Modelos e Meta-Modelos para Transações em Aplicações Web[1]

Mark Douglas Jacyntho[2]
Departamento de Informática, PUC-Rio
Av. Marquês de São Vicente, 225
Rio de Janeiro - RJ, Brasil
55 21 3527-1500
mjacyntho@inf.puc-rio.br

Daniel Schwabe
Departamento de Informática, PUC-Rio
Av. Marquês de São Vicente, 225
Rio de Janeiro - RJ, Brasil
55 21 3527-1500
dschwabe@inf.puc-rio.br

ABSTRACT
In this paper, we present a Domain Specific Language (DSL) to specify business and Web transactions in a systematic way, addressing both structural and behavioral perspectives. Our meta-model is based on the reification of transactions, where transactions are modeled as first-class types, supporting attributes, associations, operations and state machines. Treating transactions as domain type instances facilitates the interplay with other models, such as the navigational model, which is a view of the domain type model.

RESUMO
Neste artigo, apresentamos uma Linguagem Específica de Domínio (DSL) para especificar transações de forma sistemática, abordando ambas as perspectivas: estrutural e comportamental. Nosso meta-modelo baseia-se na reificação de transações, onde transações são modeladas como tipos de primeira ordem, que possuem atributos, associações, operações e máquinas de estado. Tratar transações como instâncias de tipos de domínio facilita a interação com outros modelos, como o modelo navegacional, que consiste em uma visão do modelo de tipos de domínio.

Categories and Subject Descriptors
D.2.2 [**Software Engineering**]: Design Tools and Techniques – *object-oriented design methods, petri nets.*

D.2.11 [**Software Engineering**]: Software Architectures – *domain-specific architectures.*

H.3.5 [**Information Storage and Retrieval**]: Online Information Services – *Web-based services.*

General Terms
Documentation, Design, Reliability, Human Factors.

Keywords
Business Process, Workflow, Business Transaction, Workflow Transaction, Web Transaction, Conceptual Modeling, Process Modeling, Transaction Modeling, Meta-model, Ontology, Semantic Web, DSL, MDE, MDWE, MDD, MDA.

1. INTRODUÇÃO
Hoje em dia, aplicações Web são desenvolvidas para realizar tarefas que são parte de um fluxo de trabalho (*workflow*) de transação de negócio bem definido, respeitando suas regras e restrições, servindo diferentes usuários, cujo trabalho conjunto precisa ser coordenado. Questões como distribuição e concorrência devem ser consideradas, e conflitos resolvidos. Por exemplo, em um site de comércio eletrônico, duas sessões concorrentes podem estar simultaneamente emitindo um pedido contendo a última unidade do mesmo produto, e a última sessão a enviar a confirmação não terá sucesso na aquisição do produto (que estava disponível há apenas uma fração de tempo).

De acordo com [7], uma transação de negócio envolve um ou mais sistemas, e, para cada sistema, uma ou mais transações de sistema são disparadas. No caso de aplicações da Web, estas transações de sistema pode ser chamadas de transações Web. Baseado em [3], uma transação de Web pode ser definida como "uma seqüência de atividades realizadas por um usuário, para alcançar uma meta específica relacionada com uma transação de negócio, por meio de uma aplicação Web".

Relacionando esses conceitos, podemos afirmar que uma transação de negócio é composta de uma ou mais transações Web, que por sua vez, é composta de uma ou mais chamadas de operação em objetos de domínio. Uma transação Web é uma interação completa, com limites bem definidos, que se estende por várias requisições. Quando se está lendo ou escrevendo dados em um banco de dados (ou outro tipo de camada de persistência), a requisição é processada no âmbito de uma transação *on-line* de banco de dados. A última requisição da transação Web é responsável pela efetivação (*commitment*) da transação Web,

[1] Models and Meta Models for Transactions in Web Applications

[2] Universidade Cândido Mendes, Núcleo de Pesquisa e Desenvolvimento, Rua Anita Pessanha 100, Campos dos Goytacazes, RJ, Brasil
Instituto Federal de Educação, Ciência e Tecnologia Fluminense, Av. Dário Vieira Borges, 235, Bom Jesus do Itabapoana, RJ, Brasil

gravando as alterações e resolvendo conflitos de concorrência (*off-line*) entre sessões de usuário.

Outra ponto relevante, porém pouco discutido, é como combinar transação e navegação. Estritamente falando, navegação é um comportamento como qualquer outro. A única diferença é que sua semântica é conhecida a priori, tal como previsto pelos modelos de navegação hiper-textual de nó e link padrão, exemplificados pela Web. Por isso, "modelos de navegação" especializados foram propostos para simplificar a especificação desta parte do comportamento da aplicação. Tipicamente, navegação é implementada como transação aninhada para selecionar itens a serem processados por uma transação irmã subsequente. Por exemplo, a transação Web "Criação de Pedido" tem duas transações aninhadas: "Seleção de Itens" e "*Checkout*". Sob este ponto de vista, navegação é um tipo especial de transação Web.

A falta de um modelo faz com que transações sejam frequentemente tratadas ad-hoc, como simples seqüência de requisições e respostas, com toda a lógica espalhada no código e não modelada explicitamente. A consequência é comportamento errôneo, com violação de restrições importantes que não são consideradas.

Este documento apresenta uma DSL (definida por um meta-modelo e uma notação gráfica) para atender a modelagem explícita de transações de negócio e Web, reificando o conceito de transação em um tipo de primeira ordem no meta-modelo. Instâncias do tipo transação correspondem às ocorrências da transação e armazenam todo o estado desta ocorrência. Em tempo de execução, instâncias de transação são criadas através de um controlador. É importante dizer que o foco são transações Web, não transações da camada de persistência (por exemplo, bancos de dados). É assumido a existência de uma camada de persistência transacional.

