo de implementação pode-se deparar com a falta de algumas funções eficientes, na interface de desenvolvimento disponibilizada. Na implementaçãos das operações alguns métodos, ainda equivalentes aos apresentados a seguir, seriam menos extenso, com número menor iterações para verificações necessárias no escopo de cada funcionalidade desejada podendo resultar em códigos menores. Mas essa deficiência não afeta de maneira significativa, visto a eficiência do plug-in que poderá ser constatada na secção 5.4.

O processo de criação de cada método se resumiu na aplicação do entendimento do conceito relacionado na RI. Em seguida, compreendia-se o significado de cada operação através da leitura do respectivo caso de uso da mesma. De forma que o passo seguinte extendia-se por leitura na documentação do *EA*, paralelamente a familiarização com o uso do ambiente, na intenção de se identificar cada elemento e as características (atributos, métodos) dos mesmos segundo o diagrama conceitual (apêndice A) de desenvolvimento.

Dentre os objetivos, seria importante o entendimento dos eventos possíveis para cada função base (clique de menu, *drag and drop* de elementos, seleção de diversos elementos em diferentes componentes) além daqueles que influeciavam diretamente (criação de elementos e associações entre os mesmos nos diferentes componentes) as operações da RI a serem implementadas. Além de buscar compreender os métodos disponíveis na interface de desenvolvimento, no que se refere a manipulação geral de elementos e componentes (diagramas, *project browser*). (apêndice A)

Na seção seguinte será apresentado o código que implementa a descrição e funcionalidades das operações previstas, segundo conceitos da RI apresentado anteriormente.

5.1 DESCRIÇÃO DAS OPERAÇÕES E CASOS DE USO

5.1.1 Criação de Elemento base

Segundo os autores (SANTOS; WAZLAWICK, 2009), a característica desse método é inserir um elemento de qualquer tipo em um artefato como um diagrama. No exemplo mostrado na seção 5.3, a modelagem dessa funcionalidade abstraída refere-se a elementos que não se possa encontrar uma relação causal vinda de outro elemento, ou seja, sua causa é externa. Conforme mostra a (figura 8), nenhuma nova relação de rastreabilidade é criada. Vale ressaltar que é preciso clareza sobre qual tipo de elemento de modelagem representará um conceito tido como base.

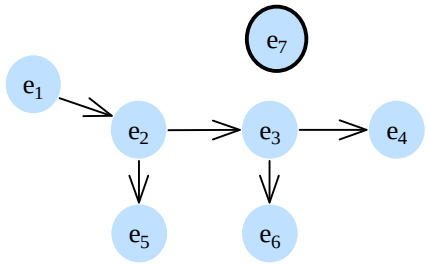


Figura 8: Criação de um elemento-base (e_7).

Exemplo: um requisito é criado no diagrama de requisitos a partir de entrevistas que o desenvolvedor realizou com o cliente. Nenhum artefato até o momento referenciava esse requisito ou suas consequências. Então ele pode ser inserido como um elemento-base.

A criação do elemento base representa a abstração de alguma informação, peculiar por ser base e passará a ter uma representação visual além de poder conter uma descrição textual. Pode ser realizada por funções básicas do *EA*, acessíveis via menus ou uma ação *drag and drop* no elemento desejado na barra de *toolbox*.A representação será vista pela disposição de uma figura representativa daquele elemento em um diagrama, ou no □*roject browser* do ambiente.

Fluxo principal:

1. O usuário seleciona na *toolbox* o elemento base a ser inserido e o arrasta para um artefato destino, geralmente um diagrama disposto. Outra maneira de inserir elementos é via menu, clicando com botão direito do mouse sob algum elemento "pacote", selecionado ação "*Add*" e escolhendo o tipo de elemento.
2. O usuário preenche as informações dos atributos do novo elemento.

5.1.2 Derivação

Como foi ressaltado no capitulo quatro, a criação de um elemento a partir de outro, com preservação do elemento original dá origem à operação de derivação. No contexto de utilização do *EA* e da própria RI, a derivação necessita uma especificação de destino no qual será inserido o novo elemento derivado. Podendo esse destino ser de três naturezas diferentes: diagrama, pacote ou elementos quaisquer.

> Quando um elemento é criado em função de outro, seja no mesmo artefato ou em artefatos diferentes, o elemento original é denominado *elemento-causa* e o elemento originado *elemento-efeito*. É criada uma relação de rastreabilidade entre o elemento-causa e o elemento-efeito no sentido do elemento-causa para o elemento-efeito (Figura 5).
> Exemplo: um caso de uso (elemento-efeito) é criado em função de um requisito funcional (elemento-causa).

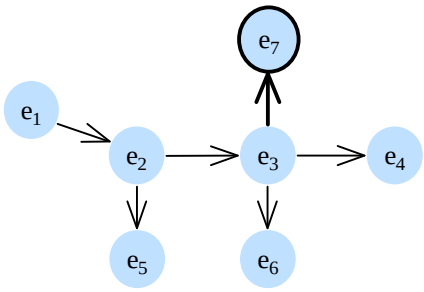


Figura 9. Derivação de um elemento-efeito (e_7) a partir de um elemento-causa (e_3).

Derivar (requisito, caso de uso, classe) para um pacote qualquer.

Fluxo Principal:

1. Usuário deixa previamente selecionado no *Project Browser* o pacote destino desejado.
2. Clica com botão direito no elemento-causa (visível no diagrama atual) a ser derivado e seleciona no menu “Add-Ins”, na aba “Rastreabilidade Add-in” o comando de derivação de acordo com o tipo do elemento que se quer derivar.

Garantia de sucesso: O elemento desejado rastreável foi criado no pacote selecionado.

Derivar (requisito, caso de uso, classe) para um diagrama qualquer:

Fluxo Principal:

1. Usuário deixa previamente selecionado no *Project Browser* o diagrama destino desejado.
2. Clica com botão direito no elemento-causa (visível no diagrama atual) a ser derivado e seleciona no menu “Add-Ins”, na aba “Rastreabilidade Add-in” o comando de derivação de acordo com o tipo do elemento que se quer derivar.

Garantia de sucesso: O diagrama atual passa a ser o selecionado no *Project Browser* e o elemento derivado rastreável está disposto no mesmo, além disso, o elemento-efeito é criado no pacote que contém o diagrama previamente selecionado como destino no *PROJECT BROWSER*.

Derivar (requisito, caso de uso, classe) associado a um elemento qualquer:

Fluxo Principal:

1. Usuário deixa previamente selecionado no *Project Browser* o elemento desejado.
2. Clica com botão direito no elemento-causa (visível no diagrama atual) a ser derivado e seleciona no menu “Add-Ins”, na aba “Rastreabilidade Add-in” o comando de derivação de acordo com o tipo do elemento que se quer derivar.

Garantia de sucesso: O elemento a ser derivado foi criado associado ao elemento previamente selecionado no *Project Browser*.

5.1.3 Criação de Relação de Rastreabilidade

Dois elementos que já existam nos artefatos podem ser identificados como elemento-causa e elemento-efeito independentemente de que o elemento-efeito tenha sido criado pelo processo de derivação.

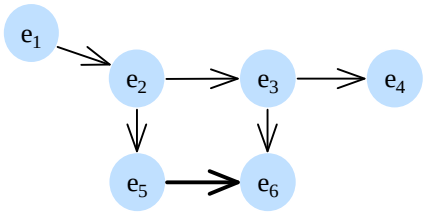


Figura 10 – Criação de relação entre um elemento-efeito (e_6) e um elemento-causa pré-existente (e_5).

Essa operação seria útil diante ao não uso da operação de derivação para que um novo elemento-efeito rastreável fosse criado a partir de um elemento-causa já existente. Esse cenário ocorreria ou devido à desatenção no planejamento ou uma posterior mudança necessária.

Exemplo: Um requisito foi identificado como elemento-causa de uma classe. Posteriormente, percebe-se que outro requisito também contribui para a definição dessa classe, sendo também elemento-causa dela. Assim, a classe em questão terá dois elementos-causa. Um efetivamente causou a criação da classe, o outro foi

> posteriormente relacionado a ela. Mas para efeito da relação de rastreabilidade, não há distinção de ordem ou importância entre os dois elementos-causa. (SANTOS; WAZLAWICK, 2009)

Fluxo Principal:

1. Usuário deixa previamente selecionado no *Project Browser* o elemento com o qual se quer estabelecer uma relação de rastreabilidade, sendo esse o elemento-efeito.
2. Clica com botão direito no elemento-causa (visível no diagrama atual) a ser associado com outro elemento e seleciona no menu “Add-Ins”, na aba “Rastreabilidade Add-in” o comando “Associar Elementos”.

Garantia de sucesso: Os dois elementos devidamente identificados como causa e origem mantém uma relação de rastreabilidade no escopo da modelagem.

.

5.2 CÓDIGO IMPLEMENTADO

Essa seção apresenta o código que implementa as operações abordadas na RI e que fazem parte dos objetivos propostos no presente trabalho.

Operação: Criar Relação de Rastreabilidade

```
  Sub Associar(ByVal Repository As EA.Repository)

   "{ Declaração de Variáveis }

   Dim objDiagAtual As Diagram
   Dim elemDest As Element
   Dim iteraDiagObj As Object
   Dim objTypeDest As ObjectType
   Dim conTeste As Connector
   Dim ctrlDiagAberto As Boolean
   Dim disparaErro As String
   Dim elemOrigem As Element
   Dim ob As Object
   Dim iteraConnObj As Connector
   Dim elementoD As Element
   Dim ctrlConnExiste As Boolean
   "{ Inicializações }

   ctrlDiagAberto = False

   " { Método }

   On Error GoTo TrataErroDiagFechado
   Set objDiagAtual = Repository.GetCurrentDiagram
   disparaErro = objDiagAtual.Name
   ctrlDiagAberto = True

TrataErroDiagFechado:

   If (ctrlDiagAberto = True) Then
      If (objDiagAtual.SelectedObjects.Count > 0) Then
         If (Repository.GetTreeSelectedItemType = otElement) Then
            objTypeDest = Repository.GetTreeSelectedItem(elemDest)
            For Each iteraDiagObj In objDiagAtual.SelectedObjects()
               Set ob = Repository.GetElementByID(iteraDiagObj.ElementID)
               Set elemOrigem = Repository.GetElementByID(ob.ElementID)
               For Each iteraConnObj In elemOrigem.Connectors
                  Set elementoD = Repository.GetElementByID(iteraConnObj.SupplierID)
                  ctrlConnExiste = False
                  If elementoD.Name = elemDest.Name Then
                     ctrlConnExiste = True
                     MsgBox "Já existe uma relação de rastreabilidade entre [" + elemOrigem.Name + "] e [" +
elemDest.Name + "]"
                     Exit For
                  End If

               Next
               If (ctrlConnExiste = False) Then
                  Set conTeste = elemOrigem.Connectors.AddNew("", "Dependency")
```

```
                conTeste.SupplierID = elemDest.ElementID
                conTeste.Update
            End If

        Next
        Repository.SaveDiagram (Repository.GetCurrentDiagram.DiagramID)
        Repository.ReloadDiagram (Repository.GetCurrentDiagram.DiagramID)
        Repository.ActivateDiagram (Repository.GetCurrentDiagram.DiagramID)
        Repository.OpenDiagram (Repository.GetCurrentDiagram.DiagramID)
      Else
        MsgBox "[ASSOCIACAO] Elemento destino da associação não pode ser associado.", vbExclamation

      End If
    Else
      MsgBox "[ASSOCIACAO] Elemento(s) origem da associação dever ser selecionado pelo diagrama.",
vbInformation

    End If
  Else
    MsgBox "[ASSOCIAÇÃO] Selecione o(s) elemento(s) através de um diagrama", vbInformation

  End If

End Sub
```

Operação: Derivar Elemento

```
Sub Derivacao(ByVal Repository As EA.Repository, ByVal TipoE As String)

  "{ Declaração de Variáveis }

  Dim objPackAux As Package
  Dim objElemPackDerivarAux As Package
  Dim objDiagAux As Diagram
  Dim objDiagAtual As Diagram
  Dim elemOrigem As Element
  Dim objElemAux As Element
  Dim objElemDerivarAux As Element
  Dim novoElemDerivado As Element
  Dim elemAuxBase As Element
  Dim objType As ObjectType
  Dim objTypeAux As ObjectType
  Dim capazBase As Object
  Dim v As DiagramObject
  Dim disparaErroX As String
  Dim testaNome As String
  Dim ctrlDiagOpen As Boolean
  Dim ctrlNome As Boolean
  Dim conTeste As Connector
  Dim ob As Object
  Dim iteraDiagObj As Object

    "{ Inicializações }

  ctrlDiagOpen = False
  ctrlNome = False
  On Error GoTo TrataErroX
  Set objDiagAtual = Repository.GetCurrentDiagram
  disparaErroX = objDiagAtual.Name
```

```
   ctrlDiagOpen = True

TrataErroX:
   If (ctrlDiagOpen = True) Then
      If (objDiagAtual.SelectedObjects.Count > 0) Then

         If (Repository.GetContextItemType = otPackage) Then
            objType = Repository.GetContextItem(objElemPackDerivarAux)
            testaNome = objElemPackDerivarAux.Name
            testaNome = Trim$(testaNome)
            testaNome = Left$(testaNome, 3)
            If (testaNome = "[D]") Then
               ctrlNome = True
               testaNome = objElemPackDerivarAux.Name
               testaNome = Trim$(testaNome)
            Else
               ctrlNome = False
               testaNome = objElemPackDerivarAux.Name
               testaNome = Trim$(testaNome)
            End If
         Else
            objType = Repository.GetContextItem(objElemDerivarAux)
            testaNome = objElemDerivarAux.Name
            testaNome = Trim$(testaNome)
            testaNome = Left$(testaNome, 3)
            If (testaNome = "[D]") Then
               ctrlNome = True
               testaNome = objElemDerivarAux.Name
               testaNome = Trim$(testaNome)
            Else
               ctrlNome = False
               testaNome = objElemDerivarAux.Name
               testaNome = Trim$(testaNome)
            End If
         End If

         Select Case TipoE

         "---------------------------------------------------------------------
            Case "r"
               If (Repository.GetTreeSelectedItemType = otPackage) Then
                  objTypeAux = Repository.GetTreeSelectedItem(objPackAux)
                  If (ctrlNome = True) Then
                        Set novoElemDerivado = objPackAux.Elements.AddNew(testaNome, "Requirement")
                        objPackAux.Elements.Refresh
                        novoElemDerivado.Update
                  Else
                        Set novoElemDerivado = objPackAux.Elements.AddNew("[D] " + testaNome,
"Requirement")
                        objPackAux.Elements.Refresh
                        novoElemDerivado.Update
                  End If

               End If

               If (Repository.GetTreeSelectedItemType = otDiagram) Then
                  objTypeAux = Repository.GetTreeSelectedItem(objDiagAux)
                  Set objPackAux = Repository.GetPackageByID(objDiagAux.PackageID)

                  If (ctrlNome = True) Then
```

```
            Set novoElemDerivado = objPackAux.Elements.AddNew(testaNome, "Requirement")
            Set v = objDiagAux.DiagramObjects.AddNew("", "")
            objPackAux.Elements.Refresh
            v.DiagramID = objDiagAux.DiagramID
            v.ElementID = novoElemDerivado.ElementID
            v.Update
            objDiagAux.Update
            objDiagAux.DiagramObjects.Refresh
         Else
            Set novoElemDerivado = objPackAux.Elements.AddNew("[D] " + testaNome, "Requirement")
            Set v = objDiagAux.DiagramObjects.AddNew("", "")
            objPackAux.Elements.Refresh
            v.DiagramID = objDiagAux.DiagramID
            v.ElementID = novoElemDerivado.ElementID
            v.Update
            objDiagAux.Update
            objDiagAux.DiagramObjects.Refresh
         End If

      End If

      If (Repository.GetTreeSelectedItemType = otElement) Then
         Set objElemAux = Repository.GetTreeSelectedObject
         Set objPackAux = Repository.GetPackageByID(objElemAux.PackageID)

         If (ctrlNome = True) Then
            Set novoElemDerivado = objPackAux.Elements.AddNew(testaNome, "Requirement")
            novoElemDerivado.ParentID = objElemAux.ElementID
            novoElemDerivado.Update
            novoElemDerivado.Refresh
            objPackAux.Elements.Refresh
         Else
            Set novoElemDerivado = objPackAux.Elements.AddNew("[D] " + testaNome, "Requirement")
            novoElemDerivado.ParentID = objElemAux.ElementID
            novoElemDerivado.Update
            novoElemDerivado.Refresh
            objPackAux.Elements.Refresh
         End If

      End If

"-------------------------------------------------------------------
   Case "u"
      If (Repository.GetTreeSelectedItemType = otPackage) Then
         objTypeAux = Repository.GetTreeSelectedItem(objPackAux)
         If (ctrlNome = True) Then
            Set novoElemDerivado = objPackAux.Elements.AddNew(testaNome, "UseCase")
            objPackAux.Elements.Refresh
            novoElemDerivado.Update
         Else
            Set novoElemDerivado = objPackAux.Elements.AddNew("[D] " + testaNome, "UseCase")
            objPackAux.Elements.Refresh
            novoElemDerivado.Update
         End If
      End If

      If (Repository.GetTreeSelectedItemType = otDiagram) Then
         objTypeAux = Repository.GetTreeSelectedItem(objDiagAux)
         Set objPackAux = Repository.GetPackageByID(objDiagAux.PackageID)
         If (ctrlNome = True) Then
```

```
            Set novoElemDerivado = objPackAux.Elements.AddNew(testaNome, "UseCase")
            Set v = objDiagAux.DiagramObjects.AddNew("", "")
            objPackAux.Elements.Refresh
            v.DiagramID = objDiagAux.DiagramID
            v.ElementID = novoElemDerivado.ElementID
            v.Update
            objDiagAux.Update
            objDiagAux.DiagramObjects.Refresh
         Else
            Set novoElemDerivado = objPackAux.Elements.AddNew("[D] " + testaNome, "UseCase")
            Set v = objDiagAux.DiagramObjects.AddNew("", "")
            objPackAux.Elements.Refresh
            v.DiagramID = objDiagAux.DiagramID
            v.ElementID = novoElemDerivado.ElementID
            v.Update
            objDiagAux.Update
            objDiagAux.DiagramObjects.Refresh

         End If
      End If

      If (Repository.GetTreeSelectedItemType = otElement) Then
         Set objElemAux = Repository.GetTreeSelectedObject
         Set objPackAux = Repository.GetPackageByID(objElemAux.PackageID)
            If (ctrlNome = True) Then
               Set novoElemDerivado = objPackAux.Elements.AddNew(testaNome, "UseCase")
               novoElemDerivado.ParentID = objElemAux.ElementID
               novoElemDerivado.Update
               novoElemDerivado.Refresh
               objPackAux.Elements.Refresh
            Else
               Set novoElemDerivado = objPackAux.Elements.AddNew("[D] " + testaNome, "UseCase")
               novoElemDerivado.ParentID = objElemAux.ElementID
               novoElemDerivado.Update
               novoElemDerivado.Refresh
               objPackAux.Elements.Refresh
            End If
      End If

"---------------------------------------------------------------------
   Case "c"
      If (Repository.GetTreeSelectedItemType = otPackage) Then
         objTypeAux = Repository.GetTreeSelectedItem(objPackAux)
         If (ctrlNome = True) Then
            Set novoElemDerivado = objPackAux.Elements.AddNew(testaNome, "Class")
            objPackAux.Elements.Refresh
            novoElemDerivado.Update
         Else
            Set novoElemDerivado = objPackAux.Elements.AddNew("[D] " + testaNome, "Class")
            objPackAux.Elements.Refresh
            novoElemDerivado.Update
         End If
      End If

      If (Repository.GetTreeSelectedItemType = otDiagram) Then
         objTypeAux = Repository.GetTreeSelectedItem(objDiagAux)
         Set objPackAux = Repository.GetPackageByID(objDiagAux.PackageID)
         If (ctrlNome = True) Then
            Set novoElemDerivado = objPackAux.Elements.AddNew(testaNome, "Class")
            Set v = objDiagAux.DiagramObjects.AddNew("", "")
```

```
            objPackAux.Elements.Refresh
            v.DiagramID = objDiagAux.DiagramID
            v.ElementID = novoElemDerivado.ElementID
            v.Update
            objDiagAux.Update
            objDiagAux.DiagramObjects.Refresh
         Else
            Set novoElemDerivado = objPackAux.Elements.AddNew("[D] " + testaNome, "Class")
            Set v = objDiagAux.DiagramObjects.AddNew("", "")
            objPackAux.Elements.Refresh
            v.DiagramID = objDiagAux.DiagramID
            v.ElementID = novoElemDerivado.ElementID
            v.Update
            objDiagAux.Update
            objDiagAux.DiagramObjects.Refresh
         End If

      End If

      If (Repository.GetTreeSelectedItemType = otElement) Then
         Set objElemAux = Repository.GetTreeSelectedObject
         Set objPackAux = Repository.GetPackageByID(objElemAux.PackageID)
            If (ctrlNome = True) Then
               Set novoElemDerivado = objPackAux.Elements.AddNew(testaNome, "Class")
               novoElemDerivado.ParentID = objElemAux.ElementID
               novoElemDerivado.Update
               novoElemDerivado.Refresh
               objPackAux.Elements.Refresh
            Else
               Set novoElemDerivado = objPackAux.Elements.AddNew("[D] " + testaNome, "Class")
               novoElemDerivado.ParentID = objElemAux.ElementID
               novoElemDerivado.Update
               novoElemDerivado.Refresh
               objPackAux.Elements.Refresh
            End If
      End If

"---------------------------------------------------------------------
End Select
For Each iteraDiagObj In objDiagAtual.SelectedObjects()

   Set ob = Repository.GetElementByID(iteraDiagObj.ElementID)
   Set elemOrigem = Repository.GetElementByID(ob.ElementID)
   Set conTeste = elemOrigem.Connectors.AddNew("", "Dependency")
   conTeste.SupplierID = novoElemDerivado.ElementID
   "conTeste.Stereotype = "Dependency"
   conTeste.Update

Next

If (Repository.GetTreeSelectedItemType = otDiagram) Then
   Repository.SaveDiagram (objDiagAux.DiagramID)
   Repository.ReloadDiagram (objDiagAux.DiagramID)
   Repository.ActivateDiagram (objDiagAux.DiagramID)
   Repository.OpenDiagram (objDiagAux.DiagramID)

End If

If Not conTeste.Update Then
   MsgBox "" + conTeste.GetLastError, vbExclamation
```

```
        End If

      Else
        GoTo X
      End If

  "--------------------------------------------------------------------------
  Else " DIAGRAMA FECHADO
X:      If (Repository.GetTreeSelectedItemType = otElement) Then
        objType = Repository.GetTreeSelectedItem(objElemDerivarAux)
        testaNome = objElemDerivarAux.Name
        testaNome = Trim$(testaNome)
        testaNome = Left$(testaNome, 3)
        If (testaNome = "[D]") Then
          ctrlNome = True
          testaNome = Trim$(objElemDerivarAux.Name)
          testaNome = Trim$(testaNome)
        Else
          ctrlNome = False
          testaNome = objElemDerivarAux.Name
          testaNome = Trim$(testaNome)
        End If


        If TipoE = "r" Then
          Set objElemAux = Repository.GetTreeSelectedObject
          Set objPackAux = Repository.GetPackageByID(objElemAux.PackageID)
          If (ctrlNome = True) Then
            Set novoElemDerivado = objPackAux.Elements.AddNew(objElemDerivarAux.Name,
"Requirement")
            novoElemDerivado.ParentID = objElemAux.ElementID
            novoElemDerivado.Update
            novoElemDerivado.Refresh
            objPackAux.Elements.Refresh
          Else
            Set novoElemDerivado = objPackAux.Elements.AddNew("[D] " + objElemDerivarAux.Name,
"Requirement")
            novoElemDerivado.ParentID = objElemAux.ElementID
            novoElemDerivado.Update
            novoElemDerivado.Refresh
            objPackAux.Elements.Refresh
          End If
        Else
          If TipoE = "u" Then
            Set objElemAux = Repository.GetTreeSelectedObject
            Set objPackAux = Repository.GetPackageByID(objElemAux.PackageID)
            If (ctrlNome = True) Then
              Set novoElemDerivado = objPackAux.Elements.AddNew(objElemDerivarAux.Name, "UseCase")
              novoElemDerivado.ParentID = objElemAux.ElementID
              novoElemDerivado.Update
              novoElemDerivado.Refresh
              objPackAux.Elements.Refresh
            Else
              Set novoElemDerivado = objPackAux.Elements.AddNew("[D] " + objElemDerivarAux.Name,
"UseCase")
              novoElemDerivado.ParentID = objElemAux.ElementID
              novoElemDerivado.Update
              novoElemDerivado.Refresh
              objPackAux.Elements.Refresh
            End If
```

```
            Else
               Set objElemAux = Repository.GetTreeSelectedObject
               Set objPackAux = Repository.GetPackageByID(objElemAux.PackageID)
               If (ctrlNome = True) Then
                  Set novoElemDerivado = objPackAux.Elements.AddNew(objElemDerivarAux.Name, "Class")
                  novoElemDerivado.ParentID = objElemAux.ElementID
                  novoElemDerivado.Update
                  novoElemDerivado.Refresh
                  objPackAux.Elements.Refresh
               Else
                  Set novoElemDerivado = objPackAux.Elements.AddNew("[D] " + objElemDerivarAux.Name,
"Class")
                  novoElemDerivado.ParentID = objElemAux.ElementID
                  novoElemDerivado.Update
                  novoElemDerivado.Refresh
                  objPackAux.Elements.Refresh
               End If

            End If

         End If

      Else
         MsgBox "[DERIVACAO] Objeto selecionado nao pode ser derivado", vbExclamation
      End If

   End If

End Sub
```

O código a seguir tem a finalidade de documentar o modo de interação, no que se refere aos acessos às operações implementadas no plug-in, através dos menus do ambiente *EA*. Outra funcionalidade do mesmo é a de aumentar a usabilidade do plug-in, na medida em que sugestiona a operação que seria o passo seguinte, dado um contexto e momentos da modelagem.

Função: Controle de Menu

```
Public Function EA_GetMenuItems(Repository As EA.Repository, MenuLocation As String, ByVal MenuName
As String) As Variant

   "{ Declaração de Variáveis }

   Dim objTreeItem As Object
   EA_GetMenuItems = ""
   Dim objTypeAux As ObjectType
   Dim objDiagAux As Diagram
   Dim objTes As Diagram
   Dim objType As ObjectType
   Dim elemAuxBase As Element
   Dim packReqDiagAux As Package
   Dim ctrlDiagOpen As Boolean

   "{ Método }

   If (Repository.GetTreeSelectedItemType = otDiagram) Then
      objTypeAux = Repository.GetTreeSelectedItem(objDiagAux)
      Dim aux As String
      aux = objDiagAux.Type
      Select Case aux
         Case "Use Case"
            Select Case MenuName
               Case ""
                  EA_GetMenuItems = "-&Rastreabilidade Add-in"
               Case "-&Rastreabilidade Add-in"
                  EA_GetMenuItems = Array("[> Derivar Caso de &Uso   Alt+A+R+U <]", "Derivar Re&quisito
Alt+A+R+Q", "Derivar &Classe   Alt+A+R+C", "A&ssociar Elementos   Alt+A+R+S")
               Case Else
                  MsgBox "Invalid Menu", vbCritical
            End Select

         Case "Logical"
            Select Case MenuName
               Case ""
                  EA_GetMenuItems = "-&Rastreabilidade Add-in"
               Case "-&Rastreabilidade Add-in"
                  EA_GetMenuItems = Array("[> Derivar &Classe   Alt+A+R+C <]", "Derivar Re&quisito
Alt+A+R+Q", "Derivar Caso de &Uso   Alt+A+R+U", "A&ssociar Elementos   Alt+A+R+S")
               Case Else
                  MsgBox "Invalid Menu", vbCritical
            End Select

         Case "Custom"
            Select Case MenuName
               Case ""
                  EA_GetMenuItems = "-&Rastreabilidade Add-in"
               Case "-&Rastreabilidade Add-in"
                  EA_GetMenuItems = Array("[> Derivar Re&quisito   Alt+A+R+Q <]", "Derivar Caso de &Uso
Alt+A+R+U", "Derivar &Classe   Alt+A+R+C", "A&ssociar Elementos   Alt+A+R+S")
               Case Else
                  MsgBox "Invalid Menu", vbCritical
            End Select

      End Select

   Else
```

```
    Select Case MenuName
        Case ""
            EA_GetMenuItems = "-&Rastreabilidade Add-in"
        Case "-&Rastreabilidade Add-in"
            EA_GetMenuItems = Array("Derivar Re&quisito   Alt+A+R+Q", "Derivar Caso de &Uso
Alt+A+R+U", "Derivar &Classe   Alt+A+R+C", "A&ssociar Elementos   Alt+A+R+S")
        Case Else
            MsgBox "Invalid Menu", vbCritical
    End Select

  End If

End Function
```

5.3 FUNCIONAMENTO DO PLUG-IN IMPLEMENTADO

No intuito de exemplificar de forma mais simples um contexto de uso para o plug-in implementado, nessa seção será apresentada a forma de interação entre um usuário do *EA* e as ações equivalentes executadas pelo plug-in. Assim, para o exemplo mostrado a seguir é deixada de lado a utilidade de reproduzir por completo as tarefas de etapas como planejamento, análise e projeto, pois foge do escopo de objetivo do exemplo.

Entretanto, vale ressaltar que a atividade de modelagem sempre irá refletir a forma como o software foi planejado, projetado e bem determinado seu contexto de uso e usuários. As práticas e as preocupações da engenharia de software, citadas na introdução do trabalho auxiliam uma modelagem mais precisa, diferentemente caso a prática fosse “pensando e modelando diretamente”, ou seja uma abordagem mais sujeita a alterações e ambiguidades de modelagem, possível falta de reuso de componente modelado. Ter condições para controlar o tempo necessário para que o software seja planejado, modelado e desenvolvido com o passar do tempo confere habilidade à empresa em estabelecer e cumprir prazos pré-determinados, previstos para todas as etapas de criação até a consequente entrega do produto final.

A (figura 12) mostra o que seria a modelagem em uma ferramenta na qual o suporte a rastreabilidade não fosse semi-automático ou que isso necessitasse ser feito posteriormente, além de denotar que toda inserção de elementos fora realizada por uma ação *drag and drop* de cada elemento que ficam dispostos na *toolbox* (figura 3) do EA.

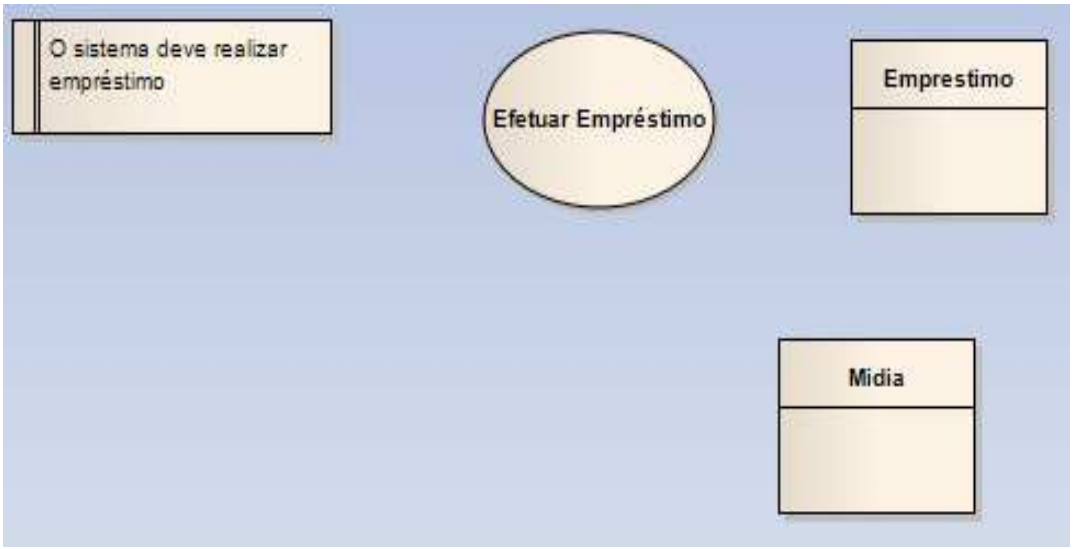


Figura 12: Elementos de modelagem

O contexto de uso do exemplo é o de controle para uma vídeo-locadora, na qual após um planejamento passou-se a modelagem de uma funcionalidade preterida para a aplicação. A (figura 12) mostra a modelagem de uma funcionalidade, vale ressaltar que o fato desses elementos, que representam diferentes abstrações de aspectos necessários estarem dispostos conjuntamente é para melhor entendimento do exemplo, ou seja, não são incomuns modelagens na qual cada elemento é criado separadamente em artefatos diferentes. Assim, quando desejado, eventualmente reunir os elementos de uma modelagem que pertençam ou não a artefatos diferentes em um único artefato diagrama, permite uma visualização abrangente de todos os elementos até então criados.

Ter uma visualização dos relacionamentos que possa existir entre os elementos de modelagem ajuda a manter uma visão geral da modelagem. Uma ferramenta CASE pode oferecer suporte a rastreabilidade de dois modos diferentes para que sejam registradas as RR entre elementos de modelagem: O registro das RR feito posteriormente a (técnica de matriz de relacionamentos, disponível no *EA*) modelagem dos aspectos necessários ou gradativamente (técnica implementada no trabalho, RI) à medida que cada elemento é criado, exceto na inserção do elemento base, se desejado estabelecer uma RR com outro elemento, então essa relação será modelada já no momento dessa inserção garantida pela operação de derivação.

Como já foi dito, a visualização de RR é importante em uma análise de impacto de mudanças e permitir essa funcionabilidade semi-automática de criação de RR entre elementos de modelagem de software é o diferencial frente a outras técnicas.

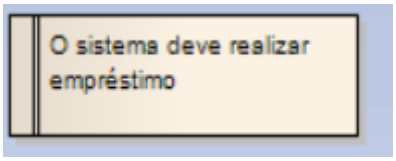


Figura 13: Elemento requisito “O sistema deve realizar empréstimo”

Nas decisões de planejamento para o software de controle de vídeo-locadora chegou-se a conclusão da necessidade de um elemento base requisito: “O sistema deve realizar empréstimo” (figura 13), posteriormente (durante iterações de processo, refinamento de modelagem, citado na introdução) ter-se-ia mais claro o que precisaria ser feito para atender tal requisito, descrevendo-o em um elemento caso de uso “Efetuar Empréstimo” (figura 14), um próximo passo seria a tarefa de modelar as classes necessárias, no exemplo um elemento classe “Empréstimo” (figura 15) foi determinada para que atendesse ao caso de uso.

Figura 14: Elemento caso de uso “Efetuar Empréstimo”

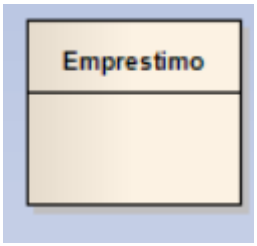


Figura 15: Elemento classe “Empréstimo”

Assumindo como referência os passos da sequências de ações que resultaram na criação de cada elemento e seguindo a ordem descrita, uma ferramenta com suporte semi-automático

de rastreabilidade, disponível a partir de agora no *EA* com o plug-in desenvolvido, necessitaria dos seguintes passos e produziria a seguinte modelagem:

A operação de derivar um caso permite determinar que seja escolhido um artefato diferente (figura 16) para o caso de uso derivado, no exemplo pela (figura 16) deseja-se que o elemento derivado, caso de uso "[D] O sistema deve realizar empréstimo", seja disposto no artefato "Diagrama de Caso de Uso" contido no pacote Casos de Uso", devendo então ser previamente selecionado. O elemento requisito "O sistema deve realizar empréstimo" (figura 13), assumido como elemento base, inserido conforme descrição na seção 5.1.1., será também o elemento-causa da operação de derivação (que estabelece uma RR entre elemento-causa e elemento derivado) e deve ser selecionado no diagrama o qual ele está disposto, com botão direito escolhendo a operação desejada (figura 17) em Add-Ins / Rastreabilidade Add-in / [> Derivar Caso de Uso Alt + A + R + U <] . Dispondo os elementos até então criados (figura 18) em um mesmo artefato diagrama, a visualização de relacionamentos fica explicita entre os elementos existentes que tem RR.

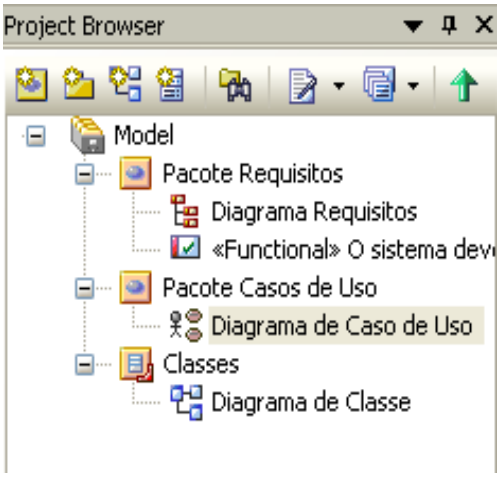


Figura 16: Derivação (artefato destino)

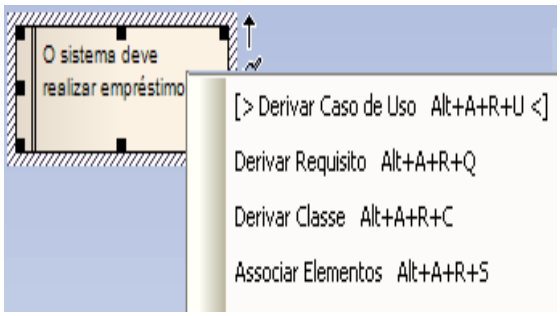


Figura 17: Derivação (elemento-causa)

(Vale ressaltar que o plug-in permite que um elemento que seja a causa da derivação não necessite ser selecionado via um diagrama, assim será entendido que trata de uma derivação a qual o elemento derivado não será disposto em um artefato diferente e será criada uma RR entre o elemento-causa com o elemento-efeito desejado. Para tal seleciona-se o elemento-causa através do *Project Browser* clicando-se com o botão direito do mouse nele e escolhendo a derivação para o tipo de elemento derivado requerido).

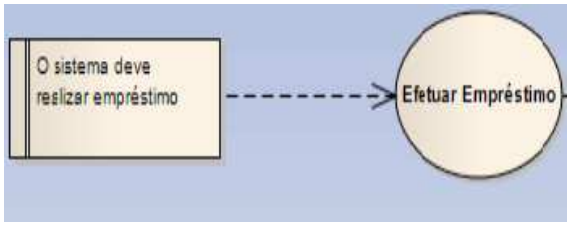


Figura 18: Rastreabilidade entre elementos

A operação de derivar uma classe, no exemplo é previamente especificado um artefato destino (figura 19) para o elemento classe derivado e deseja-se que esse elemento classe "[D] O sistema deve realizar empréstimo" esteja contido no pacote "Classes" e seja disposto em um Diagrama "Diagrama Classes" (figura 20). O elemento caso de uso "[D] O sistema deve realizar empréstimo" o qual será o elemento-causa da derivação, deve ser selecionado no diagrama no qual ele está disposto, com botão direito do mouse e escolhendo a operação desejada em Add-Ins / Rastreabilidade Add-in / [> Derivar Caso de Uso Alt + A + R + U <] . Dispondo os elementos até então criados (figura 21) a visualização de relacionamentos fica explicita entre os elementos existentes que tem relacionamento de rastreabilidade

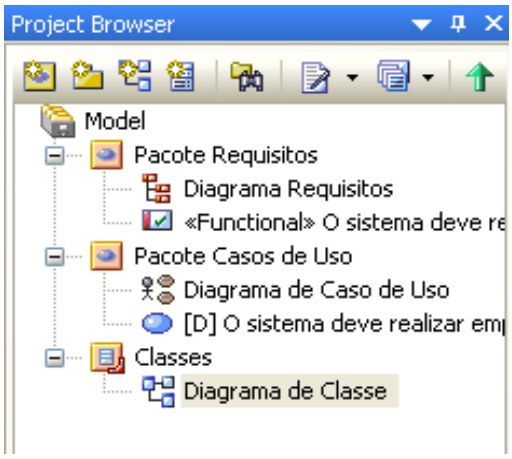


Figura 19: Derivação (artefato destino)

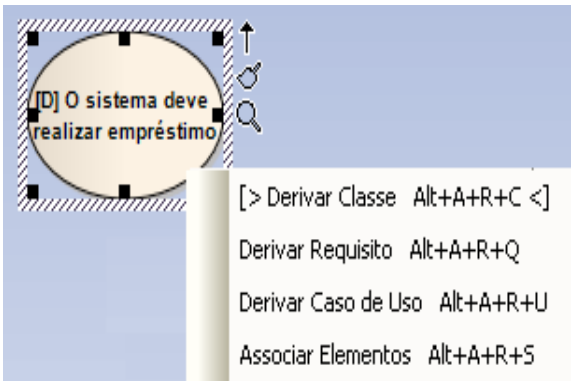


Figura 20: Derivação (elemento-causa)

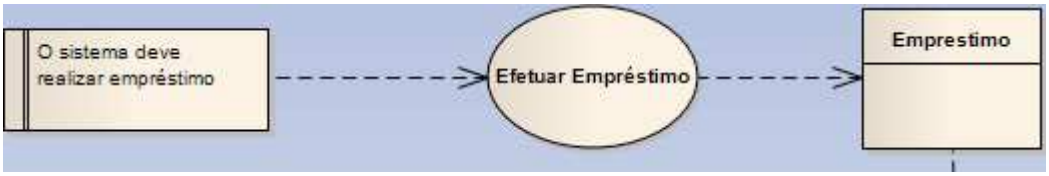


Figura 21: Rastreabilidade entre elementos

Outra operação permitida pelo plug-in (descrita na seção 5.1) é “Associar elementos” já existentes e que possivelmente devido a decisões de projeto foi necessária alguma mudança na modelagem, que seria a criação de RR entre dois elementos já modelados. No exemplo em questão, desejam-se associar o elemento-efeito classe “Mídia” com o elemento-causa classe “[D] O sistema deve realizar empréstimo”. Para tal seleciona-se previamente o elemento-efeito (figura 22), que pertence ao pacote “Classes” e posteriormente o elemento-causa, pelo artefato diagrama que o contenha “Diagrama de Classe”, com botão direito do mouse e escolhendo a operação em Add-Ins / Rastreabilidade Add-in / “Associar Elementos Alt+A+R+S” (figura 23). Dispondo os elementos até então criados (figura 24) a visualização de relacionamentos fica explicita entre os elementos existentes que tem algum RR.

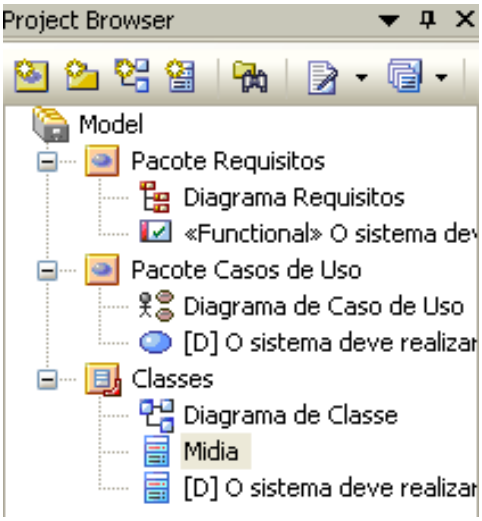


Figura 22: Escolha elemento-efeito

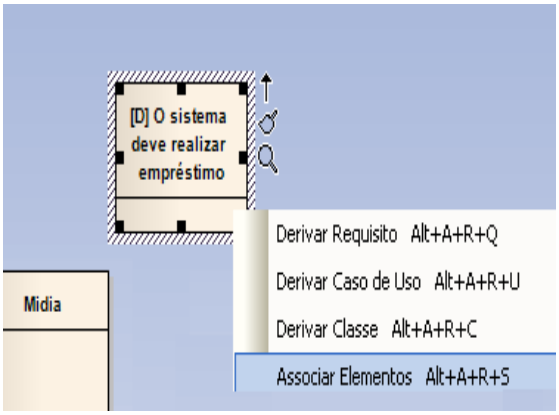


Figura 23: Associar elementos

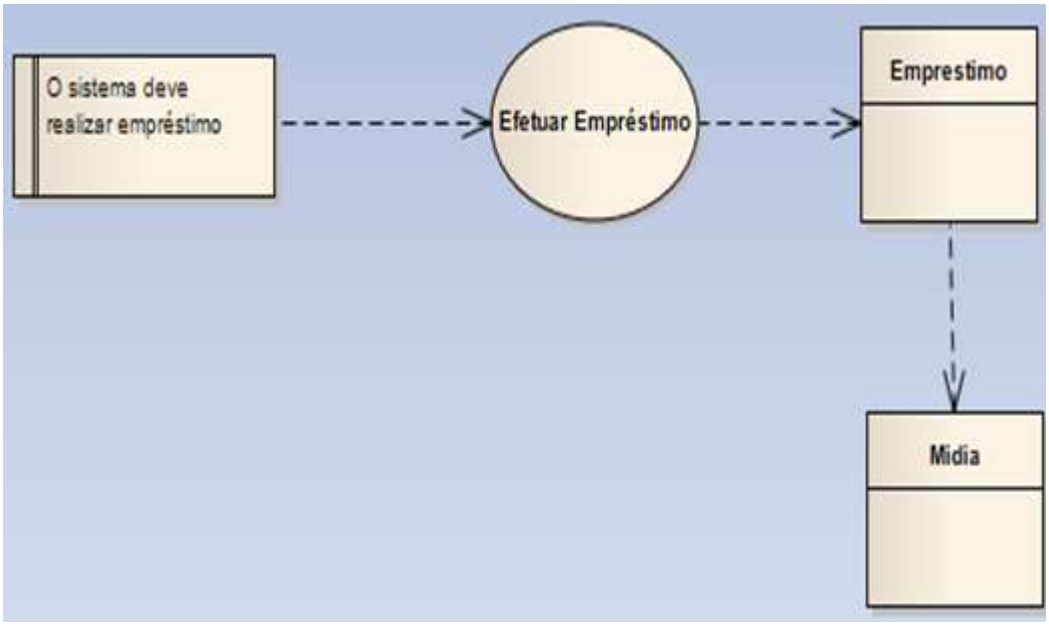


Figura 24: Rastreabilidade entre elementos

5.4 RESULTADOS DE UM ESTUDO DE CASO

Com base na obra de (SANTOS; WAZLAWICK, 2009) e na parte de implementação desenvolvida no escopo desse trabalho, foi realizado pelos mesmos autores um estudo de caso, a fim de medir a eficiência da aplicabilidade prática dessa teoria, no contexto de desenvolvimento de software. Para mensurar tal eficiência comparou-se a eficiência das operações propostas na RI frente às operações mais convencionais levando-se em conta o tempo para cumprir a tarefa determinada além de estimar número de passos observados para o completá-la.

A descrição de uma tarefa tipicamente praticada em ambientes de desenvolvimento de software, que está exemplificada no tópico 5.3, na qual para o contexto de uso buscado, o importante era a modelagem grandes quantidades de relações de rastreabilidade previamente planejadas. Para o estudo em questão os elementos do *EA* utilizados foram: diagrama de requisitos, diagrama de casos de uso, e modelo conceitual (no exemplo, equivalente ao pacote “Classes”).

No estudo de caso realizado a objetivo seria medir o tempo levado na tarefa simulada de desenvolvimento de um sistema para vídeo-locadora, a qual consistia na criação de rastreabilidade entre os elementos dos artefatos modelados.

O resultado do estudo, que pode ser consultado por completo no apêndice B contém detalhadamente em que consistia a tarefa que deveria ser realizada utilizando-se a técnica de RI (detalhada ao longo desse trabalho) além da descrição de outra técnica a ser comparada e sua respectiva aplicação na tarefa determinada. Por fim os autores (SANTOS; WAZLAWICK, 2009) reportam como resultado:

> "Considerando-se que a criação de elementos base (requisitos) tende a ocorrer com menos frequência do que a derivação de outros elementos durante o processo de desenvolvimento de software, já que artefatos são gerados uns a partir dos outros ao longo da maioria dos processos existentes, pode-se inferir que a técnica indutiva deve reduzir significativamente o número de operações que devem ser realizadas pelo desenvolvedor caso este deseje manter as relações de rastreabilidade entre os artefatos criados. Este fato pode ser observado na Tabela 2, onde das doze operações executadas para a técnica indutiva dez são de derivação."

> "Em cinco situações a técnica indutiva mostrou-se equivalente à técnica convencional. Mas em outras três, ela apresenta vantagens em relação à técnica convencional, pois o desenvolvedor pode executar a mesma tarefa com menos operações."

> "Uma das principais vantagens da técnica indutiva encontra-se na operação de derivação, que apresentou um significativo desempenho, pois ao criar um novo elemento em um artefato ela também cria automaticamente a relação de rastreabilidade entre este elemento e sua causa. A derivação é uma operação frequente no desenvolvimento de sistemas. Portanto, a vantagem da técnica indutiva no caso desta operação deve redundar em significativo ganho de produtividade."

6. CONCLUSÃO E TRABALHOS FUTUROS

Diante ao que foi proposto como objetivos para esse trabalho e pela constatação prática do potencial que a implementação da técnica de RI, sob a forma de um plug-in, apresentou tanto no exemplo da seção 5.3 quanto no estudo de caso realizado (apêndice B), melhor relação no número de passos exigidos para tornar os elementos rastreáveis (aqueles que tenham RR com outros) em comparação a outras técnicas. Assim sendo, pode-se dizer que objetivos foram atingidos e o plug-in proposto apresenta potencial de uso mercadológico, dentro do contexto de uso descrito ao longo do trabalho.

Para atingir os objetivos gerais traçados, a busca para completar também os objetivos específicos foi satisfatória, como dominar a parte de desenvolvimento de *add-ins* (o plug-in se mostrou estável, proporcionando uma interação clara com o usuário) do *EA* e assim aprimorar o conhecimento dos elementos de modelagem, abordados na *UML*, disponíveis na ferramenta. O que reforçou o significado de cada elemento, além da importância frente ao uso (que não é obrigatória) dos mesmos, para que mesmo com aumento gradativo das especificações não se perca a clareza de cada elemento modelado, diante um processo de criação de software.

Os possíveis trabalhos futuros que podem ser realizados dizem respeito à elaboração de uma interface ágil atuando como alternativa a aplicação das operações da rastreabilidade indutiva via menu e seleção de destino. Ainda a formulação de ações semânticas visuais denotando que diante alterações que eventualmente envolvam elementos que tenham alguma RR, seja sinalizado que o efeito de tal alteração poderá afetar outros elementos ou uma RR existente. Outra considerável opção para trabalhos futuros, seria desenvolver atalhos de comando mais acessíveis para as operações implementadas, mas essa alternativa somente seria possível caso o *EA* passasse a permitir a criação de atalhos para aplicações *add-in*.

.

REFERÊNCIAS BIBLIOGRÁFICAS

CERRI, Elisa C. e. *Um modelo de rastreabilidade entre o documento de especificação de requisitos e o modelo de casos de uso do sistema*. 2007. 190f. Dissertação (Mestrado) – Fac. de Informática, PUCRS, Porto Alegre, 2007.

RUMBAUGH, James; JACOBSON, Ivar; BOOCH, Grady. *The unified modeling language reference manual.* Boston: Addison-Wesley, 1999.

CRUZ, J. L. e JINO, M. e Crespo, A. N. e ARGOLLO, M. *Suporte automatizado à rastreabilidade em um processo de teste de software baseado em documentação,* em: V Simpósio Brasileiro de Qualidade de Software – SBQS´2006.

LARMAN, C. *Utilizando UML e padrões: uma introdução à análise e ao projeto orientados a objetos e ao desenvolvimento iterativo*. 3. ed. Porto Alegre: Bookman, 2007.

PRESSMAN, Roger S. *Engenharia de Software.* 3. ed. São Paulo: Ed Makron Books, 1995.

SANTOS, R. N. e WAZLAWICK, R. S. *Rastreabilidade Indutiva Aplicada a Artefatos de Software e Relações de Rastreabilidade,* em: *EXPERIMENTAL SOFTWARE ENGINEERING LATIN AMERICAN WORKSHOP,* VI, 2009, São Carlos: DC/UFSCAR, 2009.

SAYÃO, M. e LEITE, J. C. S. P. *Rastreabilidade de Requisitos.* Relatório Técnico 2005, DI/PUC-Rio, 2005.

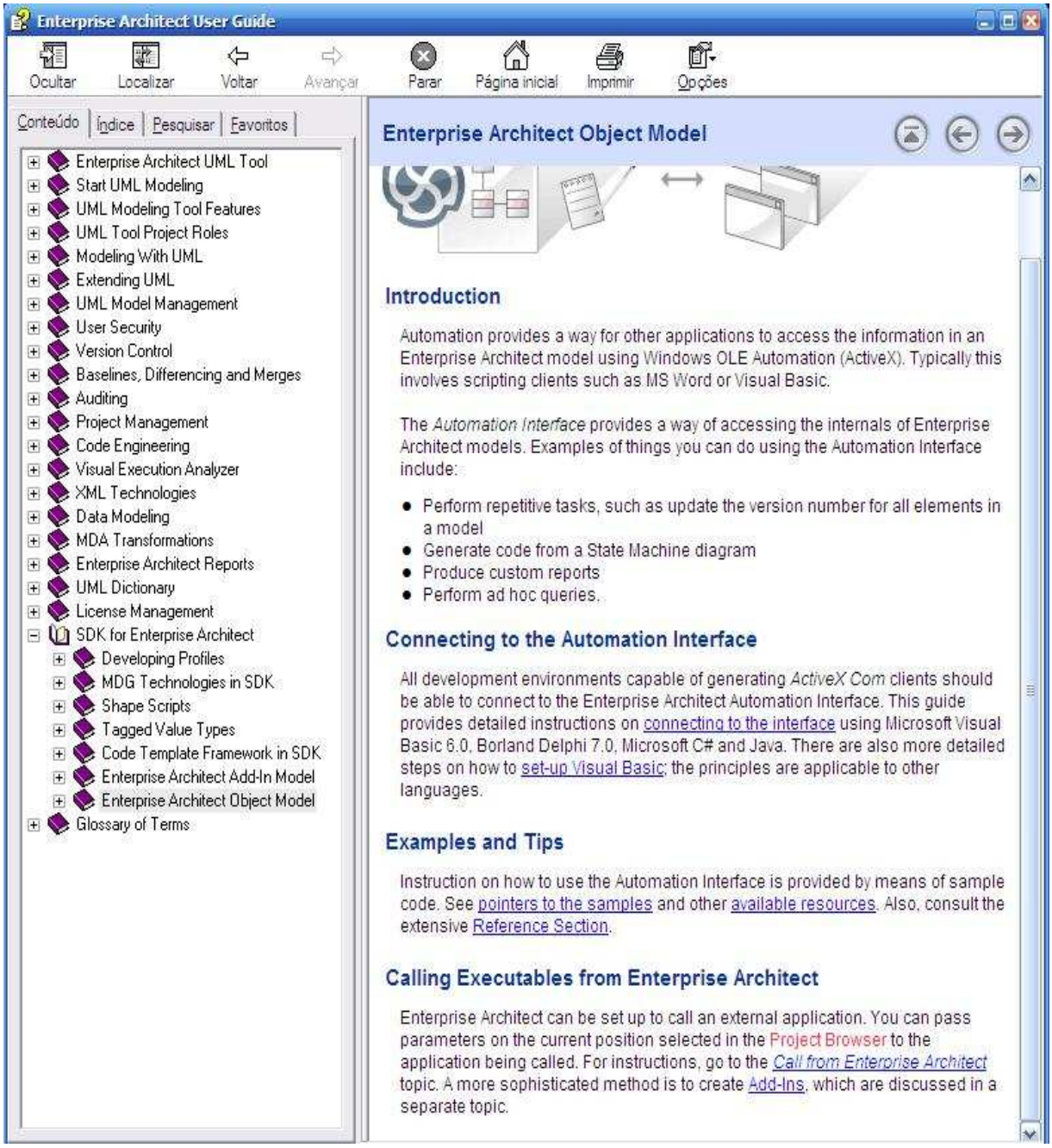
SOMMERVILLE, Ian. *Software Engineering*. 5.ed. : Addison Wesley, 2000.

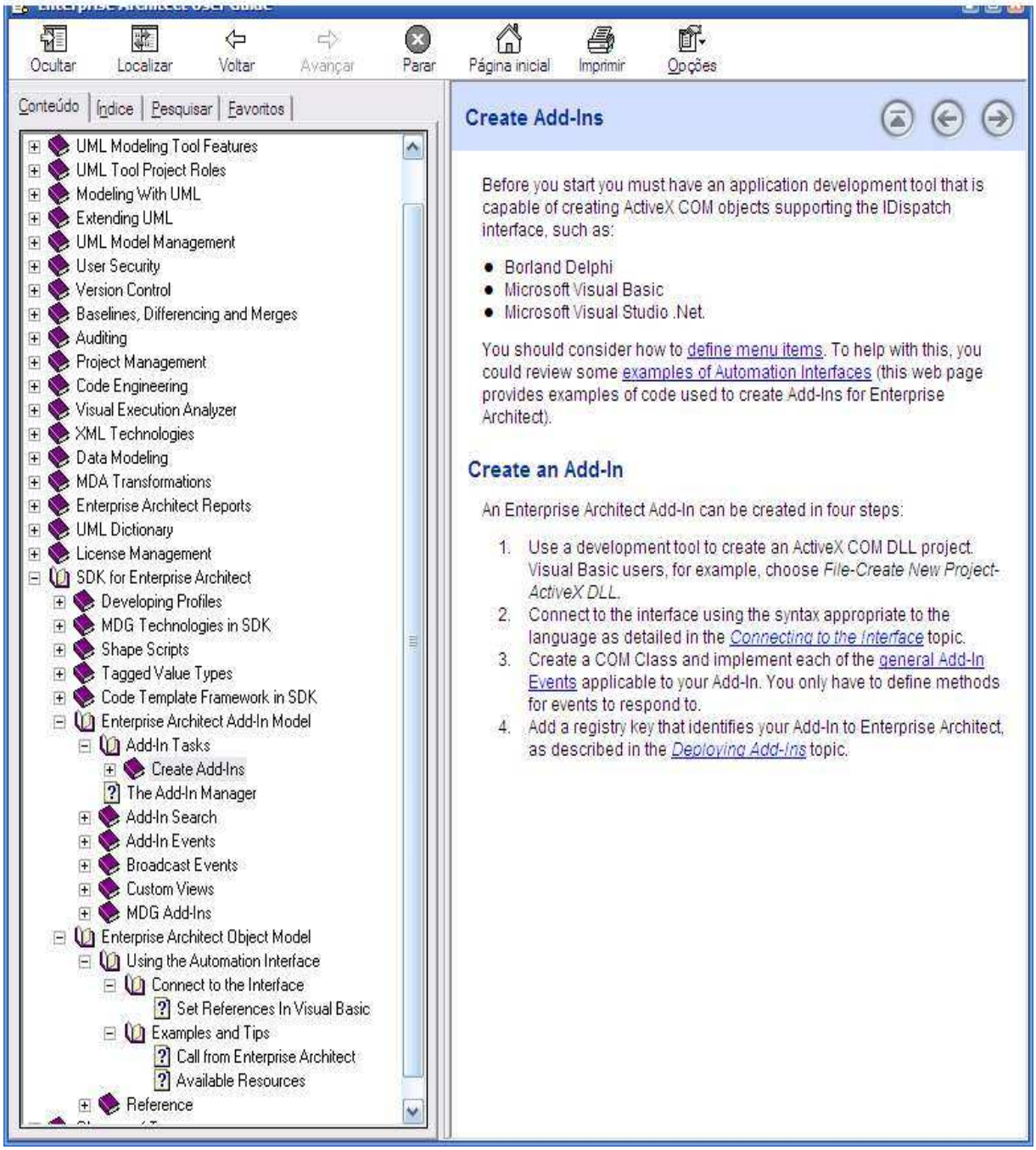
WAZLAWICK, R. S. *Análise e Projeto de Sistemas de Informação Orientados a Objetos*. 2.ed. : Campus, 2004.