Este artigo está estruturado da seguinte forma: na secção 2, o meta-modelo da DSL é apresentado. Em seguida, na seção 3, um exemplo e algumas observações sobre os benefícios de usar a DSL. A seção 4 cita alguns trabalhos relacionados e, finalmente, a seção 5 conclui este documento com observações finais e trabalhos futuros.

2. META-MODELO

O meta-modelo é baseado na noção de reificação, promovendo o conceito de transação a um tipo primeira ordem, responsável por manter o estado da execução (parâmetros, variáveis locais, status de execução, etc.), controlando todos os passos da execução. Transações reificadas podem ser usadas como qualquer outro tipo de domínio e também podem ser persistidas.

Outra característica significativa deste meta-modelo é a clara separação, em meta-classes distintas, da especificação da transação ("o quê") de seus possíveis *enactments* ("o como"). Especificação aqui significa um contrato definido por pré/pós-condições e invariantes de domínio. *Enactment* significa um ou mais protocolos de execução que satisfaçam o contrato, atingindo as mesmas pós-condições. Cada *enactment* é descrito por um modelo de fluxo, onde a transação é decomposta em transações cada vez mais simples até atingir o menor nível de abstração, ou seja, chamadas de operação em objetos de domínio. Observe que para reusar uma transação no modelo de fluxo de um *enactment* de uma transação mãe, precisamos apenas conhecer a especificação (contrato) da transação invocada, pois, por definição, todos os *enactments* satisfazem a especificação.

O meta-modelo tem duas perspectivas ortogonais. A primeira é a visão estrutural ou estática abrangendo as propriedades (atributos e associações) da transação e a segunda é a visão comportamental ou de fluxo, onde o foco é o modelo de fluxo de execução de um possível *enactment* da transação que satisfaça sua especificação.

Ao longo da explanação, os conceitos serão clarificados usando trechos de um estudo de caso sobre uma transação de negócio de análise de artefatos (*artigos, workshops e palestras*) submetidos para uma edição de uma conferência, incluindo: submissão, revisão e seleção (aceitação/rejeição) de artefatos.

2.1 Perspectiva Estrutural

A perspectiva estrutural está voltada para as informações necessárias - parâmetros e variáveis locais utilizados para realizar a transação - e objetos manipulados durante o curso da transação. A Figura 1 mostra a parte estrutural do meta-modelo por meio de um diagrama de classes UML.

O conceito central deste meta-modelo é *TransactionType*, que modela a especificação de uma transação. Existem dois tipos de transação: negócio e Web.

Para especificar uma transação, temos que definir seu contrato, incluindo *Parameters*, *Exceptions* e pre/post-conditions. *Parameter* é usado para modelar os parâmetros de entrada e saída da transação. O atributo *isStreaming* indica que se trata de um parâmetro de fluxo contínuo ou não. Parâmetros de fluxo contínuo podem aceitar ou fornecer valores enquanto a transação está em execução. Seguindo a abordagem de "*design by contract*", *post-condition* é um predicado em lógica de primeira ordem (expressão booleana) que especifica o efeito da transação que tem que ser garantido por qualquer *enactment*.

Às vezes, a pós-condição faz sentido apenas sob determinadas condições iniciais, chamadas de pré-condições (*pre-conditions*) que também são predicados booleanos. Se a pré-condição não for verdadeira ao iniciar uma transação, a especificação não se aplica, portanto, o efeito é não especificado, e a pós-condição não é garantida. Invariantes também podem ser definidas, sendo automaticamente adicionadas (conector lógico AND) às pré/pós-condições de todas as transações.

Finalmente, os resultados excepcionais de uma transação podem ser especificados usando a construção *Exception*. Uma exceção pode ser considerada um tipo especial de parâmetro de retorno. Para cada exceção pode haver uma transação que desempenha o papel de tratador da exceção.

Para ilustrar estas primitivas, a transação Web usada para submeter o artefato para a edição da conferência - "*Artifact Submission Transaction*" - tem três parâmetros de entrada (*authors*, *first author*, *conference edition*), um parâmetro de saída (*new artifact*), uma exceção (*artifact wrong format exception*). A pré-condição é que a data corrente seja anterior a data final de submissão da edição e a pós-condição é que uma nova instância de artefato seja criada, com status *submitted*.

A primitiva *Event* é usada para modelar eventos que a aplicação envia para ou recebe de entidades externas (atores). Um evento pode ser comparado a um *trigger* que inicia uma transação. Por exemplo, a retirada de um artefato anteriormente submetido. Nesta caso, um evento de entrada "*Withdrawal Event*" inicia a

transação Web "*Artifact Withdrawal Transaction*", suspendendo qualquer outra transação que esteja em curso ("*Reviewers Assignment*", "*Obtaining Artifact to Review*", "*Artifact Review Submission*").

O tipo transação pode ter subtipos, que herdam todas as especificações da transação supertipo. Uma vez que é uma herança de especificação, redefinição (*overriding*) não é permitido, apenas extensão. Um subtipo de transação tem que satisfazer o *Liskov Substitution Principle*, de forma que uma instância do subtipo possa ser apropriadamente utilizada em qualquer contexto onde se conheça apenas a especificação do supertipo, e nada sobre os possíveis subtipos. Portanto, a pós-condição de um subtipo somente pode estender a pós-condição do supertipo, não redefini-la. Além disso, a pré-condição do subtipo não pode estender a pré-condição do supertipo. A pré-condição do subtipo pode ser mais fraca, mas não pode ser mais forte do que a pré-condição do supertipo. No estudo de caso, para cada subtipo de *Artifact* (*Paper, Workshop, Talk*), existe uma transação Web correspondente que é um subtipo de "*Artifact Submission Transaction*".

Finalmente, uma transação pode ter uma ou mais transações alternativas compensatórias. Uma transação compensatória é uma forma de "reverter" os efeitos de uma transação confirmada no caso de um aborto futuro. É diferente de *rollback* porque o efeito não é desfeito, porém substituído por outro efeito que compensa o primeiro.

Uma vez que a transação é especificada, a próxima etapa é modelar pelo menos uma possível realização (*enactment*) de como alcançar a pós-condição. Isto é feito instanciando-se a meta-classe *TransactionEnactmentFlow*. Estaticamente falando, um *enactment* define as propriedades ACID (Atomicidade, Consistência, Isolamento, Durabilidade) e as transações aninhadas

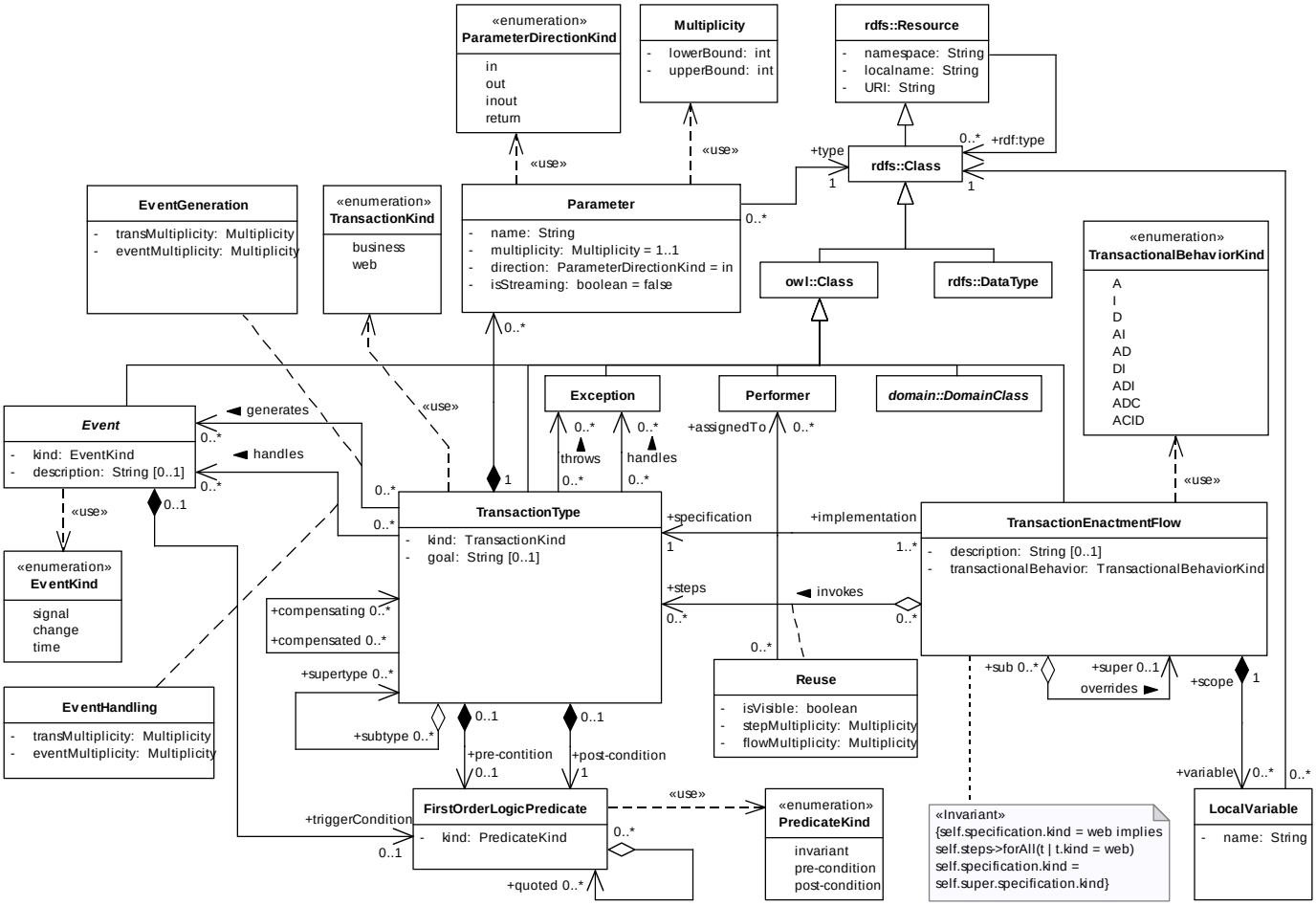

Figura 1. Perspectiva estrutural do meta-modelo.

(invocadas/reusadas) que definem os passos da transação em questão. É perfeitamente possível definir *enactments* ACID e não-ACID. Cada *enactment* define quais transações aninhadas são (re) usadas para realizar a transação mãe especificada, bem como que *perfomers* (atores) podem executá-las. Por exemplo, a transação de negócio de análise de artefato - "*Artifact Analysis Transaction*" - inclui submissão, revisão e seleção de artefatos. Portanto, um possível *enactment* desta transação de negócio - "*Artifact Analysis Enactment*" - possui, como passos, as seguintes transações Web

aninhadas: "*Artifact Submission*", "*Reviewers Assignment*", "*Obtaining Artifact to Review*", "*Artifact Review Submission*", "*Artifact Decision*", "*Artifact Final Version Submission*", "*Artifact Withdrawal*" e "*Wrong Format Exception Handling*". Estas duas últimas são, respectivamente, o tratador de evento (*event handler*) associado ao evento "*Withdrawal Event*", se autor decidir retirar seu artefato, e o tratador de exceção (*exception handler*) associado à exceção "*Artifact Wrong Format Exception*", que é a exceção gerada pela transação Web "*Artifact Submission*", caso o artefato submetido não esteja em conformidade com o formato requerido pela conferência.