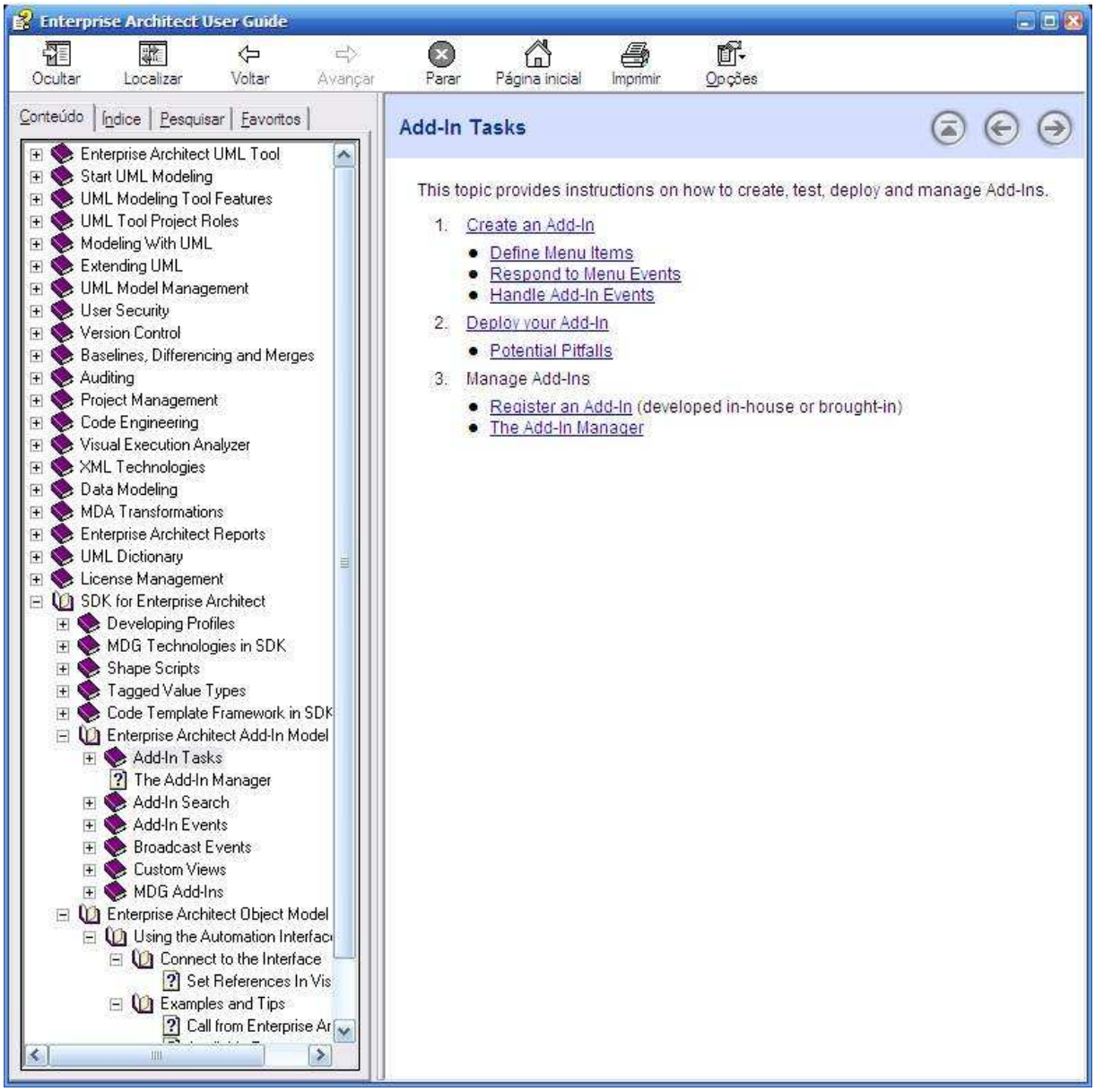
ANEXOS

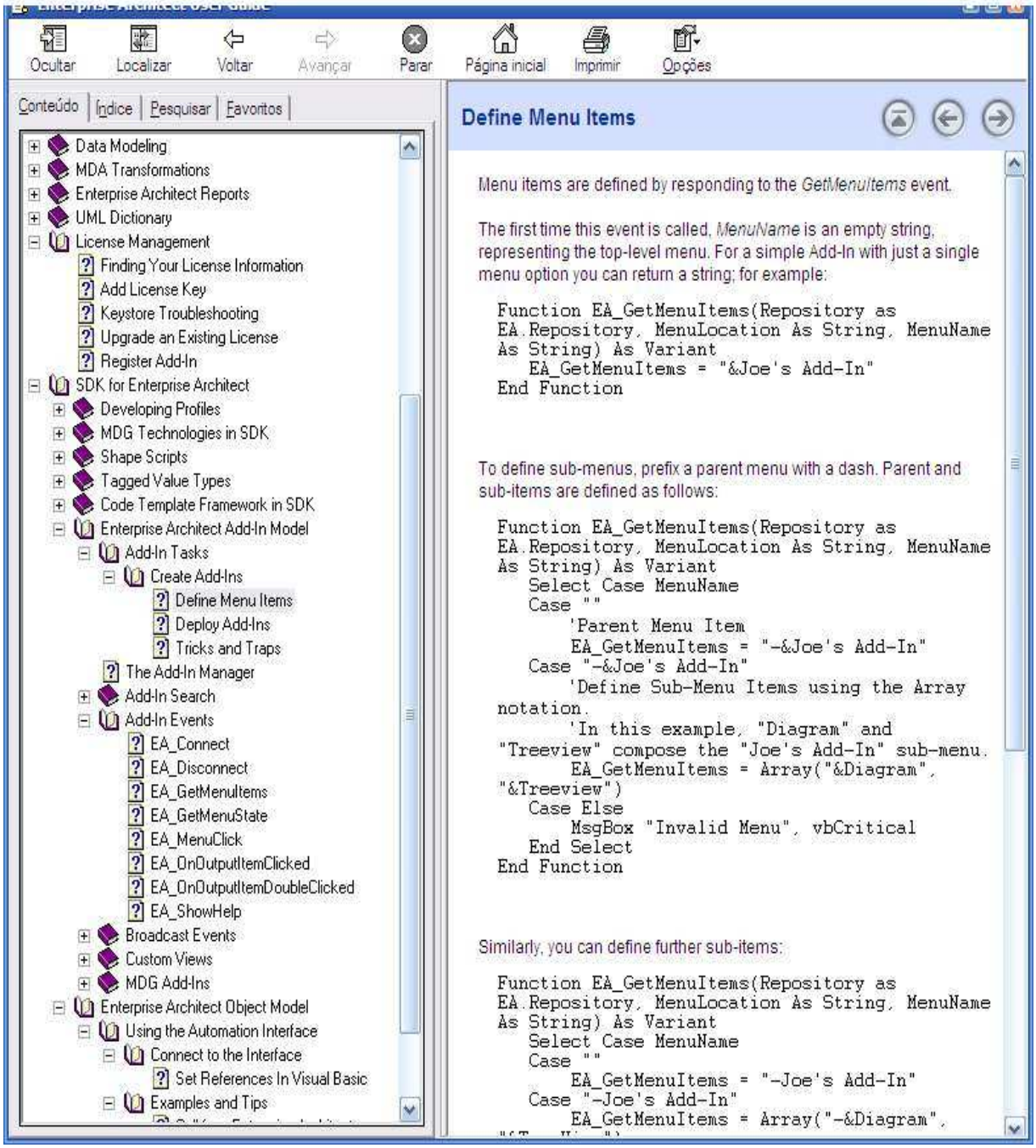
ANEXO A

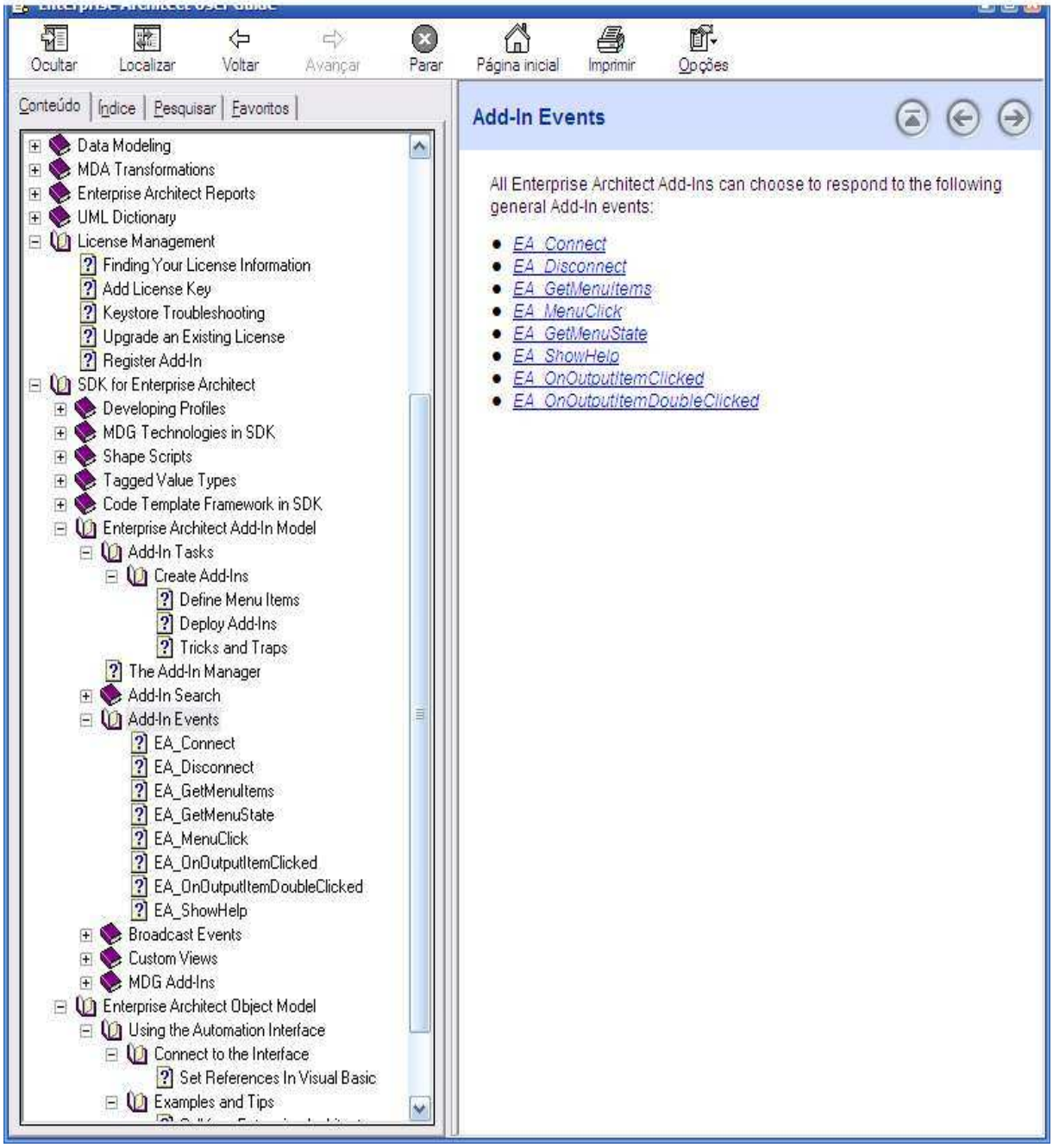
INTERFACE DE DESENVOLVIMENTO DO AMBIENTE ENTERPRISE ARCHITECT

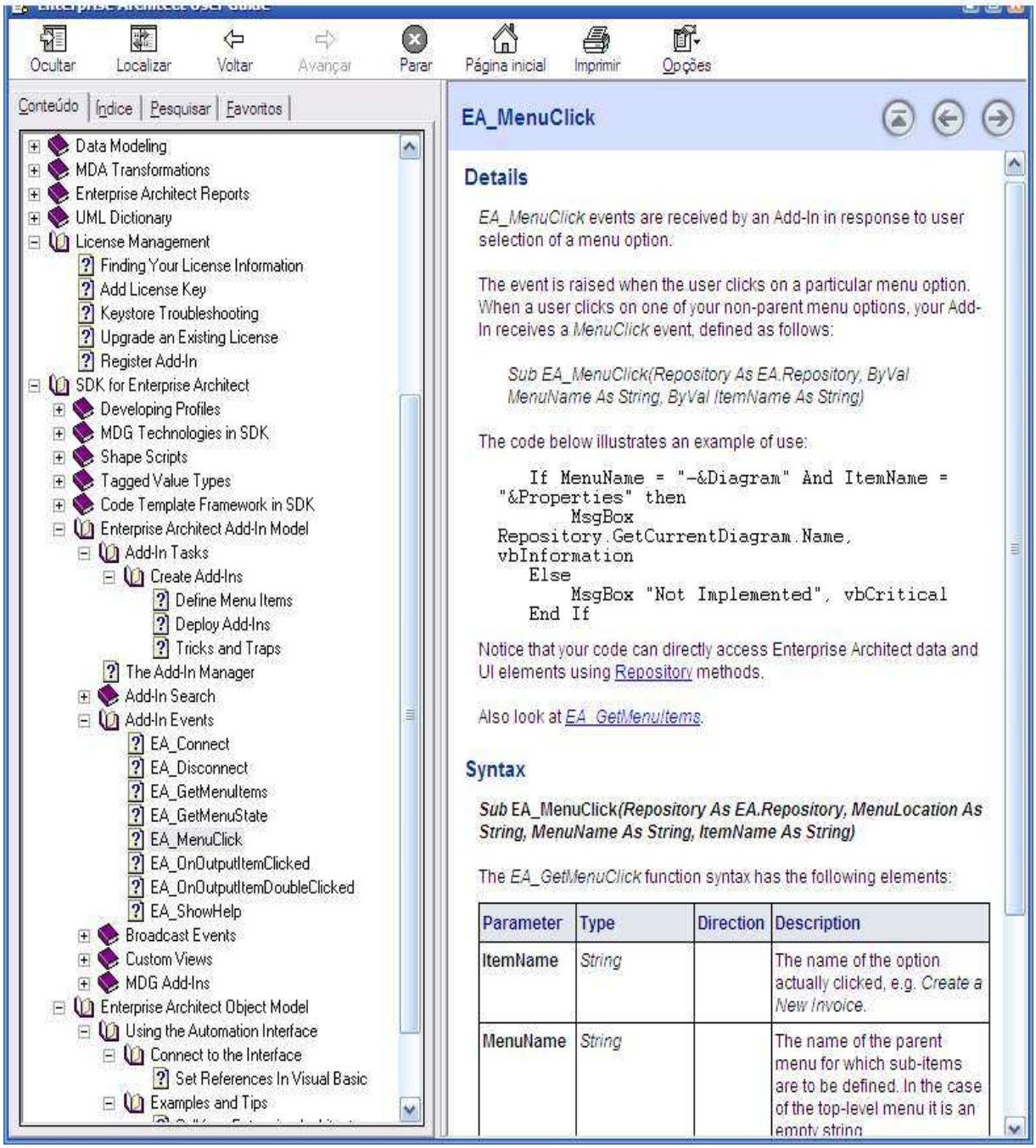

Enterprise Architect User Guide
Ocultar
Localizar
Voltar
Avançar
Parar
Página inicial
Imprimir
Opções
Conteúdo
Índice
Pesquisar
Favoritos
Enterprise Architect UML Tool
Start UML Modeling
UML Modeling Tool Features
UML Tool Project Roles
Modeling With UML
Extending UML
UML Model Management
User Security
Version Control
Baselines, Differencing and Merges
Auditing
Project Management
Code Engineering
Visual Execution Analyzer
XML Technologies
Data Modeling
MDA Transformations
Enterprise Architect Reports
UML Dictionary
License Management
SDK for Enterprise Architect
Developing Profiles
MDG Technologies in SDK
Shape Scripts
Tagged Value Types
Code Template Framework in SDK
Enterprise Architect Add-In Model
Enterprise Architect Object Model
Glossary of Terms
Enterprise Architect Object Model
Introduction
Automation provides a way for other applications to access the information in an Enterprise Architect model using Windows OLE Automation (ActiveX). Typically this involves scripting clients such as MS Word or Visual Basic.
The Automation Interface provides a way of accessing the internals of Enterprise Architect models. Examples of things you can do using the Automation Interface include:
Perform repetitive tasks, such as update the version number for all elements in a model
Generate code from a State Machine diagram
Produce custom reports
Perform ad hoc queries.
Connecting to the Automation Interface
All development environments capable of generating ActiveX Com clients should be able to connect to the Enterprise Architect Automation Interface. This guide provides detailed instructions on connecting to the interface using Microsoft Visual Basic 6.0, Borland Delphi 7.0, Microsoft C# and Java. There are also more detailed steps on how to set-up Visual Basic; the principles are applicable to other languages.
Examples and Tips
Instruction on how to use the Automation Interface is provided by means of sample code. See pointers to the samples and other available resources. Also, consult the extensive Reference Section.
Calling Executables from Enterprise Architect
Enterprise Architect can be set up to call an external application. You can pass parameters on the current position selected in the Project Browser to the application being called. For instructions, go to the Call from Enterprise Architect topic. A more sophisticated method is to create Add-Ins, which are discussed in a separate topic.

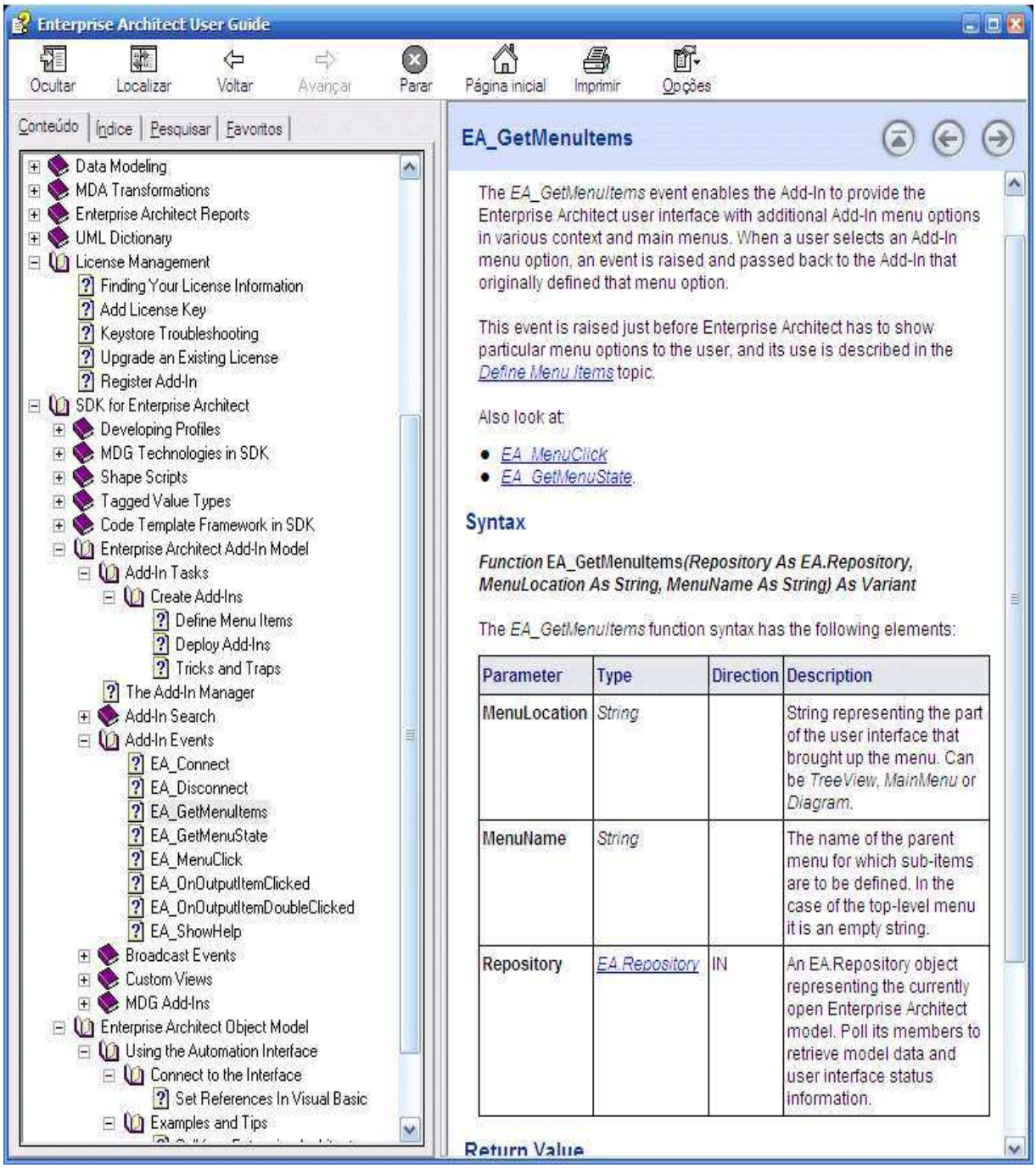

Ocultar
Localizar
Voltar
Avançar
Parar
Página inicial
Imprimir
Opções
Conteúdo
Índice
Pesquisar
Favoritos
UML Modeling Tool Features
UML Tool Project Roles
Modeling With UML
Extending UML
UML Model Management
User Security
Version Control
Baselines, Differencing and Merges
Auditing
Project Management
Code Engineering
Visual Execution Analyzer
XML Technologies
Data Modeling
MDA Transformations
Enterprise Architect Reports
UML Dictionary
License Management
SDK for Enterprise Architect
Developing Profiles
MDG Technologies in SDK
Shape Scripts
Tagged Value Types
Code Template Framework in SDK
Enterprise Architect Add-In Model
Add-In Tasks
Create Add-Ins
The Add-In Manager
Add-In Search
Add-In Events
Broadcast Events
Custom Views
MDG Add-Ins
Enterprise Architect Object Model
Using the Automation Interface
Connect to the Interface
Set References In Visual Basic
Examples and Tips
Call from Enterprise Architect
Available Resources
Reference
Create Add-Ins
Before you start you must have an application development tool that is capable of creating ActiveX COM objects supporting the IDispatch interface, such as:
Borland Delphi
Microsoft Visual Basic
Microsoft Visual Studio .Net.
You should consider how to define menu items. To help with this, you could review some examples of Automation Interfaces (this web page provides examples of code used to create Add-Ins for Enterprise Architect).
Create an Add-In
An Enterprise Architect Add-In can be created in four steps:
1. Use a development tool to create an ActiveX COM DLL project. Visual Basic users, for example, choose File-Create New Project-ActiveX DLL.
2. Connect to the interface using the syntax appropriate to the language as detailed in the Connecting to the Interface topic.
3. Create a COM Class and implement each of the general Add-In Events applicable to your Add-In. You only have to define methods for events to respond to.
4. Add a registry key that identifies your Add-In to Enterprise Architect, as described in the Deploying Add-Ins topic.

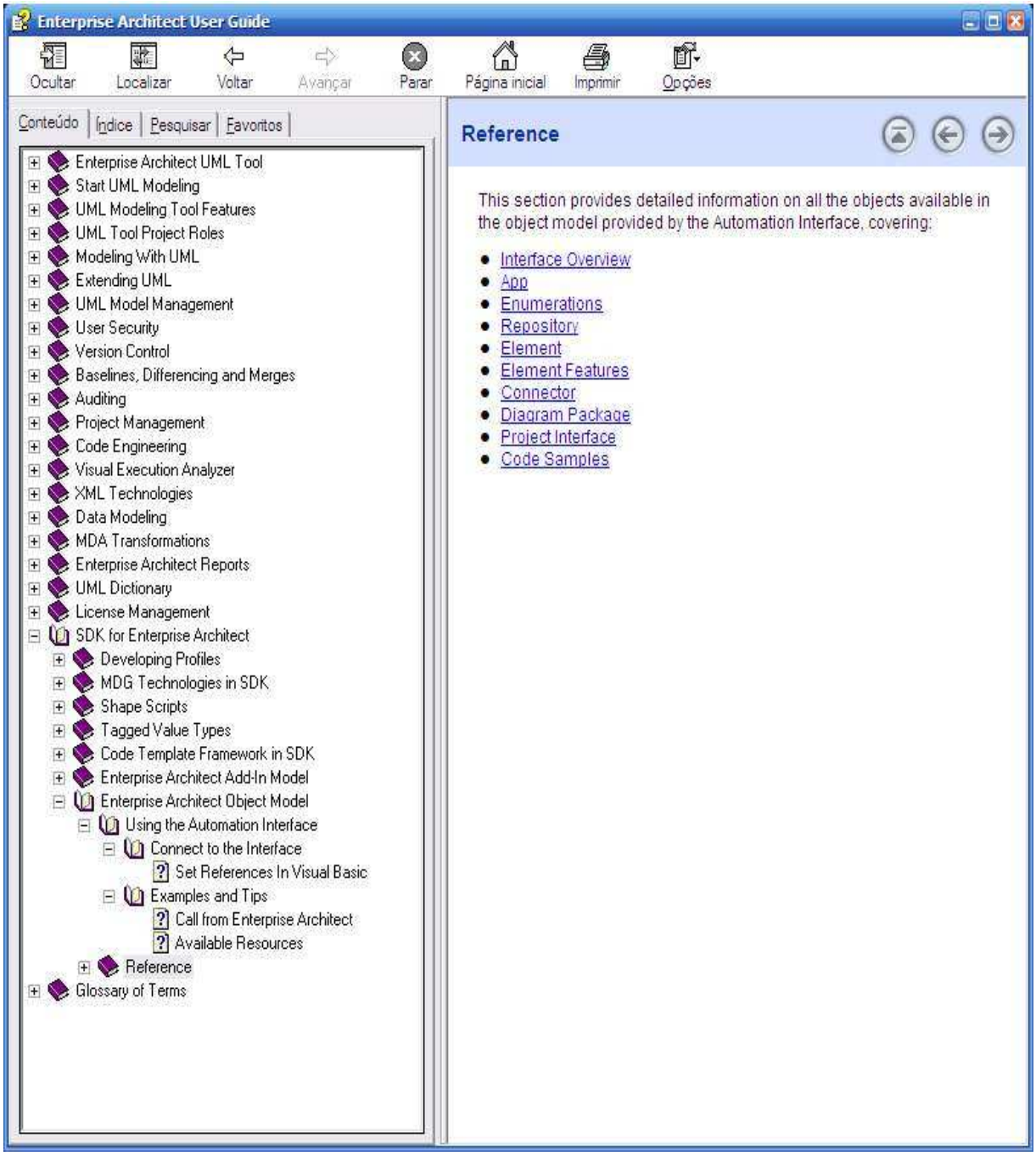

Enterprise Architect User Guide
Ocultar
Localizar
Voltar
Avançar
Parar
Página inicial
Imprimir
Opções
Conteúdo
Índice
Pesquisar
Favoritos
Enterprise Architect UML Tool
Start UML Modeling
UML Modeling Tool Features
UML Tool Project Roles
Modeling With UML
Extending UML
UML Model Management
User Security
Version Control
Baselines, Differencing and Merges
Auditing
Project Management
Code Engineering
Visual Execution Analyzer
XML Technologies
Data Modeling
MDA Transformations
Enterprise Architect Reports
UML Dictionary
License Management
SDK for Enterprise Architect
Developing Profiles
MDG Technologies in SDK
Shape Scripts
Tagged Value Types
Code Template Framework in SDK
Enterprise Architect Add-In Model
Add-In Tasks
The Add-In Manager
Add-In Search
Add-In Events
Broadcast Events
Custom Views
MDG Add-Ins
Enterprise Architect Object Model
Using the Automation Interfac
Connect to the Interface
Set References In Vis
Examples and Tips
Call from Enterprise Ar
Add-In Tasks
This topic provides instructions on how to create, test, deploy and manage Add-Ins.
1. Create an Add-In
Define Menu Items
Respond to Menu Events
Handle Add-In Events
2. Deploy your Add-In
Potential Pitfalls
3. Manage Add-Ins
Register an Add-In (developed in-house or brought-in)
The Add-In Manager

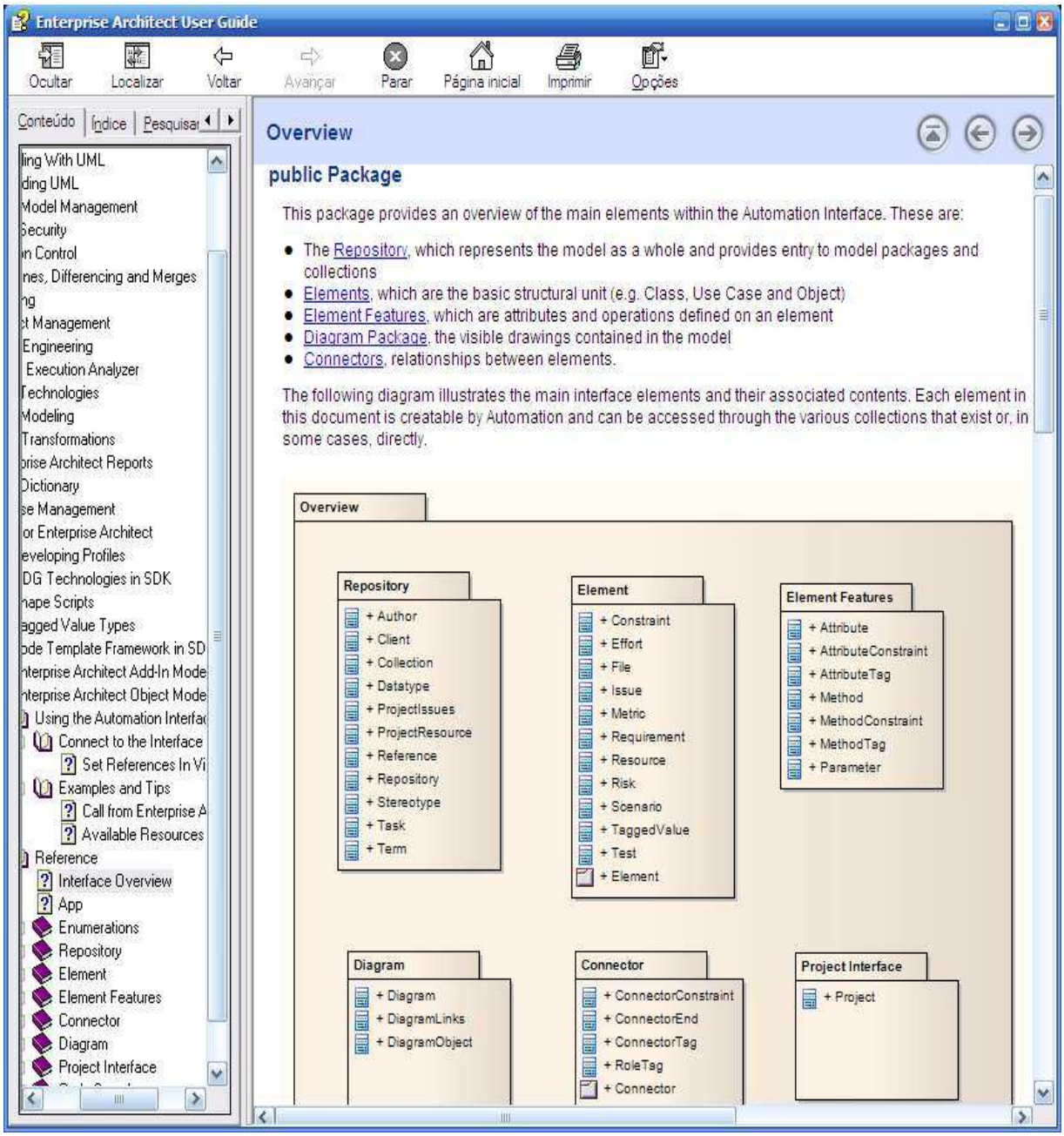

Ocultar
Localizar
Voltar
Avançar
Parar
Página inicial
Imprimir
Opções
Conteúdo
Índice
Pesquisar
Favoritos
Data Modeling
MDA Transformations
Enterprise Architect Reports
UML Dictionary
License Management
Finding Your License Information
Add License Key
Keystore Troubleshooting
Upgrade an Existing License
Register Add-In
SDK for Enterprise Architect
Developing Profiles
MDG Technologies in SDK
Shape Scripts
Tagged Value Types
Code Template Framework in SDK
Enterprise Architect Add-In Model
Add-In Tasks
Create Add-Ins
Define Menu Items
Deploy Add-Ins
Tricks and Traps
The Add-In Manager
Add-In Search
Add-In Events
EA_Connect
EA_Disconnect
EA_GetMenuItems
EA_GetMenuState
EA_MenuClick
EA_OnOutputItemClicked
EA_OnOutputItemDoubleClicked
EA_ShowHelp
Broadcast Events
Custom Views
MDG Add-Ins
Enterprise Architect Object Model
Using the Automation Interface
Connect to the Interface
Set References In Visual Basic
Examples and Tips
Define Menu Items
Menu items are defined by responding to the GetMenuItems event.
The first time this event is called, MenuName is an empty string, representing the top-level menu. For a simple Add-In with just a single menu option you can return a string; for example:
Function EA_GetMenuItems(Repository as EA.Repository, MenuLocation As String, MenuName As String) As Variant
EA_GetMenuItems = "&Joe's Add-In"
End Function
To define sub-menus, prefix a parent menu with a dash. Parent and sub-items are defined as follows:
Function EA_GetMenuItems(Repository as EA.Repository, MenuLocation As String, MenuName As String) As Variant
Select Case MenuName
Case ""
'Parent Menu Item
EA_GetMenuItems = "-&Joe's Add-In"
Case "-&Joe's Add-In"
'Define Sub-Menu Items using the Array notation.
'In this example, "Diagram" and "Treeview" compose the "Joe's Add-In" sub-menu.
EA_GetMenuItems = Array("&Diagram", "&Treeview")
Case Else
MsgBox "Invalid Menu", vbCritical
End Select
End Function
Similarly, you can define further sub-items:
Function EA_GetMenuItems(Repository as EA.Repository, MenuLocation As String, MenuName As String) As Variant
Select Case MenuName
Case ""
EA_GetMenuItems = "-Joe's Add-In"
Case "-Joe's Add-In"
EA_GetMenuItems = Array("-&Diagram",

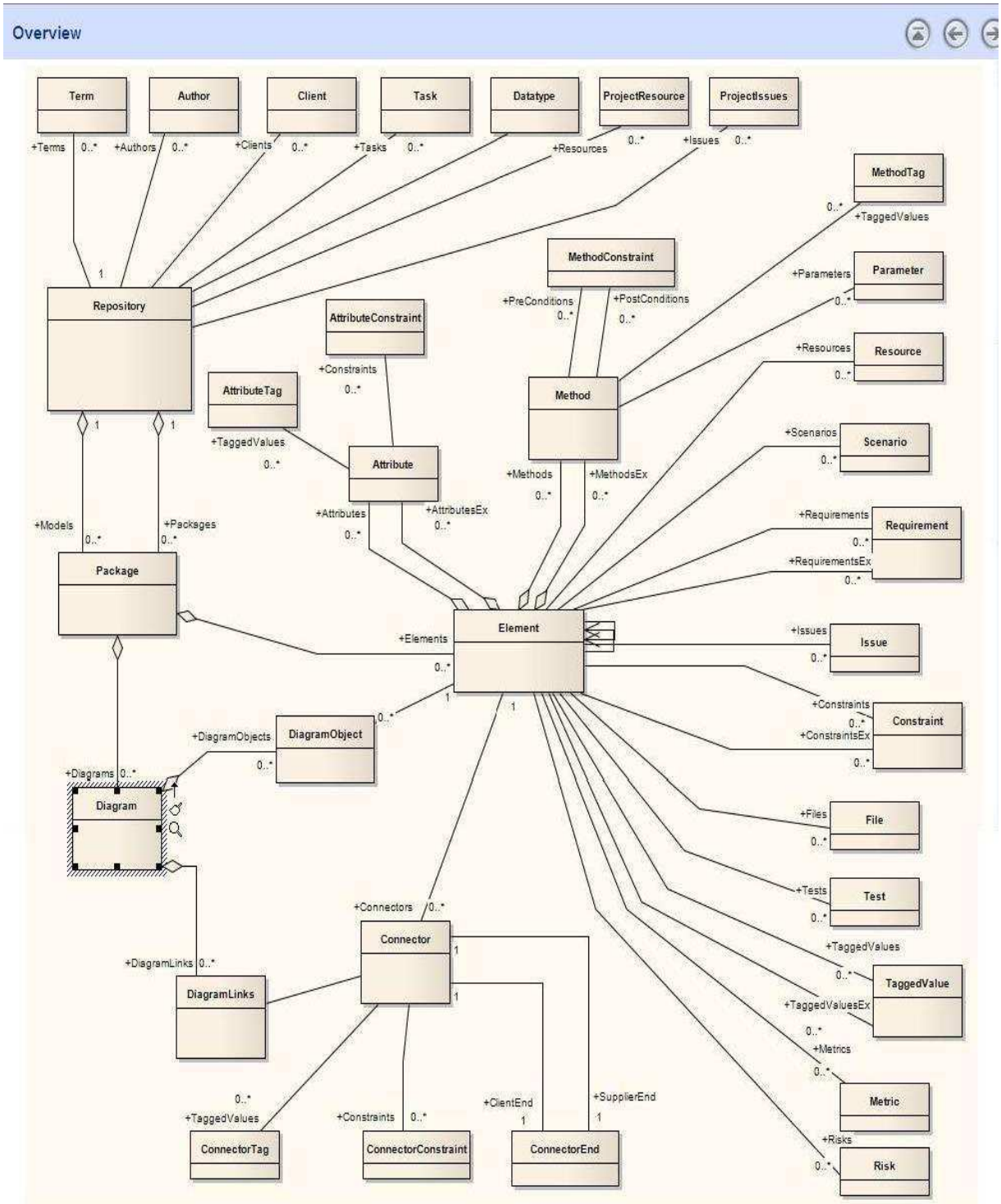

Ocultar
Localizar
Voltar
Avançar
Parar
Página inicial
Imprimir
Opções
Conteúdo
Índice
Pesquisar
Favoritos
Data Modeling
MDA Transformations
Enterprise Architect Reports
UML Dictionary
License Management
Finding Your License Information
Add License Key
Keystore Troubleshooting
Upgrade an Existing License
Register Add-In
SDK for Enterprise Architect
Developing Profiles
MDG Technologies in SDK
Shape Scripts
Tagged Value Types
Code Template Framework in SDK
Enterprise Architect Add-In Model
Add-In Tasks
Create Add-Ins
Define Menu Items
Deploy Add-Ins
Tricks and Traps
The Add-In Manager
Add-In Search
Add-In Events
EA_Connect
EA_Disconnect
EA_GetMenuItems
EA_GetMenuState
EA_MenuClick
EA_OnOutputItemClicked
EA_OnOutputItemDoubleClicked
EA_ShowHelp
Broadcast Events
Custom Views
MDG Add-Ins
Enterprise Architect Object Model
Using the Automation Interface
Connect to the Interface
Set References In Visual Basic
Examples and Tips
Add-In Events
All Enterprise Architect Add-Ins can choose to respond to the following general Add-In events:
EA_Connect
EA_Disconnect
EA_GetMenuItems
EA_MenuClick
EA_GetMenuState
EA_ShowHelp
EA_OnOutputItemClicked
EA_OnOutputItemDoubleClicked


Ocultar
Localizar
Voltar
Avançar
Parar
Página inicial
Imprimir
Opções
Conteúdo
Índice
Pesquisar
Favoritos
Data Modeling
MDA Transformations
Enterprise Architect Reports
UML Dictionary
License Management
Finding Your License Information
Add License Key
Keystore Troubleshooting
Upgrade an Existing License
Register Add-In
SDK for Enterprise Architect
Developing Profiles
MDG Technologies in SDK
Shape Scripts
Tagged Value Types
Code Template Framework in SDK
Enterprise Architect Add-In Model
Add-In Tasks
Create Add-Ins
Define Menu Items
Deploy Add-Ins
Tricks and Traps
The Add-In Manager
Add-In Search
Add-In Events
EA_Connect
EA_Disconnect
EA_GetMenuItems
EA_GetMenuState
EA_MenuClick
EA_OnOutputItemClicked
EA_OnOutputItemDoubleClicked
EA_ShowHelp
Broadcast Events
Custom Views
MDG Add-Ins
Enterprise Architect Object Model
Using the Automation Interface
Connect to the Interface
Set References In Visual Basic
Examples and Tips
EA_MenuClick
Details
EA_MenuClick events are received by an Add-In in response to user selection of a menu option.
The event is raised when the user clicks on a particular menu option. When a user clicks on one of your non-parent menu options, your Add-In receives a MenuClick event, defined as follows:
Sub EA_MenuClick(Repository As EA.Repository, ByVal MenuName As String, ByVal ItemName As String)
The code below illustrates an example of use:
If MenuName = "-&Diagram" And ItemName = "&Properties" then
MsgBox Repository.GetCurrentDiagram.Name, vbInformation
Else
MsgBox "Not Implemented", vbCritical
End If
Notice that your code can directly access Enterprise Architect data and UI elements using Repository methods.
Also look at EA_GetMenuItems.
Syntax
Sub EA_MenuClick(Repository As EA.Repository, MenuLocation As String, MenuName As String, ItemName As String)
The EA_GetMenuClick function syntax has the following elements:
Parameter
Type
Direction
Description
ItemName
String
The name of the option actually clicked, e.g. Create a New Invoice.
MenuName
String
The name of the parent menu for which sub-items are to be defined. In the case of the top-level menu it is an empty string