Um ponto importante é a restrição que transações de Web só podem ser compostas de outras transações Web. Não faz sentido colocar uma transação de negócios dentro de uma transação Web de sistema. Outro ponto é indicar se o efeito de transação aninhada concluída é visível ou não fora do escopo da transação mãe.

Por fim, um *enactment* pode ser definido por meio de herança de outro *enactment*. Dado que é uma herança de "implementação", extensões e *overriding* são permitidos. No entanto, há uma restrição que um *enactment* só pode herdar de outro *enactment* do mesmo tipo de especificação (negócio ou Web). No estudo de caso, para cada subtipo de *Artifact* existe um *enactment* correspondente que herda de "*Artifact Analysis Enactment*", e acrescenta restrições extras, como por exemplo, o passo constituído pela transação Web "*Artifact Submission*", para *Paper* tem que ser o subtipo "*Paper Submission*", para *Workshop*, o subtipo "*Workshop Submission*" e para *Talk*, o subtipo "*Talk Submission*".

Para especificar a perspectiva dinâmica, *TransactionEnactmentFlow* permite modelar a lógica de fluxo dos passos da transação, descrevendo a sequência em que as transações aninhadas entram em cena. Este é o tema da próxima seção, onde é explicada a perspectiva comportamental do meta-modelo.

2.2 Perspectiva Comportamental

Esta visão define os possíveis fluxos de execução entre os passos (transações aninhadas ou chamadas de operações) de um *enactment* da transação. Esta parte do meta-modelo é uma versão especializada do meta-modelo do diagrama de atividades da UML 2.0, que é baseado em redes de Petri [5].

Basicamente, há nós conectados por arestas formando um grafo de fluxo completo, e *tokens* de controle e de objeto (dados) fluem ao longo das arestas e são processados pelos nós, roteados para outros nós, ou armazenados temporariamente. Um *token* de controle é um valor booleano que sinaliza que o nó de atividade destino não pode iniciar até que o nó de atividade fonte termine. A Figura 2 mostra a parte comportamental do meta-modelo. Para simplificar o diagrama, alguns relacionamentos já mostrados na Figura 1 foram omitidos.

A parte estrutural do meta-conceito *TransactionEnactmentFlow* modela somente a composição estática do *enactment.* Suas transações constituintes podem ocorrer em paralelo, em sequência, pode haver caminhos alternativos, repetições, etc. O papel do meta-modelo comportamental é complementar a definição de um *enactment* da transação, estabelecendo o fluxo de execução.

Existem seis tipos de nó: *TriggerTransactionNode* - representa um disparador (*trigger*) de transação. Pode haver também um *Performer* associado, que é o ator (papel) que executa esta transação; *CallOperatinNode* - representa uma chamada de operação; *ControlNode* - roteia controle e objetos (dados) através do grafo; *ObjectBufferNode* - armazena objetos (dados) e representa o uso de uma variável local como um buffer; *Exception Nodes* - geram (*throws*) ou capturam (*catch*) uma exceção; *Event Nodes* - enviam ou recebem um evento.

Além de nós e arestas, o meta-modelo prevê a noção de Escopo (*Scope*), que é um conjunto de nós agrupados para algum propósito. Estão previstos dois tipos de escopo: *SuspensionScope*: este escopo possui um ou mais *ReceiveEventNodes* associados, e agrupa nós de modo que se ocorrer um evento, qualquer transação dentro do escopo é suspensa. *ProtectedScope*: este escopo é similar ao mecanismo "*try/catch*" de algumas linguagens programação. Os nós dentro do escopo estão protegidos contra exceções lançadas dentro do escopo. Este escopo possui um ou mais *CatchExceptionNodes* associados e os nós dentro do escopo estão protegidos para cada exceção correspondente.

Dado este meta-modelo, percebe-se, claramente, que é possível definir um *enactment* que implementa uma *engine* padrão de navegação hipermídia, que mantém o estado da navegação da aplicação em qualquer ponto. Desta forma, a navegação se torna apenas mais uma transação da aplicação.

3. UM EXEMPLO

Dado um domínio de aplicação, existem características comuns presentes na maioria dos modelos que podem ser usadas para criar um modelo padrão reutilizável. Portanto, em geral, os projetistas não instanciam um meta-modelo a partir do zero, mas criam seus modelos estendendo modelos padrão pré-existentes (instâncias abstratas do meta-modelo).

Para assistir a instanciação do meta-modelo, existe uma instância base padrão do meta-modelo para ser estendida por um modelo particular. Este modelo abstrato padrão, não mostrado por razões de espaço, define a classe abstrata *Transaction*, da qual todas as classes que representam uma transação podem derivar. O modelo padrão também prevê a classe *Flow* (*thread*), cujas instâncias são os fluxos de execução da transação. Uma transação pode ter um ou mais fluxos de execuções ou *threads*. Dois ou mais fluxos são criados quando há um nó de controle *Fork* no modelo comportamental. Cada fluxo mantém os passos (instâncias de transações) executadas no seu escopo, incluindo seu passo corrente. Uma transação pode ter mais de um fluxo em execução, mas apenas um fluxo está sendo efetivamente executado (progredindo), por sessão de usuário.

As classes possuem algumas propriedades padrão e um atributo de status de execução. A classe *Flow* tem cinco status de ciclo de vida: *ready*, *running*, *blocked*, *finished* e *interrupted*. A classe *Transaction* tem atributo de duração e os seguintes status: *definingActualParameters*, *executing*, *committed*, *aborting*, *aborted*, *compensating*, *compensated*, e *terminatedByException*.