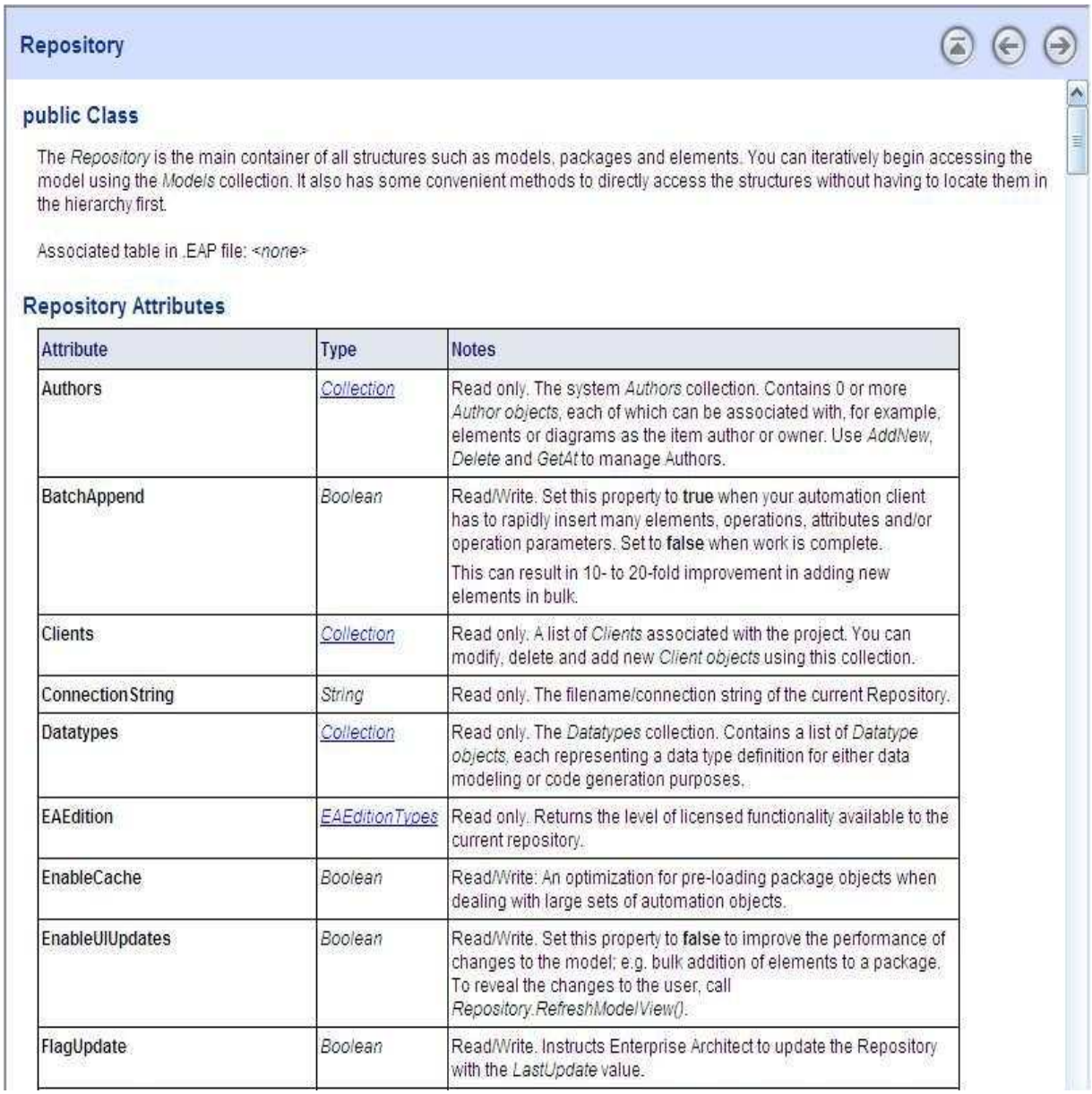

Enterprise Architect User Guide
Ocultar
Localizar
Voltar
Avançar
Parar
Página inicial
Imprimir
Opções
Conteúdo
Índice
Pesquisar
Favoritos
Data Modeling
MDA Transformations
Enterprise Architect Reports
UML Dictionary
License Management
Finding Your License Information
Add License Key
Keystore Troubleshooting
Upgrade an Existing License
Register Add-In
SDK for Enterprise Architect
Developing Profiles
MDG Technologies in SDK
Shape Scripts
Tagged Value Types
Code Template Framework in SDK
Enterprise Architect Add-In Model
Add-In Tasks
Create Add-Ins
Define Menu Items
Deploy Add-Ins
Tricks and Traps
The Add-In Manager
Add-In Search
Add-In Events
EA_Connect
EA_Disconnect
EA_GetMenuItems
EA_GetMenuState
EA_MenuClick
EA_OnOutputItemClicked
EA_OnOutputItemDoubleClicked
EA_ShowHelp
Broadcast Events
Custom Views
MDG Add-Ins
Enterprise Architect Object Model
Using the Automation Interface
Connect to the Interface
Set References In Visual Basic
Examples and Tips
EA_GetMenuItems
The EA_GetMenuItems event enables the Add-In to provide the Enterprise Architect user interface with additional Add-In menu options in various context and main menus. When a user selects an Add-In menu option, an event is raised and passed back to the Add-In that originally defined that menu option.
This event is raised just before Enterprise Architect has to show particular menu options to the user, and its use is described in the Define Menu Items topic.
Also look at:
EA_MenuClick
EA_GetMenuState.
Syntax
Function EA_GetMenuItems(Repository As EA.Repository, MenuLocation As String, MenuName As String) As Variant
The EA_GetMenuItems function syntax has the following elements:
Parameter
Type
Direction
Description
MenuLocation
String
String representing the part of the user interface that brought up the menu. Can be TreeView, MainMenu or Diagram.
MenuName
String
The name of the parent menu for which sub-items are to be defined. In the case of the top-level menu it is an empty string.
Repository
EA.Repository
IN
An EA.Repository object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
Return Value


Enterprise Architect User Guide
Ocultar
Localizar
Voltar
Avançar
Parar
Página inicial
Imprimir
Opções
Conteúdo
Índice
Pesquisar
Favoritos
Enterprise Architect UML Tool
Start UML Modeling
UML Modeling Tool Features
UML Tool Project Roles
Modeling With UML
Extending UML
UML Model Management
User Security
Version Control
Baselines, Differencing and Merges
Auditing
Project Management
Code Engineering
Visual Execution Analyzer
XML Technologies
Data Modeling
MDA Transformations
Enterprise Architect Reports
UML Dictionary
License Management
SDK for Enterprise Architect
Developing Profiles
MDG Technologies in SDK
Shape Scripts
Tagged Value Types
Code Template Framework in SDK
Enterprise Architect Add-In Model
Enterprise Architect Object Model
Using the Automation Interface
Connect to the Interface
Set References In Visual Basic
Examples and Tips
Call from Enterprise Architect
Available Resources
Reference
Glossary of Terms
Reference
This section provides detailed information on all the objects available in the object model provided by the Automation Interface, covering:
Interface Overview
App
Enumerations
Repository
Element
Element Features
Connector
Diagram Package
Project Interface
Code Samples


Enterprise Architect User Guide
Ocultar
Localizar
Voltar
Avançar
Parar
Página inicial
Imprimir
Opções
Conteúdo
Índice
Pesquisar
Using the Automation Interfac
Connect to the Interface
Set References In Vi
Examples and Tips
Call from Enterprise A
Available Resources
Reference
Interface Overview
App
Enumerations
Repository
Element
Element Features
Connector
Diagram
Project Interface
Overview
public Package
This package provides an overview of the main elements within the Automation Interface. These are:
The Repository, which represents the model as a whole and provides entry to model packages and collections
Elements, which are the basic structural unit (e.g. Class, Use Case and Object)
Element Features, which are attributes and operations defined on an element
Diagram Package, the visible drawings contained in the model
Connectors, relationships between elements.
The following diagram illustrates the main interface elements and their associated contents. Each element in this document is creatable by Automation and can be accessed through the various collections that exist or, in some cases, directly.
Overview
Repository
+ Author
+ Client
+ Collection
+ Datatype
+ ProjectIssues
+ ProjectResource
+ Reference
+ Repository
+ Stereotype
+ Task
+ Term
Element
+ Constraint
+ Effort
+ File
+ Issue
+ Metric
+ Requirement
+ Resource
+ Risk
+ Scenario
+ TaggedValue
+ Test
+ Element
Element Features
+ Attribute
+ AttributeConstraint
+ AttributeTag
+ Method
+ MethodConstraint
+ MethodTag
+ Parameter
Diagram
+ Diagram
+ DiagramLinks
+ DiagramObject
Connector
+ ConnectorConstraint
+ ConnectorEnd
+ ConnectorTag
+ RoleTag
+ Connector
Project Interface
+ Project


Overview
Term
Author
Client
Task
Datatype
ProjectResource
ProjectIssues
+Terms
+Authors
+Clients
+Tasks
+Resources
+Issues
MethodTag
+TaggedValues
MethodConstraint
+PreConditions
+PostConditions
Repository
+Parameters
Parameter
AttributeConstraint
+Constraints
+Resources
Resource
AttributeTag
+TaggedValues
Method
+Scenarios
Scenario
Attribute
+Methods
+MethodsEx
+Attributes
+AttributesEx
+Requirements
Requirement
+RequirementsEx
+Models
+Packages
Package
+Elements
Element
+Issues
Issue
+Constraints
Constraint
+ConstraintsEx
+DiagramObjects
DiagramObject
+Diagrams
Diagram
+Files
File
+Tests
Test
+Connectors
Connector
+TaggedValues
TaggedValue
+TaggedValuesEx
+DiagramLinks
DiagramLinks
+Metrics
Metric
+TaggedValues
+Constraints
+ClientEnd
+SupplierEnd
+Risks
Risk
ConnectorTag
ConnectorConstraint
ConnectorEnd

REPOSITORY



Repository

public Class

The *Repository* is the main container of all structures such as models, packages and elements. You can iteratively begin accessing the model using the *Models* collection. It also has some convenient methods to directly access the structures without having to locate them in the hierarchy first.

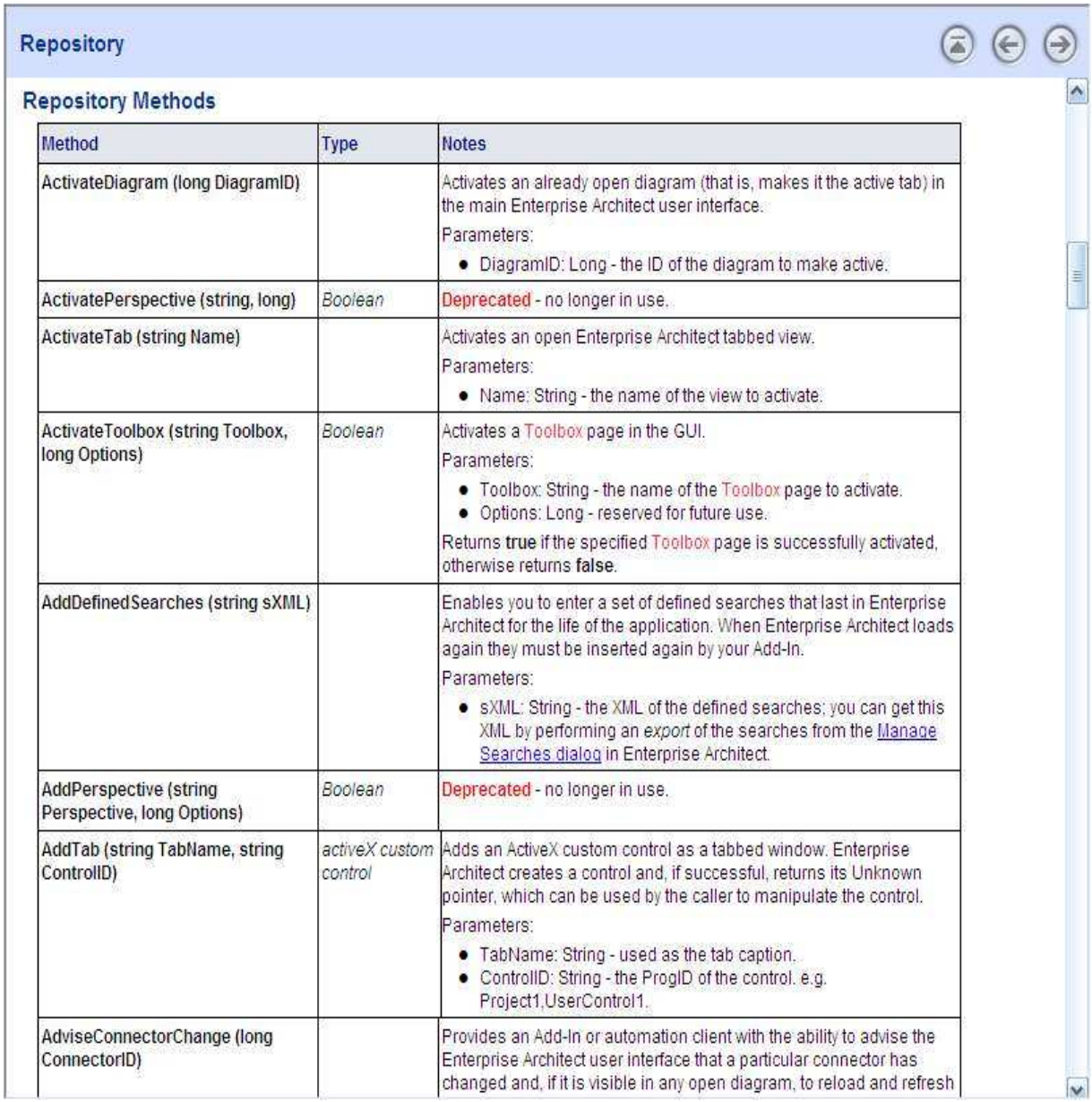
Associated table in .EAP file: *<none>*

Repository Attributes

Attribute	Type	Notes
Authors	*Collection*	Read only. The system *Authors* collection. Contains 0 or more *Author objects*, each of which can be associated with, for example, elements or diagrams as the item author or owner. Use *AddNew*, *Delete* and *GetAt* to manage Authors.
BatchAppend	*Boolean*	Read/Write. Set this property to **true** when your automation client has to rapidly insert many elements, operations, attributes and/or operation parameters. Set to **false** when work is complete. This can result in 10- to 20-fold improvement in adding new elements in bulk.
Clients	*Collection*	Read only. A list of *Clients* associated with the project. You can modify, delete and add new *Client objects* using this collection.
ConnectionString	*String*	Read only. The filename/connection string of the current Repository.
Datatypes	*Collection*	Read only. The *Datatypes* collection. Contains a list of *Datatype objects*, each representing a data type definition for either data modeling or code generation purposes.
EAEdition	*EAEditionTypes*	Read only. Returns the level of licensed functionality available to the current repository.
EnableCache	*Boolean*	Read/Write: An optimization for pre-loading package objects when dealing with large sets of automation objects.
EnableUIUpdates	*Boolean*	Read/Write. Set this property to **false** to improve the performance of changes to the model; e.g. bulk addition of elements to a package. To reveal the changes to the user, call *Repository.RefreshModelView()*.
FlagUpdate	*Boolean*	Read/Write. Instructs Enterprise Architect to update the Repository with the *LastUpdate* value.

Repository

InstanceGUID	*String*	Read only: The identifier string identifying the Enterprise Architect runtime session.
Issues	*Collection*	Read only. The *System Issues* list. Contains *ProjectIssues objects*, each detailing a particular issue as it relates to the project as a whole.
LastUpdate	*String*	Read only. The identifier string identifying the Enterprise Architect runtime session and the timestamp for when it was set.
LibraryVersion	*Long*	Read only. The build number of the Enterprise Architect runtime.
Models	*Collection of type Package*	Read only. *Models* are of type *package* and belong to a collection of packages. This is the top level entry point to an Enterprise Architect project file. Each model is a *root node* in the Project Browser and can contain items such as Views and packages. A model is a special form of a package; it has a *ParentID* of **0**. By iterating through all models, you can access all the elements within the project hierarchy. You can also use the *AddNew* function to create a new model. A model can be deleted, but remember that everything contained in the model is deleted as well.
ObjectType	*ObjectType*	Read only. Distinguishes objects referenced through the Dispatch interface.
ProjectGUID	*String*	Read only: Returns a unique ID for the project.
PropertyTypes	*Collection*	Read only. Collection of *Property Types* available to the Repository.
Resources	*Collection*	Read only. Contains available *ProjectResource* objects to assign to work items within the project. Use the **add new**, **modify** and **delete** functions to manage resources.
Stereotypes	*Collection*	Read only. The *Stereotypes* collection. A list of *Stereotype objects* that contain information on a stereotype and which elements it can be applied to.
SuppressEADialogs	*Boolean*	Read/Write. Set this property in the *EA_OnPostNewElement* or *EA_OnPostNewConnector* broadcast events to control whether Enterprise Architect should suppress showing the default Properties dialogs to the user when an element or connector is created.
Tasks	*Collection*	Read only. A list of system tasks (to do list). Each entry is a *Task Item*; you can modify, delete and add new tasks.
Terms	*Collection*	Read only. The project *Glossary*. Each *Term* object is an entry in the Glossary. Add, modify and delete Terms to maintain the Glossary.



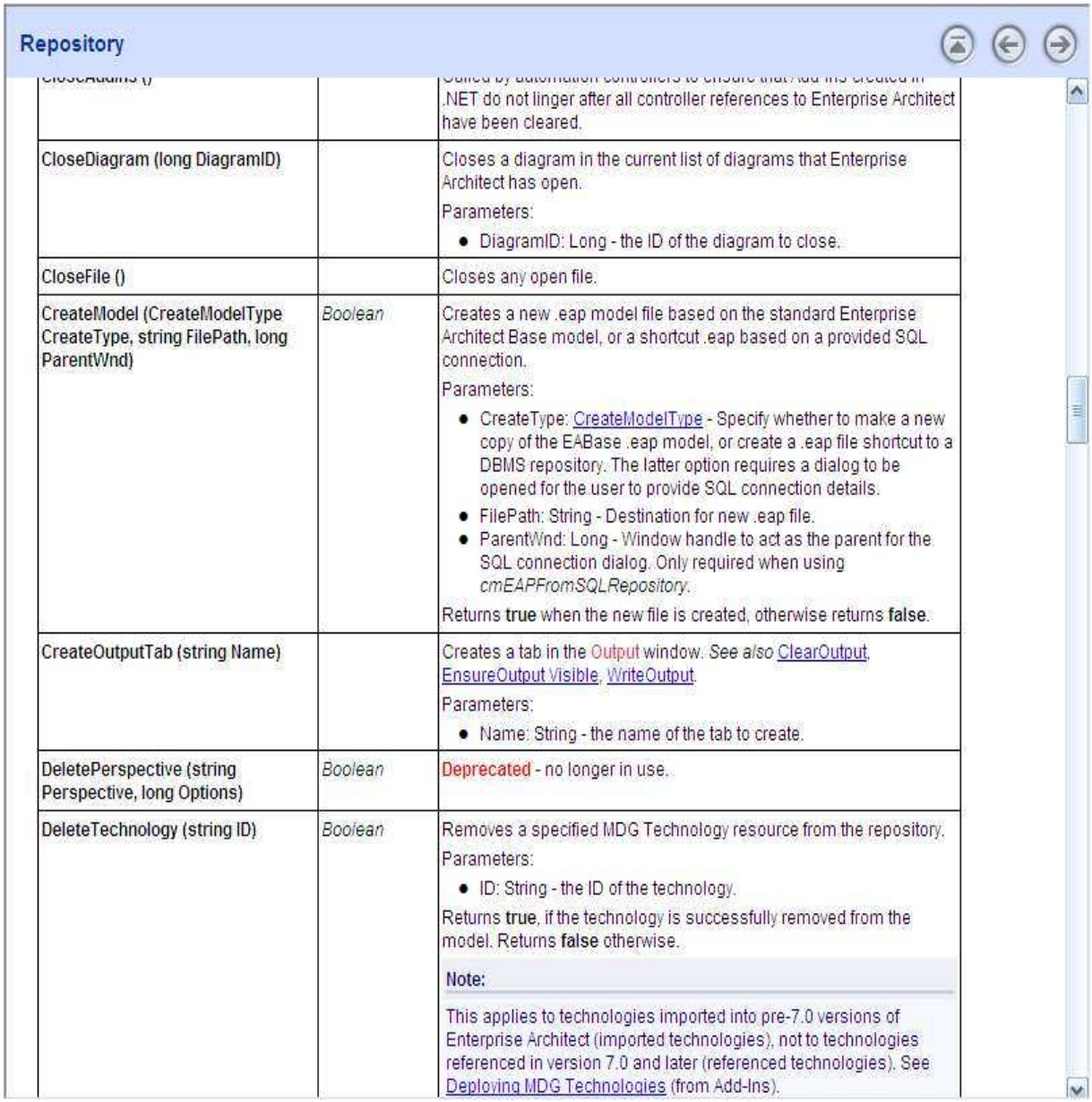
Repository

Repository Methods

Method	Type	Notes
ActivateDiagram (long DiagramID)		Activates an already open diagram (that is, makes it the active tab) in the main Enterprise Architect user interface. Parameters: • DiagramID: Long - the ID of the diagram to make active.
ActivatePerspective (string, long)	*Boolean*	Deprecated - no longer in use.
ActivateTab (string Name)		Activates an open Enterprise Architect tabbed view. Parameters: • Name: String - the name of the view to activate.
ActivateToolbox (string Toolbox, long Options)	*Boolean*	Activates a Toolbox page in the GUI. Parameters: • Toolbox: String - the name of the Toolbox page to activate. • Options: Long - reserved for future use. Returns **true** if the specified Toolbox page is successfully activated, otherwise returns **false**.
AddDefinedSearches (string sXML)		Enables you to enter a set of defined searches that last in Enterprise Architect for the life of the application. When Enterprise Architect loads again they must be inserted again by your Add-In. Parameters: • sXML: String - the XML of the defined searches; you can get this XML by performing an *export* of the searches from the Manage Searches dialog in Enterprise Architect.
AddPerspective (string Perspective, long Options)	*Boolean*	Deprecated - no longer in use.
AddTab (string TabName, string ControlID)	*activeX custom control*	Adds an ActiveX custom control as a tabbed window. Enterprise Architect creates a control and, if successful, returns its Unknown pointer, which can be used by the caller to manipulate the control. Parameters: • TabName: String - used as the tab caption. • ControlID: String - the ProgID of the control. e.g. Project1,UserControl1.
AdviseConnectorChange (long ConnectorID)		Provides an Add-In or automation client with the ability to advise the Enterprise Architect user interface that a particular connector has changed and, if it is visible in any open diagram, to reload and refresh

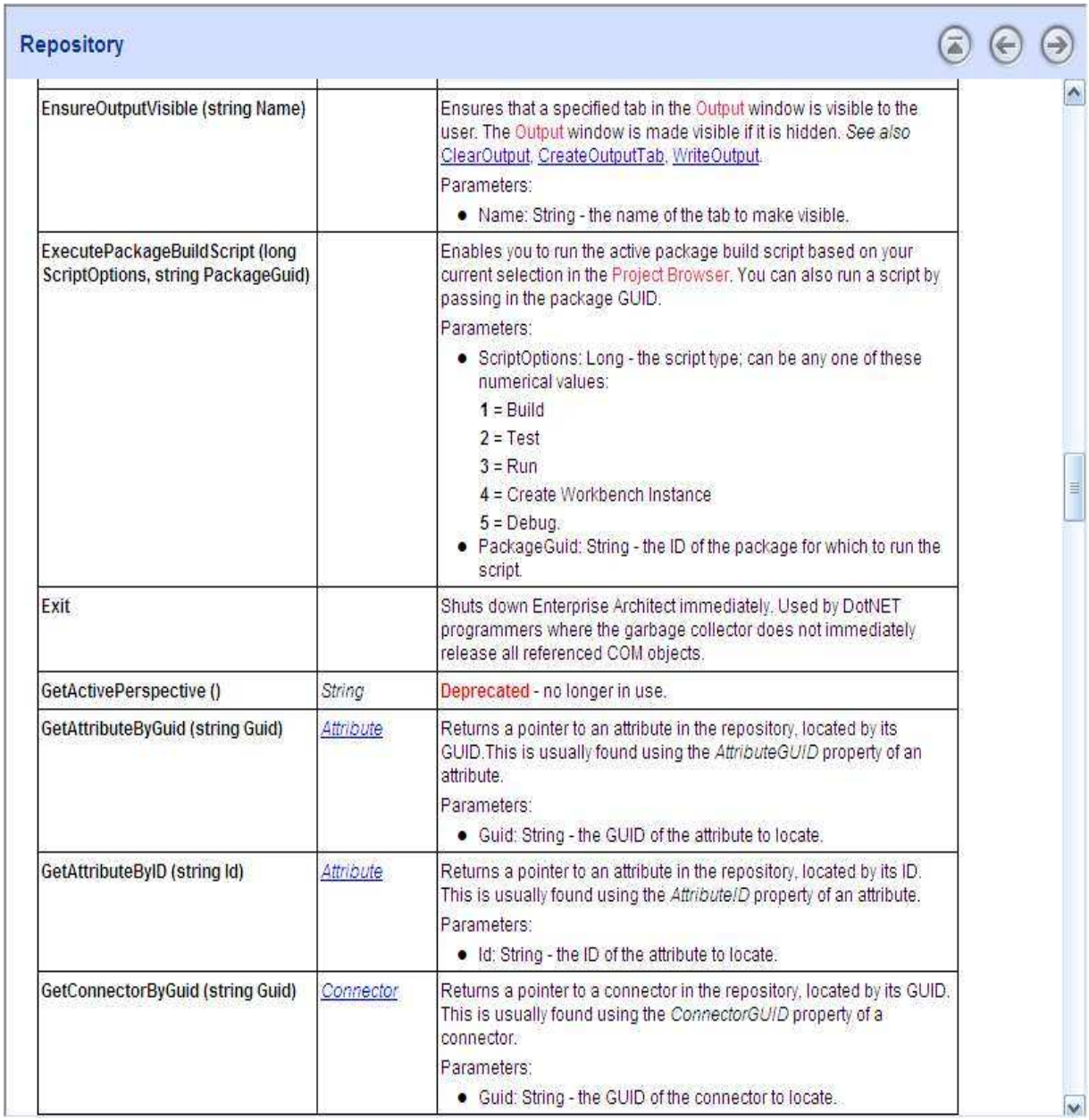
Repository

		that connector for the user. Parameters: • ConnectorID: Long - the ID of the connector.
AdviseElementChange (long ObjectID)		Provides an Add-In or automation client with the ability to advise the Enterprise Architect user interface that a particular element has changed and, if it is visible in any open diagram, to reload and refresh that element for the user. Parameters: • ObjectID: Long - the ID of the element.
ChangeLoginUser (string Name, string Password)	*Boolean*	Sets the currently logged on user to be that specified by a name and password. This logs the user into the repository when security is enabled. If security is not enabled an exception (*Security not enabled*) is thrown. Parameters: • Name: String - the name of the user. • Password: String - the password of the user.
ClearAuditLogs (Object StartDateTime, Object EndDateTime)	*Boolean*	Clears all Audit Logs from the model. Parameters: • StartDateTime: Variant [DateTime] - the earliest date and time of log entries to clear. • EndDateTime: Variant [DateTime] - the latest date and time of log entries to clear. If *StartDateTime* and *EndDateTime* are not null then only log items that fall into this period are cleared. Returns **true** for success, **false** for failure. **Notes:** • This method cannot be undone. It is strongly advised that you call *SaveAuditLogs* first to backup the logs. • This method might fail if the user logged into the model does not have the correct access permission.
ClearOutput (string Name)		Removes all the text from a tab in the Output window. *See also* CreateOutputTab, EnsureOutput Visible, WriteOutput. Parameters: • Name: String - the name of the tab to remove text from.
CloseAddins ()		Called by automation controllers to ensure that Add-Ins created in .NET do not linger after all controller references to Enterprise Architect



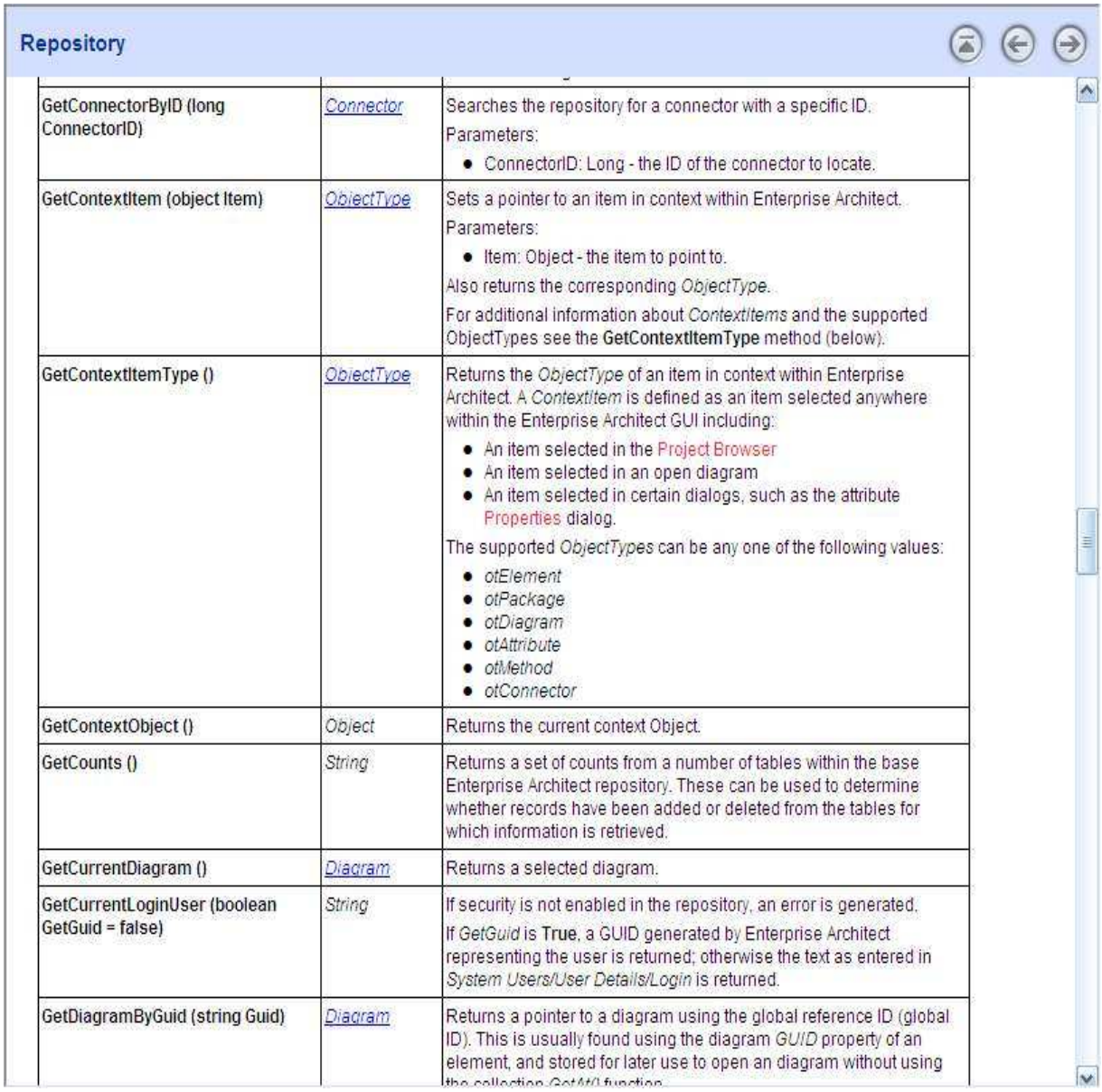
Repository

		.NET do not linger after all controller references to Enterprise Architect have been cleared.
CloseDiagram (long DiagramID)		Closes a diagram in the current list of diagrams that Enterprise Architect has open. Parameters: • DiagramID: Long - the ID of the diagram to close.
CloseFile ()		Closes any open file.
CreateModel (CreateModelType CreateType, string FilePath, long ParentWnd)	*Boolean*	Creates a new .eap model file based on the standard Enterprise Architect Base model, or a shortcut .eap based on a provided SQL connection. Parameters: • CreateType: CreateModelType - Specify whether to make a new copy of the EABase .eap model, or create a .eap file shortcut to a DBMS repository. The latter option requires a dialog to be opened for the user to provide SQL connection details. • FilePath: String - Destination for new .eap file. • ParentWnd: Long - Window handle to act as the parent for the SQL connection dialog. Only required when using *cmEAPFromSQLRepository*. Returns **true** when the new file is created, otherwise returns **false**.
CreateOutputTab (string Name)		Creates a tab in the Output window. *See also* ClearOutput, EnsureOutput Visible, WriteOutput. Parameters: • Name: String - the name of the tab to create.
DeletePerspective (string Perspective, long Options)	*Boolean*	Deprecated - no longer in use.
DeleteTechnology (string ID)	*Boolean*	Removes a specified MDG Technology resource from the repository. Parameters: • ID: String - the ID of the technology. Returns **true**, if the technology is successfully removed from the model. Returns **false** otherwise. **Note:** This applies to technologies imported into pre-7.0 versions of Enterprise Architect (imported technologies), not to technologies referenced in version 7.0 and later (referenced technologies). See Deploying MDG Technologies (from Add-Ins).