Como uma extensão deste modelo padrão, foi criado o modelo do estudo de caso sobre a transação de negócio de análise de artefatos submetidos para uma conferência. Neste exemplo, esta transação de negócio é chamada *ArtifactAnalysis* e um possível *enactment* engloba as seguintes transações Web como passos: *ArtifactSubmission* - um dos autores submete o artefato;

ReviewersAssignment - um *PC-chair* aloca os revisores para avaliar o artefato; *AcquiringArtifactToReview* - cada revisor obtém o artefato para avaliar; *ArtifactReviewSubmission* - cada revisor submete sua avaliação; *ArtifactDecision* - um *PC-chair* aceita ou rejeita o artefato; *ArtifactFinalVersionSubmission* - se o artefato for aceito, o autor submete a versão final; *ArtifactWithdrawal* - o autor desiste e retira o artefato.

Por falta de espaço, o modelo de domínio não será mostrado. O tipo principal de domínio é artefato (*Artifact*) e seus subtipos (*Paper*, *Workshop* e *Talk*). Entre outras propriedades, um artefato tem autores (*Author*) e um status (*submitted*, *waitingReviewers*, *inReview*, *reviewed*, *acceptedWaitingLastVersion*, *withoutLastVersion*, *accepted*, *rejected*, *withdrawn*). Cada artefato é associado (submetido) a uma edição da conferência (*Edition*).

O primeiro passo é modelar a visão estrutural das transações. A Figura 3[3] mostra a visão estrutural da transação de negócio *ArtifactAnalysis* e da transação Web *ArtifactSubmission* (omitindo suas pré/pós-condições, por falta de espaço) e seus subtipos. Para simplificar, foi modelado apenas um possível *enactment* (*ArtifactAnalysisEnactment*) da especificação de *ArtifactAnalysis* e foram omitidos os *enactments* ACID da transação Web *ArtifactSubmission* e seus subtipos. O comportamento ACID de *ArtifactAnalysisEnactment* tem apenas "D" (durabilidade) e, vale lembrar que este *enactment* tem que honrar a "nota" de pré/pós condições associada com a transação *ArtifactAnalysis*. O símbolo pontilhado de subtipo entre *ArtifactAnalysisEnactment* e *ArtifactAnalysis* representa realização (uma possível implementação) da transação. Trata-se de classificação de tipo do

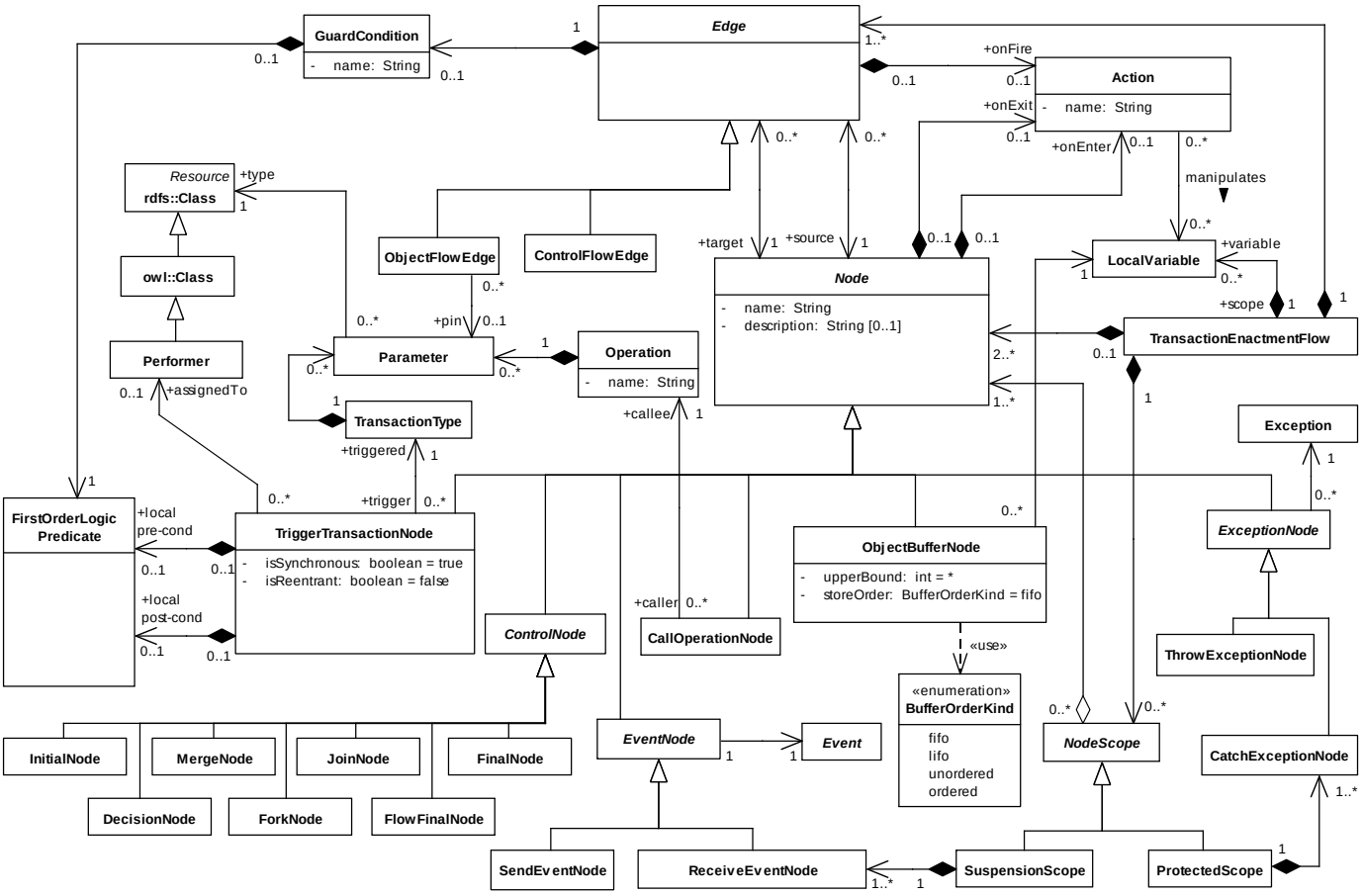

Figura 2. Perspectiva comportamental do meta-modelo.

Cada artefato passa por três revisões (*Review*), cada uma feita por um revisor (*Reviewer*). Finalmente, o artefato é aceito ou rejeitado por um presidente (*PC-Chair*) da edição. Caso seja aceito, uma versão final deve ser submetida, caso contrário o artefato não é considerado aceito.

[3] Notação da DSL: oval dupla - transação de negócio; oval simples - transação Web; retângulo arredondado - *enactment*; seta cima/baixo - exceção, retângulo - objeto de domínio; seta losango - passo; linha sólida/triângulo - subtipo/*overriding*; linha tracejada/triângulo - realização de uma especificação por um enactment.

enactment, não uma herança, ou seja, todas as instâncias de um enactment são instâncias da correspondente transação (tipo). Perceba que *ArtifactAnalysis* e *ArtifactSubmission* são subclasses da classe *Transaction* definida no modelo abstrato padrão.

O *ArtifactAnalysisEnactment* é composto (símbolo losango de composição) por todas as transações Web mencionadas anteriormente (cujos respectivos *enactments* não foram mostrados). Um aspecto interessante é o *overriding* de *ArtifactAnalysisEnactment* para cada subtipo de *Artifact*. De forma similar, a relação com *Artifact*, por meio do parâmetro de saída *artifactResult*, também é refinada para cada subtipo de *ArtifatctSubmission*. Outro ponto notável é o refinamento de nível de abstração de *ArtifactAnalysis* para *ArtifactSubmission*. No nível de abstração de *ArtifactAnalysis*, existem apenas dois parâmetros (*artifactResult* e *edition*), mas no nível de *ArtifactSubmission* (e seus subtipos), há dois parâmetros extras (*authors* e *firstAuthor*). Se o artefato for submetido no formato incorreto, a transação Web *ArtifactSubmission* gera a exceção *ArtifatWrongFormatException* que é tratada pela transação Web *WrongFormatExceptionHandling*, onde o artefato é rejeitado.

Supondo que esta conferência possua vários *PC-chairs*, temos que evitar conflitos de concorrência quando um *PC-chair* aloca revisores para um artefato (transação Web *ReviewersAssignment*). Esse problema é elegantemente resolvido, especificando que o