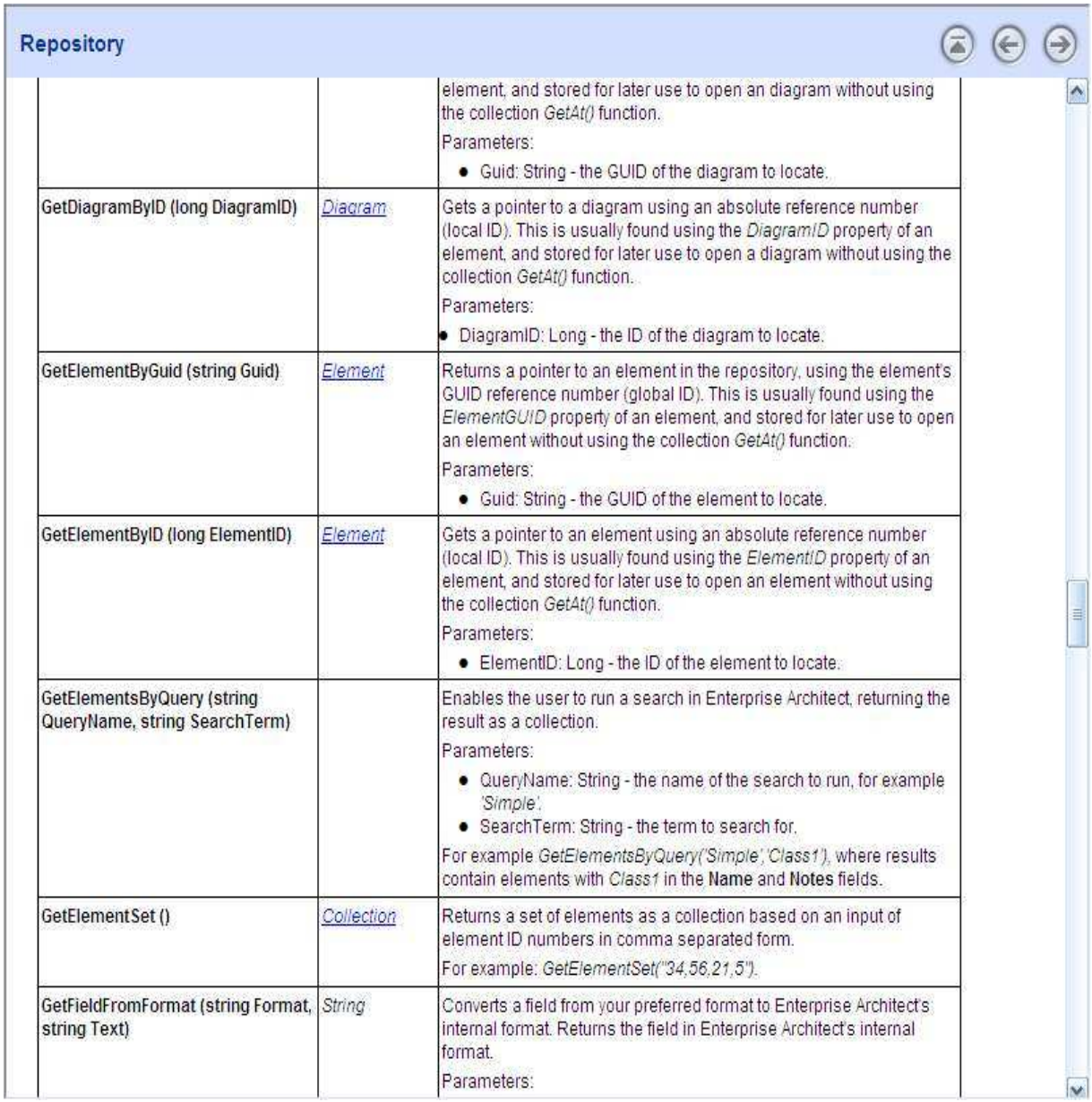
Repository

EnsureOutputVisible (string Name)		Ensures that a specified tab in the Output window is visible to the user. The Output window is made visible if it is hidden. *See also* ClearOutput, CreateOutputTab, WriteOutput. Parameters: • Name: String - the name of the tab to make visible.
ExecutePackageBuildScript (long ScriptOptions, string PackageGuid)		Enables you to run the active package build script based on your current selection in the Project Browser. You can also run a script by passing in the package GUID. Parameters: • ScriptOptions: Long - the script type; can be any one of these numerical values: 1 = Build 2 = Test 3 = Run 4 = Create Workbench Instance 5 = Debug. • PackageGuid: String - the ID of the package for which to run the script.
Exit		Shuts down Enterprise Architect immediately. Used by DotNET programmers where the garbage collector does not immediately release all referenced COM objects.
GetActivePerspective ()	*String*	Deprecated - no longer in use.
GetAttributeByGuid (string Guid)	*Attribute*	Returns a pointer to an attribute in the repository, located by its GUID.This is usually found using the *AttributeGUID* property of an attribute. Parameters: • Guid: String - the GUID of the attribute to locate.
GetAttributeByID (string Id)	*Attribute*	Returns a pointer to an attribute in the repository, located by its ID. This is usually found using the *AttributeID* property of an attribute. Parameters: • Id: String - the ID of the attribute to locate.
GetConnectorByGuid (string Guid)	*Connector*	Returns a pointer to a connector in the repository, located by its GUID. This is usually found using the *ConnectorGUID* property of a connector. Parameters: • Guid: String - the GUID of the connector to locate.



Repository

GetConnectorByID (long ConnectorID)	*Connector*	Searches the repository for a connector with a specific ID. Parameters: • ConnectorID: Long - the ID of the connector to locate.
GetContextItem (object Item)	*ObjectType*	Sets a pointer to an item in context within Enterprise Architect. Parameters: • Item: Object - the item to point to. Also returns the corresponding *ObjectType*. For additional information about *ContextItems* and the supported ObjectTypes see the **GetContextItemType** method (below).
GetContextItemType ()	*ObjectType*	Returns the *ObjectType* of an item in context within Enterprise Architect. A *ContextItem* is defined as an item selected anywhere within the Enterprise Architect GUI including: • An item selected in the Project Browser • An item selected in an open diagram • An item selected in certain dialogs, such as the attribute Properties dialog. The supported *ObjectTypes* can be any one of the following values: • *otElement* • *otPackage* • *otDiagram* • *otAttribute* • *otMethod* • *otConnector*
GetContextObject ()	*Object*	Returns the current context Object.
GetCounts ()	*String*	Returns a set of counts from a number of tables within the base Enterprise Architect repository. These can be used to determine whether records have been added or deleted from the tables for which information is retrieved.
GetCurrentDiagram ()	*Diagram*	Returns a selected diagram.
GetCurrentLoginUser (boolean GetGuid = false)	*String*	If security is not enabled in the repository, an error is generated. If *GetGuid* is **True**, a GUID generated by Enterprise Architect representing the user is returned; otherwise the text as entered in *System Users/User Details/Login* is returned.
GetDiagramByGuid (string Guid)	*Diagram*	Returns a pointer to a diagram using the global reference ID (global ID). This is usually found using the diagram *GUID* property of an element, and stored for later use to open an diagram without using the collection *GetAt()* function.



Repository

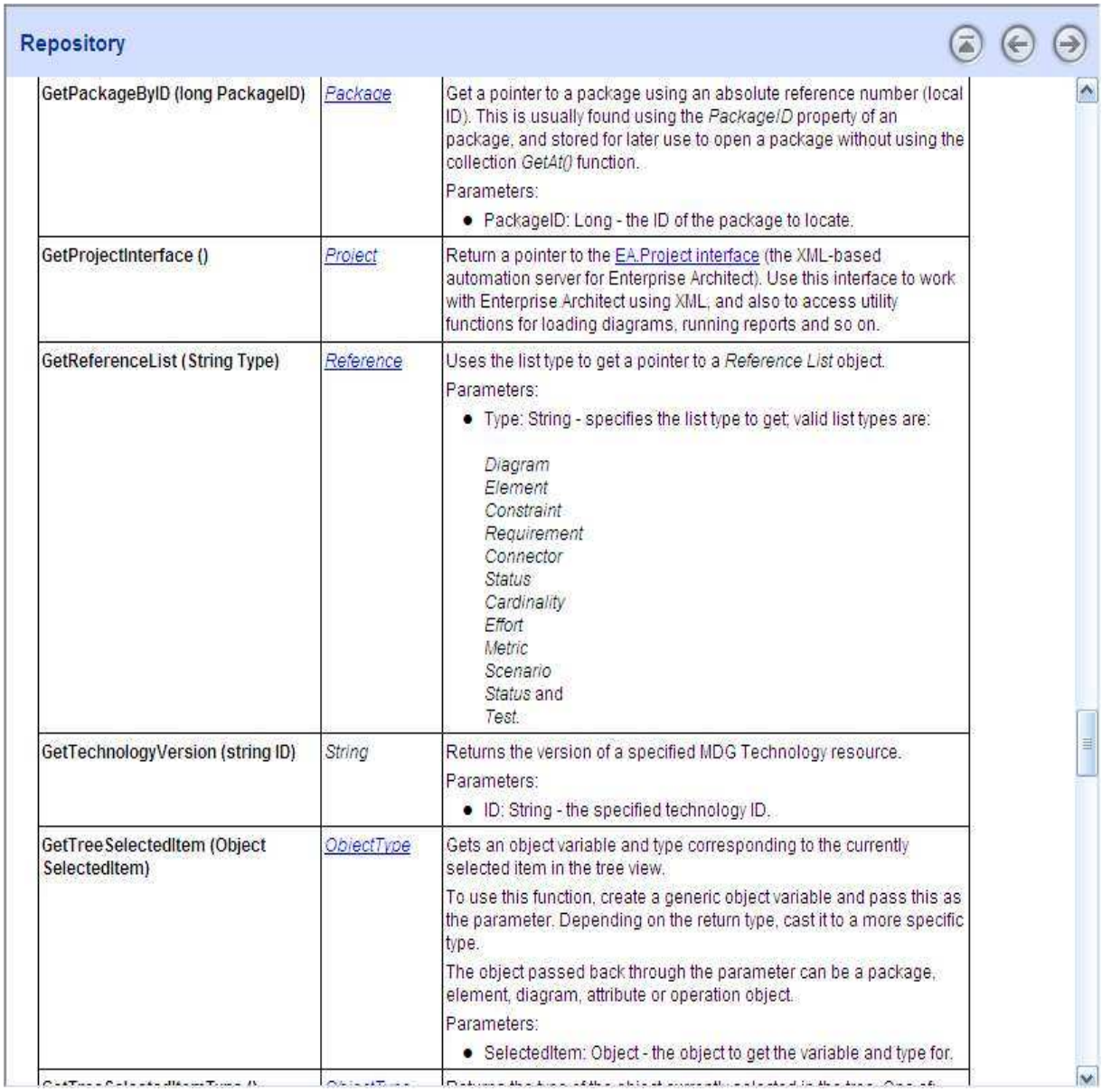
		element, and stored for later use to open an diagram without using the collection *GetAt()* function. Parameters: • Guid: String - the GUID of the diagram to locate.
GetDiagramByID (long DiagramID)	*Diagram*	Gets a pointer to a diagram using an absolute reference number (local ID). This is usually found using the *DiagramID* property of an element, and stored for later use to open a diagram without using the collection *GetAt()* function. Parameters: • DiagramID: Long - the ID of the diagram to locate.
GetElementByGuid (string Guid)	*Element*	Returns a pointer to an element in the repository, using the element's GUID reference number (global ID). This is usually found using the *ElementGUID* property of an element, and stored for later use to open an element without using the collection *GetAt()* function. Parameters: • Guid: String - the GUID of the element to locate.
GetElementByID (long ElementID)	*Element*	Gets a pointer to an element using an absolute reference number (local ID). This is usually found using the *ElementID* property of an element, and stored for later use to open an element without using the collection *GetAt()* function. Parameters: • ElementID: Long - the ID of the element to locate.
GetElementsByQuery (string QueryName, string SearchTerm)		Enables the user to run a search in Enterprise Architect, returning the result as a collection. Parameters: • QueryName: String - the name of the search to run, for example *'Simple'*. • SearchTerm: String - the term to search for. For example *GetElementsByQuery('Simple','Class1')*, where results contain elements with *Class1* in the **Name** and **Notes** fields.
GetElementSet ()	*Collection*	Returns a set of elements as a collection based on an input of element ID numbers in comma separated form. For example: *GetElementSet("34,56,21,5")*.
GetFieldFromFormat (string Format, string Text)	*String*	Converts a field from your preferred format to Enterprise Architect's internal format. Returns the field in Enterprise Architect's internal format. Parameters:

Repository

		Parameters:
		• Format: String - The format to convert the field from. Valid formats are:
		  • HTML - Full HTML
		  • RTF - Rich Text Format
		  • TXT - Plain text
		• Text: String - The field to be converted.

GetFormatFromField (string Format, string Text)	*String*	After accessing a field that contains formatting, use this method to convert it to your preferred format. Returns the field in the format specified.
		Parameters:
		• Format: String - The format to convert the field to. Valid formats are:
		  • HTML - Full HTML
		  • RTF - Rich Text Format
		  • TXT - Plain text
		• Text: String - The field to be converted.

GetLastError ()	*String*	Returns a string value describing the most recent error that occurred in relation to this object. This function is rarely used as an exception is thrown when an error occurs.
GetMethodByGuid (string Guid)	*Method*	Returns a pointer to a method in the repository. This is usually found using the *MethodGUID* property of a method. Parameters: • Guid: String - the GUID of the method to look for.
GetMethodById (string Id)	*Method*	Returns a pointer to a method in the repository. This is usually found using the *MethodID* property of a method. Parameters: • Id: String - the ID of the method to look for.
GetPackageByGuid (string Guid)	*Package*	Returns a pointer to a package in the repository using the package's GUID reference number (global ID). This is usually found using the *PackageGUID* property of the package. Parameters: • Guid: String - the GUID of the package to look for. Each package in the model also has an associated element with the same GUID, so if you have an element with *Type="Package"* then you can load the package by calling:

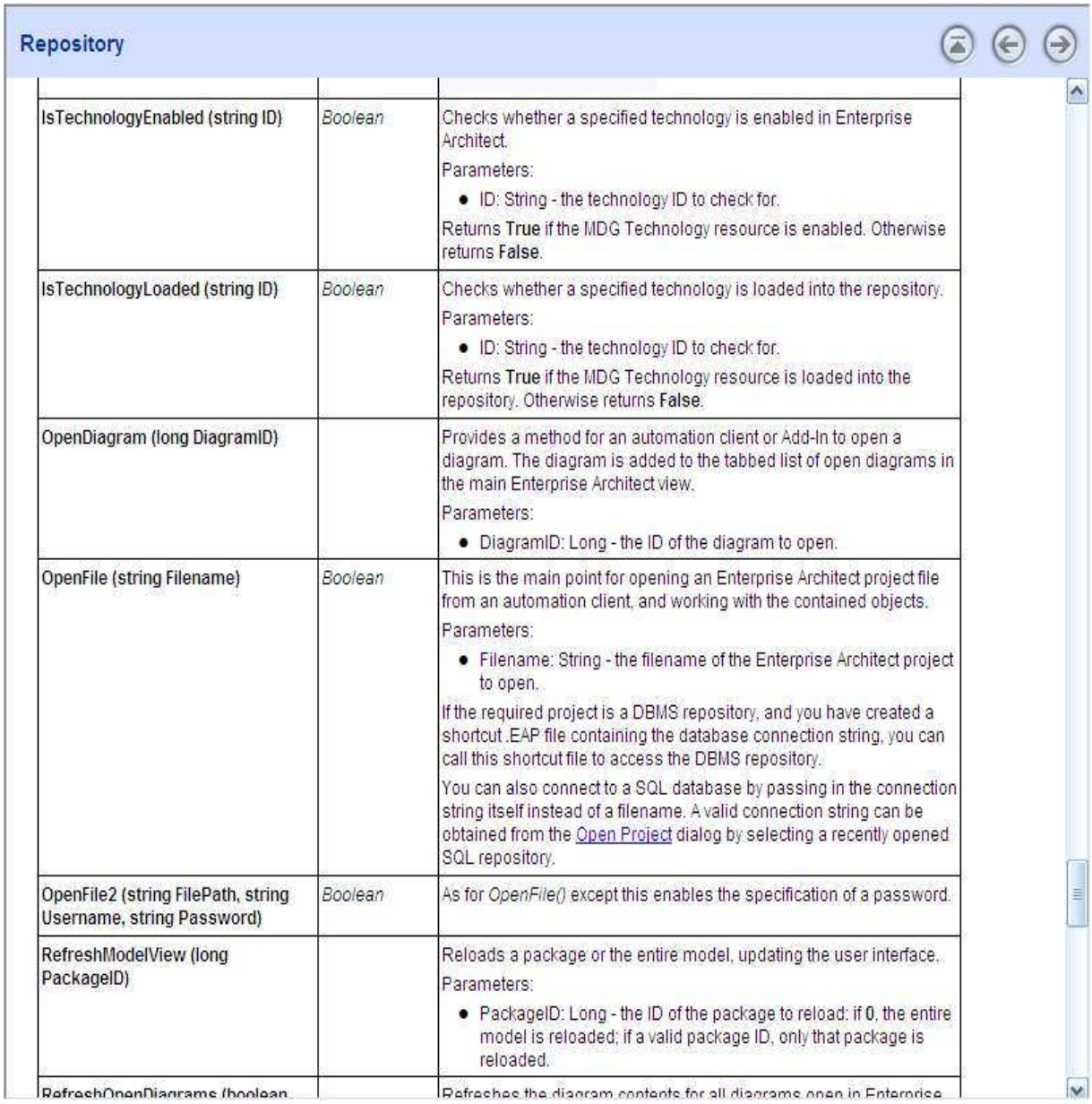


Repository

GetPackageByID (long PackageID)	*Package*	Get a pointer to a package using an absolute reference number (local ID). This is usually found using the *PackageID* property of an package, and stored for later use to open a package without using the collection *GetAt()* function. Parameters: • PackageID: Long - the ID of the package to locate.
GetProjectInterface ()	*Project*	Return a pointer to the EA.Project interface (the XML-based automation server for Enterprise Architect). Use this interface to work with Enterprise Architect using XML, and also to access utility functions for loading diagrams, running reports and so on.
GetReferenceList (String Type)	*Reference*	Uses the list type to get a pointer to a *Reference List* object. Parameters: • Type: String - specifies the list type to get; valid list types are: *Diagram* *Element* *Constraint* *Requirement* *Connector* *Status* *Cardinality* *Effort* *Metric* *Scenario* *Status* and *Test*.
GetTechnologyVersion (string ID)	*String*	Returns the version of a specified MDG Technology resource. Parameters: • ID: String - the specified technology ID.
GetTreeSelectedItem (Object SelectedItem)	*ObjectType*	Gets an object variable and type corresponding to the currently selected item in the tree view. To use this function, create a generic object variable and pass this as the parameter. Depending on the return type, cast it to a more specific type. The object passed back through the parameter can be a package, element, diagram, attribute or operation object. Parameters: • SelectedItem: Object - the object to get the variable and type for.

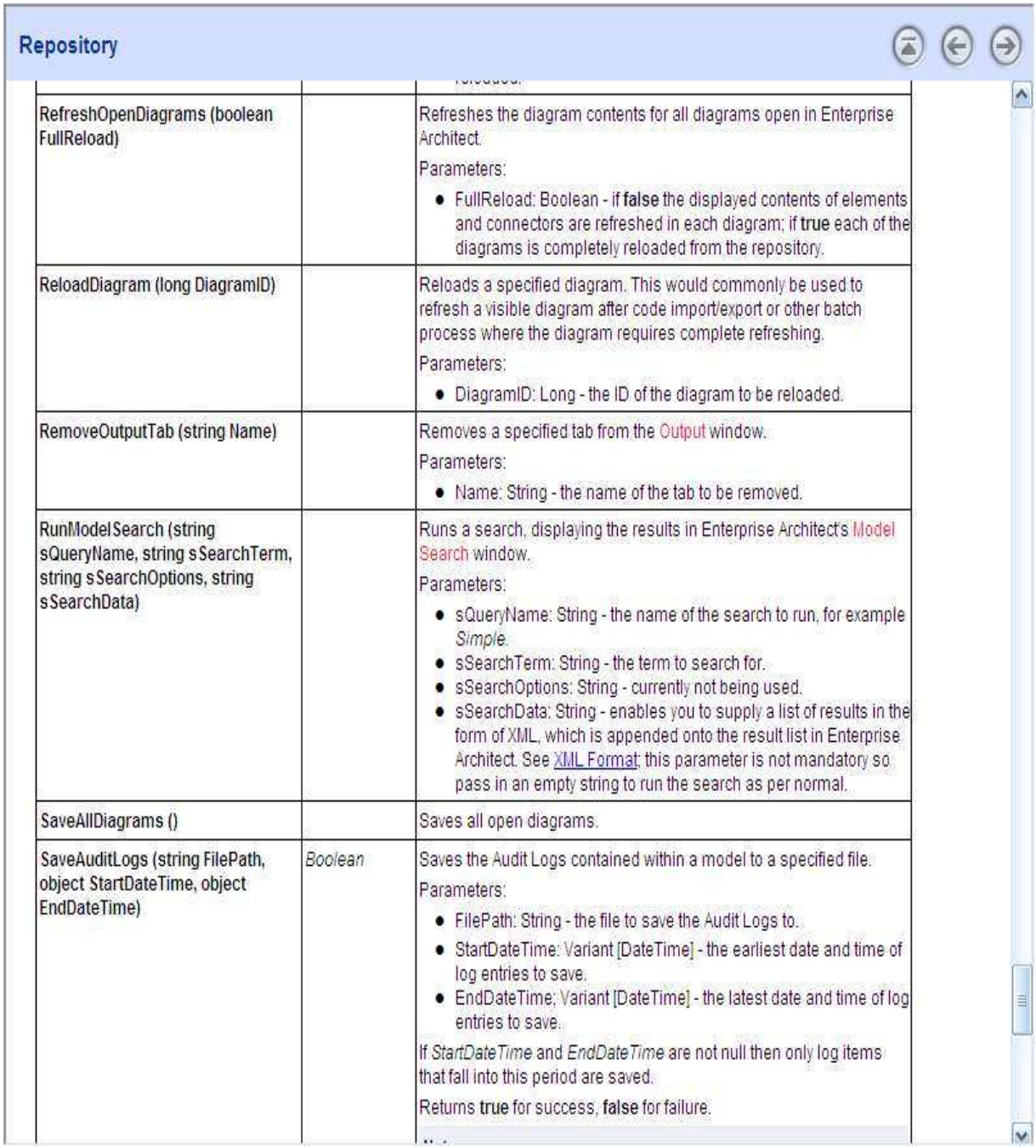
Repository

GetTreeSelectedItemType ()	*ObjectType*	Returns the type of the object currently selected in the tree. One of: • *otDiagram* • *otElement* • *otPackage* • *otAttribute* • *otMethod*.
GetTreeSelectedObject ()	*Object*	The related method *GetTreeSelectedItem()* has an output parameter that is inaccessible by some scripting languages. As an alternative, this method provides the selected item through the return value.
GetTreeSelectedPackage ()	*Package*	Returns the package in which the currently selected tree view object is contained.
HasPerspective (string Perspective)	*String*	Deprecated - no longer in use.
ImportPackageBuildScripts (string PackageGuid, string BuildScriptXML)		Imports build scripts into a package in Enterprise Architect. Parameters: • PackageGuid: String - the GUID of the package into which to import the build scripts. • BuildScriptXML: String - the build script XML data, which you can export from within Enterprise Architect.
ImportTechnology (string Technology)	*Boolean*	Installs a given MDG Technology resource into the repository. Parameters: • Technology: String - the contents of the technology resource file. Returns **True**, if the technology is successfully loaded into the model. Otherwise returns **False**. **Note:** This applies to technologies imported into pre-7.0 versions of Enterprise Architect (imported technologies), not to technologies referenced in version 7.0 and later (referenced technologies). See Deploying MDG Technologies (from Add-Ins).
IsTabOpen ()	*String*	Checks whether a named tab is open and active. Parameters: • TabName: String - the name of the tab to check for. Returns: • **2** to indicate that a tab is open and active (top-most) • **1** to indicate that it is open but not top-most, or • **0** to indicate that it is not visible at all.



Repository

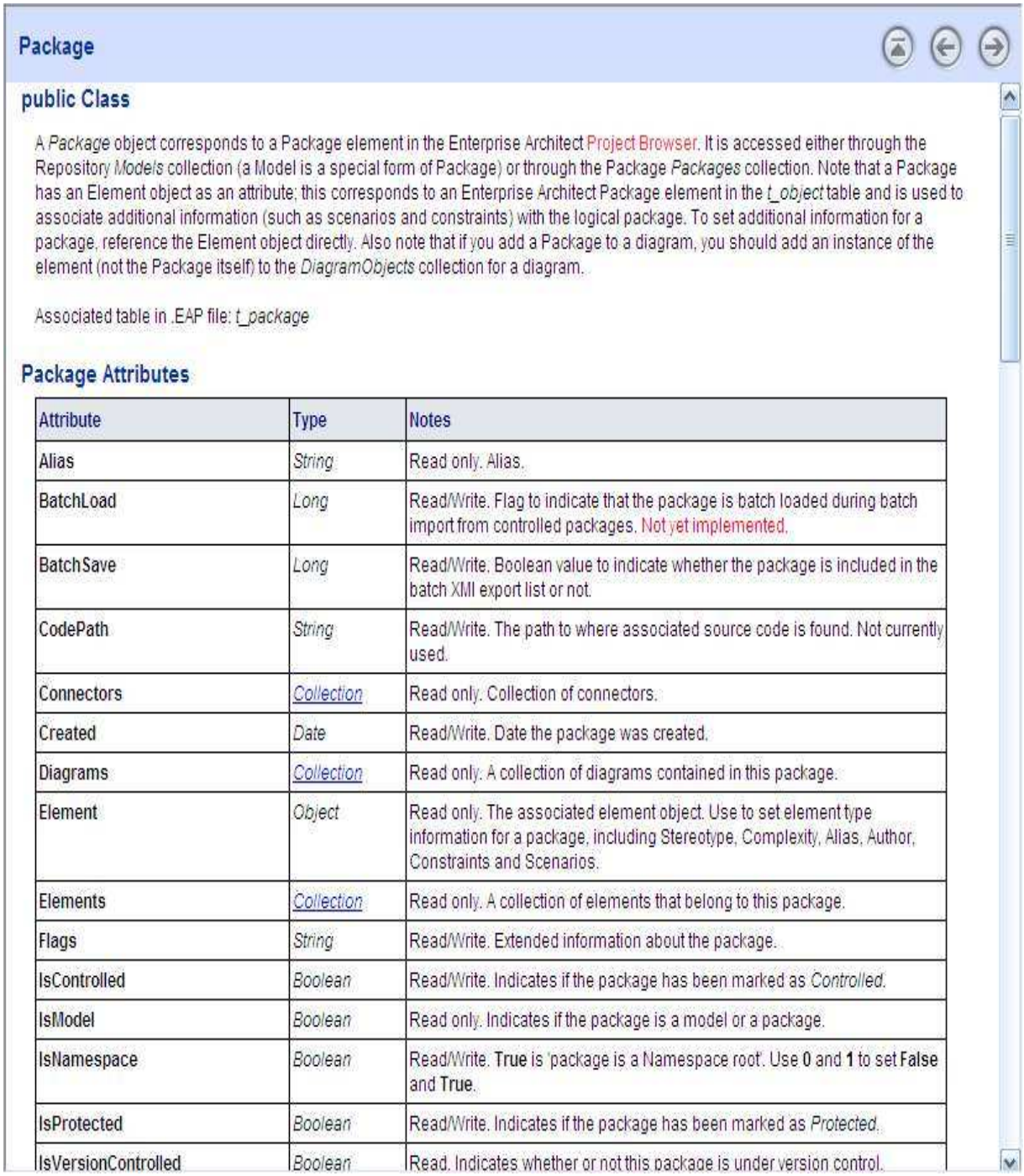
IsTechnologyEnabled (string ID)	*Boolean*	Checks whether a specified technology is enabled in Enterprise Architect. Parameters: • ID: String - the technology ID to check for. Returns **True** if the MDG Technology resource is enabled. Otherwise returns **False**.
IsTechnologyLoaded (string ID)	*Boolean*	Checks whether a specified technology is loaded into the repository. Parameters: • ID: String - the technology ID to check for. Returns **True** if the MDG Technology resource is loaded into the repository. Otherwise returns **False**.
OpenDiagram (long DiagramID)		Provides a method for an automation client or Add-In to open a diagram. The diagram is added to the tabbed list of open diagrams in the main Enterprise Architect view. Parameters: • DiagramID: Long - the ID of the diagram to open.
OpenFile (string Filename)	*Boolean*	This is the main point for opening an Enterprise Architect project file from an automation client, and working with the contained objects. Parameters: • Filename: String - the filename of the Enterprise Architect project to open. If the required project is a DBMS repository, and you have created a shortcut .EAP file containing the database connection string, you can call this shortcut file to access the DBMS repository. You can also connect to a SQL database by passing in the connection string itself instead of a filename. A valid connection string can be obtained from the Open Project dialog by selecting a recently opened SQL repository.
OpenFile2 (string FilePath, string Username, string Password)	*Boolean*	As for *OpenFile()* except this enables the specification of a password.
RefreshModelView (long PackageID)		Reloads a package or the entire model, updating the user interface. Parameters: • PackageID: Long - the ID of the package to reload: if 0, the entire model is reloaded; if a valid package ID, only that package is reloaded.
RefreshOpenDiagrams (boolean		Refreshes the diagram contents for all diagrams open in Enterprise



Repository

RefreshOpenDiagrams (boolean FullReload)		Refreshes the diagram contents for all diagrams open in Enterprise Architect. Parameters: • FullReload: Boolean - if **false** the displayed contents of elements and connectors are refreshed in each diagram; if **true** each of the diagrams is completely reloaded from the repository.
ReloadDiagram (long DiagramID)		Reloads a specified diagram. This would commonly be used to refresh a visible diagram after code import/export or other batch process where the diagram requires complete refreshing. Parameters: • DiagramID: Long - the ID of the diagram to be reloaded.
RemoveOutputTab (string Name)		Removes a specified tab from the Output window. Parameters: • Name: String - the name of the tab to be removed.
RunModelSearch (string sQueryName, string sSearchTerm, string sSearchOptions, string sSearchData)		Runs a search, displaying the results in Enterprise Architect's Model Search window. Parameters: • sQueryName: String - the name of the search to run, for example *Simple*. • sSearchTerm: String - the term to search for. • sSearchOptions: String - currently not being used. • sSearchData: String - enables you to supply a list of results in the form of XML, which is appended onto the result list in Enterprise Architect. See XML Format; this parameter is not mandatory so pass in an empty string to run the search as per normal.
SaveAllDiagrams ()		Saves all open diagrams.
SaveAuditLogs (string FilePath, object StartDateTime, object EndDateTime)	*Boolean*	Saves the Audit Logs contained within a model to a specified file. Parameters: • FilePath: String - the file to save the Audit Logs to. • StartDateTime: Variant [DateTime] - the earliest date and time of log entries to save. • EndDateTime: Variant [DateTime] - the latest date and time of log entries to save. If *StartDateTime* and *EndDateTime* are not null then only log items that fall into this period are saved. Returns **true** for success, **false** for failure.

SaveDiagram (long DiagramID)		Saves an open diagram. Assumes the diagram is open in the main user interface Tab list. Parameters: • DiagramID: Long - the ID of the diagram to save.
ShowDynamicHelp (string Topic)		Shows a help topic as a view. Parameters: • Topic: String - specifies the help topic.
ShowInProjectView (Object Item)		Selects a specified object in the Project Browser. Parameters: • Item: Object - the object to highlight. Accepted object types are *Package*, *Element*, *Diagram*, *Attribute*, and *Method*. An exception is thrown if the object is of an invalid type.
ShowProfileToolbox (string Technology, string Profile, boolean Show)		Shows/hides the contents of a specified technology or profile in the Enterprise Architect UML Toolbox. Parameters: • Technology: String - the ID of the technology. • Profile: String - the ID of the profile. • Show: Boolean - if **true**, show the technology or profile; if **false**, hide the technology or profile. To show/hide a profile in the Toolbox, specify the profile's ID value in the *Profile* parameter and set the *Technology* parameter to a null string. To show/hide a technology in the Toolbox, specify the technology's ID in the *Technology* parameter and set the *Profile* parameter to a null string.
ShowWindow (long Show)		Shows or hides Enterprise Architect. Parameters: • Show: Long.
SQLQuery (string SQL)	*String*	Enables execution of a SQL *select* statement against the current repository. Returns an XML formatted string value of the resulting recordset. Parameters: • SQL: String - contains the SQL Select statement.
WriteOutput (string Name, string String, long ID)		Writes text to a specified tab in the Output window, and associates the text with an ID. *See also* ClearOutput, CreateOutputTab, EnsureOutputVisible. Parameters: • Name: String - specifies the tab on which to display the text. • String: String - specifies the text to display. • ID: Long - specifies the ID the text is associated with.



Package

public Class

A *Package* object corresponds to a Package element in the Enterprise Architect Project Browser. It is accessed either through the Repository *Models* collection (a Model is a special form of Package) or through the Package *Packages* collection. Note that a Package has an Element object as an attribute; this corresponds to an Enterprise Architect Package element in the *t_object* table and is used to associate additional information (such as scenarios and constraints) with the logical package. To set additional information for a package, reference the Element object directly. Also note that if you add a Package to a diagram, you should add an instance of the element (not the Package itself) to the *DiagramObjects* collection for a diagram.

Associated table in .EAP file: *t_package*

Package Attributes

Attribute	Type	Notes
Alias	*String*	Read only. Alias.
BatchLoad	*Long*	Read/Write. Flag to indicate that the package is batch loaded during batch import from controlled packages. Not yet implemented.
BatchSave	*Long*	Read/Write. Boolean value to indicate whether the package is included in the batch XMI export list or not.
CodePath	*String*	Read/Write. The path to where associated source code is found. Not currently used.
Connectors	*Collection*	Read only. Collection of connectors.
Created	*Date*	Read/Write. Date the package was created.
Diagrams	*Collection*	Read only. A collection of diagrams contained in this package.
Element	*Object*	Read only. The associated element object. Use to set element type information for a package, including Stereotype, Complexity, Alias, Author, Constraints and Scenarios.
Elements	*Collection*	Read only. A collection of elements that belong to this package.
Flags	*String*	Read/Write. Extended information about the package.
IsControlled	*Boolean*	Read/Write. Indicates if the package has been marked as *Controlled*.
IsModel	*Boolean*	Read only. Indicates if the package is a model or a package.
IsNamespace	*Boolean*	Read/Write. **True** is 'package is a Namespace root'. Use **0** and **1** to set **False** and **True**.
IsProtected	*Boolean*	Read/Write. Indicates if the package has been marked as *Protected*.
IsVersionControlled	*Boolean*	Read. Indicates whether or not this package is under version control.

Package

LastLoadDate	*Date*	Read/Write. The date XML was last loaded for the package.
LastSaveDate	*Date*	Read/Write. The date XML was last saved from the package.
LogXML	*Boolean*	Read/Write. Indicates if XMI export information is to be logged.
Modified	*Date*	Read/Write. Date the package was last modified.
Name	*String*	Read/Write. The name of the package.
Notes	*String*	Read/Write. Notes about this package.
ObjectType	*ObjectType*	Read only. Distinguishes objects referenced through a Dispatch interface.
Owner	*String*	Read/Write. The package owner when using controlled packages.
PackageGUID	*Variant*	Read only. The global Package ID. Valid across models.
PackageID	*Long*	Read only. The local Package ID number. Valid only in this model file.
Packages	*Collection*	Read only. A collection of contained packages that can be walked through.
ParentID	*Long*	Read/Write. The ID of the package that is the parent of this one. **0** indicates this package is a *model* (that is, it has no parent).
TreePos	*Long*	Read/Write. The relative position in the tree compared to other packages (use to sort packages).
UMLVersion	*String*	Read/Write. The UML version for XMI export purposes.
UseDTD	*Boolean*	Read/Write. Indicates if a DTD is to be used when exporting XMI.
Version	*String*	Read/Write. The version of the package.
XMLPath	*String*	Read/Write. The path to where the XML is saved when using controlled packages.

Package

Package Methods

Method	Type	Notes
ApplyGroupLock (string aGroupName)	*Boolean*	Applies a group lock to the package object, for the specified group, on behalf of the current user. Throws an exception if the operation fails. Use *GetLastError()* to retrieve error information. Parameter: • aGroupName: String - The name of the security group for which to apply the lock.
ApplyUserLock ()	*Boolean*	Applies a user lock to the package object for the current user. Throws an exception if the operation fails. Use *GetLastError()* to retrieve error information.
Clone	*LDISPATCH*	Inserts a copy of the package into the same parent as the original package. Returns the newly-created package.
FindObject (string DottedID)	*LPDISPATCH*	Returns a package, element, attribute or operation matching the parameter *DottedID*. Parameter: • DottedID: String - Is in the form *object.object.object* where *object* is replaced by the name of a package, element attribute or operation. Examples include MyNamespace.Class1, CStudent.m_Name, MathClass.DoubleIt(int) If the DottedID is not found, an error is returned: *Can't find matching object.*
GetLastError ()	*String*	Returns a string value describing the most recent error that occurred in relation to this object. This function is rarely used as an exception is thrown when an error occurs.
ReleaseLock ()	*Boolean*	Removes an existing User or Group lock from the package object. Throws an exception if the operation fails. Use *GetLastError()* to retrieve error information.

Package

Update ()	*Boolean*	Update the current package object after modification or appending a new item. If **false** is returned, check the *GetLastError* function for more information. Note that a package object also has an *element* component that must be taken into account. The package object contains information about the package attributes such as hierarchy or contents. The element attribute contains information about, for example, Stereotype, Constraints or Files - all the attributes of a typical element.
VersionControlAdd (string ConfigGuid, string XMLFile, bool KeepCheckedOut)	*Void*	Places the package under version control, using the specified Version Control Configuration and the specified XMI filename. Throws an exception if the operation fails. Use *GetLastError( )* to retrieve error information. It is recommended that the package be saved using *Update()* before calling *VersionControlAdd()*, so that any outstanding changes are not lost. Parameters: • ConfigGuid: String - Name corresponding to the Unique ID of the version control configuration to use. • XMLFile: String - Name of the XML file to use for this package. This filename is relative to the Working Copy folder specified for the Config. • KeepCheckedOut: Boolean - Specify **True** to add to version control and keep package checked-out.
VersionControlCheckin (string Comment)	*Void*	Perform checkin of the version controlled package. Throws an exception if the operation fails. Use *GetLastError( )* to retrieve error information. Parameters: • Comment: String - Log message that is added to the version controlled file's history (where applicable).
VersionControlCheckout (string Comment)	*Void*	Perform checkout of the version controlled package. Throws an exception if the operation fails. Use *GetLastError( )* to retrieve error information. Parameters: • Comment: String - Log message that is added to the version controlled file's history (where applicable).

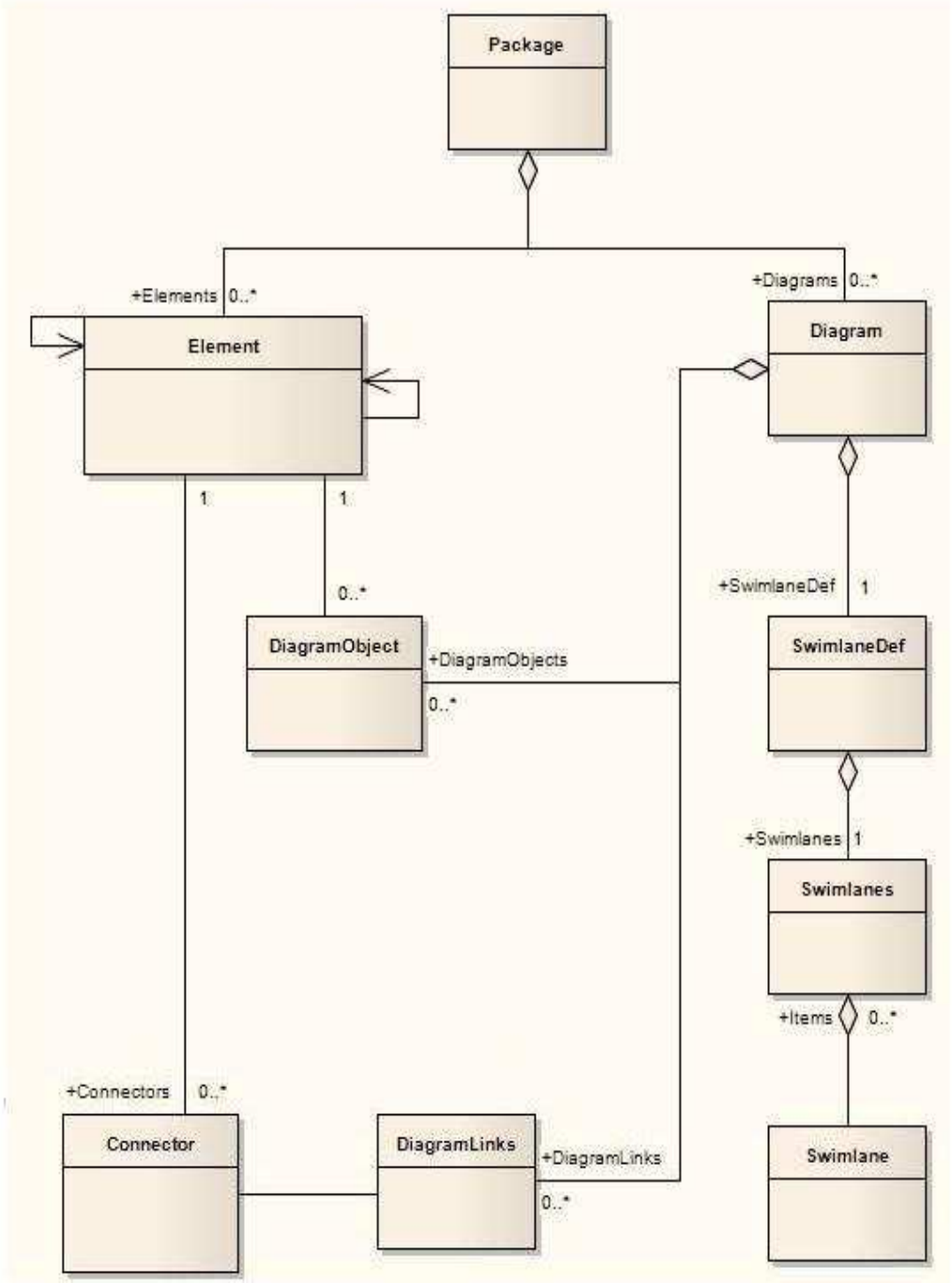
Package

VersionControlGetStatus ()	*Long*	Returns the version control status of the package. Throws an exception if the operation fails. Use *GetLastError( )* to retrieve error information. Return value maps to the following enumerated type: *enum EnumCheckOutStatus* *{* *csUncontrolled = 0,* *csCheckedIn,* *csCheckedOutToThisUser,* *csReadOnlyVersion,* *csCheckedOutToAnotherUser,* *csOfflineCheckedOutToThisUser,* *csOfflineNotCheckedOutToThisUser,* *csDeleted* *}* *csUncontrolled* - Either unable to communicate with the version control provider associated with the package or the package file is unknown to the provider. *csReadOnlyVersion* - Package is marked as read-only. An earlier revision of the package has been retrieved from version control. *csOfflineCheckedOutToThisUser* - Indicates that the package was "checked out" by this user whilst disconnected from version control. *csOfflineNotCheckedOutToThisUser* - Indicates that Enterprise Architect can not currently connect to the version control config and the package was not previously checked out to this user. *csDeleted* - The package file has been deleted from version control.
VersionControlRemove ()	*Void*	Removes version control from the package. Throws an exception if the operation fails. Use *GetLastError( )* to retrieve error information.

Diagram

public Package

The *Diagram* package has information on a diagram and on *DiagramObjects* and *DiagramLinks*, which are the instances of elements within a diagram.



Diagram

public Class

A *Diagram* corresponds to a single Enterprise Architect diagram. It is accessed through the *Package Diagrams* collection and in turn contains a collection of diagram objects and diagram connectors. Adding to the *DiagramObjects* collection adds an element to the diagram (the element must already exist). When adding a new diagram, you must set the diagram type to a valid type; these are:

- Activity
- Analysis
- Component
- Custom
- Deployment
- Logical
- Sequence
- Statechart
- Use Case

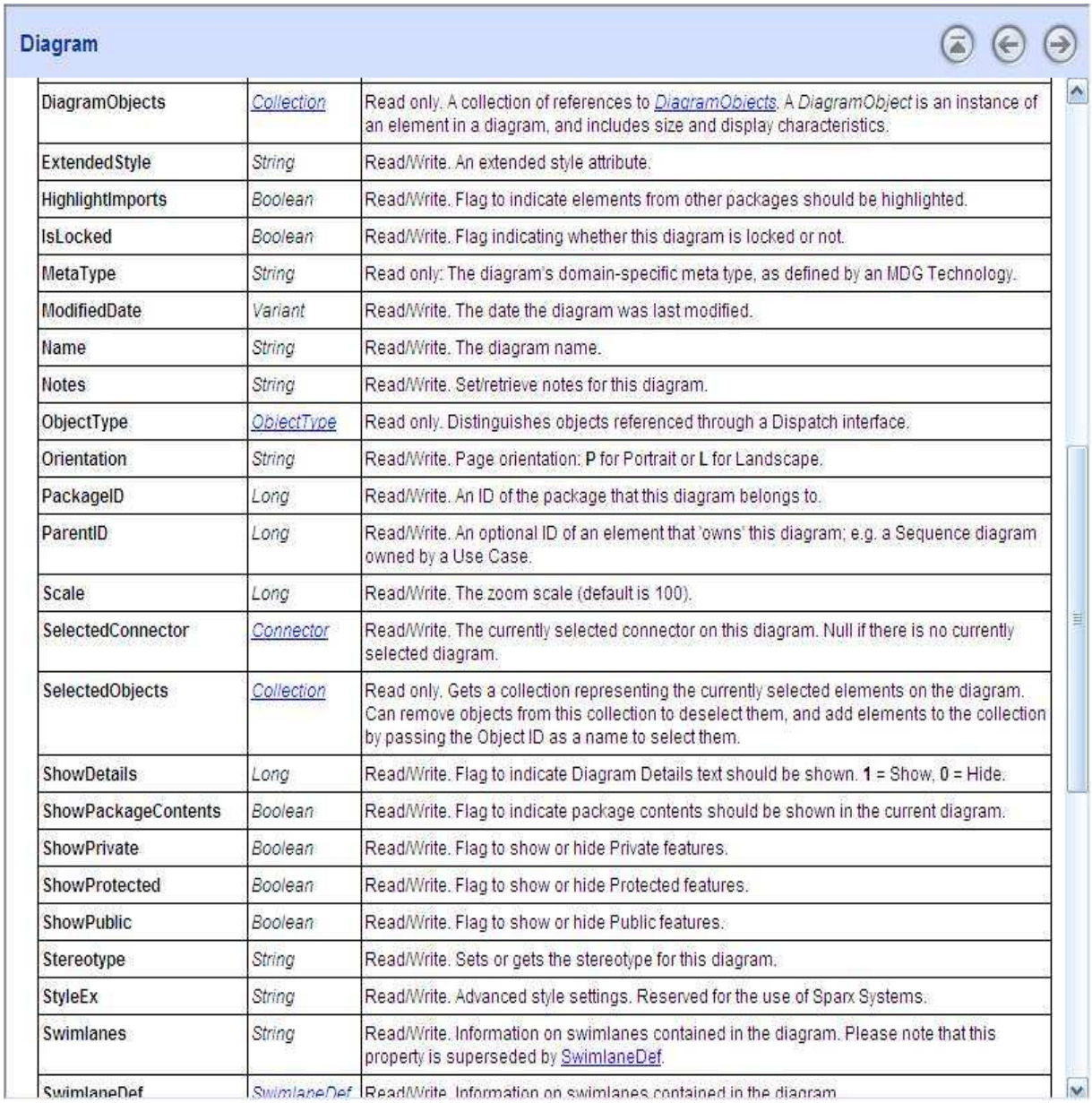
Note:

Use the Analysis type for a Collaboration Diagram.

Associated table in .EAP file: *t_diagram*

Diagram Attributes

Attribute	Type	Notes
Author	*String*	Read/Write. The author.
CreatedDate	*Date*	Read/Write. The date the diagram was created.
cx	*Long*	Read/Write. The *X* dimension of the diagram (default is 800).
cy	*Long*	Read/Write. The *Y* dimension of the diagram (default is 1100).
DiagramGUID	*Variant*	Read/Write. A globally unique ID for this diagram.
DiagramID	*Long*	Read only. A local ID for the diagram.
DiagramLinks	*Collection*	Read only. A list of *DiagramLink* objects, each containing information about the display characteristics of a connector in a diagram. **Note:** A DiagramLink is only created once a user modifies a connector in a diagram in some way. Until this condition has been met default values are used and the DiagramLink is not in use.



Diagram

DiagramObjects	Collection	Read only. A collection of references to DiagramObjects. A DiagramObject is an instance of an element in a diagram, and includes size and display characteristics.
ExtendedStyle	String	Read/Write. An extended style attribute.
HighlightImports	Boolean	Read/Write. Flag to indicate elements from other packages should be highlighted.
IsLocked	Boolean	Read/Write. Flag indicating whether this diagram is locked or not.
MetaType	String	Read only: The diagram's domain-specific meta type, as defined by an MDG Technology.
ModifiedDate	Variant	Read/Write. The date the diagram was last modified.
Name	String	Read/Write. The diagram name.
Notes	String	Read/Write. Set/retrieve notes for this diagram.
ObjectType	ObjectType	Read only. Distinguishes objects referenced through a Dispatch interface.
Orientation	String	Read/Write. Page orientation: **P** for Portrait or **L** for Landscape.
PackageID	Long	Read/Write. An ID of the package that this diagram belongs to.
ParentID	Long	Read/Write. An optional ID of an element that 'owns' this diagram; e.g. a Sequence diagram owned by a Use Case.
Scale	Long	Read/Write. The zoom scale (default is 100).
SelectedConnector	Connector	Read/Write. The currently selected connector on this diagram. Null if there is no currently selected diagram.
SelectedObjects	Collection	Read only. Gets a collection representing the currently selected elements on the diagram. Can remove objects from this collection to deselect them, and add elements to the collection by passing the Object ID as a name to select them.
ShowDetails	Long	Read/Write. Flag to indicate Diagram Details text should be shown. **1** = Show, **0** = Hide.
ShowPackageContents	Boolean	Read/Write. Flag to indicate package contents should be shown in the current diagram.
ShowPrivate	Boolean	Read/Write. Flag to show or hide Private features.
ShowProtected	Boolean	Read/Write. Flag to show or hide Protected features.
ShowPublic	Boolean	Read/Write. Flag to show or hide Public features.
Stereotype	String	Read/Write. Sets or gets the stereotype for this diagram.
StyleEx	String	Read/Write. Advanced style settings. Reserved for the use of Sparx Systems.
Swimlanes	String	Read/Write. Information on swimlanes contained in the diagram. Please note that this property is superseded by SwimlaneDef.
SwimlaneDef	SwimlaneDef	Read/Write. Information on swimlanes contained in the diagram.

Diagram

ShowPublic	*Boolean*	Read/Write. Flag to show or hide Public features.
Stereotype	*String*	Read/Write. Sets or gets the stereotype for this diagram.
StyleEx	*String*	Read/Write. Advanced style settings. Reserved for the use of Sparx Systems.
Swimlanes	*String*	Read/Write. Information on swimlanes contained in the diagram. Please note that this property is superseded by SwimlaneDef.
SwimlaneDef	*SwimlaneDef*	Read/Write. Information on swimlanes contained in the diagram.
Type	*String*	Read only. The diagram type. See the *t_diagramtypes* table in the .EAP file for more information.
Version	*String*	Read/Write. The version of the diagram.

Diagram Methods

Method	Type	Notes
ApplyGroupLock (string aGroupName)	*Boolean*	Applies a group lock to this diagram object, for the specified group, on behalf of the current user. Throws an exception if the operation fails. Use *GetLastError()* to retrieve error information. Parameter: • aGroupName: String - the name of the user group for which to set the group lock.
ApplyUserLock ()	*Boolean*	Applies a user lock to this diagram object, for the current user. Throws an exception if the operation fails. Use *GetLastError()* to retrieve error information.
GetLastError ()	*String*	Returns a string value describing the most recent error that occurred in relation to this object. This function is rarely used as an exception is thrown when an error occurs.
ReleaseLock ()	*Boolean*	Releases a group lock or user lock on this diagram object. Throws an exception if the operation fails. Use *GetLastError()* to retrieve error information.
ReorderMessages ()	*Void*	Resets the display order of Sequence and Collaboration messages. Typically used after inserting or deleting messages in the diagram.
ShowAsElementList (bool ShowAsList, bool Persist)	*Boolean*	Toggles the diagram display between diagram format and Element List depending on the value of *ShowAsList*. If *Persist* is set, the display format is written to the database so the diagram always opens in that format (diagram or list). Otherwise, the display format falls back to the default (diagram) once the display is closed.
Update ()	*Boolean*	Updates this diagram object after modification or appending a new item. If **false** is returned, use *GetLastError()* to retrieve error information.

DiagramObjects

public Class

The *DiagramObjects* collection holds a list of element IDs and presentation information that indicates what is displayed in a diagram and how it is shown.

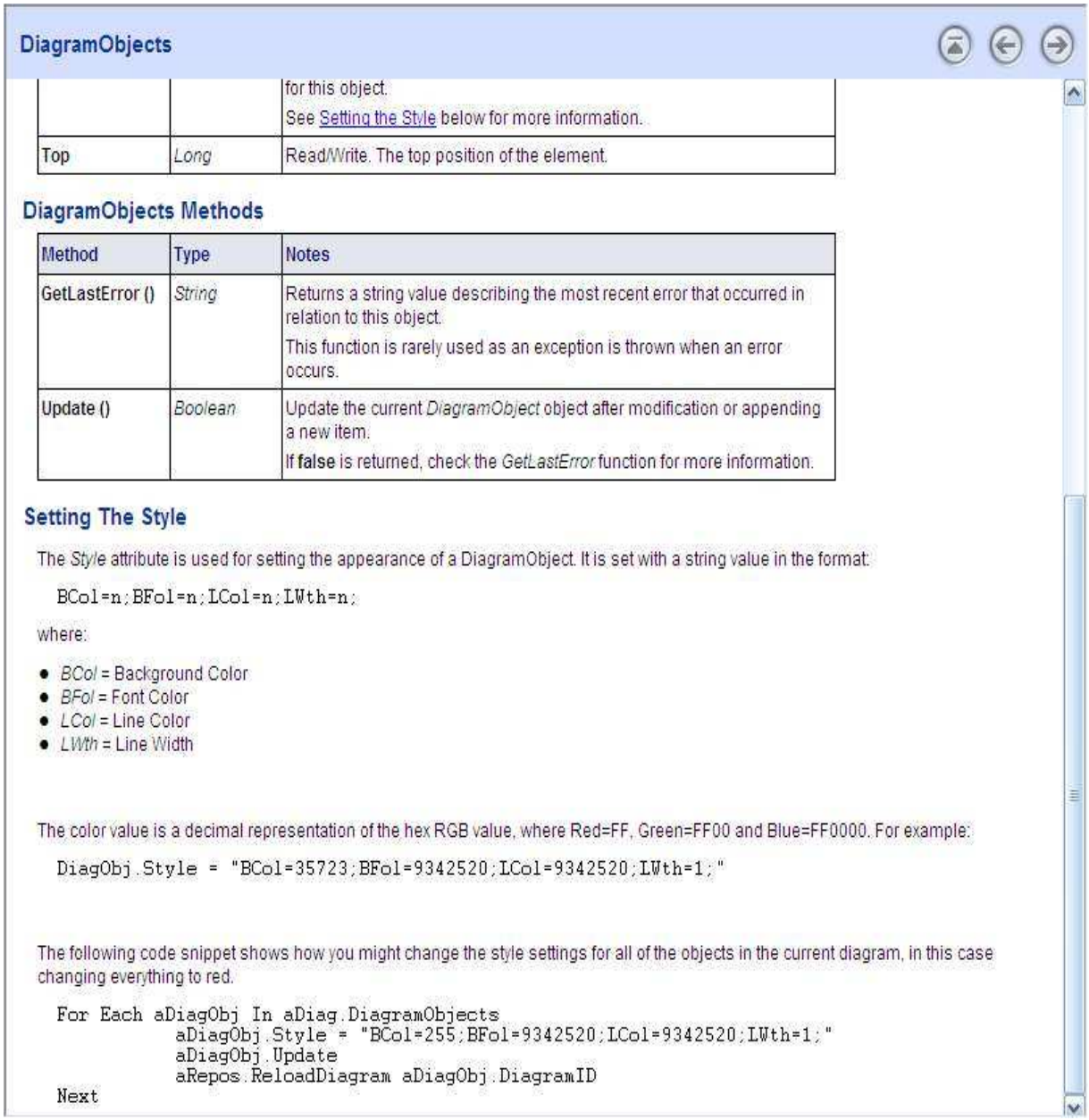
Associated table in .EAP file: *t_diagramobjects*

DiagramObjects Attributes

Attribute	Type	Notes
Bottom	*Long*	Read/Write. The bottom position of the element.
DiagramID	*Long*	Read/Write. The ID of the associated diagram (long).
ElementID	*Long*	Read/Write. The *ElementID* of the object instance in this diagram.
InstanceID	*Long*	Read/Write. Read only attribute. Holds the connector identifier for the current model.
Left	*Long*	Read/Write. The left position of the element.
ObjectType	*ObjectType*	Read only. Distinguishes objects referenced through a Dispatch interface.
Right	*Long*	Read/Write. The right position of the element.
Sequence	*Long*	Read/Write. The sequence position when loading into diagram (affects Z order). The Z-order is one-based and the lowest value is in the foreground.
Style	*Variant*	Write only (reading this value gives undefined results). Style information for this object. See Setting the Style below for more information.
Top	*Long*	Read/Write. The top position of the element.

DiagramObjects Methods

Method	Type	Notes
GetLastError ()	*String*	Returns a string value describing the most recent error that occurred in relation to this object. This function is rarely used as an exception is thrown when an error occurs.
Update ()	*Boolean*	Update the current *DiagramObject* object after modification or appending a new item.



DiagramObjects

		for this object. See Setting the Style below for more information.
Top	*Long*	Read/Write. The top position of the element.

DiagramObjects Methods

Method	Type	Notes
GetLastError ()	*String*	Returns a string value describing the most recent error that occurred in relation to this object. This function is rarely used as an exception is thrown when an error occurs.
Update ()	*Boolean*	Update the current *DiagramObject* object after modification or appending a new item. If **false** is returned, check the *GetLastError* function for more information.

Setting The Style

The *Style* attribute is used for setting the appearance of a DiagramObject. It is set with a string value in the format:

```
BCol=n;BFol=n;LCol=n;LWth=n;
```

where:

- *BCol* = Background Color
- *BFol* = Font Color
- *LCol* = Line Color
- *LWth* = Line Width

The color value is a decimal representation of the hex RGB value, where Red=FF, Green=FF00 and Blue=FF0000. For example:

```
DiagObj.Style = "BCol=35723;BFol=9342520;LCol=9342520;LWth=1;"
```

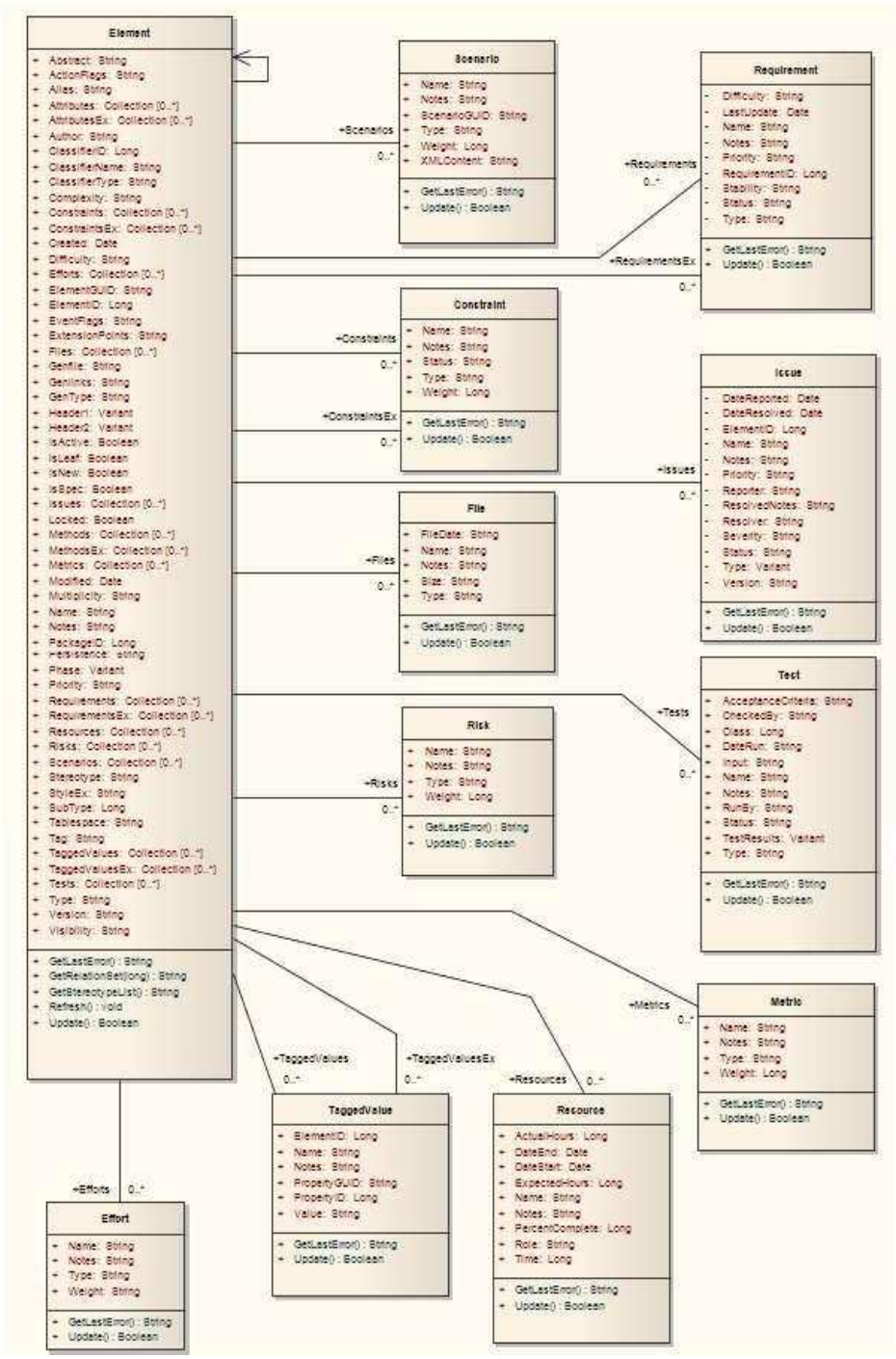
The following code snippet shows how you might change the style settings for all of the objects in the current diagram, in this case changing everything to red.

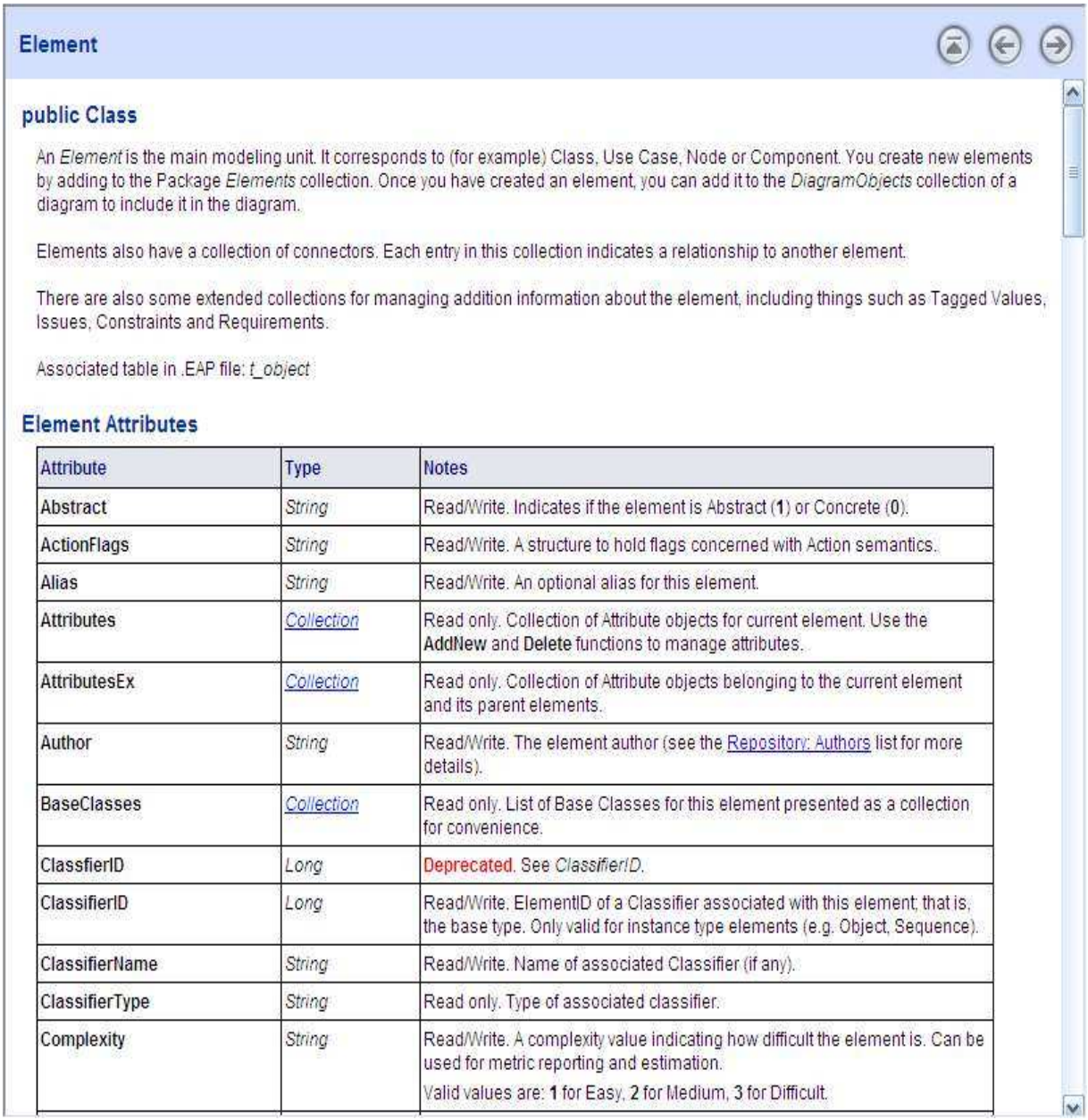
```
For Each aDiagObj In aDiag.DiagramObjects
          aDiagObj.Style = "BCol=255;BFol=9342520;LCol=9342520;LWth=1;"
          aDiagObj.Update
          aRepos.ReloadDiagram aDiagObj.DiagramID
Next
```

Element

The *Element* package contains information about an element and its associated extended properties such as testing and project management information. An element is the basic item in an Enterprise Architect model. Classes, Use Cases and Components are all different types of UML element.