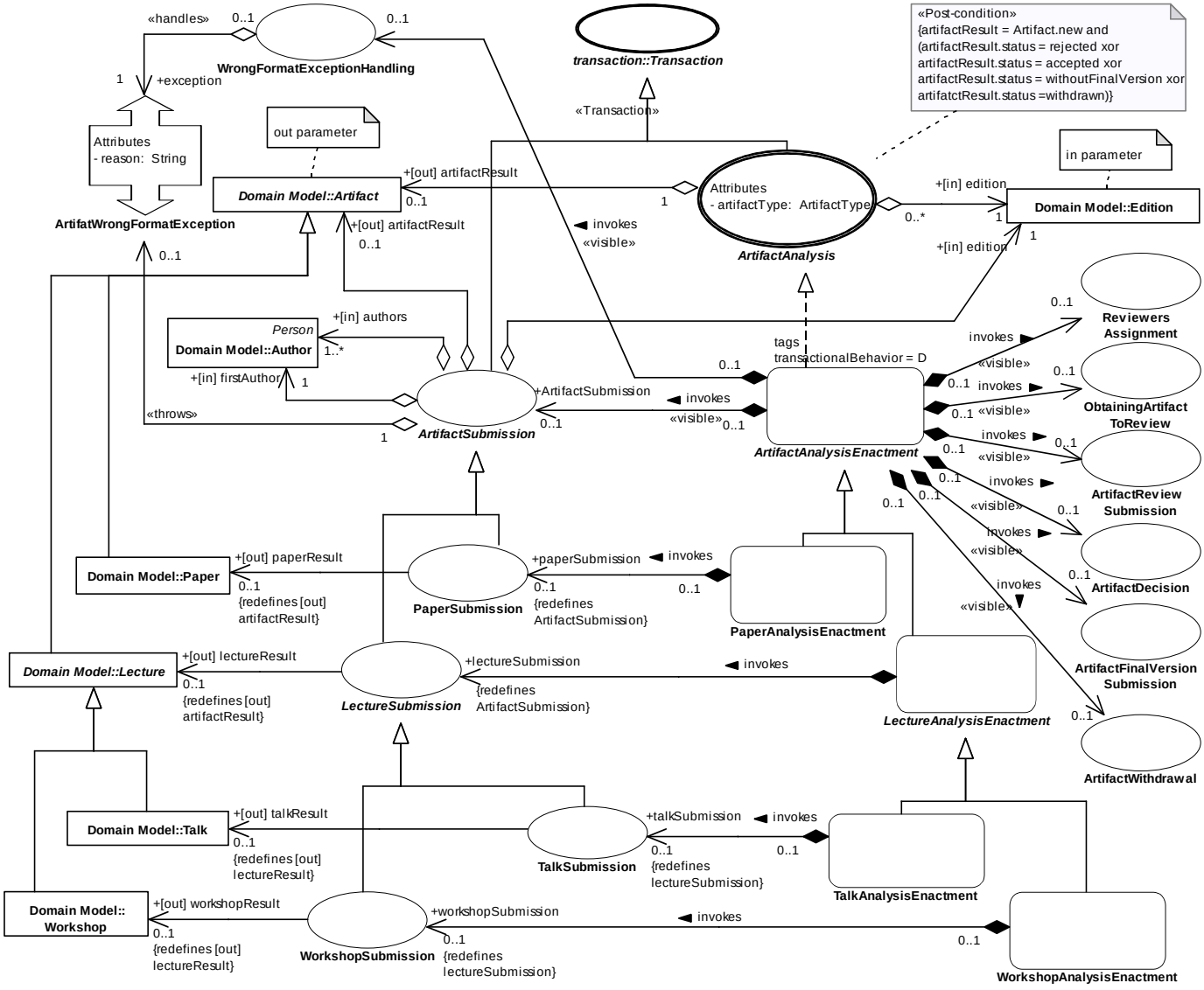

Figura 3. Modelo estrutural da transação de negócio "*Artifact Analysis*".

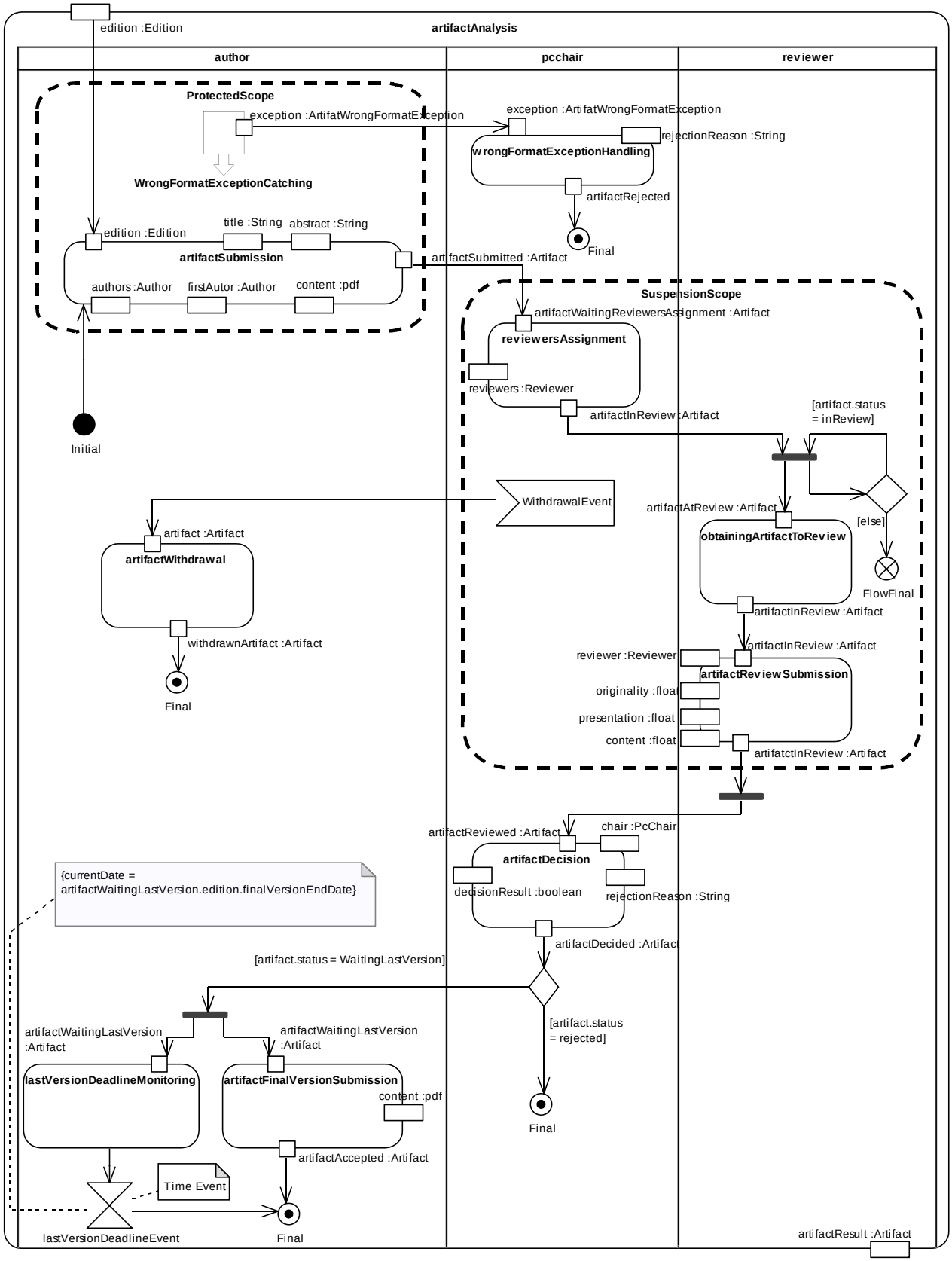

Figura 4. Modelo comportamental da transação de negócio "*Artifact Analysis*".

comportamento ACID do respectivo *enactment* tem, pelo menos, "I", significando que esta transação Web deve ser isolada. Isto implica algum mecanismo de implementação de bloqueio (*lock*), optimista ou pessimista, na aplicação. Teoricamente, "I" significa *serializable*, mas, na prática, isso pode ser flexibilizado para aumentar a concorrência e pode-se especificar o nível de

isolamento: *read uncommitted*, *read committed*, *repeatable read* e *serializable*. Neste caso específico, por exemplo, poderíamos usar *repeatable read*, indicando que esta transação Web terá êxito somente se nenhuma outra sessão concorrente atualizar o artefato.