The diagram below illustrates the relationships between an *element* and its associated extended information. The related information is accessed through the collections owned by the element (e.g. Scenarios and Tests). It also includes a full description of the element object (the basic model structural unit).





Element

public Class

An *Element* is the main modeling unit. It corresponds to (for example) Class, Use Case, Node or Component. You create new elements by adding to the Package *Elements* collection. Once you have created an element, you can add it to the *DiagramObjects* collection of a diagram to include it in the diagram.

Elements also have a collection of connectors. Each entry in this collection indicates a relationship to another element.

There are also some extended collections for managing addition information about the element, including things such as Tagged Values, Issues, Constraints and Requirements.

Associated table in .EAP file: *t_object*

Element Attributes

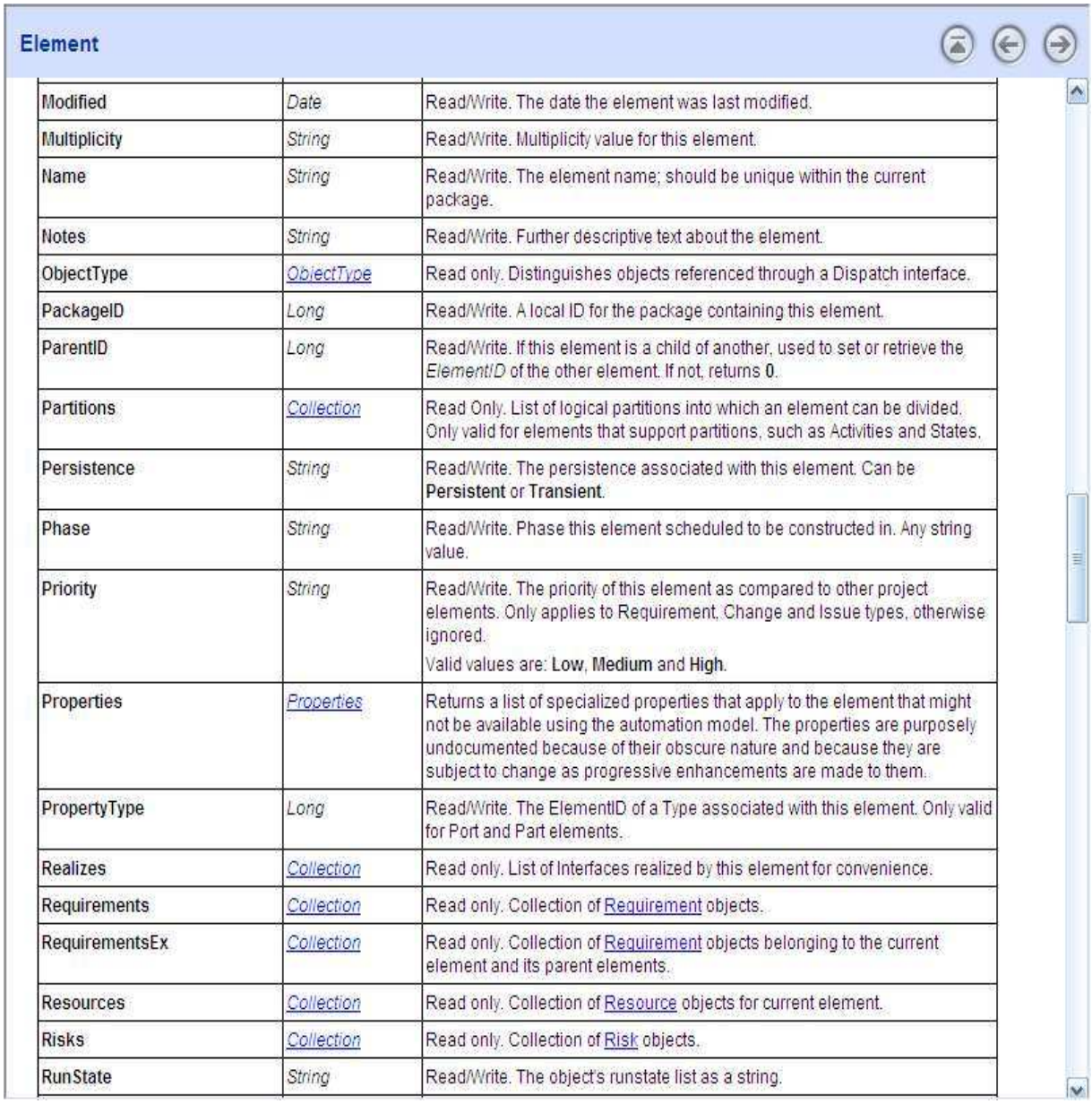
Attribute	Type	Notes
Abstract	*String*	Read/Write. Indicates if the element is Abstract (**1**) or Concrete (**0**).
ActionFlags	*String*	Read/Write. A structure to hold flags concerned with Action semantics.
Alias	*String*	Read/Write. An optional alias for this element.
Attributes	*Collection*	Read only. Collection of Attribute objects for current element. Use the **AddNew** and **Delete** functions to manage attributes.
AttributesEx	*Collection*	Read only. Collection of Attribute objects belonging to the current element and its parent elements.
Author	*String*	Read/Write. The element author (see the Repository: Authors list for more details).
BaseClasses	*Collection*	Read only. List of Base Classes for this element presented as a collection for convenience.
ClassfierID	*Long*	Deprecated. See *ClassifierID*.
ClassifierID	*Long*	Read/Write. ElementID of a Classifier associated with this element; that is, the base type. Only valid for instance type elements (e.g. Object, Sequence).
ClassifierName	*String*	Read/Write. Name of associated Classifier (if any).
ClassifierType	*String*	Read only. Type of associated classifier.
Complexity	*String*	Read/Write. A complexity value indicating how difficult the element is. Can be used for metric reporting and estimation. Valid values are: **1** for Easy, **2** for Medium, **3** for Difficult.

Element

CompositeDiagram	*Diagram*	Read only. If the element is Composite, returns its associated diagram; otherwise returns null.
Connectors	*Collection*	Read only. Returns a collection containing the connectors to other elements.
Constraints	*Collection*	Read only. Collection of Constraint objects.
ConstraintsEx	*Collection*	Read only. Collection of Constraint objects belonging to the current element and its parent elements.
Created	*Date*	Read/Write. The date the element was created.
CustomProperties	*Collection*	Read only. List of advanced properties for an element. The collection of advanced properties differs depending on element type; for example, an Action and an Activity have different advanced properties. Currently only editable from the user interface.
Diagrams	*Collection*	Read only. Returns a collection of sub-diagrams (child diagrams) attached to this element as seen in the tree view.
Difficulty	*String*	Read/Write. A difficulty level associated with this element for estimation/metrics; only useable for Requirement, Change and Issue element types, otherwise ignored. Valid values are: **Low**, **Medium**, **High**.
Efforts	*Collection*	Read only. Collection of Effort objects.
ElementGUID	*String*	Read only. A globally unique ID for this element; that is, unique across all model files. If you have to set this value manually, you should only do so when the element is first created, and make sure you format the GUID exactly as Enterprise Architect expects.
ElementID	*Long*	Read only. The local ID of the Element. Valid for this file only.
Elements	*Collection*	Read only. Returns a collection of child elements (sub-elements) attached to this element as seen in the tree view.
EmbeddedElements	*Collection*	Read only. List of elements that are embedded into this element, such as Ports, Parts, Pins and Parameter Sets.
EventFlags	*String*	Read/Write. A structure to hold a variety of flags to do with signals or events.
ExtensionPoints	*String*	Read/Write. Optional extension points for a Use Case as a comma-separated list.
Files	*Collection*	Read only. Collection of File objects.
GenFile	*String*	Read/Write. The file associated with this element for code generation and synchronization purposes. Can include macro expansion tags for local

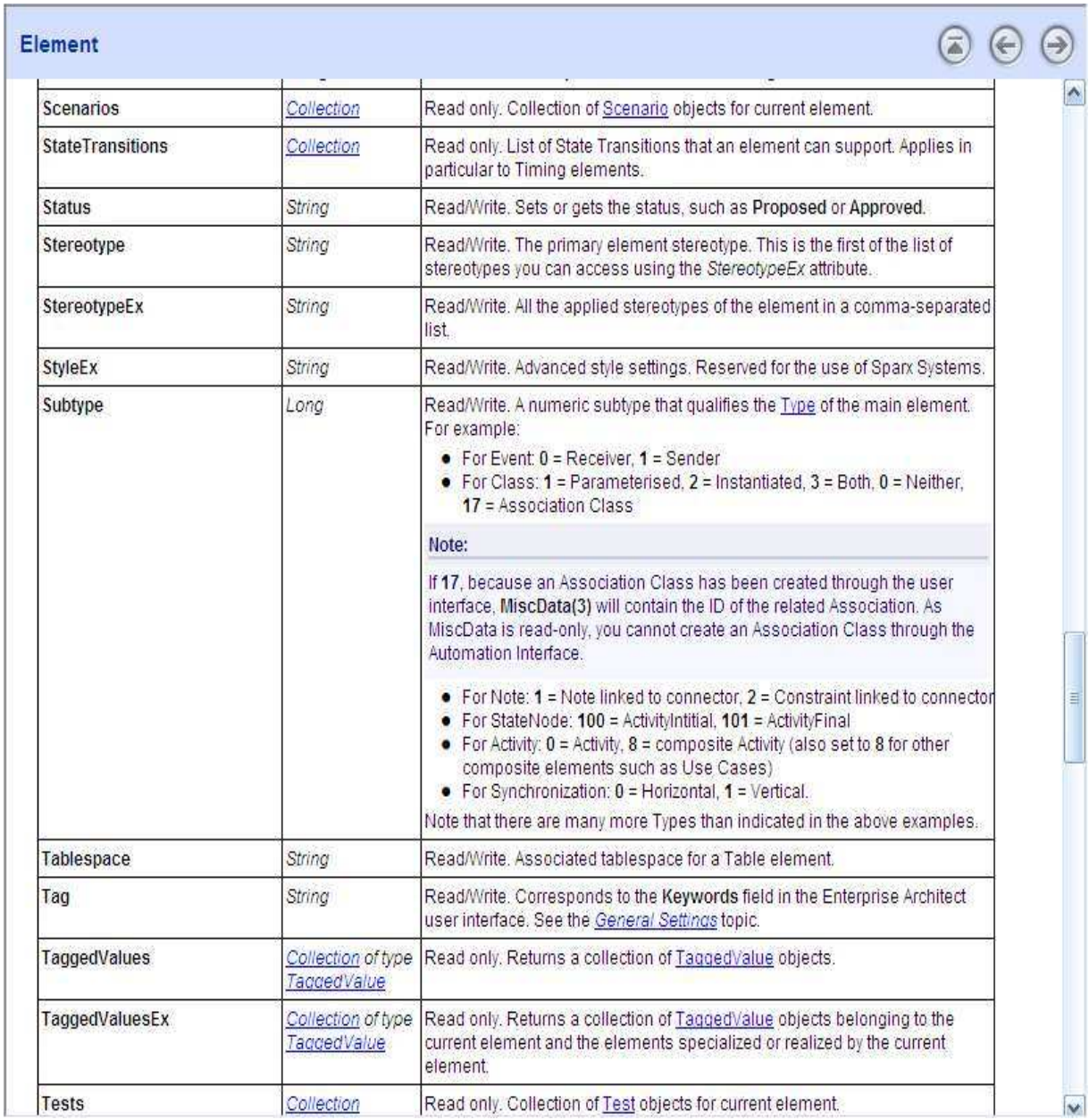
Element

Genlinks	*String*	Read/Write. Links to other Classes discovered at code reversing time; Parents and Implements connectors only.
GenType	*String*	Read/Write. The code generation type; e.g. Java, C++, C#, VBNet, Visual Basic, Delphi.
Header1	*Variant*	Read/Write. A user defined string for inclusion as header in the source files generated.
Header2	*Variant*	Read/Write. Same as for **Header1**, but used in the CPP source file.
IsActive	*Boolean*	Read/Write. Boolean value indicating whether the element is active or not. **1** = True, **0** = False.
IsLeaf	*Boolean*	Read/Write. Boolean value indicating whether the element is in leaf node or not. **1** = True, **0** = False.
IsNew	*Boolean*	Read/Write. Boolean value indicating whether the element is new or not. **1** = True, **0** = False.
IsSpec	*Boolean*	Read/Write. Boolean value indicating whether the element is a specification or not. **1** = True, **0** = False.
Issues	*Collection*	Read only. Collection of Issue objects.
Locked	*Boolean*	Read/Write. Indicates if the element has been locked against further change.
MetaType	*String*	Read only: The element's domain-specific meta type, as defined by an applied stereotype from an MDG Technology.
Methods	*Collection*	Read only. Collection of Method objects for current element.
MethodsEx	*Collection*	Read only. Collection of Method objects belonging to the current element and its parent elements.
Metrics	*Collection*	Read only. Collection of Metric elements for current element.
MiscData	*String*	Read only. This low-level property provides information about the contents of the **PDatax** fields. These database fields are not documented and developers must gain understanding of these fields through their own endeavors to use this property. **MiscData** is zero based, therefore: • **MiscData(0)** corresponds to **PDATA1** • **MiscData(1)** to **PDATA2** and so on.



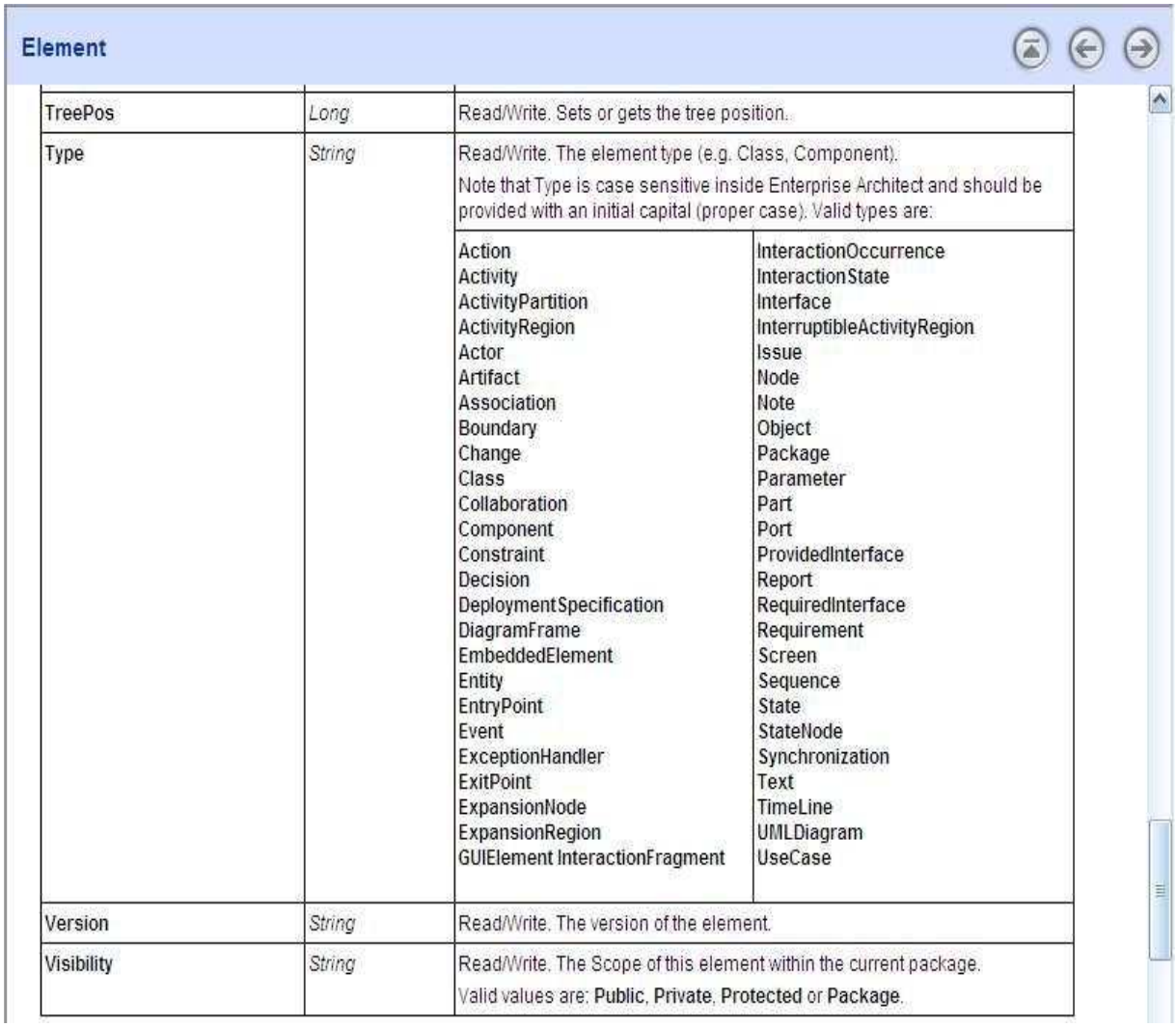
Element

Modified	*Date*	Read/Write. The date the element was last modified.
Multiplicity	*String*	Read/Write. Multiplicity value for this element.
Name	*String*	Read/Write. The element name; should be unique within the current package.
Notes	*String*	Read/Write. Further descriptive text about the element.
ObjectType	*ObjectType*	Read only. Distinguishes objects referenced through a Dispatch interface.
PackageID	*Long*	Read/Write. A local ID for the package containing this element.
ParentID	*Long*	Read/Write. If this element is a child of another, used to set or retrieve the *ElementID* of the other element. If not, returns **0**.
Partitions	*Collection*	Read Only. List of logical partitions into which an element can be divided. Only valid for elements that support partitions, such as Activities and States.
Persistence	*String*	Read/Write. The persistence associated with this element. Can be **Persistent** or **Transient**.
Phase	*String*	Read/Write. Phase this element scheduled to be constructed in. Any string value.
Priority	*String*	Read/Write. The priority of this element as compared to other project elements. Only applies to Requirement, Change and Issue types, otherwise ignored. Valid values are: **Low**, **Medium** and **High**.
Properties	*Properties*	Returns a list of specialized properties that apply to the element that might not be available using the automation model. The properties are purposely undocumented because of their obscure nature and because they are subject to change as progressive enhancements are made to them.
PropertyType	*Long*	Read/Write. The ElementID of a Type associated with this element. Only valid for Port and Part elements.
Realizes	*Collection*	Read only. List of Interfaces realized by this element for convenience.
Requirements	*Collection*	Read only. Collection of Requirement objects.
RequirementsEx	*Collection*	Read only. Collection of Requirement objects belonging to the current element and its parent elements.
Resources	*Collection*	Read only. Collection of Resource objects for current element.
Risks	*Collection*	Read only. Collection of Risk objects.
RunState	*String*	Read/Write. The object's runstate list as a string.



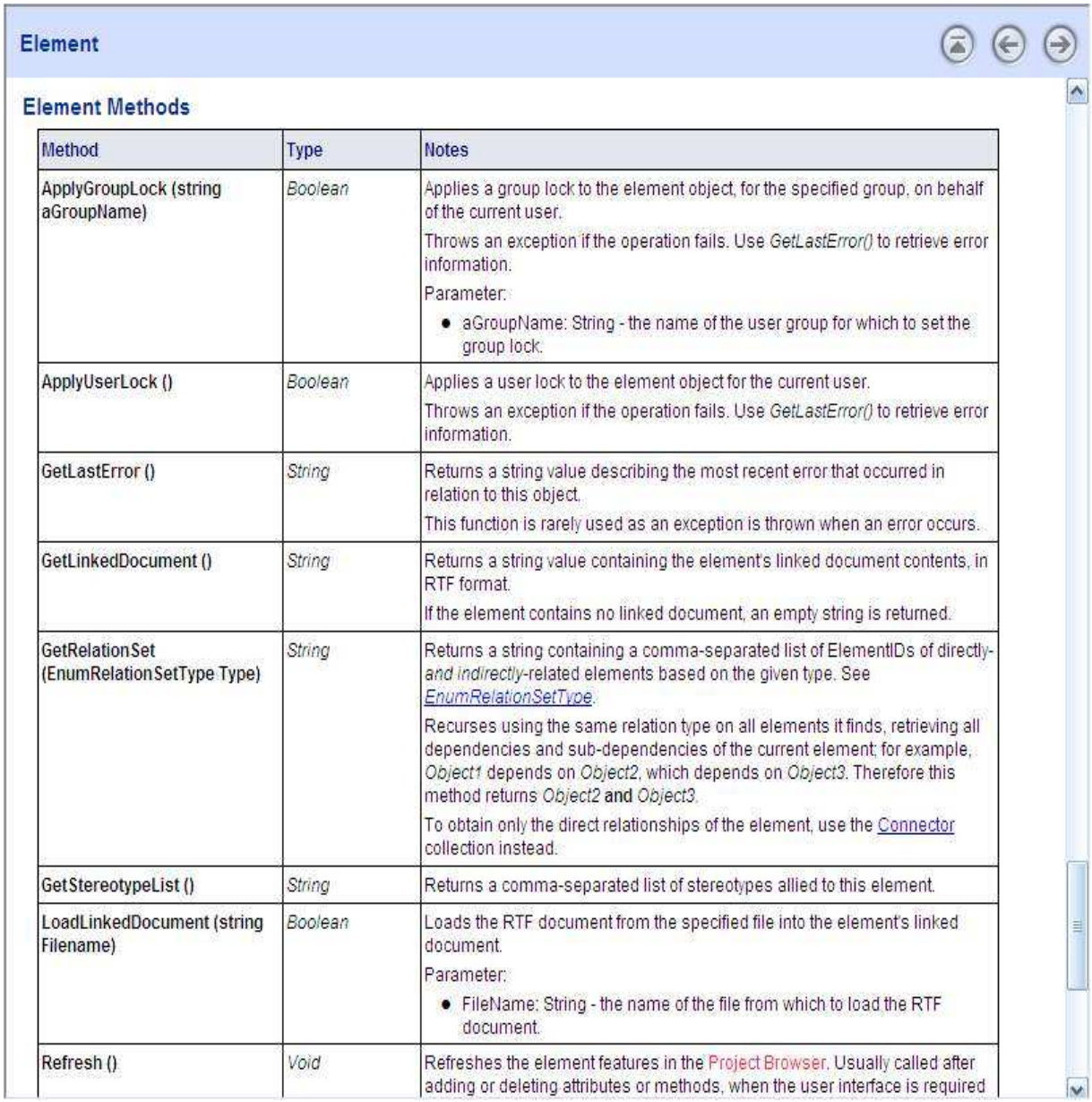
Element

Scenarios	*Collection*	Read only. Collection of Scenario objects for current element.
StateTransitions	*Collection*	Read only. List of State Transitions that an element can support. Applies in particular to Timing elements.
Status	*String*	Read/Write. Sets or gets the status, such as **Proposed** or **Approved**.
Stereotype	*String*	Read/Write. The primary element stereotype. This is the first of the list of stereotypes you can access using the *StereotypeEx* attribute.
StereotypeEx	*String*	Read/Write. All the applied stereotypes of the element in a comma-separated list.
StyleEx	*String*	Read/Write. Advanced style settings. Reserved for the use of Sparx Systems.
Subtype	*Long*	Read/Write. A numeric subtype that qualifies the Type of the main element. For example: • For Event: **0** = Receiver, **1** = Sender • For Class: **1** = Parameterised, **2** = Instantiated, **3** = Both, **0** = Neither, **17** = Association Class **Note:** If **17**, because an Association Class has been created through the user interface, **MiscData(3)** will contain the ID of the related Association. As MiscData is read-only, you cannot create an Association Class through the Automation Interface. • For Note: **1** = Note linked to connector, **2** = Constraint linked to connector • For StateNode: **100** = ActivityIntitial, **101** = ActivityFinal • For Activity: **0** = Activity, **8** = composite Activity (also set to **8** for other composite elements such as Use Cases) • For Synchronization: **0** = Horizontal, **1** = Vertical. Note that there are many more Types than indicated in the above examples.
Tablespace	*String*	Read/Write. Associated tablespace for a Table element.
Tag	*String*	Read/Write. Corresponds to the **Keywords** field in the Enterprise Architect user interface. See the *General Settings* topic.
TaggedValues	*Collection of type TaggedValue*	Read only. Returns a collection of TaggedValue objects.
TaggedValuesEx	*Collection of type TaggedValue*	Read only. Returns a collection of TaggedValue objects belonging to the current element and the elements specialized or realized by the current element.
Tests	*Collection*	Read only. Collection of Test objects for current element.



Element

TreePos	*Long*	Read/Write. Sets or gets the tree position.
Type	*String*	Read/Write. The element type (e.g. Class, Component). Note that Type is case sensitive inside Enterprise Architect and should be provided with an initial capital (proper case). Valid types are: Action, Activity, ActivityPartition, ActivityRegion, Actor, Artifact, Association, Boundary, Change, Class, Collaboration, Component, Constraint, Decision, DeploymentSpecification, DiagramFrame, EmbeddedElement, Entity, EntryPoint, Event, ExceptionHandler, ExitPoint, ExpansionNode, ExpansionRegion, GUIElement InteractionFragment, InteractionOccurrence, InteractionState, Interface, InterruptibleActivityRegion, Issue, Node, Note, Object, Package, Parameter, Part, Port, ProvidedInterface, Report, RequiredInterface, Requirement, Screen, Sequence, State, StateNode, Synchronization, Text, TimeLine, UMLDiagram, UseCase
Version	*String*	Read/Write. The version of the element.
Visibility	*String*	Read/Write. The Scope of this element within the current package. Valid values are: **Public**, **Private**, **Protected** or **Package**.



Element

Element Methods

Method	Type	Notes
ApplyGroupLock (string aGroupName)	*Boolean*	Applies a group lock to the element object, for the specified group, on behalf of the current user. Throws an exception if the operation fails. Use *GetLastError()* to retrieve error information. Parameter: • aGroupName: String - the name of the user group for which to set the group lock.
ApplyUserLock ()	*Boolean*	Applies a user lock to the element object for the current user. Throws an exception if the operation fails. Use *GetLastError()* to retrieve error information.
GetLastError ()	*String*	Returns a string value describing the most recent error that occurred in relation to this object. This function is rarely used as an exception is thrown when an error occurs.
GetLinkedDocument ()	*String*	Returns a string value containing the element's linked document contents, in RTF format. If the element contains no linked document, an empty string is returned.
GetRelationSet (EnumRelationSetType Type)	*String*	Returns a string containing a comma-separated list of ElementIDs of directly- *and indirectly*-related elements based on the given type. See *EnumRelationSetType*. Recurses using the same relation type on all elements it finds, retrieving all dependencies and sub-dependencies of the current element; for example, *Object1* depends on *Object2*, which depends on *Object3*. Therefore this method returns *Object2* **and** *Object3*. To obtain only the direct relationships of the element, use the Connector collection instead.
GetStereotypeList ()	*String*	Returns a comma-separated list of stereotypes allied to this element.
LoadLinkedDocument (string Filename)	*Boolean*	Loads the RTF document from the specified file into the element's linked document. Parameter: • FileName: String - the name of the file from which to load the RTF document.
Refresh ()	*Void*	Refreshes the element features in the Project Browser. Usually called after adding or deleting attributes or methods, when the user interface is required

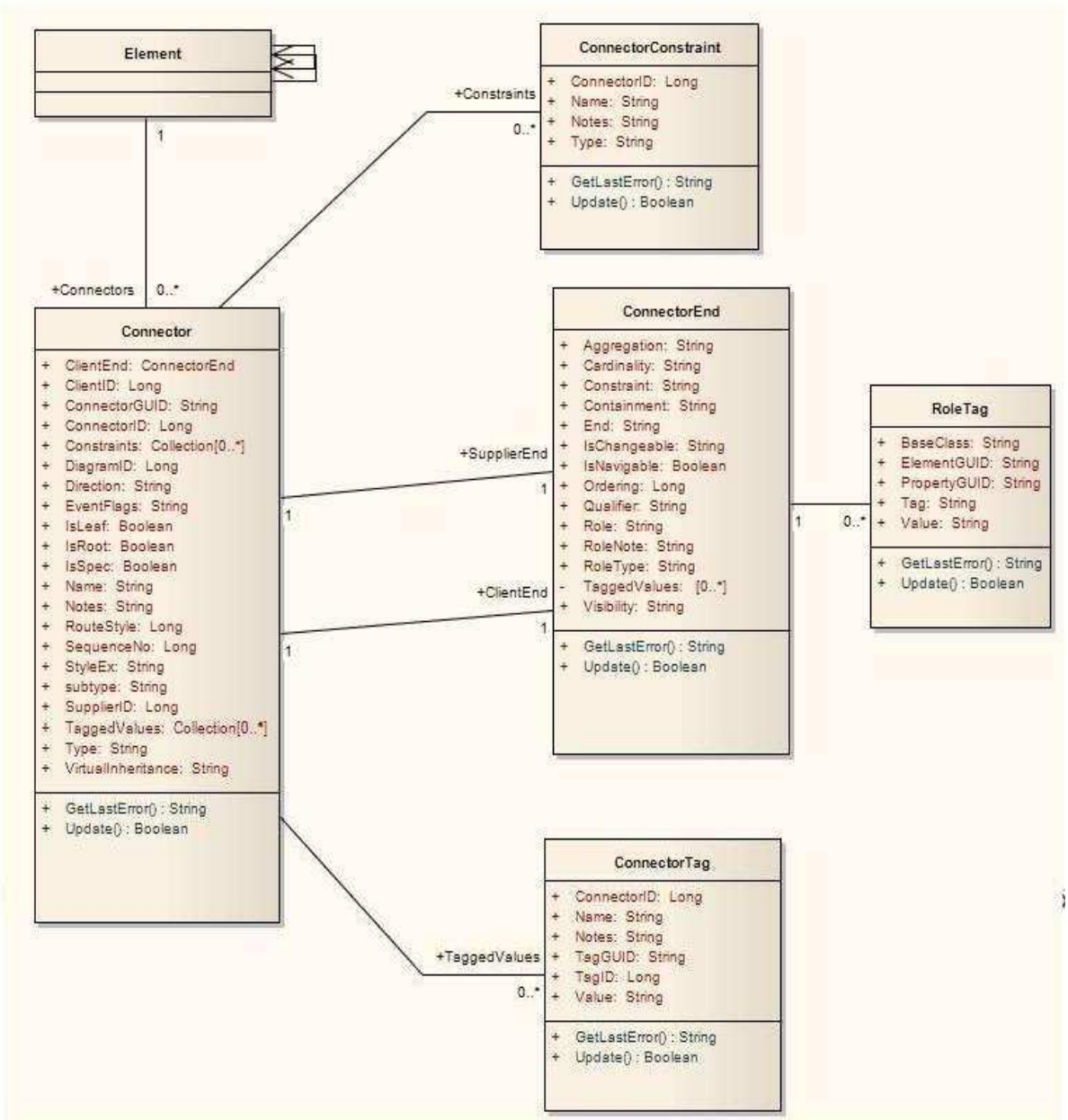
Element

GetStereotypeList ()	*String*	Returns a comma-separated list of stereotypes allied to this element.
LoadLinkedDocument (string Filename)	*Boolean*	Loads the RTF document from the specified file into the element's linked document. Parameter: • FileName: String - the name of the file from which to load the RTF document.
Refresh ()	*Void*	Refreshes the element features in the Project Browser. Usually called after adding or deleting attributes or methods, when the user interface is required to be updated as well.
ReleaseLock ()	*Boolean*	Releases a user lock or group lock on the element object. Throws an exception if the operation fails. Use *GetLastError()* to retrieve error information.
SaveLinkedDocument (string Filename)	*Boolean*	Saves the linked document for this element to the specified RTF file. Parameter: • FileName: String - the name of the RTF file to which to save the linked document.
SetAppearance (long Scope, long Item, long Value)	*Void*	Sets the visual appearance of the element. Parameter: • Scope: Long - Scope of appearance set to modify **0** – Local (Diagram-local appearance) **1** – Base (Default appearance across entire model) • Item: Long - Appearance item to modify **0** – Background color **1** – Font Color **2** – Border Color **3** – Border Width • Value: Long - Value to set appearance to.
Update ()	*Boolean*	Update the current element object after modification or appending a new item. If **false** is returned, check the *GetLastError* function for more information.

Connector

public Package

The *Connector* package details how connectors between elements are accessed and managed.



Connector

public Class

A *Connector* object represents the various kinds of connectors between UML elements. It is accessed from either the *Client* or *Supplier* element, using the *Connectors* collection of that element. When creating a new connector you must assign it a valid type from the following list:

- Aggregation
- Association
- Collaboration
- Dependency
- Generalization
- Instantiation
- Nesting
- NoteLink
- Realisation
- Sequence
- Transition
- UseCase

Associated table in .EAP file: *t_connector*

Connector Attributes

Attribute	Type	Notes
Alias	*String*	Read/Write. An optional alias for this connector.
ClientEnd	*ConnectorEnd*	Read only. A pointer to the *ConnectorEnd* object representing the source end of the relationship.
ClientID	*Long*	Read/Write. *ElementID* of the element at the source end of this connector.
Color	*Long*	Read/Write. Sets the color of the connector.
ConnectorGUID	*Variant*	Read only. A globally unique ID for the current connector. System generated.
ConnectorID	*Long*	Read only. Local identifier for the current connector. System generated.
Constraints	*Collection*	Read only. Collection of constraint objects.
CustomProperties	*Collection*	Read only. Returns a collection of advanced properties associated with an element in the form of CustomProperty objects.
DiagramID	*Long*	Read/Write. The *DiagramID* of the connector.

Connector

Direction	*String*	Read/Write. Connector direction. Can be set to one of the following: • Unspecified • Bi-Directional • Source -> Destination • Destination -> Source
EventFlags	*String*	Read/Write. Structure to hold a variety of flags concerned with event signaling on messages.
IsLeaf	*Boolean*	Read/Write. Flag indicating connector is a *leaf*.
IsRoot	*Boolean*	Read/Write. Flag indicating connector is a *root*.
IsSpec	*Boolean*	Read/Write. Flag indicating connector is a specification.
MetaType	*String*	Read only: The connector's domain-specific meta type, as defined by an applied stereotype from an MDG Technology.
Name	*String*	Read/Write. The connector name.
Notes	*String*	Read/Write. Descriptive notes about the connector.
ObjectType	*ObjectType*	Read only. Distinguishes objects referenced through a Dispatch interface.
Properties	*Properties*	Returns a list of specialized properties that apply to the connector that might not be available using the automation model. The properties are purposely undocumented because of their obscure nature and because they are subject to change as progressive enhancements are made to them.
RouteStyle	*Long*	Read/Write. The route style.
SequenceNo	*Long*	Read/Write. The *SequenceNo* of the connector.
Stereotype	*String*	Read/Write. Sets or gets the stereotype for this connector end.
StereotypeEx	*String*	Read/Write. All the applied stereotypes of the connector in a comma-separated list.
StyleEx	*String*	Read/Write. Advanced style settings. Reserved for the use of Sparx Systems.
Subtype	*String*	Read/Write. A possible subtype to refine the meaning of the connector.
SupplierEnd	*ConnectorEnd*	Read only. A pointer to the *ConnectorEnd* object representing the target end of the relationship.
SupplierID	*Long*	Read/Write. *ElementID* of the element at the target end of this connector.
TaggedValues	*Collection*	Read only. Collection of *ConnectorTag* objects.
TransitionAction	*String*	Read/Write. See the *Transition* topic in the *Enterprise Architect User Guide* for appropriate values.
TransitionEvent	*String*	Read/Write. See the *Transition* topic in the *Enterprise Architect User Guide* for appropriate

Connector

Properties	*Properties*	Returns a list of specialized properties that apply to the connector that might not be available using the automation model. The properties are purposely undocumented because of their obscure nature and because they are subject to change as progressive enhancements are made to them.
RouteStyle	*Long*	Read/Write. The route style.
SequenceNo	*Long*	Read/Write. The *SequenceNo* of the connector.
Stereotype	*String*	Read/Write. Sets or gets the stereotype for this connector end.
StereotypeEx	*String*	Read/Write. All the applied stereotypes of the connector in a comma-separated list.
StyleEx	*String*	Read/Write. Advanced style settings. Reserved for the use of Sparx Systems.
Subtype	*String*	Read/Write. A possible subtype to refine the meaning of the connector.
SupplierEnd	*ConnectorEnd*	Read only. A pointer to the *ConnectorEnd* object representing the target end of the relationship.
SupplierID	*Long*	Read/Write. *ElementID* of the element at the target end of this connector.
TaggedValues	*Collection*	Read only. Collection of *ConnectorTag* objects.
TransitionAction	*String*	Read/Write. See the *Transition* topic in the *Enterprise Architect User Guide* for appropriate values.
TransitionEvent	*String*	Read/Write. See the *Transition* topic in the *Enterprise Architect User Guide* for appropriate values.
TransitionGuard	*String*	Read/Write. See the *Transition* topic in the *Enterprise Architect User Guide* for appropriate values.
Type	*String*	Read/Write. Connector type. Valid types are held in the *t_connectortypes* table in the .EAP file.
VirtualInheritance	*String*	Read/Write. For *Generalization*, indicates if inheritance is virtual.
Width	*Long*	Read/Write. Specifies the width of the connector.

Connector Methods

Method	Type	Notes
GetLastError ()	*String*	Returns a string value describing the most recent error that occurred in relation to this object. This function is rarely used as an exception is thrown when an error occurs.
Update ()	*Boolean*	Update the current *ConnectorObject* after modification or appending a new item. If **false** is returned, check the *GetLastError* function for more information.

ANEXO B

RESULTADOS DO ESTUDO DE CASO REALIZADO

1. Comparação entre as Técnicas Convencional e Indutiva

Nesta seção encontra-se descrito um estudo de caso que faz uma comparação entre as operações convencionais de ferramentas CASE e as operações propostas pela Rastreabilidade Indutiva. Neste estudo de caso é mostrado como acontece a criação da rastreabilidade entre os elementos dos artefatos de software com as duas técnicas. Os artefatos utilizados são "diagrama de requisitos", "diagrama de casos de uso" e "modelo conceitual".
Inicialmente é apresentada a descrição de uma tarefa na qual o desenvolvedor inicia o projeto de um sistema para uma vídeolocadora. A Tabela 1 descreve a tarefa do desenvolvedor ao modelar um sistema para o qual ele deverá também mapear a rastreabilidade entre os elementos.

Tabela 1. Descrição da Tarefa.

Modelagem de Requisitos:
Criação do requisito RF001: "O sistema deve permitir o empréstimo e devolução de filmes".
Criação do requisito RF002: "O sistema deve permitir o cadastro de filmes e clientes".
Modelagem de Casos de Uso:
O requisito RF001 implica na criação do caso de uso UC001: "Emprestar filme".
O requisito RF001 implica na criação do caso de uso UC002: "Devolver filme".
O requisito RF002 implica na criação do caso de uso UC003: "Cadastrar cliente".
O requisito RF002 implica na criação do caso de uso UC004: "Cadastrar filme".
Modelagem Conceitual:
O caso de uso UC001 implica na criação das classes Emprestimo, Cliente e Filme.
O caso de uso UC002 implica a criação da classe Devolucao.
O caso de uso UC003 afeta a estrutura interna da classe Cliente.
O caso de uso UC004 afeta a estrutura interna da classe Filme.

A Figura 1 apresenta o diagrama de rastreabilidade esperado após a execução da tarefa apresentada na Tabela 1. O diagrama de rastreabilidade objetiva apresentar as relações de rastreabilidade entre os elementos.

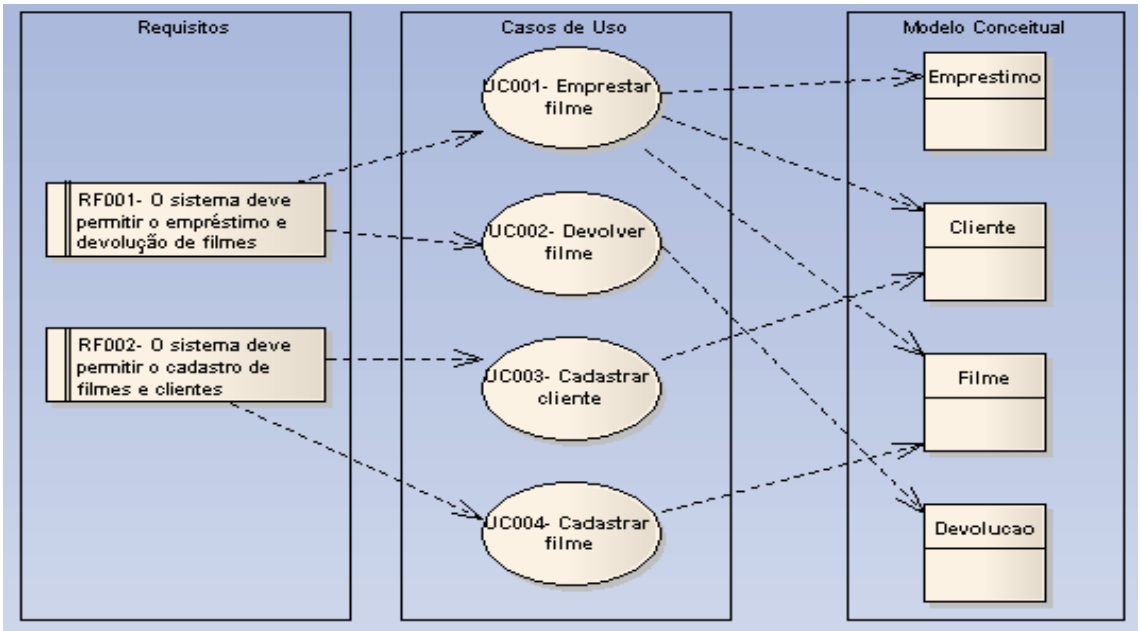


Figura 1. Diagrama de rastreabilidade.

A Tabela 2 apresenta um comparativo entre as operações que devem ser realizadas pelo desenvolvedor com a técnica convencional, única possibilidade nas ferramentas CASE existentes, e com a técnica indutiva. Em ambos os casos, considera-se que o desenvolvedor deseja criar e manter um registro das relações de rastreabilidade ao longo da tarefa.

Tabela 2. Comparativo entre a técnica convencional e a indutiva.

Técnica Convencional	Técnica Indutiva
Modelagem de Requisitos: 1) Foi criado o requisito RF001. 2) Foi criado o requisito RF002.	**Modelagem de Requisitos:** 1) Foi criado o elemento-base RF001. 2) Foi criado o elemento-base RF002.
Modelagem de Casos de Uso: 3) Foi criado o UC001. 4) Foi criado o UC002. 5) Foi criado o UC003. 6) Foi criado o UC004. 7) Foi criada uma relação de rastreabilidade entre RF001 e UC001. 8) Foi criada uma relação de rastreabilidade entre RF001 e UC002. 9) Foi criada uma relação de rastreabilidade entre RF002 e UC003. 10) Foi criada uma relação de rastreabilidade entre RF002 e UC004.	**Modelagem de Casos de Uso:** 11) A partir do RF001 é derivado o UC001. 12) A partir do RF001 é derivado o UC002. 13) A partir do RF002 é derivado o UC003. 14) A partir do RF002 é derivado o UC004.
Modelagem Conceitual: 15) Foi criada a classe Emprestimo. 16) Foi criada a classe Cliente. 17) Foi criada a classe Filme. 18) Foi criada a classe Devolucao.	**Modelagem Conceitual:** 25) A partir do UC001 é derivada a classe Emprestimo. 26) A partir do UC001 é derivada a classe Cliente. 27) A partir do UC001 é derivada a classe Filme. 28) A partir do UC002 é derivada a classe

19) Foi criada uma relação de rastreabilidade entre UC001 e a classe Emprestimo. 20) Foi criada uma relação de rastreabilidade entre UC001 e a classe Cliente. 21) Foi criada uma relação de rastreabilidade entre UC001 e a classe Filme. 22) Foi criada uma relação de rastreabilidade entre UC002 e a classe Devolucao. 23) Foi criada uma relação de rastreabilidade entre UC003 e a classe Cliente. 24) Foi criada uma relação de rastreabilidade entre UC004 e a classe Filme.	Devolucao. 29) Foi criada uma relação de rastreabilidade entre UC003 e a classe Cliente. 30) Foi criada uma relação de rastreabilidade entre UC004 e a classe Filme.

A técnica convencional implica em inserir os elementos nos diagramas e somente após a criação desses elementos criar as relações de rastreabilidade entre eles.

No total foram executadas 20 operações com a técnica convencional e 12 operações com a técnica indutiva, para o desenvolvimento dos artefatos de acordo com a tarefa descrita, com a respectiva criação das relações de rastreabilidade apresentadas na Figura 1. Para este estudo de caso a técnica indutiva apresentou um desempenho significativamente melhor, considerando o número de operações da técnica convencional e da técnica indutiva. A Tabela 3 faz uma comparação mais direta entre as duas técnicas, mostrando, para cada tipo de operação as vantagens e desvantagens destas.

Tabela 3 - Comparação entre as operações na técnica convencional e indutiva.

Operação indutiva	Operação convencional equivalente	Comparação
Criação de elemento-base	Criação de elemento novo	As técnicas são equivalentes.
Divisão	Criação de elemento novo + Transformação interna de elemento + Criação de relações de rastreabilidade	A técnica indutiva utiliza menos operações. Ambas as técnicas necessitam que as relações de rastreabilidade criadas sejam revisadas.
Derivação	Criação de elemento novo + Criação de relação de rastreabilidade	A técnica indutiva utiliza menos operações.
Transformação Interna	Transformação de elemento	As técnicas são equivalentes.
Criação de relação de rastreabilidade	Criação de relação de rastreabilidade	As técnicas são equivalentes.
Destruição de relação de rastreabilidade	Destruição de relação de rastreabilidade	As técnicas são equivalentes.
Junção	Destruição de elemento + Modificação de Elemento	A técnica indutiva utiliza menos operações.
Destruição	Destruição de elemento	As técnicas são equivalentes.

Portanto, conforme visto na Tabela 3, em cinco situações as técnicas são equivalentes, mas em três casos: divisão, derivação e junção a técnica indutiva utiliza menos operações. Não há situações onde a técnica indutiva tenha desempenho inferior, no que se refere ao número de operações que devem ser executadas pelo desenvolvedor.

Considerando-se que a criação de elementos base (requisitos) tende a ocorrer com menos frequência do que a derivação de outros elementos durante o processo de desenvolvimento de software, já que artefatos são gerados uns a partir dos outros ao longo da maioria dos processos existentes, pode-se inferir que a técnica indutiva deve reduzir significativamente o número de operações que devem ser realizadas pelo desenvolvedor caso este deseje manter as relações de rastreabilidade entre os artefatos criados. Este fato pode ser observado na Tabela 2, onde das doze operações executadas para a técnica indutiva dez são de derivação.

APÊNDICES

APÊNDICE A

ARTIGO

Plug-in para criação semi-automática de rastreabilidade entre artefatos de software no ambiente Enterprise Architect, sob a ótica da Rastreabilidade Indutiva

Trabalho de Conclusão de Curso

Diogo Dantas Fonseca dos Santos

Departamento de Informática e Estatística – Graduação em Ciência da Computação –
Universidade Federal de Santa Catarina (UFSC) – Cx. P. 476 –
Florianópolis, SC – Brasil

{diogu}@inf.ufsc.br

Abstract. *This work is based on the concepts of traceability inductive technique. This technique aims to promote traceability between elements and artifacts of modeling a process of creating software, so inductive. The work consists in implementing a plug-in can offer, the environment modeling Enterprise Architect, support semi-automatic traceability relationships between project artifacts.*

Resumo. Este trabalho está fundamentado nos conceitos da técnica de rastreabilidade indutiva. Essa técnica tem por objetivo favorecer a rastreabilidade entre elementos e artefatos de modelagem em um processo de criação de software, de modo indutivo. O trabalho consiste na implementação de um plug-in capaz de oferecer, no ambiente de modelagem Enterprise Architect, suporte semi-automático de relações de rastreabilidade entre artefatos de projeto.

1. Introdução

A qualidade nas atividades exercidas de gerenciamento das fases (comumente: projeto, planejamento e do desenvolvimento) de criação de um produto de software, e da modelagem das informações que compõe a especificação de um software contribui consideravelmente para o sucesso do mesmo. (SOMMERVILLE, 2000)

Empresas desenvolvedoras de software mantêm sempre uma preocupação positiva em diminuir as chances de fazer algo que não reflita a real necessidade do cliente, evitando o adiamento de prazos, aumento de custos e a insatisfação do cliente. Para tal, exige-se a habilidade, geralmente percebida em empresas com metodologias e processos bem definidos, de captar o que o cliente quer e transformar isso em uma solução na forma de software. Passando a agregar maior confiabilidade aos produtos além de reconhecimento mercadológico

Aspectos e funcionalidades levantadas inicialmente e desenvolvida durante fases posteriores de desenvolvimento representam decisões de planejamento e projeto, sendo primordial ter-las especificadas. Documentar e organizar essas informações faz parte dos objetivos da prática de modelagem, a qual faz uso conjunto de descrição

textual com o uso de ferramentas CASE de modelagem que dispõe de diversos elementos (previstos na UML) específicos de modelagem de aspectos de software.

Dado o caráter evolutivo do desenvolvimento de software, há um gradual crescimento já esperado de especificações e alterações das mesmas (LARMAN, 2007) sendo preciso uma estrutura que reaja de forma coesa tanto com relação a esse aumento quanto às operações mais comuns (criação, modificação, exclusão) envolvendo artefatos de modelagem de software. Garantir tal estrutura pode ser complexo em determinados projetos fazendo com que seja inviável utilizar a rastreabilidade a partir de certo do desenvolvimento.

Para contribuir na ideal estrutura da organização descrita, verifica-se no conceito de rastreabilidade uma solução viável, a ser aplicada durante o desenvolvimento de um software, quanto ao aspecto de "registrar" e representar, a cada operação, possíveis relações entre elementos de modelagem, tornando-as também flexível a futuras alterações. Assim, sob a ótica da Rastreabilidade Indutiva é possível a viabilidade de um modelo de rastreabilidade, implementado nesse trabalho em um plug-in que permite a criação semi-automática de relações entre elementos dependentes em algum aspecto de uma abstração modelada. A tarefa de registrar essas relações, às vezes deixada de lado por alguns desenvolvedores, também pode ser feita após a modelagem (fator de diferença da técnica RI frente às outras). Ressaltando (SANTOS; WAZLAWICK, 2009):

A intenção da técnica indutiva é reduzir o esforço do desenvolvedor no mapeamento da rastreabilidade, tornando a atividade menos penosa em comparação às técnicas atualmente utilizadas. Com isto espera-se que a resistência à utilização da rastreabilidade possa diminuir.

2. Modelagem de Software

A prática de modelagem busca favorecer a captura e compreensão de requisitos e funcionalidades, descrever aspectos dinâmicos e estáticos de um software, documentar o desenvolvimento e melhorar a comunicação à fase de programação de código sobre o que deve ser desenvolvido. O completo entendimento de requisitos contribui para que o desenvolvimento seja bem sucedido (PRESSMAN, 1995), (SOMMERVILLE, 2000). "Construímos modelos de sistemas complexos porque não é possível compreendê-los em sua totalidade" (RUMBAUGH; JACOBSON; BOOCH, 1999, p. 7).

Os resultados dessa prática também podem ser usados como roteiro de revisão, verificação da consistência e precisão do que foi especificado. Permitir que erros sejam visualizados antes da fase de codificação do produto, comunicar decisões sem ambigüidade de interpretação. A modelagem de software é representada por notação gráfica ou textual e pode também ser direcionada por modelos que ajudam a abordar aspectos importantes de um software em construção (requisitos, conceitual, caso de uso, funcional.

Os elementos de modelagem podem estar relacionados a outros seja para especificar um aspecto desejado para o software ou para que se identifique em um elemento, a causa para a criação de outro. Constantes operações, comuns em todo

processo de desenvolvimento podem afetar diretamente tanto o artefato que esteja sendo trabalhado como indiretamente a outros artefatos que tenham alguma relação com este. Levando em conta os aspectos levantados, há então a necessidade de algo que apóie as operações práticas da modelagem e paralelamente contribua no gerenciamento do conteúdo resultante das atividades que se referem ao planejamento e desenvolvimento.

3. Rastreabilidade

Por se um trabalho de entendimento da técnica de rastreabilidade e posterior implementação de suas premissas e conceitos, essa seção e a próxima reproduzem definições descritas na obra de (SANTOS; WAZLAWICK, 2009) para que qualquer interpretação possa ser feita com base no que foi originalmente definido. Assim fica definido:

Um elemento e é uma unidade de informação que compõe um artefato. O universo de todos os elementos possíveis é denotado por E. Exemplos de elementos: um caso de uso, uma classe, um requisito, um protótipo de tela.

Um artefato a é definido como sendo um conjunto de elementos de E. Um sistema de software pode então ser modelado por um conjunto de artefatos A = {a1, a2, ... an} cada qual contendo um conjunto de elementos, ou seja, A = { {e1,1, e1,2, ...}, {e2,1, e2,2, ...}, ... {en,1, en,2, ...} }. As eventuais associações entre elementos de um artefato (composição, generalização, associação simples, etc.) são também consideradas elementos dos artefatos. Faz-se exceção apenas às relações de rastreabilidade, definidas a seguir, que são consideradas externas aos artefatos, não sendo, portanto, elementos destes.

A relação de rastreabilidade R □E □E é uma relação acíclica e transitiva que estabelece relações entre elementos de artefatos. A relação de rastreabilidade se dá entre os elementos: mesmo que um elemento esteja presente em um ou mais artefatos, suas relações permanecem as mesmas.

3.1 Rastreabilidade Indutiva

> [...] é possível automatizar a criação e manutenção de ligações de rastreabilidade sob a hipótese de que a inserção de novos elementos nos artefatos não consiste simplesmente em criar um novo elemento no diagrama, mas em uma ação que em muitos casos tem uma causa bem definida a partir de algum outro elemento. Por exemplo, uma classe pode estar sendo inserida no modelo conceitual devido à existência de um caso de uso que a menciona. Ou ainda, um caso de uso pode estar sendo inserido no diagrama de casos de uso em função de um ou mais requisitos que lhe dão origem.
>
> [...] as operações de inserção de elementos nos artefatos sejam indutivas. Ou seja, com exceção dos elementos iniciais (base) a inserção de um elemento em um artefato deverá ocorrer sempre a partir de outro elemento, o qual consiste em sua causa. Por exemplo, os elementos iniciais são aqueles que surgem de fontes externas, como os requisitos. Os demais elementos como classes e casos de uso seriam criados em função de elementos já existentes (ou seja, por indução).
>
> [...] um conjunto de operações para criação e evolução de artefatos do ponto de vista da área que pode ser definida como Rastreabilidade Indutiva, mostrando

que é possível conceber um processo de desenvolvimento e evolução de software baseado nestas premissas.

4. Plug-in Implementado

Encontrar uma ferramenta CASE que dispunha todos elementos previstos na UML acabou por determinar a escolha pelo ambiente de modelagem Enterprise Architect, o qual mais importante ainda, oferece disponibilidade de desenvolvimento de plug-ins, desenvolvidos por terceiros, que executam conjuntamente a execução da ferramenta principal. O EA identifica esse tipo de aplicação externa como um“add-in”

O plug-in ou add-in que rode dentro do ambiente EA pode controlar desde algumas funcionalidades originais do ambiente bem como manipular todos elementos e artefatos de modelagem dispníveis. Dada ainda à acessibilidade a documentação (apêndice A) do EA no que se refere ao desenvolvimento de aplicações add-in ser satisfatóriamente completa a escolha por essa ferramenta oferece as condições suficientes para utilização da técnica aplicar-se um modelo de rastreabilidade indutiva.O plug-in foi desenvolvido como um componente ActiveX (COM) e a linguagem de programação usada foi o Visual Basic 6.

Assim o próximo passo foi a implementação da descrição, citada na seção 3.1, de possíveis métodos que satisfizesse a técnica de Rastreabilidade Indutiva. Resultando em 4 operações que têm a seguinte descrição de uso: derivar requisito / derivar caso de uso/ derivar classe/ associar elementos. Um outro método indentificado pela técnica seria a inserção de elementos um elemento base, mas seria redundante implementá-lo uma vez que essa é uma função original do EA, representada na ação de drag and drop de um elemento para um artefato.

Vale ressaltar que a RI não estabelece que sempre determinado tipo de elemento seja considerado o elemento base (requisitos por exemplo, mesmo sendo útil em um desenvolvimento de software usar essa noçao para ajudar na abstração de funcionalidades para o mesmo). Assim, o plug-in suporta essa característica. Outra consideração a ser feita é a dinâmica da operação de derivação da RI, quando se deseja que o elemento derivado fique disposto em um artefato diferente do qual está dispondo o elemento causa selecionado então é necessário selecionar tal artefato destino previamente a derivação.

Para exemplificar o funcionamento do plug-in será assumido um contexto de uso: Será o de um controle para uma vídeo locadora, na qual após um planejamento passou-se a modelagem de uma funcionalidade, empréstimo de filmes, preterida para a aplicação. Assim, essa funcionalidade foi abstraída como sendo um requisito base chamado de “O sistema deve realizar empréstimo”, esse elemento é adicionado por uma ação drag and drop do mesmo para um artefato diagrama “Diagrama de requisitos” contido em um pacote “Requisitos”.

O próximo passo após outras abstrações e decisões de projeto feitas no sentido de compreender esse requisito em passos necessários para satisfazê-lo, é naturalmente derivar a partir de um elemento causa (requisito) um elemento derivado (caso de uso) que atenda tal propósito. Então o que deve ser feito é selecionar um artefato destino (figura 1), já que queremos ter organizado todos os casos de uso da aplicação, em um artefato digrama “Diagrama de Caso de Uso” pertecente ao pacote “Casos de Uso” e

depois selecionar com o botão direito do mouse o elemento requisito e a opção Add-Ins / Rastreabilidade Add-in / [> Derivar Caso de Uso Alt + A + R + U <] (figura 2).

Agora um caso de uso “[D] O sistema deve realizar Empréstimo” estará disposto no artefato destino especificado. Tendo a partir de agora um elemento que abstrai a especificação dos passos para atender o caso de uso derivado, o próximo passo será usar a abstração e a noção de classe para dar origem posteriormente a classes com métodos e atributos. Seguindo o mesmo raciocínio queremos agora derivar uma classe “[D] O sistema deve realizar empréstimo” para um pacote “Classes” que contém o artefato destino diagrama “Diagrama de Classe” sendo esse previamente selecionado (figura 3) antes de selecionar com o botão direito do mouse o elemento causa (caso de uso) e a opção Add-Ins / Rastreabilidade Add-in / [> Derivar Classe Alt + A + R + C <] (figura 4).

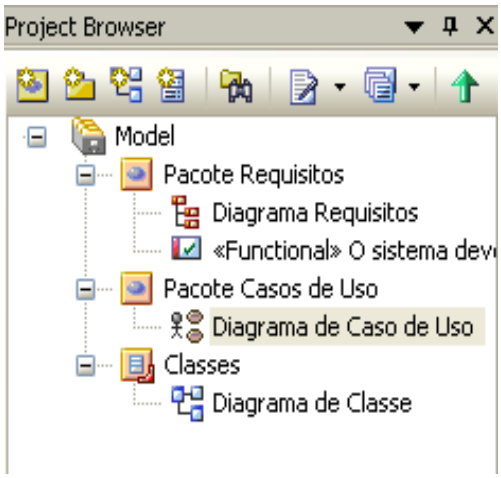


Figura 1: Derivação (escolha destino)

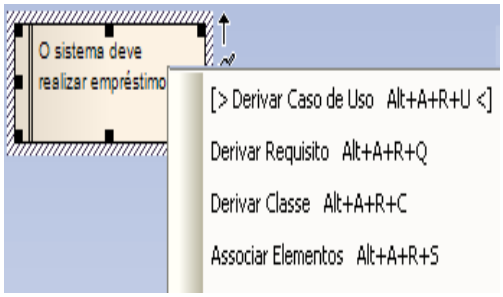


Figura 2: Derivação (elemento causa)

Caso em algum momento da modelagem fosse necessário associar um elemento já previamente modelado a outro também já existente, pois caso não existisse poderia ser usada a operação de derivação, deve-se selecionar previamente o elemento efeito (figura 5) e com botão direito do mouse selecionar o elemento causa e escolher a opção Add-Ins / Rastreabilidade Add-in / [> Associar Elementos Alt + A + R + S <] (figura 6).

A visualização de todas as relações entre elementos pode ser realizada apenas juntando todos os elementos de modelagem criados em um artefato diagrama qualquer.

No momento em que os elementos são arrastados para esse diagrama é possível ver os relacionamentos entre elementos causa e efeito (figura 7). Essa visualização contribui eventualmente para análises que precisem ser realizadas frente a mudanças que possam envolver elementos que se relacionem a outros.

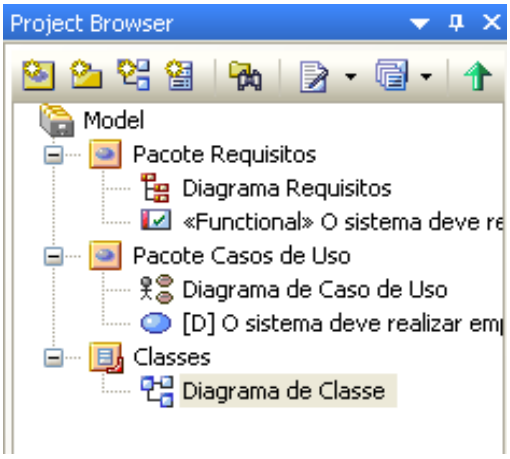


Figura 3: Derivação (escolha destino)

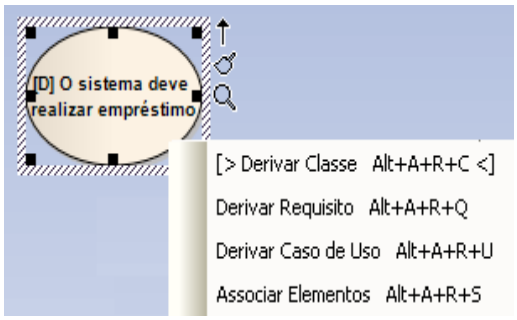


Figura 4: Derivação (elemento causa)

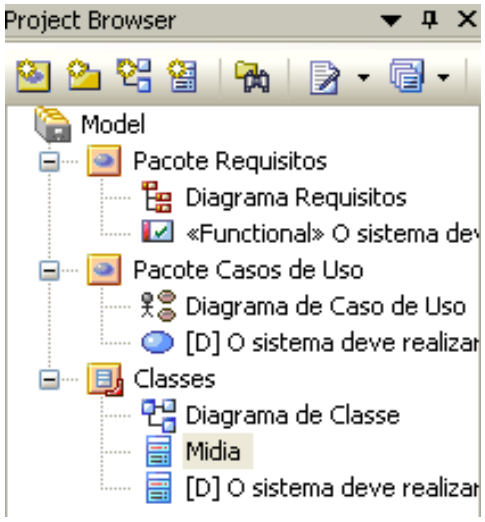


Figura 5: Escolha elemento

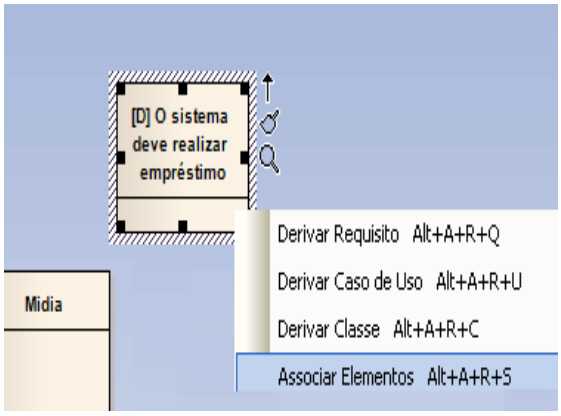


Figura 6: Associar elementos

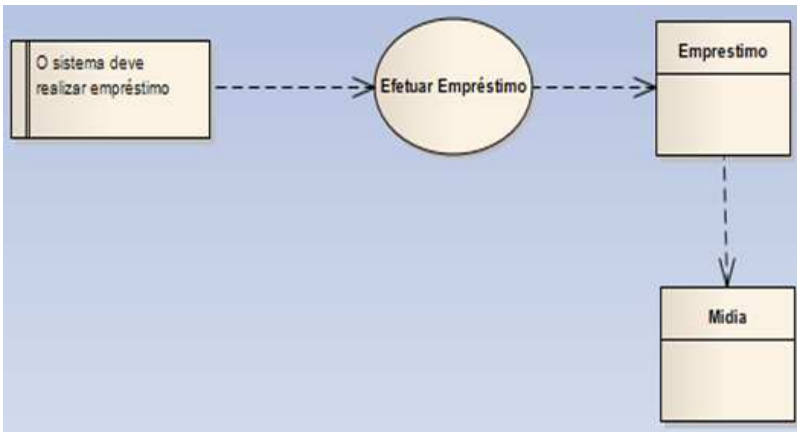


Figura 7: Rastreabilidade entre elementos

5. Conclusão

O plug-in que suporta a criação semi-automática de relações de rastreabilidade em experimento prático realizado e relatado no artigo que também apresenta a técnica de RI (SANTOS; WAZLAWICK, 2009) obteve a melhor relação no número de passos exigidos para tornar os elementos rastreáveis em comparação ao número de passos exigidos em outras técnicas que tratam da rastreabilidade.

O exemplo mostrado no presente artigo já dá sinais dessa constatação, uma vez que usando uma técnica que faça o registro das relações a posteriori, nesta seria executado o mesmo número de passos para criação de cada elemento mais uma quantidade de passos necessários para tratar as relações de rastreabilidade. Enquanto, na técnica de rastreabilidade indutiva, as relações seriam registradas à medida que elementos (que se relacionassem a outros) fossem criados usando operação de derivação de elementos.

Entretanto, como relatam os mesmos autores, há a seguinte limitação: “A técnica indutiva proposta é eficaz apenas em sistemas cuja documentação ainda vai ser produzida, onde se tem a oportunidade de utilizar a técnica desde o início”. Mesmo assim, empresas e desenvolvedores de software que identifiquem importância e

benefícios na modelagem de software e na rastreabilidade têm uma opção que suporte esses aspectos.

7. Referências Bibliográficas

CRUZ, J. L. e JINO, M. e Crespo, A. N. e ARGOLLO, M. Suporte automatizado à rastreabilidade em um processo de teste de software baseado em documentação, em: V Simpósio Brasileiro de Qualidade de Software – SBQS´2006.

LARMAN, C. Utilizando UML e padrões: uma introdução à análise e ao projeto orientados a objetos e ao desenvolvimento iterativo. 3. ed. Porto Alegre: Bookman, 2007.

PRESSMAN, Roger S. Engenharia de Software. 3. ed. São Paulo: Ed Makron Books, 1995.

RUMBAUGH, James; JACOBSON, Ivar; BOOCH, Grady. The unified modeling language reference manual. Boston: Addison-Wesley, 1999.

SANTOS, R. N. e WAZLAWICK, R. S. Rastreabilidade Indutiva Aplicada a Artefatos de Software e Relações de Rastreabilidade, em: EXPERIMENTAL SOFTWARE ENGINEERING LATIN AMERICAN WORKSHOP, VI, 2009, São Carlos: DC/UFSCAR, 2009.

SOMMERVILLE, Ian. Software Engineering. 5.ed. : Addison Wesley, 2000.
WAZLAWICK, R. S. Análise e Projeto de Sistemas de Informação Orientados a Objetos. 2.ed. : Campus, 2004