Para concluir a modelagem, a Figura 4[4] mostra o modelo de fluxo para a transação de negócio *ArtifactAnalysis* que define o fluxo de execução dos passos da transação. Observe que existem dois escopos: um escopo protegido e um de suspensão. O primeiro escopo captura a exceção *ArtifatWrongFormatException* lançada pelo passo *ArtifactSubmission* e a envia para o passo *WrongFormatExceptionHandling*. O outro escopo monitora a chegada do evento *WithdrawalEvent* que é gerado pelo usuário (autor) quando ele pretende retirar o artefato. O restante é a modelagem de fluxo tradicional. É importante dizer que, para cada subtipo de artefato (*Paper, Workshop e Talk*), o subtipo *ArtifactSubmission* correspondente substitui o passo *ArtifactSubmission* neste modelo de fluxo.

Finalmente, para ilustrar a interação entre os paradigmas de transação e navegação, imagine que o designer queira permitir ao revisor, quando este estiver executando a transação Web *ArtifactReviewSubmission*, navegar para outros artefatos que ele reviu anteriormente. Para conseguir isso, temos três modelos independentes: o modelo de transação, onde está a transação Web *ArtifactReviewSubmission*, o modelo de navegação, onde está o nó de navegação de artefato e um modelo de interface, onde há um elemento que, quando ativado, envia dois eventos de interface, um para iniciar a transação Web *ArtifactReviewSubmission* e outro para alterar o estado de navegação para o nó de navegação do artefato. A interface resultante é a composição dos dois estados de execução de "transação": *ArtifactReviewSubmission* e navegação, que pode ser interpretada como uma transação Web especial. Desta forma, o revisor pode navegar por meio do nó de navegação de artefato, suspendendo a transação Web *ArtifactReviewSubmission*, que pode ser retomada mais tarde no ponto que foi suspensa.

Ao final deste exemplo fica claro que ao seguir o *roadmap* sugerido pelo meta-modelo, o projetista acabar por definir, explicitamente, todas as propriedades e fluxo de cada transação, sendo levado a pensar em várias nuances importantes que outrora passavam desapercebidas, vindo a tona apenas na fase de testes (ou produção), na forma de erro (ou defeito).

[4] Notação da DSL: retângulo arredondado - TriggerTransactionNode ou CallOperationNode, retângulo arredondado tracejado - Scope, seta - Edge; diamante - Decision/Merge Nodes; barra - Fork/Join Nodes, círculo sólido - InitialNode, círculo com X - FlowFinalNode; círculos sólido/branco - FinalNode; retângulo seta p/ cima - ThrowNode; retângulo seta p/ baixo - CatchNode, Pentágono - SendEventNode, retângulo côncavo - ReceiveEventNode; pequeno retângulo - ObjectBufferNode.

Já em termos de arquitetura de software, poder-se-ia vislumbrar um componente controlador de execução de transação - "*Transaction Engine*" - cuja lógica já estaria definida pelos modelos: "fluxo de execução" - modelo de fluxo pré-carregado (*loaded*); "cache dos parâmetros (http)" - atributos dos objetos transação reificados e "controle de concorrência e gerência de locks" - propriedades ACID das transações.

4. TRABALHOS RELACIONADOS

Métodos conhecidos de design de aplicações Web têm evoluído para contemplar transações: Brambilla et al [2], OO-H e UWE [6], UWA [1], não detalhados aqui por falta de espaço. Nenhum deles trata navegação como uma transação Web especial, misturando os aspectos de transação e navegação, sem uma clara separação entre especificação de transação e suas possíveis realizações (*enactments*). Nenhuma destas propostas contém todas as primitivas presentes no meta-modelo deste trabalho.

5. CONSIDERAÇÕES FINAIS

Neste documento, foi apresentada uma abordagem de reificação para modelar transações de negócios e Web, abrangendo as perspectivas estrutural e comportamental de transações. Foram propostos uma notação gráfica e um meta-modelo, definindo uma DSL para modelar transações. Pretende-se aprimorar esta DSL em vários protótipos e criar outras DSLs (diálogo, segurança, etc.), combinando-as, sem misturar os diversos aspectos envolvidos.

Reconhecimento: Esta pesquisa foi parcialmente financiada pelo CNPq, número de processo 142192/2007-4 e 302.352/85.6.

6. REFERÊNCIAS

[1] Baresi, L.: Ubiquitous Web applications. In: The eBusiness and eWork Conference (e2002), Prague, Czech Republic, 2002.

[2] Brambilla, M., Ceri, S., Fraternali, P., and Manolescu, I. Process Modeling in Web Applications. ACM Transactions on Software Engineering and Methodology (TOSEM), 2006

[3] Distante, D., and Tilley, S. Conceptual Modeling of Web Application Transactions: Towards a Revised and Extended Version of the UWA Transaction Design Model. Proceedings of the 11th International Multi-Media Modeling Conference (MMM 2005), CA: IEEE Computer Society Press, Melbourne, Australia, Los Alamitos, January 2005.

[4] Gioldasis, N., and Christodoulakis, S. UTML: Unified Transaction Modeling Language. The 3rd International Conference on Web Information Systems Engineering (WISE 2002), Singapore, December 2002.

[5] Hee, K.V., and Aalst, W.V.D. Workflow Management: Models, Methods, and Systems, MIT Press, 2002, ISBN: 0-262-01189-1.

[6] Koch, N., Kraus, A., Cachero, C., and Meliá, S. Modeling Web Business Processes with OO-H and UWE. Proceedings of 3rd International Workshop on Web Oriented Software Technology (IWWOST 2003), Oviedo, Spain, July 2003

[7] Papazoglou, M. P. Web Services e Business Transactions. World Wide Web: Internet and Web Information Systems, March 2003, 6(1):49-91.